

EMAX6SPC-0001
Ver.0.1: Feb. 1 2016
Ver.1.3: Jan. 1 2022
Ver.1.4: Nov. 1 2022

EMAX6/ZYNQ64 (IMAX2) Architecture Handbook
– Interposed Memory Accelerator eXtension –

Nara Institute of Science and Technology
Computing Architecture Laboratory
Accelerator Group

Copyright Yasuhiko NAKASHIMA. All Rights Reserved.

Table of contents

1 IMAX Hardware	8
1.1 Background	8
1.2 History of CGRA	12
1.3 Policy of IMAX	13
1.4 省面積化のための列方向マルチスレッディング	15
1.5 起動オーバヘッド削減のための多重ループ一括実行	17
1.6 ローカルメモリ (LMM) の様々な使用方法	18
1.7 リング構造の UNIT 間接続と命令写像位置シフト機能	21
1.8 メモリとしてのカスケード接続機能	22
1.9 詳細構造と動作	25
1.10 シミュレータ	30
1.11 FPGA によるマルチチッププロトタイプ	30
1.12 総合評価	31
2 IMAX Software	33
2.1 主記憶上に写像される IMAX インタフェース	33
2.2 Control Registers	37
2.3 Programming model	39
2.4 Templates of instructions	40
2.5 List of instructions	44
2.6 Overlapping of prefetch and drain	46
2.7 マルチチップ対応	47
2.8 動的 DMA 連結と DMA 起動タイミング	47
3 Examples	49
3.1 Tuning of applications	49
3.1.1 C によるアルゴリズムの記述	49
3.1.2 CUDA によるアルゴリズムの記述	49
3.1.3 IMAX 向け C 言語記述 (毎サイクル 1 画素を出力)	50
3.1.4 IMAX 向け C 言語記述 (毎サイクル 2 画素を出力)	51
3.1.5 IMAX 向け C 言語記述 (複数段利用, 2 重ループ一括実行, マルチチップ実行)	52
3.2 Basics	53
3.2.1 FFT	53
3.2.2 Merge sort	57
3.2.3 文字列検索	59
3.2.4 16x16 置み込み	62
3.2.5 VBGMM_Logsum	64
3.2.6 Stochastic sgemm00	65
3.2.7 Sparse matrix multiplication	68

3.2.8	Sparse matrix compression	73
3.3	2D-imaging	75
3.3.1	Tone_curve	75
3.3.2	Hokan1 with stencil	78
3.3.3	Hokan2 with stencil	81
3.3.4	Hokan3 with stencil	84
3.3.5	Expand4k with stencil	86
3.3.6	Unsharp with stencil	90
3.3.7	Blur with stencil	93
3.3.8	Edge with stencil	96
3.3.9	Stereo with stencil	99
3.4	3D-floating-point	102
3.4.1	Grapes with stencil	102
3.4.2	Jacobi with stencil	105
3.4.3	Fd6 with stencil	108
3.4.4	Resid with stencil	111
3.4.5	Wave2d with stencil	114
3.5	実用カーネル	117
3.5.1	3x3 置み込み	117
3.5.2	行列積	123
3.5.3	逆行列 (1/3)	128
3.5.4	逆行列 (2/3)	132
3.5.5	逆行列 (3/3)	135
3.5.6	Lightfield レンダリング	138
3.5.7	Lightfield 距離画像生成	141
3.6	Graph processing on EMAX5	145
3.6.1	Triangle counting kernel0 with TCU	145
3.6.2	Triangle counting kernel1 with TCU	148
3.7	Graph processing on IMAX2	150
3.7.1	Triangle counting kernel0	150
3.7.2	Triangle counting kernel1	151
3.8	Image Recognition (ssim)	152
3.8.1	Cnn5x5	153
3.8.2	Cnn3x3	155
3.8.3	Cnn2x2	157
3.8.4	Sgemm00	159
3.8.5	Back_g_ker	161
3.8.6	Back_in	164
3.9	Crypto	167
3.9.1	SHA256	167
A	Appendix	170
A.1	Prototype systems	170
A.2	Anatomy of IMAX	171
A.3	Basic loop structure	178
A.4	Basic data flow	182
A.5	Compilation of IMAX	193
A.6	References	197

A.7 Publications	198
B Q&A	205
B.1 概要	205
B.1.1 CGRA とは何ですか	205
B.1.2 EMAX6 と IMAX はどこか違うのですか	205
B.1.3 実際速いんですか	205
B.2 プログラマとしての使い方	205
B.2.1 自分で書いたプログラムが全くコンパイルできないのですが	205
B.2.2 自分で書いたプログラムが全く動かないのですが	205
B.2.3 自分で書いたプログラムの実行結果がおかしいのですが	206
B.3 ハード設計者としての使い方	206
B.3.1 ハードが動いていないようですが	206
B.3.2 演算機能を追加したいのですが	206
B.3.3 機能追加後の csim の調べ方	207
B.4 商売としての使い方	208
B.4.1 設計データとツールを一式欲しいのですが	208

List of Figures

1.1	CPU/GPU と CGRA の命令実行モデルの違い	8
1.2	CPU/GPU と CGRA のメモリアクセスの違い	9
1.3	CGRA の理想的動作	9
1.4	複雑化するアプリケーション	10
1.5	今後 CGRA に要求される機能	10
1.6	VPP から IMAX に至る命令実行モデルの歴史	11
1.7	VPP から IMAX に至る主要機能の変遷	11
1.8	LAPP から IMAX に至る演算器ネットワークの変化	12
1.9	必要とされるメモリ参照パターンと演算	14
1.10	行列積の演算方向	15
1.11	Column multithreading for small footprint and bubble-free execution	16
1.12	IMAX におけるレジスタダブルバッファリングの重要性	16
1.13	Read-modify-write operation	17
1.14	Multilevel loop control	18
1.15	ローカルメモリ (LMM) の様々な使用方法	20
1.16	24 行構成の概要	21
1.17	Cascaded peer-to-peer AXI bus for scalable multichip CGRA	22
1.18	3Chip 構成の動作	24
1.19	IMAX の基本 UNIT 構造	25
1.20	UNIT 内演算器の構成とタイミング	26
1.21	UNIT 内ローカルメモリ (LMM) の構成とタイミング	27
1.22	CPU からのローカルメモリ (LMM) 直接参照	28
1.23	UNIT の機能	29
1.24	4 チップ構成の IMAX	31
1.25	総合評価 1 (15Gbps interface)	31
1.26	総合評価 2 (40Gbps interface)	32
2.1	物理メモリ空間	33
2.2	起動から終結までの手順	35
2.3	論理インターフェースと各行におけるデータ取り込み機構の関係	35
2.4	Control Registers	37
2.5	Configuration Registers	38
2.6	ロード-演算-ストアを基本とする IMAX のプログラミングモデル	39
2.7	レジスタと演算器の接続ネットワーク	44
2.8	プリフェッч / ドレインとのオーバラップ実行例	46
2.9	for 文によるマルチチップと多重ループの記述例	47
2.10	動的 DMA 連結と DMA 起動条件	48
3.1	Tone curve	49

3.2	Tone curve 1 pixel per cycle	50
3.3	Dual tone curve	51
3.4	Tone curve 2 pixels per cycle	51
3.5	FFT	54
3.6	FFT	56
3.7	FFT	58
3.8	String search	59
3.9	文字列検索	61
3.10	16x16 置み込み	63
3.11	Stochastic sgemm00	65
3.12	Stochastic sgemm00	66
3.13	SFMA 1st stage	67
3.14	SFMA 2nd and 3rd stage	67
3.15	Sparse matrix multiplication	68
3.16	Data path for sparse matrix multiplication	69
3.17	Timing chart (single flow)	69
3.18	Sparse matrix multiplication (single flow)	70
3.19	Timing chart (dual flow)	71
3.20	Sparse matrix multiplication (dual flow)	72
3.21	Sparse matrix compression	74
3.22	Tone curve	75
3.23	Tone curve	77
3.24	Hokan	78
3.25	Hokan1	80
3.26	Hokan2	83
3.27	Hokan3	85
3.28	Expand4k	86
3.29	Expand4k	89
3.30	Unsharp	90
3.31	Unsharp	92
3.32	Blur	93
3.33	Blur	95
3.34	Edge	96
3.35	Edge	98
3.36	Stereo	99
3.37	Stereo with stencil	101
3.38	Grapes	102
3.39	Grapes	104
3.40	Jacobi	105
3.41	Jacobi	107
3.42	Fd6	108
3.43	Fd6	110
3.44	Resid	111
3.45	Resid	113
3.46	Wave2d	114
3.47	Wave2d	116
3.48	CNN の写像	117
3.49	Cnn	118

3.50 3x3 畳み込み	121
3.51 Mm	123
3.52 行列積	125
3.53 32bit → 32bit 32bit load と SIMD 演算の組み合わせ	126
3.54 Inverse matrix	128
3.55 逆行列 (1/3)	131
3.56 逆行列 (2/3)	134
3.57 逆行列 (3/3)	137
3.58 Gather	138
3.59 Lightfield レンダリング	140
3.60 Gdepth	141
3.61 Lightfield	141
3.62 Lightfield 距離画像生成	144
3.63 Graph	145
3.64 Triangle counting kernel0 with TCU	147
3.65 Triangle counting kernel1 with TCU	149
3.66 Image recognition (training + inference)	152
3.67 5x5 Convolution	154
3.68 3x3 Convolution	156
3.69 2x2 Convolution	158
3.70 Sgemm00	160
3.71 back_g_ker	161
3.72 back_g_ker	161
3.73 Back propagation to g_ker	163
3.74 back_in	164
3.75 back_in	164
3.76 Back propagation to input	166
3.77 SHA256	169
A.1 Prototype systems	170
A.2 Whole of IMAX2	171
A.3 IMAX2 w/o sparse matrix	172
A.4 IMAX2 w/o transmission registers (TR)	173
A.5 IMAX2 w/o dual port LMM	174
A.6 IMAX2 w/o folding	175
A.7 IMAX2 w/o AXI-IF	176
A.8 IMAX2 w/o multi-threading	177
A.9 Compilation step1	193
A.10 Compilation step2	193
A.11 Compilation step3	194
A.12 Compilation step4	194
A.13 Compilation step5	195
A.14 Compilation step6	195
A.15 Compilation step7	196

List of Tables

1.1 物理メモリインタフェース	23
2.1 論理インタフェース	34
2.2 Memory operations	44
2.3 ALU operations	45

Chapter 1

IMAX Hardware

1.1 Background

IoT, ビッグデータ, AI, ロボットが注目される一方, 持続的発展に必須の低電力計算基盤技術は進展速度が極めて遅い. 主因は, 半導体微細化限界により表面化した消費電力-プログラマビリティ間の強いトレードオフ関係にある. JST 低炭素社会戦略センター (LCS) の報告書は, CPU や GPGPU 等ノイマン型に依存する状況が続ければ, 2050 年にはデータセンターの消費電力が供給可能電力を上回ると予測する. データセンター集中処理にせよエッジコンピューティングにせよ, トレードオフを精査し, 低電力計算基盤の導入を進めなければ, AI やブロックチェインを柱とする次世代持続的社会システムは画餅に帰す. 様々な物理現象を利用する数多のアナログ型計算機構がサイエンスとして探求される中, 大規模化・安定運用に要する電力も含め, 次世代低電力計算基盤に至るほぼ唯一の道は, プログラムをデータフローに細分化しハードウェアに写像して電力効率を 2 柄改善可能な非ノイマン型の汎用性・プログラマビリティ向上である.

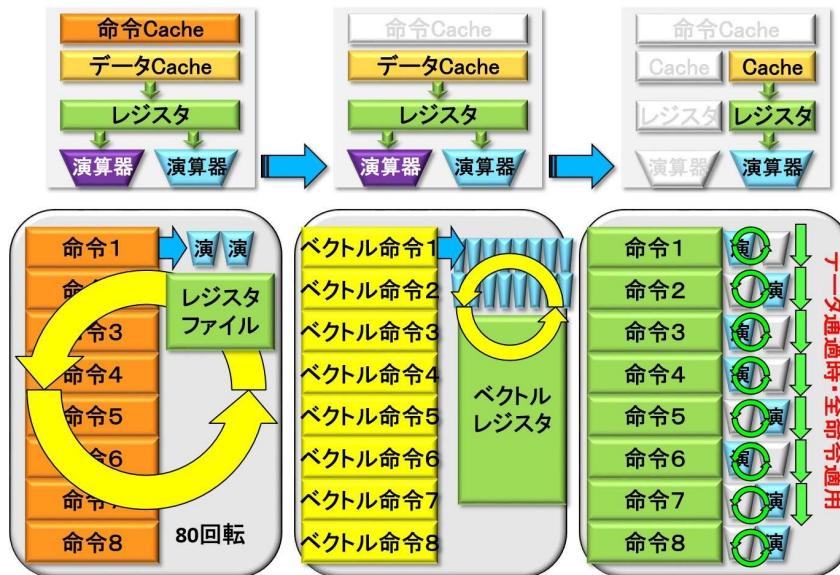


Figure 1.1: CPU/GPU と CGRA の命令実行モデルの違い

機械語命令が演算器とメモリを逐次制御するノイマン型計算基盤は, 半導体微細化と並列処理高度化の両輪が性能向上を支えてきた. 前者は物理限界に近付き後者が追求された. GPGPU は従来型プログラマビリティを維持しつつ並列処理を行うために, 大規模レジスタファイルとマルチスレッディングによる 1000 サイクルを超える主記憶遅延隠蔽機能を備える. ただし, 主記憶参照順序を規則化するチューニングが難しく, 要求性能達成のために過大な演算能力と電力を必要とする. 一方, 1982 年に H.T.Kung らが提唱した Coarse Grain Reconfigurable Architecture(CGRA) は, 最内ループの命令列を 2 次元配置の演算器ネットワークに写像して主記憶データを流し込む非ノイマン型である (図 1.1, 1.2). 多くの国内企業が着手するも

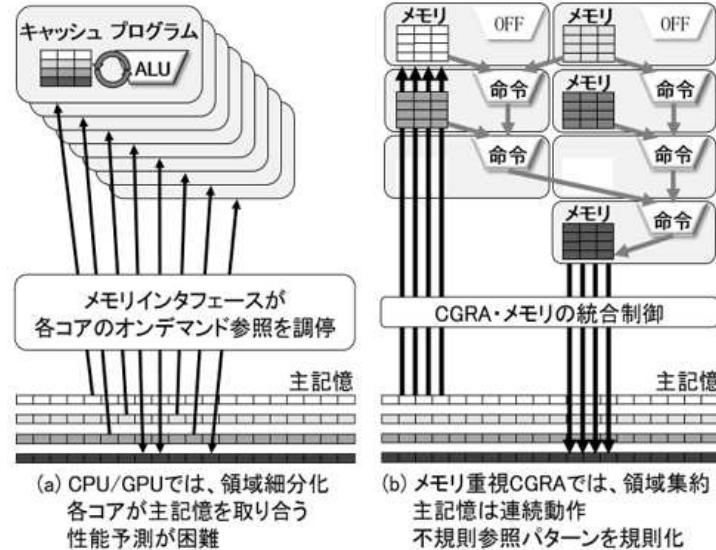


Figure 1.2: CPU/GPU と CGRA のメモリアクセスの違い

プログラマビリティの壁により撤退する中, Google 社の TPU や Wave computing 社の DPU は AI 応用に絞り高効率化に成功している. 研究室では 2008 年に VLIW 互換の LAPP を開発し, その後, 64 組のニアメモリ高機能演算器と多重ループの統合制御により面積効率(演算性能/面積)を飛躍的に高めた IMAX を開発した. 多数のコアを SIMD 動作させる GPGPU 方式が, メモリ参照を連続アドレスに統合するコアアレンジングと豪華なメモリバスを必要とするのに対し, 本方式は多数の演算器を縦に多数連結することでメモリバスの軽量化と低電力化を達成している.

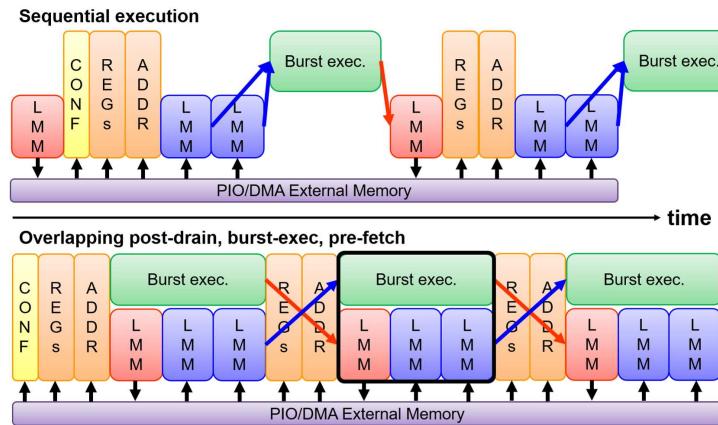
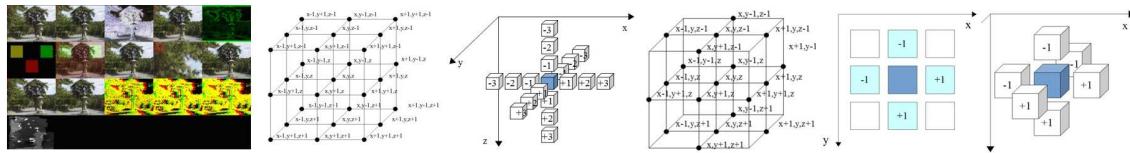


Figure 1.3: CGRA の理想的動作

CGRa は膨大な演算を一度に実行する仕組みであるため, 効率よく利用するには, 図 1.3 のように, 直前の演算結果の主記憶への追い出し, 演算, および, 次の演算に必要なデータの主記憶からの取り込みをオーバラップさせ, 演算器を稼働し続けることが極めて重要である. また, 図 1.4 のように, アプリケーションは急速に複雑化しており, CGRA には, 図 1.5 に示す多くの機能が要求されている.

IMAX に至る経緯を図 1.6, 1.7 に示す. 脱 IBM 互換の (a) 並列ベクトル VPP シリーズでは, 海外メーカーがスーパースカラに傾く中, 電力効率最大化のため VLIW を提案し採用された. VLIW+ベクトル演算機構は, メモリパイプ M と演算パイプ E のバースト動作が CGRA に酷似するものの, プログラムに対する構造的制約がない特長を有する. (b) OROCHI は, 命令デコーダ D が VLIW バッファ B に ARM 命令を動的分解投入し, キャッシュ C を共有する VLIW 共用型低電力スーパースカラである. VLIW の 2 次元拡張技術は CGRA 構成技術の基礎であり, 初の VLIW 互換 CGRA である (c)LAPP が継承した. (d) EMAX 以降は, 演算器 E とローカルメモリ LM のリング接続による LM 最大再利用や, 明示的制御によるデータフロー干渉排除等により, 離散ステンシル計算やグラフ処理の高効率化を達成した. Google 社の TPU (整



More complicated memory access for light field, graph, string search, AI, ...

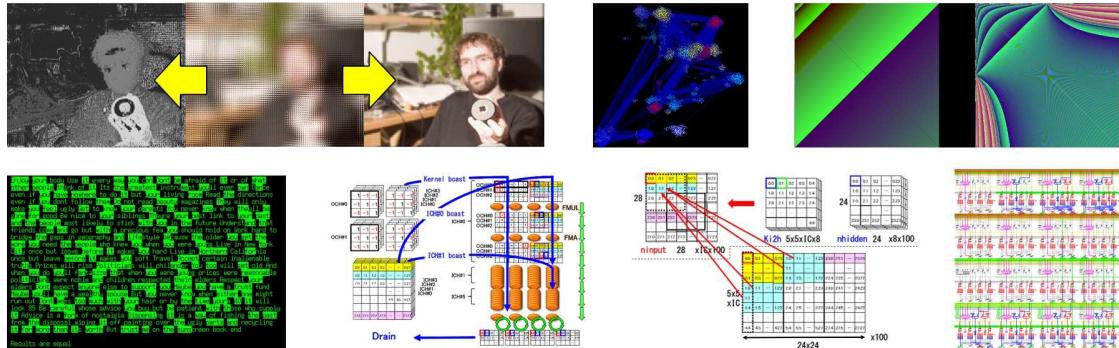


Figure.1.4: 複雑化するアプリケーション

Metrics	Requirements	IMAX2
Application	HPC, Tensor, Multimedia (8bit/16bit), String-search, ...	✓
Speed / area	100x better than SIMD/SIMT (less programmability)	✓
Speed / memory bandwidth	Minimum DDR access (full reuse of local memory)	✓
FPU utilization	No stall in accumulation	✓
Physical registers	Minimum registers for multithreading	✓
Overhead for transmission	Broadcasting + Overlapped prefetching and drain	✓
Debuggability	Compatible w/ low level language such as C	✓
Scalability and Yield	Daisy chain + chiplet (to reduce price)	✓
Connectivity	Serial slave interface (to reduce I/O cost)	✓
Compilation time	Several seconds (to save time)	✓

Figure.1.5: 今後 CGRA に要求される機能

数演算のみ)と同時期の (e)IMAX では、物理 1 列 x64 行を論理 4 列 x64 行に仮想化する多重実行を考案し、LM の最大再利用と併せて、累算型浮動小数点積和演算 E の面積効率(演算性能/面積)を劇的に改善(28nm 想定で組込み GPGPU 比 250 倍)した。 (f)Tandem CGRA は、仮想化機構によりプログラマビリティを向上させる構成である。

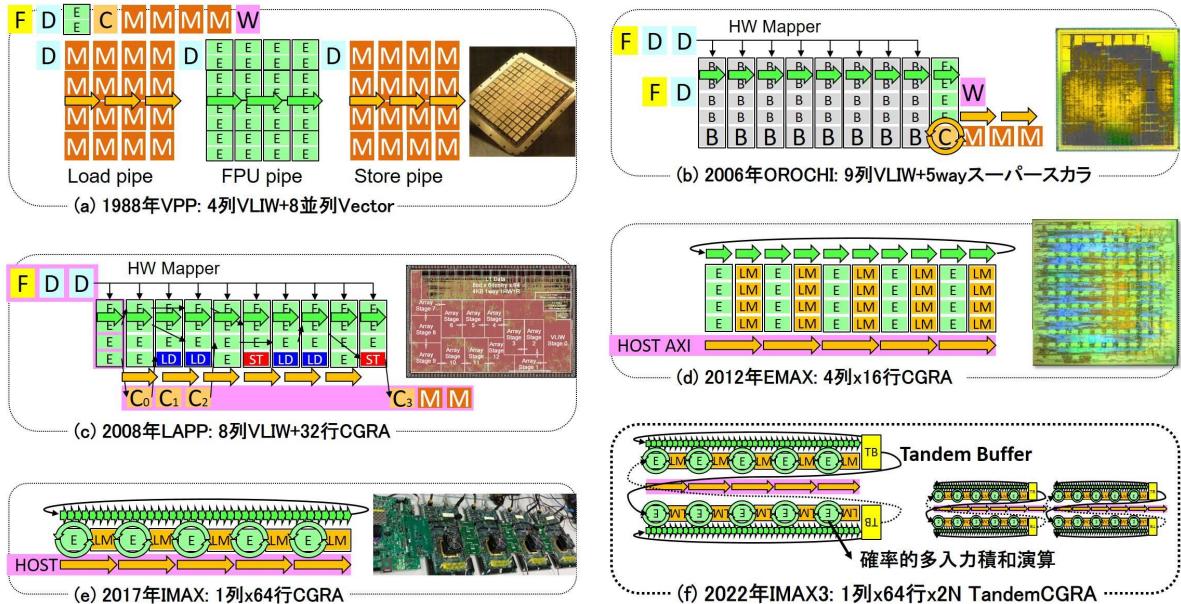


Figure 1.6: VPP から IMAX に至る命令実行モデルの歴史

	Type	# of ops / chip	# of load+store	Compatibility	Graph proc.	Dual-port	SIMD	FPU+Multi-threading	Chiplet
1988 VPP	VLIW+vector	8+8	8+8						
2006 OROCHI	VLIW+mapper	w8	1						
2008 LAPP	VLIW+compatible CGRA	w8 *h13	w1 *h13	Instr. Asm.(C)	x	x	x	x	○
2012 EMAX2	CGRA (AXI master)	w4 *h16	w4 *h16	Asm.(C)	x	x	x	△FPU	×
2014 EMAX4	CGRA (AXI master)	w4 *h16	w4 *h16	Asm.(C)	○	x	x	△FPU	×
2015 EMAX5	CGRA (AXI master)	w8 *h16	w8 *h16	Asm.(C)	○	○	○	△FPU	×
2017 IMAX	CGRA (AXI slave 15Gbps)	w24 *h64	w16 *h64	Asm.(C)	x	○	○	○8beat	○
2019 IMAX2	CGRA (AXI slave 40Gbps)	w24 *h64	w16 *h64	Asm.(C)	○	○	○	○8beat	○

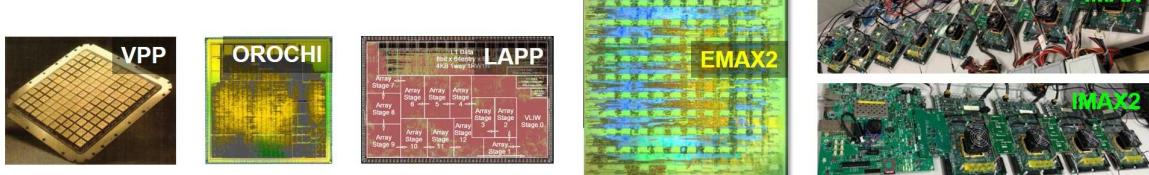


Figure 1.7: VPP から IMAX に至る主要機能の変遷

1.2 History of CGRA

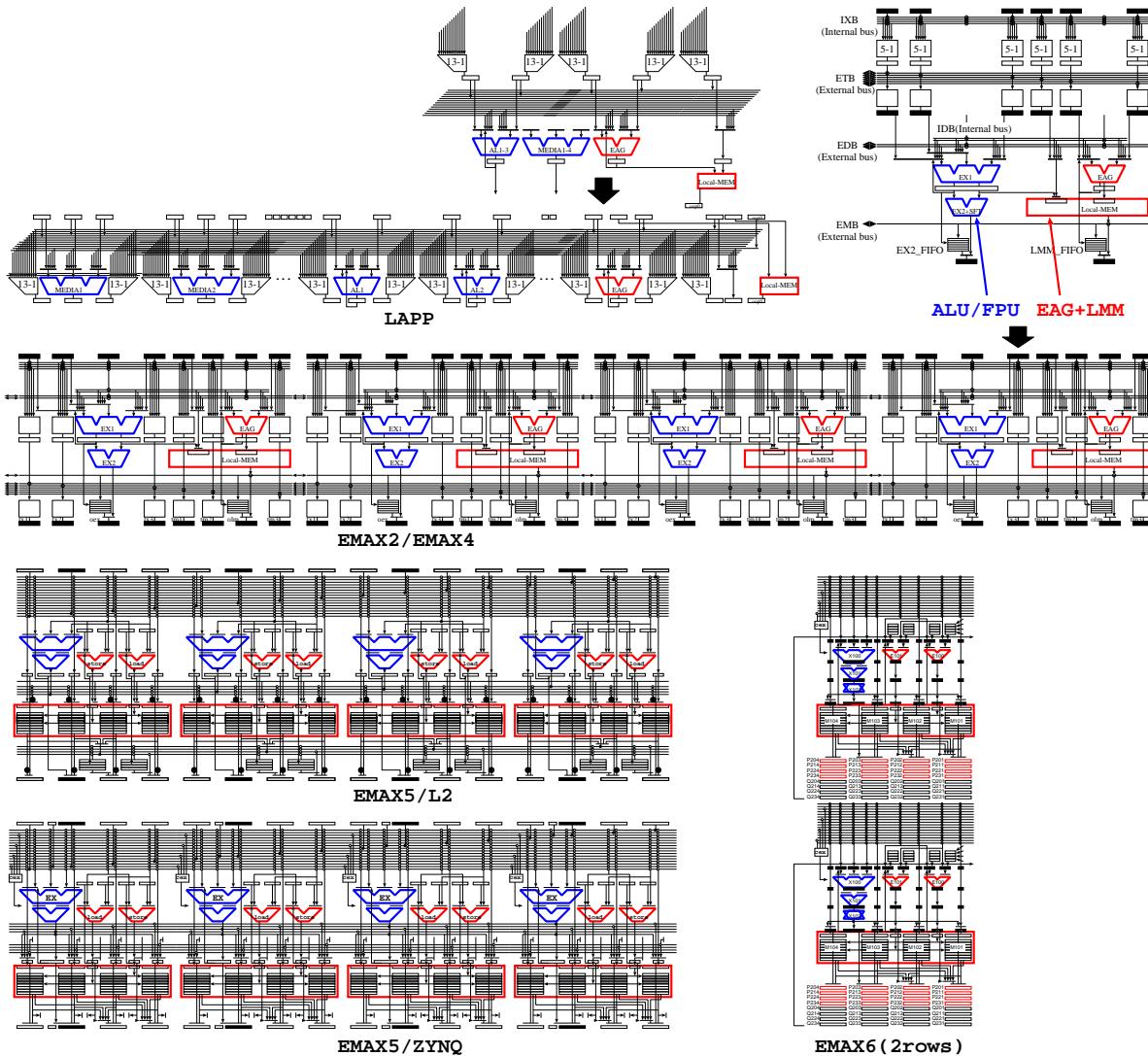


Figure 1.8: LAPP から IMAX に至る演算器ネットワークの変化

図 1.8 に、当グループが開発してきた CGRA の 1 行分の構造を示す。LAPP は VLIW 互換 CGRA であったものの、FIFO により構成される各行のローカルメモリ (LMM) 容量は高々 16 要素 x4 列と小さく、次数 (1 辺の長さ) が大きなステンシル計算には対応できなかった。EMAX2 では、VLIW 互換を放棄する代わりに各演算器に専用の中容量 LMM と FIFO を配置し、多くのステンシル計算を写像できるようにした。ただし、LMM 数を増やしたことにより行間レジスタ数が 32 本に大幅増加した。このため、レジスタから演算器出力までに 4 サイクルを要した。EMAX4 では、さらにグラフ処理に必要となるトランザクション機構を追加した。EMAX2 から EMAX4 への変更点は以下の通りである。

- 複数条件の組合せによる条件付実行の追加
- トランザクション機能の追加
- 各段と外部メモリの間のデータバスを 64bit*1 から 32bit*4 に拡張

EMAX5/L2 では、SIMD 演算機構を追加するとともに、ALU と EAG の入力レジスタを共用することで行間レジスタ数を 16 本に半減し、レジスタから演算器出力までを 2 サイクルとした。また、外部メモリバス-LMM 間スループットを向上させた。EMAX5/ZYNQ では、FIFO 初期化オーバヘッドを削減するために FIFO を削除し、代わりに、外部メモリバスから複数列の LMM へ同時に書き込むパスを追加した。これにより、複数の外部メモリバスを使用して多数の LMM を同時に初期化することが可能となった。

EMAX4 から EMAX5/ZYNQ への変更点は以下の通りである。

- 単精度浮動小数点演算を 32bit*2 演算へ SIMD 化
- 32bit 整数演算を 32bit*2 整数演算へ SIMD 化
- 8/16bit メディア演算を 4/2 倍幅 SIMD から 8/4 倍幅 SIMD へ増強
- 各 UNIT のローカルメモリ (LMM) バス幅を 32bit*1 から 64bit*4 に拡張し, LMM-外部メモリ間のスループットを 8 倍に拡張
- 各段と外部メモリの接続を 32bit*4*1 チャネルから 64bit*4*4 チャネルに拡張し, 各段-外部メモリ間のスループットを 8 倍に拡張 (conf/regv/lmmi 初期化オーバヘッドを 1/8 に削減)
- 外部メモリ直接参照のための Dual-Port-LMM の採用および CGRA 基本構成の変更
- 最大 3UNIT の LMM から 3 入力 SIMD へデータ供給しつつ LMM に出力できる UNIT 間バスネットワーク
- UNITあたりの伝搬レジスタ数を 8 から 4 に削減し, 代わりに伝搬レジスタから演算器へはクロスバスイッチを配置
- LMM の出力を水平方向に伝搬する FIFO を削除し, 代わりに外部メモリバスから同一段の LMM へブロードキャストする機構を装備 (FIFO の初期化が不要)

IMAX では, CGRA の一般的な欠点である配線の多さ, および, 自己ループ演算時の演算回路使用率の低さを改善するために, 水平方向の演算には互いに依存関係がないことを利用し, 複数列の演算機能を 1 列に束ねるマルチスレッディングを導入した. 論理的には複数列の演算機能 (論理 UNIT) を実現しつつ, 物理的には 1 列の演算器群により実装することにより, 演算器数・配線量削減, 演算器使用効率向上, 水平方向ブロードキャストの不要化, LMM 冗長利用の不要化が可能となった. また, CPU への接続方法を AXI-SLAVE に変更し, LMM への書き込みは, UNIT 毎に設けた vAddr-range に従い, 各 UNIT が自律的に取り込む方法に変更した. これにより, 垂直方向ブロードキャストも可能になった. さらに, UNIT の出力を入力に戻すパスを追加することにより, 同一 LMM に対する read-modify-write を可能とし, 置き込み演算や逆行列計算に必要となる LMM アキュムレートの写像を可能とした. なお, 複数行の LMM に対して連続する vAddr-range を設定しておき, 複数の LMM に対して同一ストアを行うことにより, アキュムレート空間を数倍に増やすことも可能となった. EMAX5/ZYNQ から IMAX への変更点は以下の通りである.

- EMAX5/ZYNQ とのプログラム互換性を維持しながら物理演算器数および LMM 数を削減
- AXI-SLAVE 化, lmring の 8 行 8 列構成, および, 垂直方向ブロードキャスト機能によるホスト主記憶 ⇌ LMM 転送効率の向上
- 同一 LMM を使用する read-modify-write 機能とアキュムレート機能
- 同一 LMM をダブルバッファとして使用するタンデム機能 (FFT と多段マージソートが使用)
- デュアルポート LMM を利用したロード結果比較およびアドレス更新機能 (多段マージソートと疎行列積が使用)
- 要素に位置情報を付加して圧縮した疎行列どうしの行列積演算機能
- マルチチップ対応およびマルチチップへの写像も含めた多重ループ一括実行機能
- パースト長が最大 8(256bit 幅の場合) である AXI3 の DMA-READ 高速化のための複数トランザクション単一化 (AXI4 化) およびバッファリング機能
- デュアル EAG を利用した非 8 バイト境界からの 64bit ロード
- SIMD 化を容易にする, 32bit/8bit ロードデータの 64bit レジスタ拡張
- 動的 DMA 連結による, 複数 LMM に対する連続データ転送

1.3 Policy of IMAX

CGRa の課題は, (1) 二次元格子構造のまま実装すると長距離配線や配線混雑が動作周波数および面積効率の低下をもたらすため省面積化の方策が必要であること, (2) 浮動小数点アキュムレート演算やローカルメモリに対する read-modify-write のように本質的に複数サイクルを要する命令があってもパイプライン実行を妨げない仕組みが必要であること, (3) 最内ループの高速化だけではベクトル長の短い行列積や

畳み込み演算の高速化に向かないため、起動オーバヘッド削減のための多重ループ一括実行機能が必要であること、(4) ローカルメモリの再利用効率を上げて外部メモリ参照を削減するために、静的解析が困難なデータ参照パターンの変動に追随できる命令写像位置シフト機能が必要であること、(5) 高機能な DMA マスター機能の内蔵を必要とせず簡素な DMA スレーブメモリとして高効率を達成できること、(6) 外部メモリバス (AXI) の増設を必要とせず、スレーブメモリの機能を生かしたカスケーディングによりスケーラブルな性能を容易に得られることである。

IMAX の第 1 のポイントは、CGRA の名称により一般に知られている 2 次元構成とは大きく異なり、低コストが要求される IoT エッジ向けに物理構造を 1 次元（リニアアレイ）としつつ高性能を維持した点である。第 2 のポイントは、メモリ機能を内部に融合し、ホストからメモリとして使用するため、通常のアクセラレータがメモリ間転送オーバヘッドに苦しむのに対して優位な点、および、メモリであることを利用したカスケード接続が容易な点である。第 3 のポイントは、メモリ機能と演算機能を一体的に取り扱う直感的プログラミングのために様々なデータパスを備え、特に、通常の CGRA が得意とする、同一ローカルメモリに対する read-modify-write や多重ループ制御をシンプルな構造により可能とした点である。第 4 のポイントは、データパスの工夫により、FPGA のような探索的コンパイルを必要とせず、コンパイル速度が高速な点である。

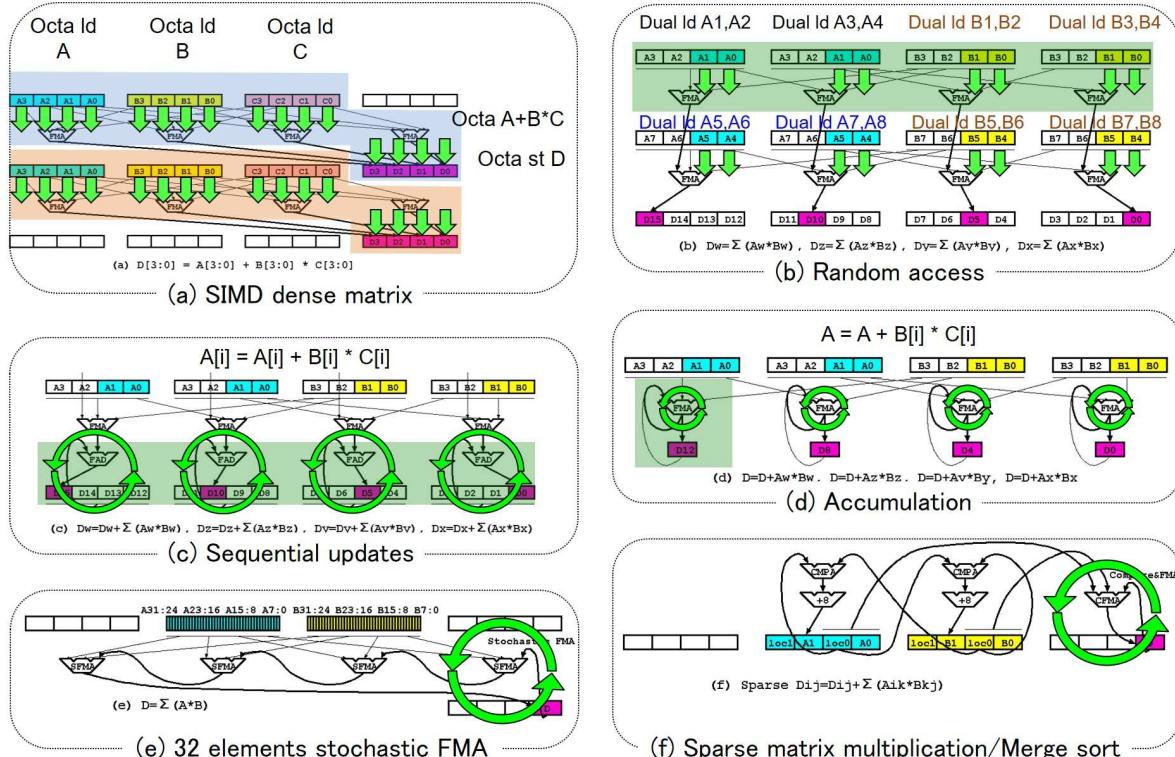


Figure 1.9: 必要とされるメモリ参照パターンと演算

IMAX には、1 列 x64 行の物理構造に対して最大 10 命令 (2 演算 x3stage + ロード x2 + ストア x2) x4 列 x64 行 = 2560 命令を一度に写像できる。IMAX は基本 unit を 1 列 x64 行 (後述) 配置する物理構造を有するものの、プログラミング時のハードウェアモデルは、4 列 x64 行である。各 unit は 3 段パイプラインの 3 入力単精度浮動小数点加減乗算器を 2 組備え、64 ビット幅のレジスタを使用して 2 つの演算を同時実行する。また、パイプラインの初段は 8/16/32 ビット単位の各種マルチメディア演算および固定小数点演算、中段は論理演算、後段はシフト演算等を行うことができる。各 unit は 64KB デュアルポートメモリを備えており、図 1.9 に示す様々なメモリ参照パターンに対応できる。(a) は、1 行のローカルメモリ (LMM) に格納された A,B,C を 4 ワードずつ読み出し、4 つの演算器により SIMD 演算を行った後、演算結果 D を次段の LMM に格納する組を 2 セット写像した状態である。A,B,C は外部メモリから供給され、D は外部メモリに取り出される。(b) は、縦方向に畳み込み演算を行う写像である。デュアルポートメモリから各列の演算器に A と B を供給し、4 系統の畳み込み演算を行っている。(c) は、LMM に 4 系統の D を累算する

写像である。一般的な CGRA とは異なり、全体としてはデータを下流へ伝搬しつつ、累算を行うことができる。(d) は、同様に LMM に 4 系統の D を累算する写像である。ただし、格納先が固定され、LMM への read-modify-write により、何度も更新を行う場合に対応する。(e) は UNIT 内多入力積和演算向けの拡張機能である。(f) は疎行列の行列積やマージソートである。

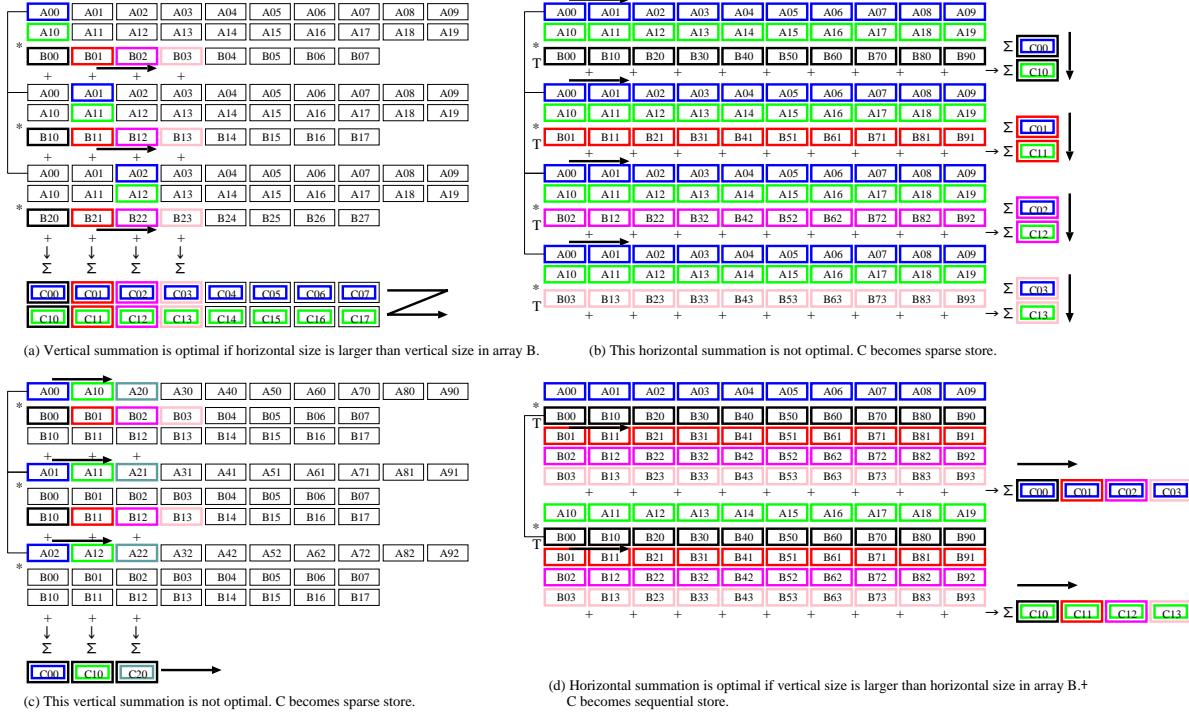


Figure 1.10: 行列積の演算方向

例えば行列積の場合、図 1.10 に示すように、最適な累算方向は配列 B の形状によって異なる。累算方向が縦 (a) の場合、図 1.9(b) および図 1.9(c) が必要となる。累算方向が横の (b) は配列 C へのストアが不連続となるため最適ではない。同様に、累算方向が縦の (c) は配列 C へのストアが不連続となるため最適ではない。複数行の B を束ねる (d) の場合、図 1.9(d) または図 1.9(e) が必要となる。改めて見ると、(a) では配列 A 複数行の垂直方向ブロードキャスト、(d) では転置後の配列 B 複数行の垂直方向ブロードキャストを必要とする点が異なるものの、配列 C へのストアは連続領域となる。

1.4 省面積化のための列方向マルチスレッディング

課題 (1) と (2) に対応するために、列方向マルチスレッディングを導入した。図 1.11(a) に、導入前の一般的な unit 構成を示す。一般に CGRA は、前後の演算との同期コストを削減するため、各演算器が毎サイクル演算結果を出力する前提で設計される。このため、浮動小数点アキュムレートなど演算時間の長い機能をパイプライン化することができず面積効率が悪化する。また、LMM から多数の読み出しを行うために、同一内容を保持する複数の LMM を配置して多ポート化する必要があり、面積効率悪化の要因となる。図 1.11(b) は、マルチスレッディング (W=4) 適用後の unit 構成である。1 つの unit が 4 サイクルを使用して論理的に (a) と同じ機能を実現する。Reg#0-15 を選択する 16-to-1 セレクタを含めて演算器を 4 段パイプライン化し水平方向 4 個の演算を時分割実行すると、浮動小数点アキュムレートが存在しても同一性能を維持でき、面積効率を 4 倍に改善できる。ただし、(a) では 1 つの unit における演算遅延時間が 2 サイクルであるのに対し、(b) では 8 サイクルである。また、4 サイクルかけて出そろう前段 unit の演算結果や LMM 読み出しデータの全てを次段 unit が参照するために、Reg#0-15 のダブルバッファリングが必要となる (grp.A および grp.B)。LMM に関しては、2 ポート LMM の 4 サイクル動作により 8 ポート化すると、水平方向 4 個の LMM に全て重複がある場合、面積効率を 4 倍に改善できる。重複がない場合は LMM の容量を分割使用するためメモリ本体の効率は変わらないものの、アドレス生成器は削減できる。

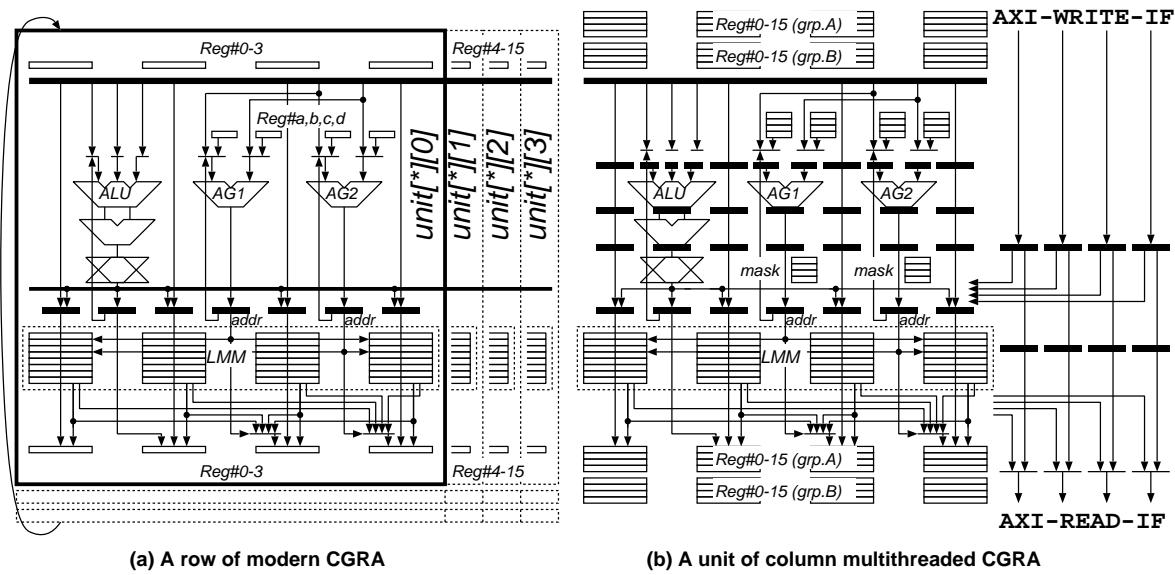


Figure 1.11: Column multithreading for small footprint and bubble-free execution

AXI-WRITE-IF および AXI-READ-IF は、演算用データパスが LMM の load/store ポートを使わないサイクルを利用して LMM を参照する。

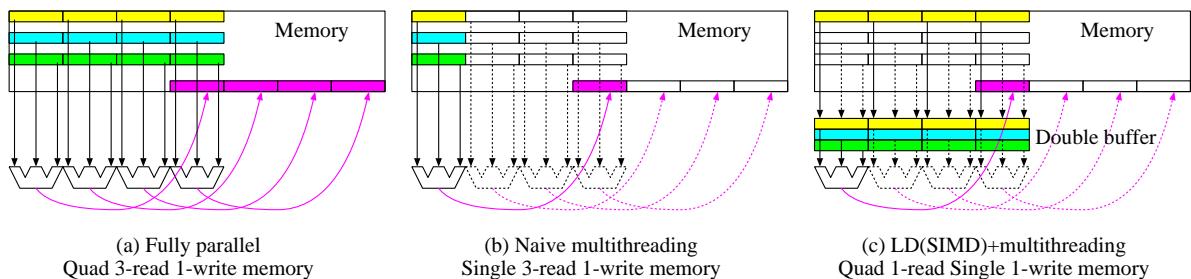


Figure 1.12: IMAX におけるレジスタダブルバッファリングの重要性

図 1.12 に、IMAX におけるダブルバッファの重要性を示す。 (a) は 4 列構成の一般的な CGRA におけるメモリと演算器の関係である。メモリの 3 箇所から各々 4 ワードを読み出し、並列動作する 4 演算器の出力(4 ワード分)をメモリに書き込むために、3-read 1-write メモリが必要である。4 段パイプライン構成の浮動小数点演算器の場合、アドレス間に依存関係がなければスループットは低下しないものの、累算のように依存関係があると、入力が自身の演算結果を待ち合わせるためパイプラインが停止し、性能は 1/4 に低下する。パイプラインを止めないためには、(b) のように 1 演算器のみを用いて 4way-マルチスレッディング化すればよい。ただし、演算器は削減できても、メモリのポート数は削減できない。(c) はポート数削減のために SIMD-LD を復活させた構成である。プログラムは SIMD-LD, SIMD-FMA, SIMD-ST の組として記述する。ハードウェアは、4 サイクルのうち 3 サイクルを使用して 3 箇所から SIMD-LD による読み出しを行う。一旦、ダブルバッファに保存することで、4 サイクルの全てにおいて各演算に必要な 3 入力が常時参照可能となる。マルチスレッディングにより SIMD-FMA は時分割実行され、演算結果が順にメモリに格納される。すなわち、SIMD-FMA と SIMD-ST は実際には SIMD 動作しないものの、ハードウェアの性能は 100% 発揮できる。

図 1.13 に read-modify-write operation と unit の対応を示す。論理的に 4 列分の read-modify-write が物理的には单一 unit に写像される。前段の演算器による累算結果 (sum0-3) が一旦 unit 下端のレジスタに送られ、load 結果 (o0-3) とタイミングを合わせて演算器入力へフィードバックされ、加算結果 (s0-3) が LMM に store される。

1. load (&o0, op0, offset); ... load (&o3, op3, offset);
2. add (&s0, o0, sum0); ... add (&s3, o3, sum3);
3. store (&s0, op0, offset); ... store (&s3, op3, offset);

(a) *(op+offset) += sum for proposed CGRA

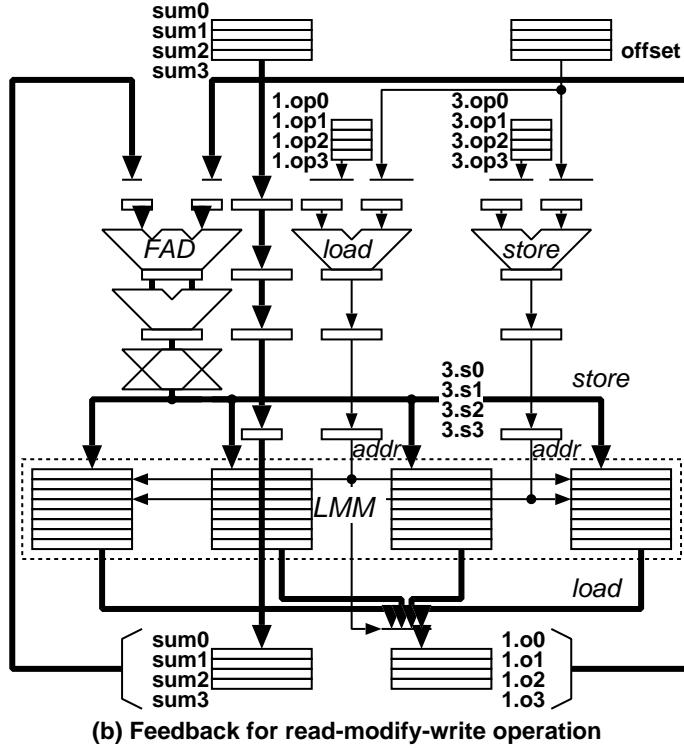


Figure.1.13: Read-modify-write operation

1.5 起動オーバヘッド削減のための多重ループ一括実行

課題（3）に対応するために、多重ループ一括実行を導入した。CGRAはバースト演算起動前に命令写像とレジスタ初期化を必要とするため、なるべくバースト演算時間を長くし起動オーバヘッドを減らしたい。しかし、バースト演算時間はLMMに格納可能なデータ量により決まり、LMMの容量にはパイプライン演算器の動作周波数を落さない上限がある。経験的に3ステージからなる単精度浮動小数点演算パイプラインと釣り合うLMMは64KB程度（かつてのベクトル演算機構もベクトルレジスタ1本あたり最大16KB程度）であり、64KBに格納できるword(4B)数で割った16Kサイクルがバースト演算時間の上限となる。Lightfield(LF)では入力が8K画像のためLMMを十分使い切ることができる一方、Matrix Multiplication(MM)はバースト演算時間に対応するcolがM/W(M=480,W=4の場合120)、Convolutional Neural Network(CNN)は同様にcolがM-2(M=242の場合240)と極めて短い。LMMを目一杯使うため、MMは回転数GRP=8のrowループを外側に挿入した二重ループとしバースト演算時間をM/WxGRP=960に、CNNも同様にGRP=8により(M-2)xGRP=1920に増加させ、ハードウェアにも多重ループ一括実行機構を追加する。

図1.14(a)にCNN冒頭部分の実際のCコード、(b)に4列構成（左から第3,2,1,0列とする）を用いた論理的多重ループ制御方法を示す。(a)の第2行から第5行が各々第1, 0, 2, 3列に写像されている（第4行のinit0?col:colはinit0による入力切替え指示でありC言語としてはnop）。第0列に写像されたloop0がCGRaのバースト演算に関与する最内ループカウンタであり、レジスタに初期値M-2が設定された後は毎サイクル自己ループによりデクリメントされ、結果が非0の場合は第1列に写像されたloop1（初期値GRP）のデクリメント値を0に維持する。loop0が0に到達すると、第0列の左側ソースに再度M-2を設定、第1列の右側ソースを-1に切り替えてloop1をデクリメント、第2列の左側ソースに再度-4を設定、第3列の右側ソースをM*4に切り替えてrowを加算し、loop1のイタレーションが1つ進む。loop0とloop1

```

1. for (chip=0; chip<N; chip++) {
2.   for (init1=1, loop1=GRP, row=0-M*4; loop1--; init1=0) {
3.     for (init0=1, loop0=M-2, col=0-4; loop0--; init0=0) {
4.       add (&col, init0?col:col, 4);
5.       add (&row, row, init0?M*4:0);

```

(a) Head of CNN code for proposed CGRA

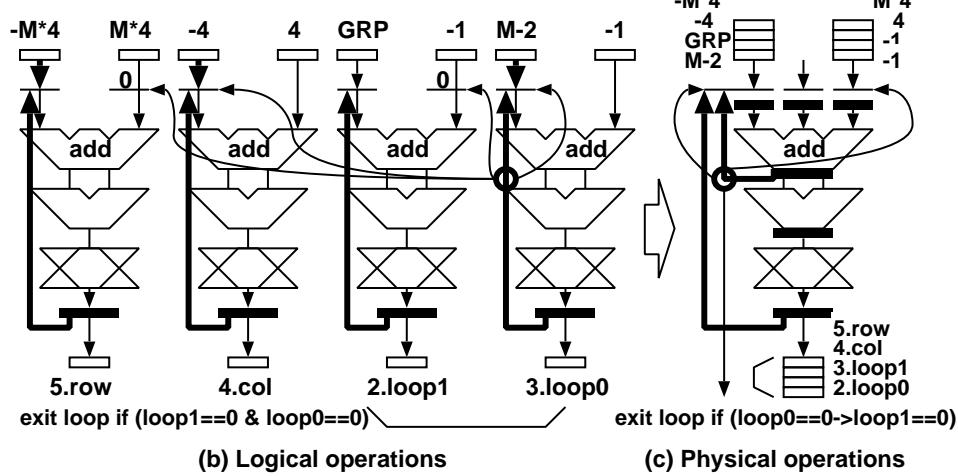


Figure 1.14: Multilevel loop control

が共に 0 になるとバースト演算が停止する。 (c) は列方向マルチスレッディング適用後の提案手法であり、以上の制御を单一 unit が行う。 (b) が 1 サイクルに 4 演算を並列実行するのに対し、(c) は 4 倍周波数の 4 サイクルにより第 0-3 列の演算を順に実行する。パイプライン演算器初段の自己ループにより列間データ依存に対応している。多重ループ一括実行機構には、同一 LMM に対する read-modify-write 機能も含まれる。MM および CNN 最終段の LMM に保存される最内ループの実行結果を一旦 DDR に追い出すことなく read-modify-write 機能により累算することにより、多重ループ一括実行の効果を引き上げる。

1.6 ローカルメモリ (LMM) の様々な使用方法

図 1.15 に、ローカルメモリ (LMM) の様々な使用方法を示す。

- 物理的なローカルメモリ空間を論理 4 列に対応して 4 分割したり、2 分割して論理 2 列で共有したり、分割せず全体で共有することができる。分割方法は混在も可能。なお、同一 LMM のプリフェッチ/ポストドレインや、ダブルバッファリング機能を使用する場合は、最大 8 分割となる。
- 入力データを DDR から LMM に読み込んだ後、LMM から演算器入力に供給する。ロード命令を配置する場合の、LMM の一般的な使用方法。なお、隣接 LMM へのプリフェッチと、後述の命令写像位置シフトを組み合わせると、DDR から LMM への転送時間を演算時間に隠蔽できる。
- LMM から演算器入力に供給しながら、次回の IMAX 起動に必要な入力データを DDR から同一 LMM の別領域に読み込む、同一 LMM に対して、演算用の読み出しとプリフェッチ用の書き込みを同時に使う使用方法。2 つの空間を LMM に同居させてるので DMA 長は、LMM 空間分割後のさらに半分になる。
- 演算結果を LMM に書き込んだ後、出力データを LMM から DDR へ書き戻す。ストア命令を配置する場合の、LMM の一般的な使用方法。なお、隣接 LMM からのポストドレインと、後述の命令写像位置シフトを組み合わせると、LMM から DDR への転送時間を演算時間に隠蔽できる。
- 演算結果を LMM に書き込みながら、前回起動時の出力データを同一 LMM の別領域から DDR に書き戻す。同一 LMM に対して、演算結果の書き込みとポストドレイン用の読み出しを同時に行う使用方法。2 つの空間を LMM に同居させてるので DMA 長は、LMM 空間分割後のさらに半分になる。

- (f) DMA 長に 0 を指定することにより, LMM と DDR の間のデータ転送を抑止できる. その上で, 一連の演算結果を LMM に書き込みながら, 前回の演算結果を同一 LMM から読み出して次の演算に使用できる. すなわち, LMM をダブルバッファとして使用することができる. FFT やマージソートのような多段処理のパイプライン実行に利用できる.

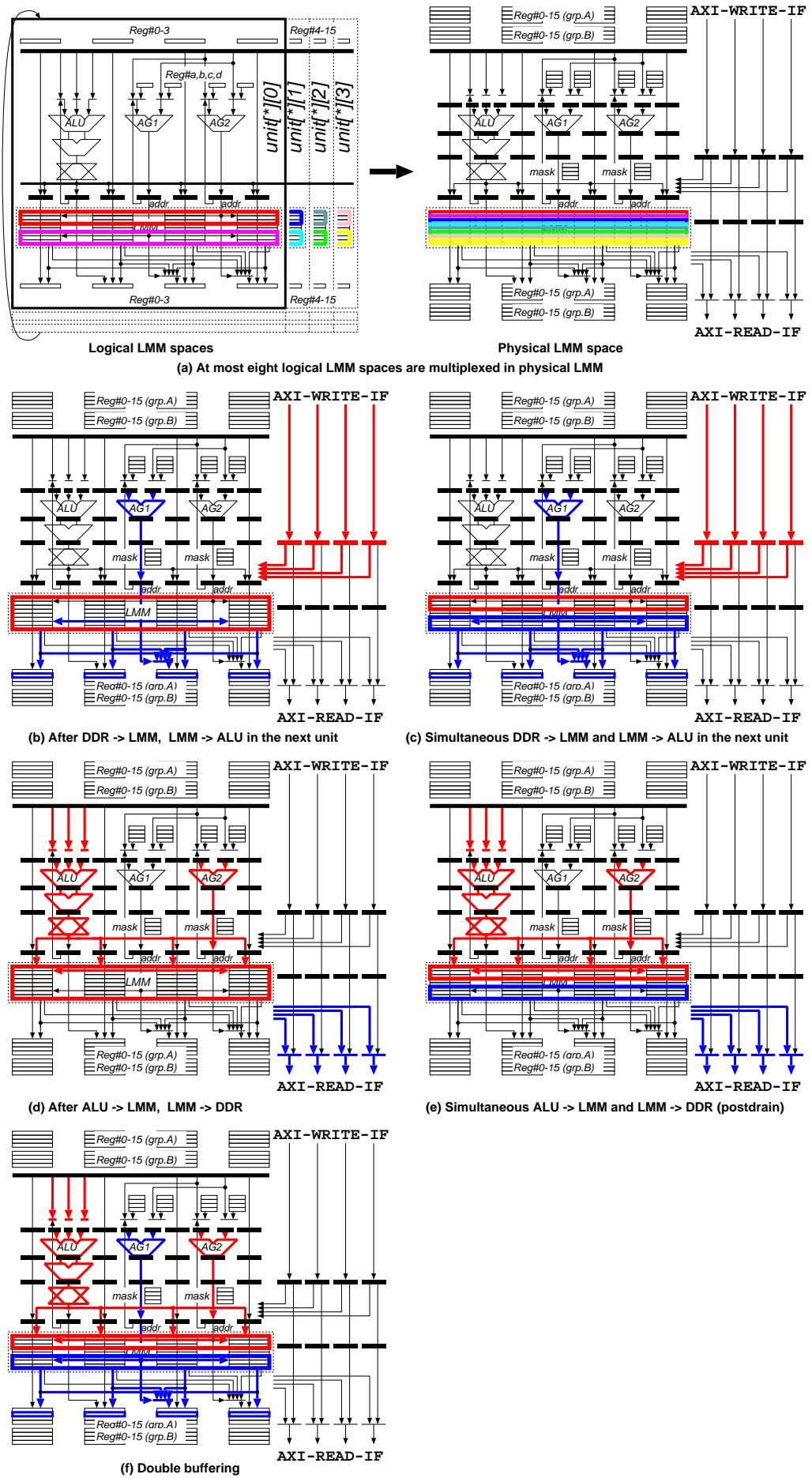


Figure 1.15: ローカルメモリ (LMM) の様々な使用方法

1.7 リング構造の UNIT 間接続と命令写像位置シフト機能

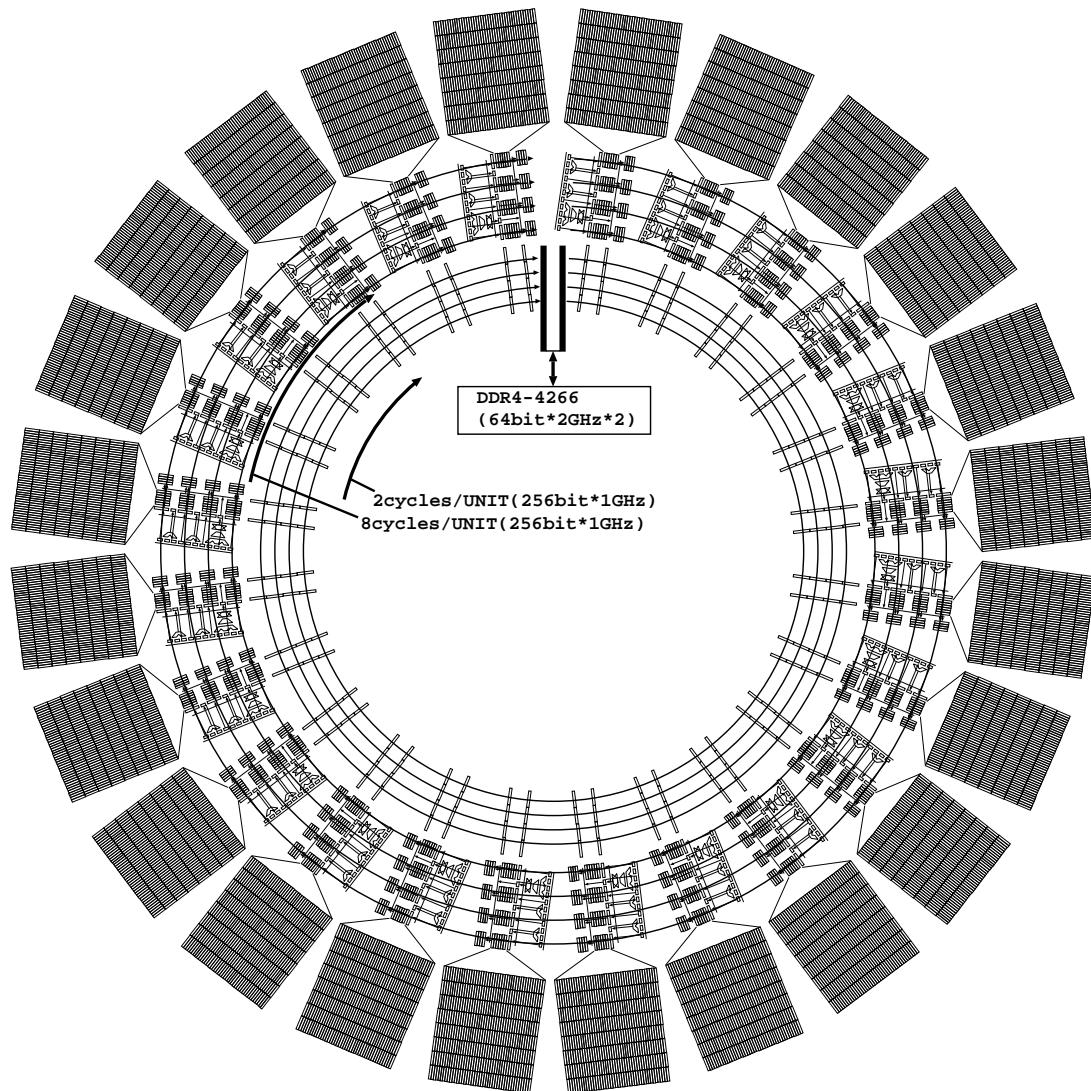


Figure.1.16: 24 行構成の概要

課題（4）に対応するために、命令写像位置シフト機能を導入した。図 1.16 は、DDR を含む 24 行構成の概要である。EMAX5 では、各列を担当する fsm が、二分木により接続された全ての LMM と主記憶の間のバースト転送を行うのに対し、IMAX では、パイプラインメモリバスにより LMM の参照を行う。二分木のための配線を削減でき、また、対象 LMM の切替えが不要であるため、DMA 転送と PIO 転送を混在させて切れ目なく実行できる。EMAX5 では DMA のみを使用していた conf, lmmi, regv の初期化に、IMAX では PIO を使用でき、部分的なレジスタ更新を高速化できる。DMA 転送の際の UNIT 間遅延は 2 サイクル、演算実行時の UNIT 間遅延は 8 サイクルである。なお、メモリバスを 1 列 24 行構成から複数列構成に組み替えることにより、メモリバス遅延時間を短縮できる。

命令写像位置シフト機能は、リング構造を利用し、LMM の内容は移動せず、命令写像を回転移動させる。従来コンパイラが、バースト演算起動前にホストが常時命令シフト指示コードを生成していた点を見直し、バースト演算起動前に、現在の LMM のアドレス範囲と次のバースト演算に必要なアドレス範囲を比較し、同一の場合に命令シフト指示を発行しないよう改良した機能である。

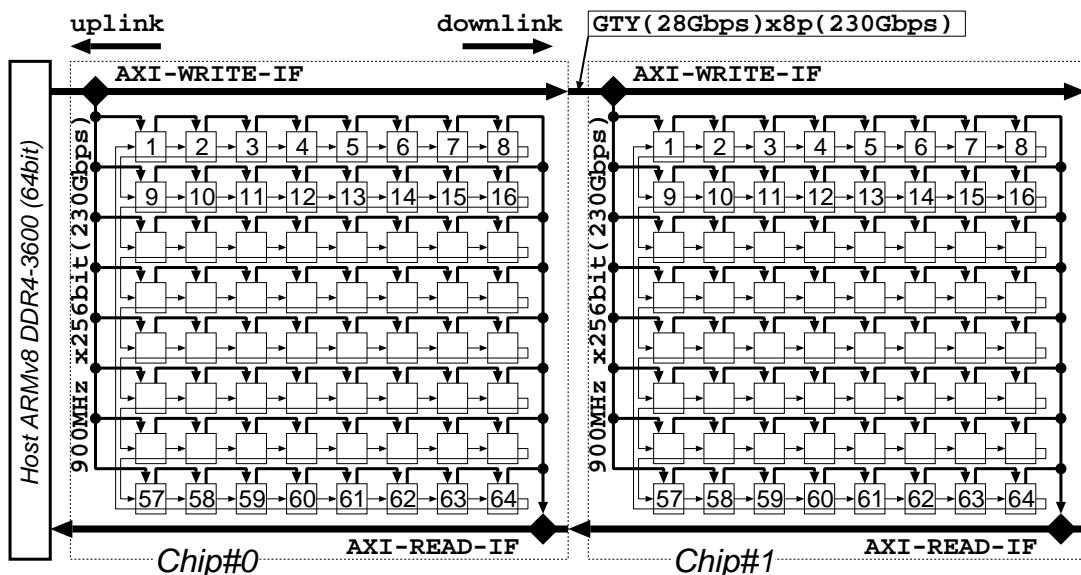


Figure 1.17: Cascaded peer-to-peer AXI bus for scalable multichip CGRA

1.8 メモリとしてのカスケード接続機能

課題（5）および（6）に対応するためには、マルチチップ構成を考案した。マルチチップ構成によりコストダウンが図れるのは、单一チップの面積を小さくすることにより歩留まりが改善されるためである。AMD の Threadripper（4 チップの MCM）が、Intel の Core-i9（シングルチップ）よりも低価格であることは記憶に新しい。

IMAX は、ARMv8 をホストとし、標準 AXI バス上のメモリデバイスとして任意の演算資源を ARM の主記憶空間上に写像できる。この際、1 つの空間に複数の LMM を写像し、ホストからの PIO/DMA により、多数の LMM に画像やパラメタを一斉に書き込むことも可能である。プログラマは C 言語によりアプリケーションを記述する際、各データ構造をどの LMM に対応付けるかを指定し、データ構造間に演算を配置して任意のデータフローを形成できる。なお、データ供給バスは 8 列 × 8 行構成として ARM との間の転送遅延を低減している。さて、IMAX は、IoT 向け高効率小型リニアアレイアクセラレータとして必要な機能を搭載し尽くしたと考えている。しかし、実用化に近づけるためには、様々な IoT の要求性能に合わせて、スケーラブルに性能を向上できる基本構造を備えることが必須である。複数チップにまたがって多数の LMM へのブロードキャスト PIO/DMA ができ、演算結果の回収も複数チップにまたがって PIO/DMA 転送できれば、行列積、ステンシル計算、画像認識に必要な畳み込み演算等を見通しよく高速化できる。ただし、GPU のように多数のメモリバスを用意して並列接続する方法は、多数の AXI バスを用意できない IoT 向けアクセラレータでは論外である。そこで、DMA スレーブ型であることを利用し、同一 AXI バスにカスケード接続して所要の性能に到達できるアーキテクチャを考案した。同様のカスケード接続構成は、商用 CAM-LSI の容量拡張に利用されてきたものの、アクセラレータに適用された例はこれまでにない。これは、GPU をはじめとする通常のアクセラレータは専らリッチなメモリバンド幅を前提とする DMA マスターとして設計されるためである。

図 1.17 に、2 チップ構成の場合の unit 間接続方法を示す。PIO/DMA 機能を有するホストおよび主記憶 (DDR) と、チップ#0 およびチップ#1 が、1 組の AXI4 バスによりカスケード接続されている。64 個の unit を内蔵する各チップは、演算用に 1 列構成のリング接続（各 unit 通過時間は 8 サイクル）、DDR-LMM 間転送用に 8 行 × 8 列構成のデータバス（同様に各 unit 通過時間は 2 サイクル）を備えている。演算用と DDR-LMM 間転送用を共有しないのは、演算と転送の同時動作に加え、チップ数増加時に問題となる 64unit 通過時間の大削減を狙うためである。AXI-WRITE-IF は、ホストからの書き込み要求をチップ内および次チップに伝搬する。また AXI-READ-IF は、チップ内 8 列および次チップからの応答を待ち合わせて結果を戻す。なお最終的なハード構成は、DDR4-3600(64bit)、チップ間物理接続に Xilinx 社の 28Gbps-GTY に相当する。

双方向差動リンク (4本)x8組、片方向がチップ内スループット（後述のように 900MHzx256bit=230Gbps）に釣り合う想定である。

図 1.18 は、3chip 構成の動作である。DDR に格納されている入力画像は、DMA により、シリアル接続された全ての IMAX に送信され、各 IMAX ではアドレス情報に基づき、該当 LMM が入力画像を自律的に取り込む。この際、同じアドレス情報をセットすることにより、複数箇所の LMM が同時に取り込むことも可能である（図 1.18(a)）。特定の LMM に対する Write も同様に、アドレス情報に基づき、該当 LMM が自律的に取り込む（図 1.18(b)）。演算実行時は、各 IMAX が独立に演算を行い、結果を LMM に格納する（図 1.18(c)）。演算結果である出力画像は、DMA により、LMM から DDR に読み出される（図 1.18(d)）。

Table 1.1: 物理メモリインタフェース

信号線名称	I/O	備考
rw	I	0:read(LDDMQ/TRANS のポーリング含む), 1:write
ty	I	0:reg/conf, 1:reg/breg, 2:reg/addr, 3:lddmq/tr, 4:lmm read&lddmq:LMM からの読み出し, write&tr:TR への書き込み
col[1:0]	I	論理列番号
sqi[15:0]	I	seq 番号（最大 64Kdwords）
avi	I	0:a/dm/di 無効, 1:有効
a[31:5]	I	register/LMM のアドレス
dm[31:0]	I	register/LMM への書き込みデータ Byte 每マスク
di[255:0]	I	register/LMM への書き込みデータ
avo	O	0:sqo/do 無効, 1:有効
sqi[15:0]	O	seq 番号（最大 64Kdwords）
do[255:0]	O	LMM からの読み出しデータ

表 1.1 に示す CPU-IMAX の物理メモリインタフェースは、CPU が外部メモリとして IMAX を参照するのに必要な配線群から構成される。rw:1bit の R/W 種別、ty:2bit の register/LMM 選択、col:参照先論理列番号、sqi:16bit の seq 番号、avi:1bit の R/W 要求、a:27bit のアドレス線 (4dword アライン)、dm:4bit の dword 単位マスク、di:256bit のデータ線 (Write)、avo:1bit の読み出しデータ有効表示、sqi:16bit の seq 番号、do:256bit のデータ線 (Read) が含まれる。

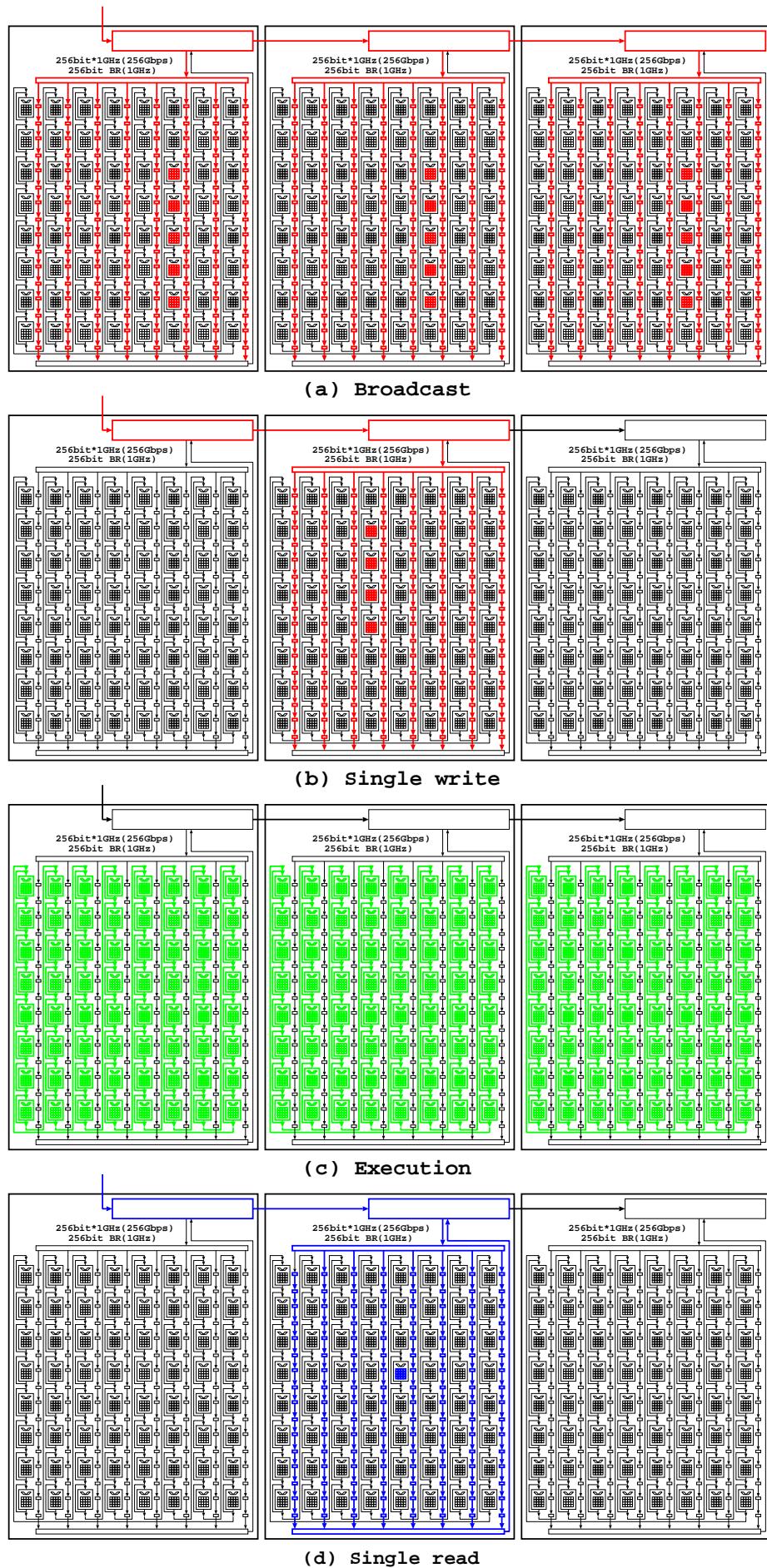


Figure 1.18: 3Chip 構成の動作

1.9 詳細構造と動作

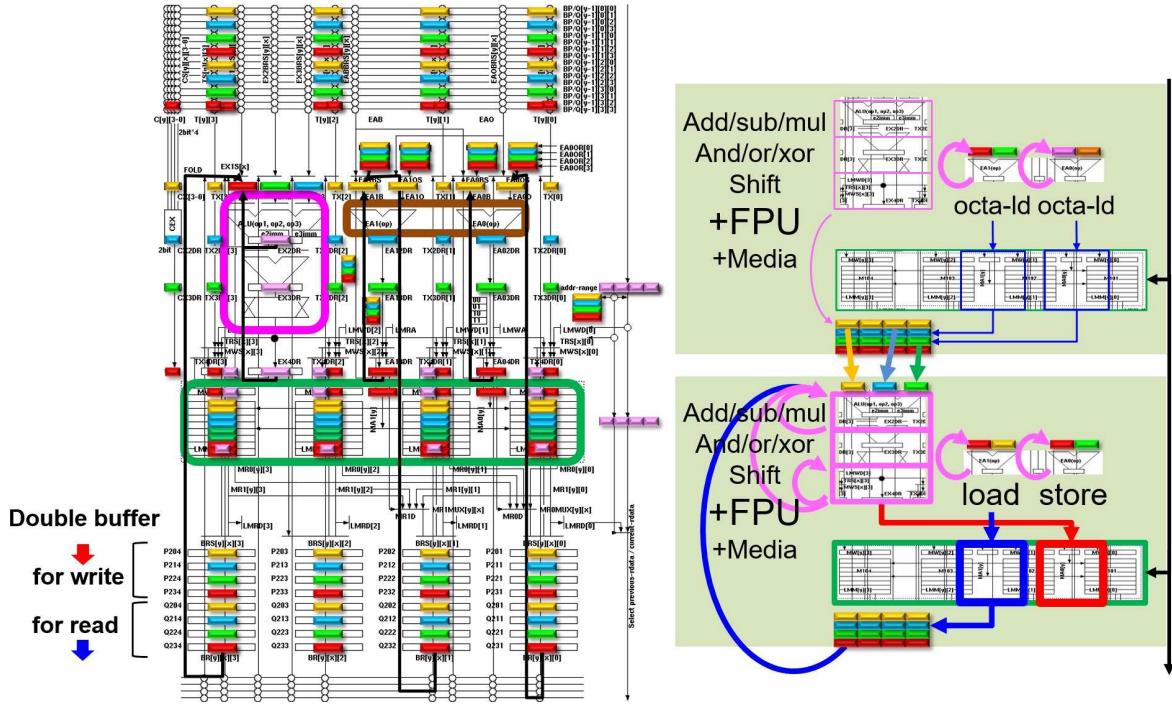
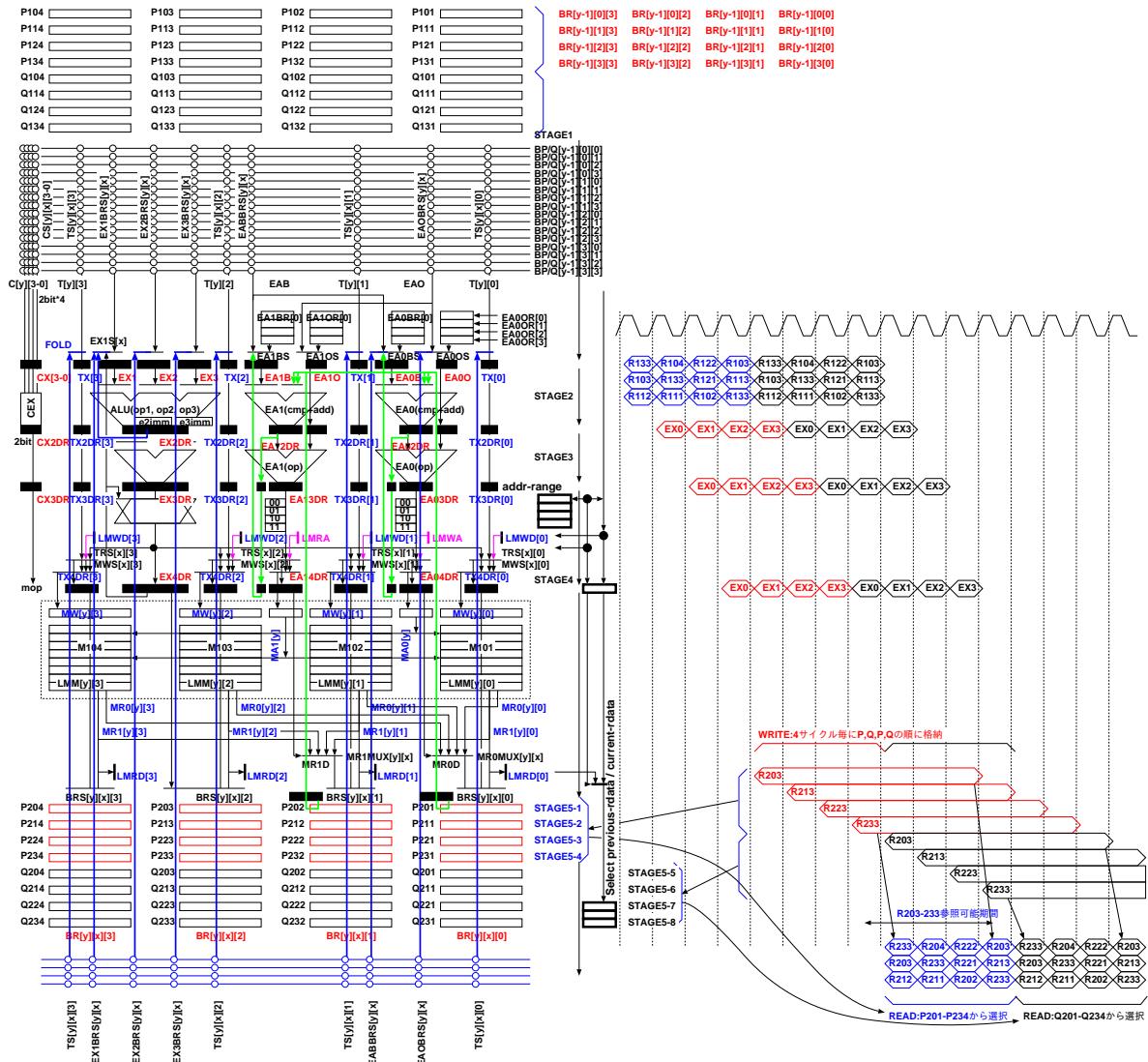


Figure 1.19: IMAX の基本 UNIT 構造

各 UNIT の演算器には、整数演算、単精度浮動小数点演算およびマルチメディア演算を収容する。図 1.20 は、演算器に着目したレジスタ参照のタイミングチャートである。4 列分の演算機能を 1 列分のハードウェアを用いた多重実行により実現している。レジスタ読み出しと演算を 4 サイクルに分割してパイプライン実行する。図 1.21 は、LMM に着目したレジスタ参照のタイミングチャートである。同様に 4 列分の LMM 機能を 1 列分のハードウェアを用いた多重実行により実現している。LMM を最大 4 分割して 4 つの参照をパイプライン実行する。LMM を分割しない場合、EA0/1 の出力 18bit がそのまま LMM のアドレス指定に使用される。LMM を 4 分割する場合、EA0/1 の出力 18bit の上位 2bit が列番号に応じて 00/01/10/11 のいずれかに上書きされ LMM のアドレス指定に使用される。図 1.22 は、CPU から LMM を直接参照する場合に使用する DMA/PIO のデータパス構成である。前行から毎サイクル、アドレス、R/W 種別、書き込みの場合書き込みデータを供給し、複数行から構成されるメモリ空間をパイプライン的に参照し、最終行から読み出しデータを取り出す。どの UNIT の LMM が目的アドレスを収容しているかは、アドレスと各 UNIT 内 vAddr-range (top,len) の比較により判定し、自 LMM が一致している場合は R/W 動作を行いアドレスおよびデータを次行に渡す。不一致の場合もアドレスおよびデータを次行に渡す。



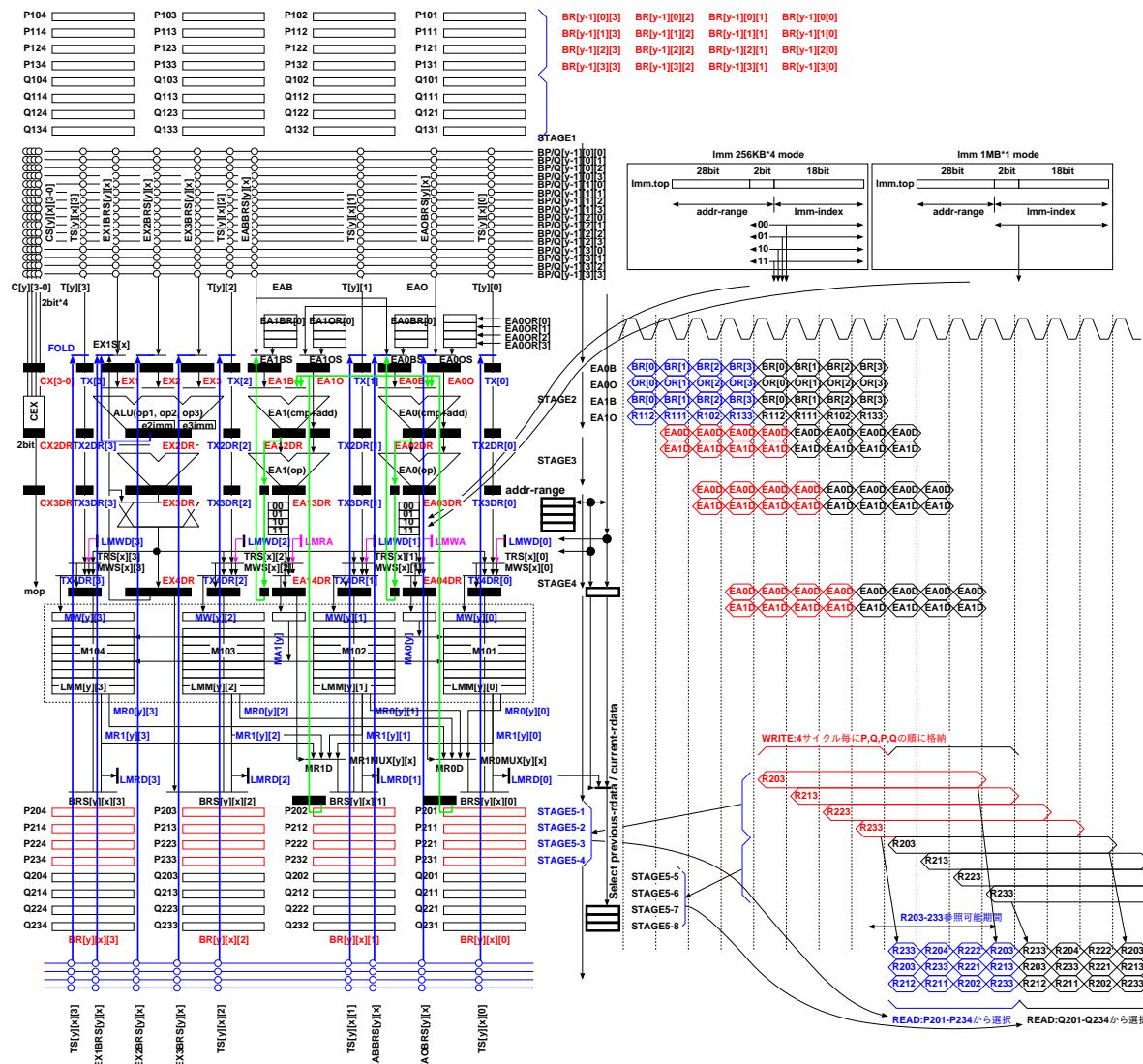


Figure 1.21: UNIT 内ローカルメモリ (LMM) の構成とタイミング

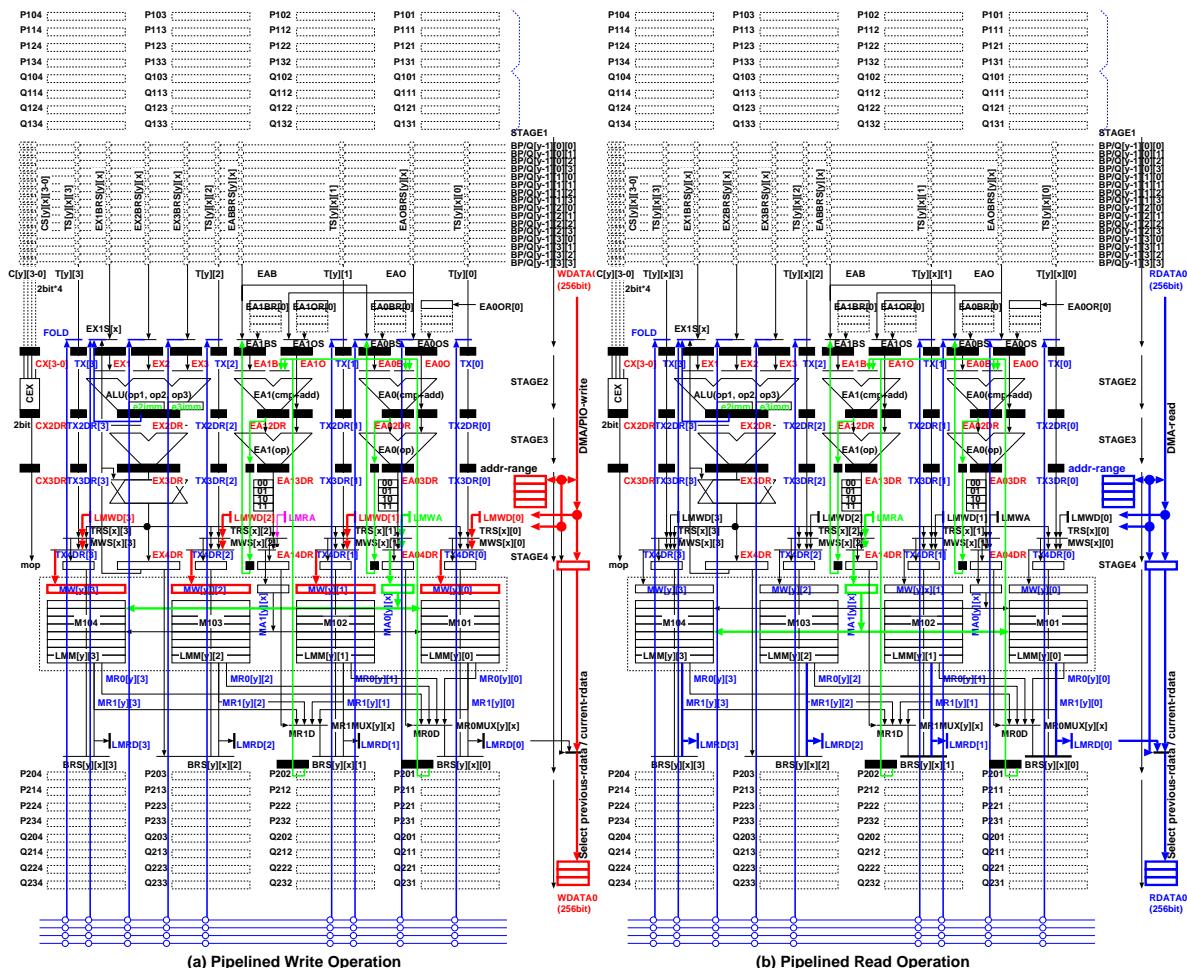


Figure 1.22: CPU からのローカルメモリ (LMM) 直接参照

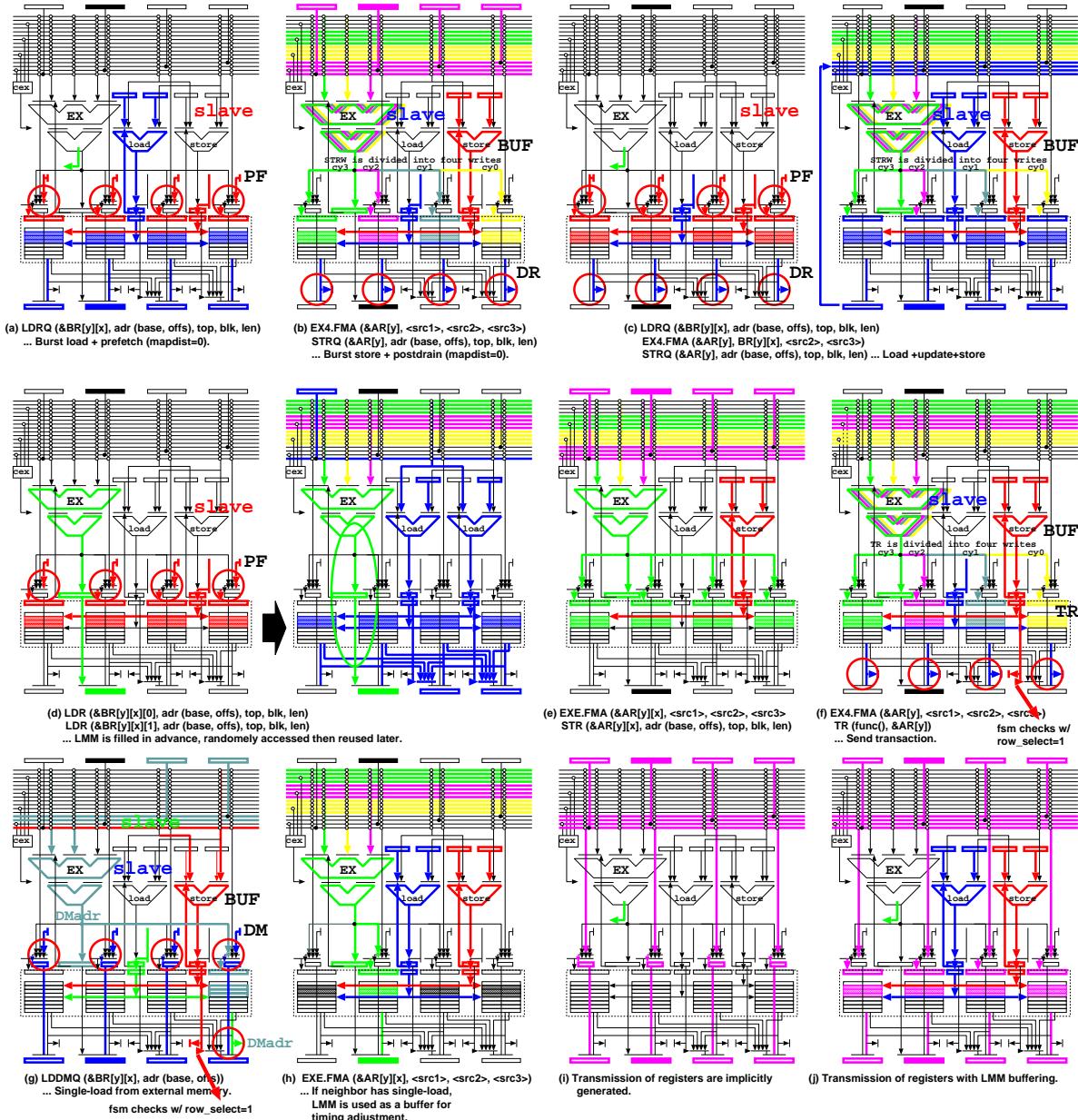


Figure 1.23: UNIT の機能

図 1.23(a) から (j) は、UNIT に写像可能な機能である。 (a) は、次の実行に必要な主記憶からのデータを LMM にプリロードしつつ、ロード済データを LDRQ (4dwords) / LDR (1dword) が次行の演算器に供給する。プリロードには、blk (0:ブロック無し, 1:16 回連続参照毎に先頭ポインタ配列を進めるブロッキング, 2:32 回連続参照毎に先頭ポインタ配列を進めるブロッキング, 3:64 回連続参照毎に先頭ポインタ配列を進めるブロッキング) および len (32bit 単位のバースト長) を指定する。

(b) は、演算結果を STRQ (4dwords) / STR (1dword) により LMM にストアしつつ、前回の実行結果を主記憶に転送する。STRQ は論理的な複数列の演算結果を LMM に格納するために、1 サイクル毎に 1dword を格納する。このため、同一行に複数の STRQ やプリロードを写像することはできない。

(c) は、同一 UNIT において LMM の read-modify-write を行っている。top, blk, len により指定された範囲を予め LMM に格納する。その後、LDRQ が読み出したデータが演算器入力に戻され、結果が同一 LMM に書き戻される。マルチスレッディング機能により、このようなアキュムレートを写像してもパイプラインが止まることはない。

(d) は、LDR が LMM からロードするのに先立ち、top, blk, len により指定された範囲を予め LMM に格納する。その後、LDR がランダムアドレス指定に基づき LMM を参照する。top, blk, len が同一の

LDR は、LMM のデュアルポート機能が利用され、同一 UNIT に写像される。

(e) は、(d) と対をなし、演算結果を STR により LMM にストアしている。全てのストアが完了後、LMM から主記憶に len により指定された量のデータがバースト転送される。

(f) は、トランザクションである。演算結果を TR (4dwords) により LMM にストアしつつ、ARM にトランザクションに必要な 4dwords を供給する。TR は論理的な複数列の演算結果を LMM に格納するために、1 サイクル毎に 1dword を格納する。このため、同一行に複数の TR やプリロードを写像することはできない。

(g) は、外部メモリからの直接ロード機能である。主記憶アドレス計算は EX に写像され、アドレスは LMM の dword0 にキューイングされる。LMM 書き込み用には EA0 を使用し、ARM は当該 AXRA (EA0 と同じ値) を監視し、新規アドレスの登録を検知して LMM から主記憶アドレスを取り出し、AXI を使用して主記憶を読み出し、LMWD にデータを送出する。UNIT では、LMWD → TR → BR を経由して 4dword を次行に送出する。

ところで、以上の基本機能のうち、(g) の写像に際しては、主記憶遅延に対する考慮が必要である。プログラミング時に特殊な記述は必要としないものの、写像時には、(g) と同一行に含まれる UNIT では、(h) のように LMM を利用した遅延同期機構が活性化される。同様に、レジスタ間伝搬が必要である場合、空きレジスタを使用して、(i) (遅延同期無) や (j) (遅延同期有) のような機能が写像される。

1.10 シミュレータ

迅速かつ正確なプロトタイプ開発のためには、まず、Verilog に 1 対 1 に変換可能なレベルの詳細シミュレータの開発が必須である。これは、Verilog シミュレータの動作速度が極めて遅いために、大規模なアプリケーションの動作検証が困難であること、また、Verilog シミュレータによるハードウェア設計検証のために、レジスタ値の正解値との比較を行うための正確な期待値が必須なことによる。IMAX の制御は ARMv8 により行うため、ARMv8 もシミュレーション可能な IMAX シミュレータを開発した (C 言語にて 17K 行の規模)。前述のアプリケーションプログラム、IMAX コンパイラ、および、本シミュレータを用いて、プロトタイプシステムの設計開始前に、ハードウェアデバッグに必須の期待値比較機能付きテストプログラムおよび Verilog シミュレーション用テストベンチを準備した。

1.11 FPGA によるマルチチッププロトタイプ

8 チップ構成を実機評価するためには、XILINX 社製 VU440 を 8 基搭載する試験環境が必要である。前述のシミュレータを元に Verilog 記述を完成させ、シミュレータにより生成した各レジスタの正解値と Verilog シミュレーション結果を比較してデバッグを進めた。ハードウェア記述量は、Verilog にて 11K 行であった。ホストには、エッジデバイス向け CPU の業界標準である ARMv8 を搭載した Xilinx 社製 Zynq UltraScale+ (ZCU102)、IMAX には、S2C 社製 Virtex Ultra Scale (S2C Single VU440 Prodigy Logic Module) を使用した (図 1.24)。現在でも、64unit 構成の IMAX を実装可能な FPGA は VU440 のみであり、ARMv8 と VU440 を高速シリアルリンクにより接続可能なシステムとして本組み合わせは最適である。ただし、カタログスペック上は、10Gbps の高速シリアルリンクを 3 組束ねて 30Gbps のスループットが出せるはずなのに対し、実際には、5Gbps x 3 レーンの合計 15Gbps のスループットでしかリンクを確立できなかった (その後 8 レーンに増速)。また、8 チップを接続すると、LMM からの読み出しに使用する AXI-READ 動作が極めて低速であることが判明した。これは、ホストの DMA 機能に対して十分な長さのバースト長を指定しても、ホストに内蔵されている AXI インタフェースが AXI3 互換であり、256bit 幅転送の場合、8beat ごとに転送が分断され、全チップからの応答が完了するまで次の AXI-READ が待たされることが原因であった。そこで、変更できないホストの AXI インタフェースは AXI3 のまま使用しつつ、IMAX 間では分断せずに元のバースト長を維持する機構を考案し、AXI-READ の大幅な高速化に成功した。



Figure 1.24: 4 チップ構成の IMAX

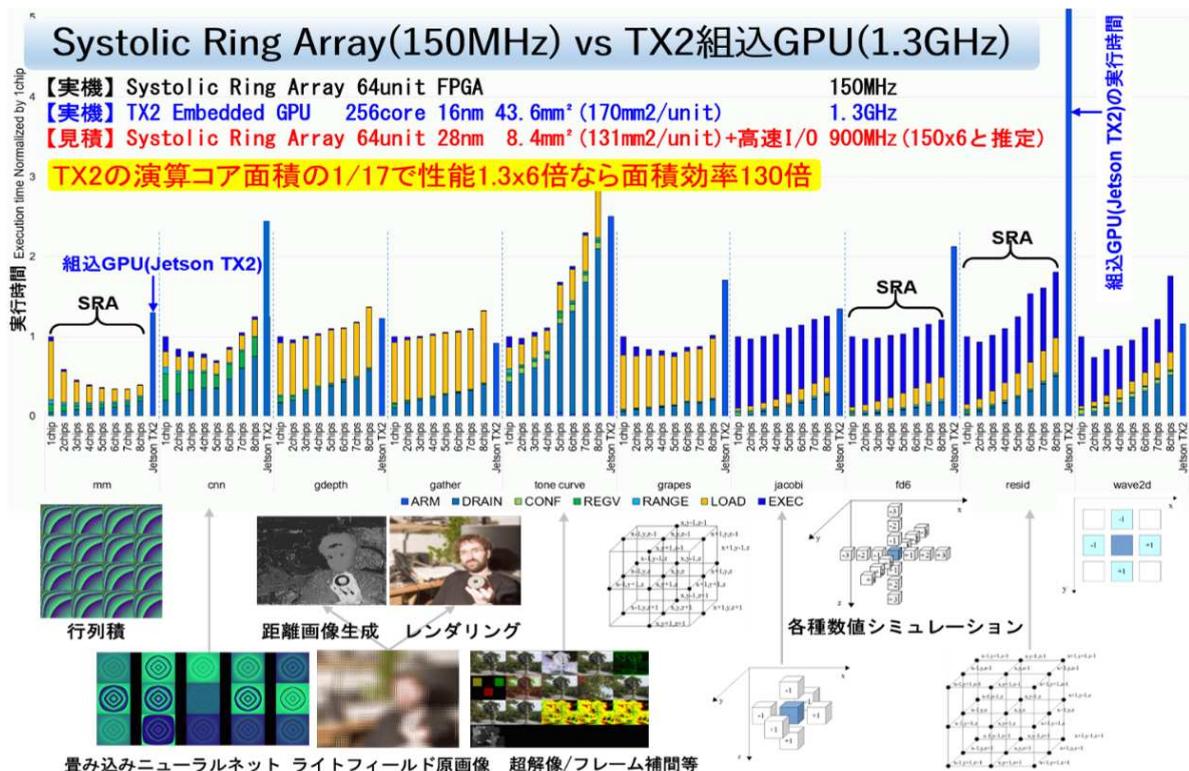


Figure 1.25: 総合評価 1 (15Gbps interface)

1.12 総合評価

以上の経緯を経て開発が完了したIMAXの8チップ構成モデルを用いて、各種プログラムを走行させた。図1.25は、5GbpsのAuroraインターフェースを3レーン使用し(計15Gbps)、IMAXのチップ数を1から8まで変化させて測定したアプリケーションの実行時間である。プログラム毎に、1チップ構成(動作周波数150MHz、演算器数64、面積は28nmにて推定8.4mm²)の実行時間を基準に正規化してあり、比較のために、組み込みGPGPU(Jetson TX2:動作周波数1.3GHz、演算器数256、演算コア部分の面積は16nm

Kernel	CPU ARMv8 1.2GHz	GPU 256core Jetson TX2 1.3GHz DDR4 480Gbps 16nm 43.6mm ²	CGRA 64core*4 IMAX2 140MHz DDR4 40Gbps [28nm想定 14.6mm ² *4] 8nm想定 1.2mm ² *4	GPU 3584core GTX1080Ti 1.5GHz GDDR5 3872Gbps 16nm 471mm ²	GPU 10496core RTX3090 1.4GHz GDDR6X 7490Gbps 8nm 628mm ²
DDR bandwidth		12	1	97	187
Power		7.5W	ARM 0.6W + [31W] 2.7W	250W	350W
MM		3160msec	170 EDP=588K	16 [3msec] [EDP=284] EDP=30	12 EDP=36K
CNN		2080msec	280 EDP=588K	23 [4msec] [EDP=505] EDP=53	18 EDP=81K
Lightfield		14500msec EDP=10.6M	1190 EDP=10.6M	754 [126msec] [EDP=501K] EDP=52K	43 EDP=462K
Sparse MM 3200 ²		-	-	333+469 [134ms] [EDP=567K] EDP=59K	Cusparse使用 2044 EDP=1045M
Sparse MM 4000 ²		-	-	2378+734 [519ms] [EDP=8.51M] EDP=889K	Cusparse使用 3492 EDP=3049M

Figure.1.26: 総合評価 2 (40Gbps interface)

にて推定 43.6mm²) の実行時間も掲載している。行列積 (mm) は 7chip 連結、畳み込み (cnn) は 5chip 連結、距離画像 (depth) は 2chip 連結、科学技術計算 (Stencil 計算) では、大気シミュレーション (grapes) は 5chip 連結、その他ステンシルは 2chip 連結が最速であった。全体の傾向として、mm や cnn のように chip 間共通データが多いとマルチチップ化の効果が大きく、ステンシルのような単純な空間分割では効果が小さい（メモリバンド幅が狭いまでのマルチチップ化は必要ない）ことがわかる。総じて、エッジデバイス前提の限られたメモリバンド幅 (Jetson TX2 の 1/32) では納得できる結果が得られた。図 1.26 は、5Gbps の Aurora インタフェースを 8 レーン使用 (40Gbps) した評価である。1.3GHz 動作の Jetson TX2 を 140MHz 動作の IMAX が性能面で凌駕していることは、CGRA のポテンシャルを十分に示している。また、IMAX が仮に 28nm の ASIC 化により 900MHz 動作した場合、Jetson TX2 の 1/17 の面積で最大 15 倍 (1 チップ構成の cnn の場合) の性能となり、面積あたり性能では 250 倍となる。消費電力が面積に比例すると考えると、同一性能あたりの消費電力も同じ比率であると考えられる。

Chapter 2

IMAX Software

2.1 主記憶上に写像されるIMAX インタフェース

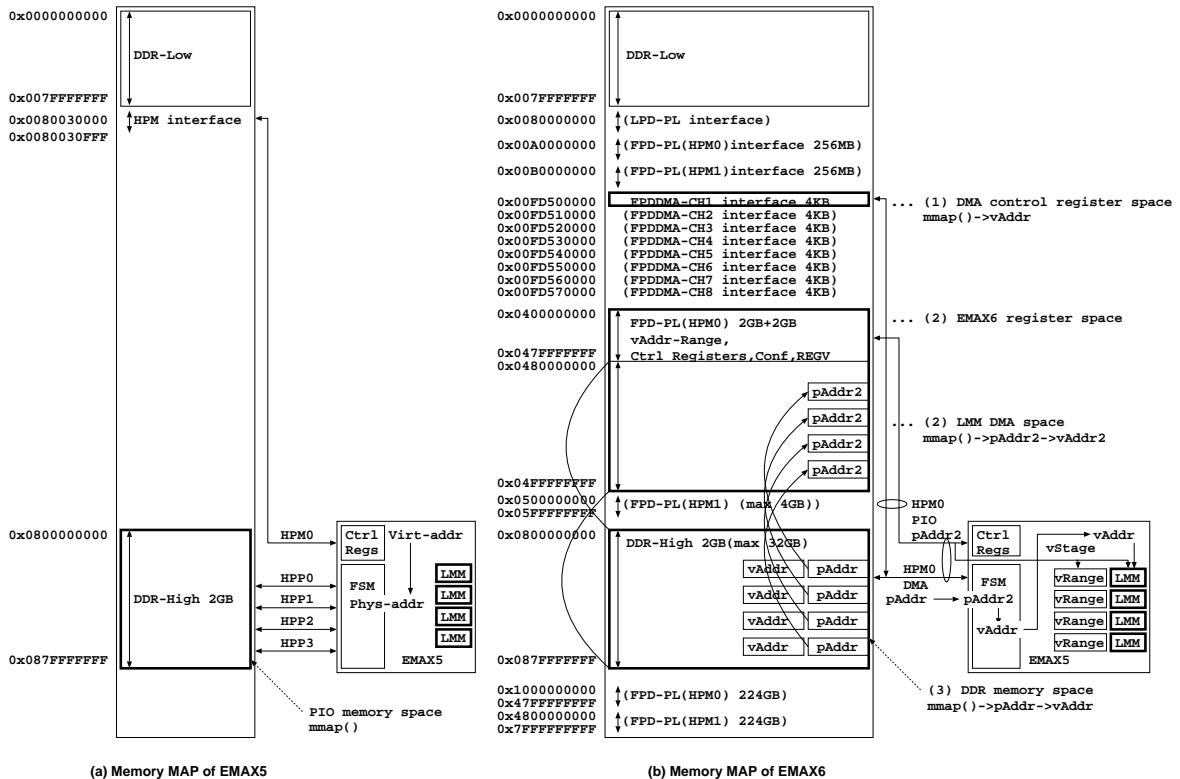


Figure 2.1: 物理メモリ空間

図 2.1 に CPU の物理メモリ空間を示す。IMAX に関する物理メモリ空間は、(1) CPU 物理メモリ空間と LMM の間の DMA を制御する FPDDMA-CH1 DMA 制御レジスタ空間 (DMA control register space:4KB), (2) LMM を CPU 物理メモリ空間に写像して DMA により参照する LMM DMA 空間 (pAddr2 space:2GB) および IMAX 内のレジスタを PIO により参照する制御レジスタ空間 (pAddr2+ space:2GB), (3) LMM との DMA が可能な DDR メモリ空間 (pAddr space:2GB) から構成される。このうち、IMAX との物理的接続により参照できる空間は、pAddr2 space:2GB および pAddr2+ space:2GB である。IMAX 内 LMM の総容量は 16MB (256KB*64stages)，データバス幅は 256bit (32B) であるため、少なくともアドレス幅に 19bit, 1dword (64bit) 每の 1bit 書き込みマスクに 4bit が必要である。これに制御レジスタ空間を加えた物理メモリ空間が、CPU の物理メモリ空間に写像される。

表 2.1 に物理メモリインターフェースを通じて提供する論理インターフェースを示す。物理メモリ空間のうち 0x4fffffff-0x4800000000 (pAddr2) は、0x87fffff-0x800000000 (pAddr) に対応付けられる DDR-High

Table.2.1: 論理インターフェース

制御レジスタ空間 (pAddr2+)	R/W	Bit 位置	備考
STATUS 0x400000007-0x400000000	R	bit3-0 bit7-4 bit11-8 bit15-12 bit63-16	EXRING 状態 0:IDLE, 1:BUSY(SCON/EXEC 動作中) LMRING 状態 0:IDLE, 1:BUSY(PIO/DMA 動作中) 0:EMAX_DEPTH=8, 1:EMAX_DEPTH=16 2:EMAX_DEPTH=32, 3:EMAX_DEPTH=64 0:LMM_SIZE=32KB, 1:LMM_SIZE=64KB, 2:LMM_SIZE=128KB reserved by 0
COMMAND 0x400000017-0x400000010	W	bit1-0 bit31-2 bit35-32 bit63-36	IMAX への指示 0:NOP, 1:RESET, 2:SCON, 3:EXEC reserved by 0 chip 番号 (0 を書き込むと chip 間を順次伝搬) reserved by 0
ADRTRANS 0x400000027-0x400000020	W	bit63-0	vAddr-pAddr2 DDR-LMM 間 DMA 時のアドレス変換情報
COLSELECT 0x400000037-0x400000030	W	bit1-0 bit63-2	vAddr-pAddr2 DMA/LDDMQ/TRANS 時の明示的論理 col 指定 reserved by 0
CONF 0x40000201f-0x400002000 0x40000203f-0x400002020 : 0x400003ff-0x400003fe0	W W : W	bit255-0 bit255-0 : bit255-0	論理 UNIT#0.0 の conf 論理 UNIT#0.1 の conf : 論理 UNIT#63.3 の conf
REGV-BR 0x40000401f-0x400004000 0x40000403f-0x400004020 : 0x400005ff-0x400005fe0	W W : W	bit255-0 bit255-0 : bit255-0	BR 書き込み 論理 UNIT#0.0 の BR[3:0] 論理 UNIT#0.1 の BR[3:0] : 論理 UNIT#63.3 の BR[3:0]
REGV-EAR/vAddr-range 0x40000600f-0x400006000 0x400006017-0x400006010 0x40000602f-0x400006020 0x400006037-0x400006030 : 0x400007fef-0x400007fe0 0x400007ff7-0x400007ff0	W W W W : W W	bit49-32,17-0 bit113-96,81-64 bit30-0 bit62-32 bit49-32,17-0 bit113-96,81-64 bit30-0 bit62-32 bit49-32,17-0 bit113-96,81-64 bit30-0 bit62-32	EAB,EAO 書き込み LMM 先頭 addr(max2GB), LMM 有効 dword 数 (max8KDW) 論理 UNIT#0.0 の ea0o,ea0b(virt-addr) 論理 UNIT#0.0 の ea1o,ea1b(virt-addr) 論理 UNIT#0.0 の top(virt-addr) 論理 UNIT#0.0 の bot(virt-addr) 論理 UNIT#0.1 の ea0o,ea0b(virt-addr) 論理 UNIT#0.1 の ea1o,ea1b(virt-addr) 論理 UNIT#0.1 の top(virt-addr) 論理 UNIT#0.1 の bot(virt-addr) : 論理 UNIT#63.3 の ea0o,ea0b(virt-addr) 論理 UNIT#63.3 の ea1o,ea1b(virt-addr) 論理 UNIT#63.3 の top(virt-addr) 論理 UNIT#63.3 の bot(virt-addr)
LDDMQ/TRANS-R 0x40000801f-0x400008000 0x40000803f-0x400008020 : 0x400009fff-0x400009fe0	R R : R	bit255-0 bit255-0 : bit255-0	LMM から LDDMQ/TRANS 要求を読み出す 論理 UNIT#0.0 の LMM 論理 UNIT#0.1 の LMM : 論理 UNIT#63.3 の LMM
LDDMQ-W 0x40000801f-0x400008000 0x40000803f-0x400008020 : 0x400009fff-0x400009fe0	W W : W	bit255-0 bit255-0 : bit255-0	TR への書き戻し 論理 UNIT#0.0 の TR 論理 UNIT#0.1 の TR : 論理 UNIT#63.3 の TR
LMM 空間 (pAddr2) 0x4fffffff-0x480000000	R/W	Addr 境界 32B	備考 DDR-High(0x87ffffffff-0x800000000) の対応位置に該当する LMM の 内容を R/W する。

と LMM との間の DMA のために FPDDMA 機構が使用する連続物理空間である。ユーザプログラムは、連続物理空間 (pAddr2) の mmap() により得られる連続仮想空間 (vAddr2) を PIO により参照することはなく、専ら、DDR-High 連続物理領域 (pAddr) の mmap() により得られる連続仮想空間 (vAddr) のみを参照する。具体的には、IMAX が演算と連動して LMM を参照する際には vAddr を使用し、LMM から DDR-High の当該アドレスと同じ内容を得るために、CPU は、あらかじめ vAddr の内容を LMM に転送しておく必要がある。転送先 LMM の特定は、UNIT 毎に設けられる vAddr-range レジスタとの比較により行われ、書き込み先アドレスには、FPDDMA から FSM に通知される pAddr2 を FSM 内アドレス変換レジスタの値 (vAddr-pAddr2) に加えて得られる vAddr が使用される。各 UNIT に設けられる vAddr-range レジスタは、各 UNIT の conf.mapdist 値を SCON 指示毎に減算した最新の論理 UNIT 番号に対応付けられ

る。このため、命令シフトを伴って LMM が理想的に再利用されている状態では、新たに有効になる LMM に該当する vAddr-range レジスタの書き込みのみを行えばよい。

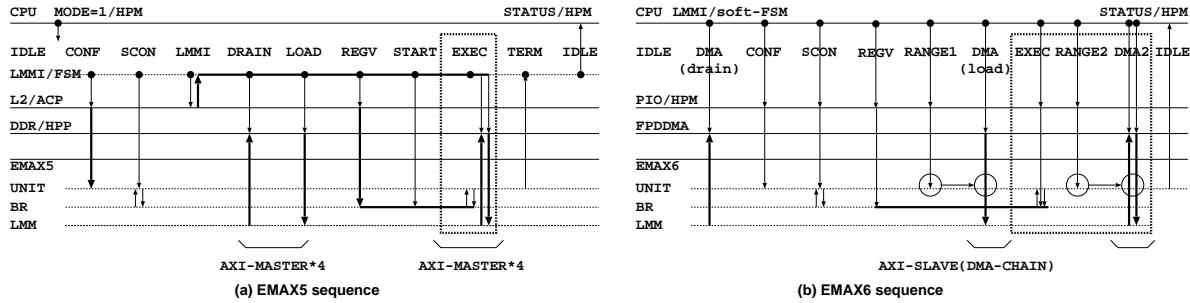


Figure.2.2: 起動から終結までの手順

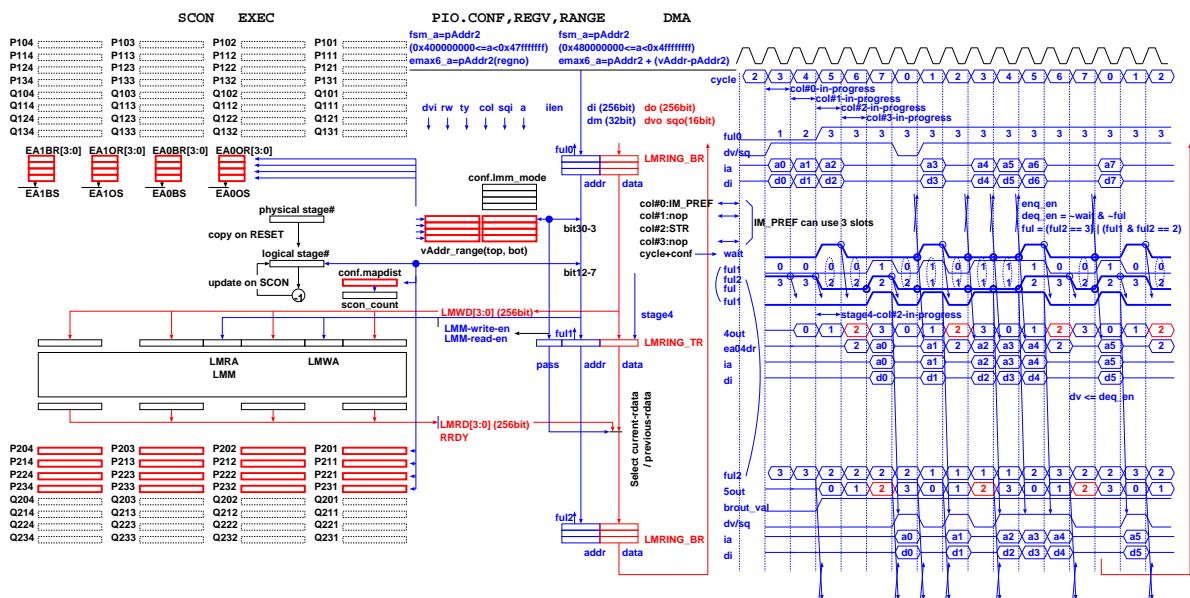


Figure.2.3: 論理インターフェースと各行におけるデータ取り込み機構の関係

図 2.2 は、EMAX5 と IMAX の起動から終結までの動作概要の比較である。EMAX5 の場合、CPU は HPM を経由して EMAX 内の FSM を起動し、IDLE 状態が CPU に通知されるまで FSM が AXI-MASTER として動作する。一方、IMAX の場合、CPU 側が AXI-MASTER となり、FSM は AXI-SLAVE として動作する。DRAIN, LOAD には FPDDMA の DMA-CHAIN 機能を使用し、CONF, SCON, REGV, RANGE, EXEC には PIO を使用する。EMAX5 内に配置されていた LMMI は、IMAX では CPU が管理しており、IMAX への送信は不要であるものの、新たに vAddr-range の更新 (RANGE) が必要となる。図 2.3 を用いて、図 2.2 に示した各手順におけるハードウェアの動作を説明する。なお、CONF, SCON, REGV および RANGE は単独動作のみ可能であり、EXEC と DMA は、同時動作 (EXEC 起動後に DMA を直ちに起動) 可能である。

RESET CPU は、COMMAND に RESET を書き込むことにより、IMAX 内部状態の初期化、および、全 UNIT の physical stage# (固定値) を logical stage# に書き込む初期化を行う。なお、本機能は IMAX 開発時のデバッグ用であり、ユーザプログラムが使用する必要はない。IMAX が RESET 動作中の場合、STATUS.EXRING に BUSY が表示される。

CONF CPU は、STATUS.EXRING および STATUS.LMRING が IDLE の時、PIO により連続的に CONF を更新することができる。CONF は論理 UNIT 番号に対応して物理アドレスが割り当てられており、指定物理アドレスに対して書き込むことにより、論理行番号 (pAddr2+ の bit12-7) に一致する

logical stage#を有する論理 UNIT の CONF が更新される。IMAX が CONF 書き込み動作中の場合、STATUS.LMRING に BUSY が表示される。

SCON CPU は、STATUS.EXRING が IDLE の時、COMMAND に SCON を書き込むことにより、全行に対して、各行が保持する conf.mapdist の値に従う CONF のシフト開始を指示することができる。全ての行は、第 1 サイクル目に、自 CONF 情報 (256bit*4set) のうち先頭 256bit を BR に書き込み、第 2 サイクル目に、前行の BR から全行の CONF 情報を自行に取り込む。以上を 4 回繰り返すことにより全ての CONF 情報を 1 行分下方にシフトする。また、logical stage#の値を 1 だけ減じる。さらに、以上を conf.mapdist 回繰り返すことにより全ての CONF 情報を conf.mapdist 行分シフトし、logical stage#の値を conf.mapdist だけ減じる。なお、SCON により REGV の内容は破壊される。IMAX が SCON 動作中の場合、STATUS.EXRING に BUSY が表示される。

REGV CPU は、STATUS.EXRING および STATUS.LMRING が IDLE の時、PIO により連続的に REGV-EAR および REGV-BR を更新することができる。REGV は論理 UNIT 番号に対応して物理アドレスが割り当てられており、指定物理アドレスに対して書き込むことにより、当該論理行番号 (pAddr2+ の bit12-7) に一致する logical stage#を有する論理 UNIT の REGV が更新される。IMAX が REGV 書き込み動作中の場合、STATUS.LMRING に BUSY が表示される。

RANGE CPU は、STATUS.EXRING および STATUS.LMRING が IDLE の時、PIO により連続的に vAddr-range を更新することができる。vAddr-range は論理 UNIT 番号に対応して物理アドレスが割り当てられており、指定物理アドレスに対して書き込むことにより、当該論理行番号 (pAddr2+ の bit12-7) に一致する logical stage#を有する論理 UNIT の vAddr-range が更新される。なお、LMM を無効、分割無、2 分割、4 分割のいずれかに指定する指示 (conf.lmm_mode) は、前述の CONF に含まれる。IMAX が vAddr-range 書き込み動作中の場合、STATUS.LMRING に BUSY が表示される。

DMA CPU は、STATUS.LMRING が IDLE の時、DMA による連続的な DDR-LMM 間転送を起動できる。EMAX5 では fsm により実現されていた機能を IMAX では CPU が DMA を用いて行う。具体的には、旧 lmmi と新 lmmi を比較し、旧 lmmi が書き込み先、または、旧 lmmi が強制 STORE 対象（次回先頭アドレスが異なる lmx）の時、当該 LMM を主記憶へ追い出す DMA を起動しなければならない。新 lmmi に対応する領域が LMM に存在しない場合、または、強制 LOAD 対象 (lmf、前回先頭アドレスが異なる lmx) の場合、主記憶から当該 LMM への DMA を起動しなければならない。IMAX が DMA 動作中の場合、STATUS.LMRING に BUSY が表示される。

EXEC CPU は、STATUS.EXRING が IDLE の時、COMMAND に EXEC を書き込むことにより、全行に対して、実行開始を指示することができる。IMAX が EXEC 動作中の場合、STATUS.EXRING に BUSY が表示される。

LDDMQ EXEC 動作中、OP_LDDMQ が写像された論理 UNIT では、LMM に主記憶参照要求がキューイングされる。キューイング状態は fsm に通知され、fsm は当該 LMM から参照要求（先頭仮想アドレス）を読み出す。この際 fsm は、imax_rw=0, imax_ty=1, imax_a[31:5] の bit12-7 に論理行番号をセットして対象論理 UNIT を特定する。読み出しデータは fsm 内部のバッファに格納され、CPU からの読み出しに備える。CPU は、適切な間隔で特定レジスタ空間に対する read 要求を発行しなければならない。LMM に LDDMQ 要求がキューイングされている場合、avo, sqo および do の組を用いて要求が出力される。CPU は当該仮想アドレスを取得後、DDR-High を参照し、当該 UNIT の TR に書き込みを行わなければならない。

TRANS EXEC 動作中、OP_TR が写像された論理 UNIT では、LMM に主記憶参照要求がキューイングされる。キューイング状態は fsm に通知され、fsm は当該 LMM から TRANS 要求を読み出す。この際 fsm は、imax_rw=0, imax_ty=2, imax_a[31:5] の bit12-7 に論理行番号をセットして対象論理 UNIT を特定する。読み出しデータは fsm 内部のバッファに格納され、CPU からの読み出しに備える。CPU は、適切な間隔で特定レジスタ空間に対する read 要求を発行しなければならない。LMM に TRANS 要求がキューイングされている場合、avo, sqo および do の組を用いて要求が出力される。CPU は当該 TRANS 要求を取得後、Transaction を実行しなければならない。

2.2 Control Registers

図 2.4 に、制御レジスタの詳細構造を示す。また、図 2.5 に、CONF により設定される UNIT 内レジスタの詳細構造を示す。

```

struct reg_ctrl {
    struct i0 {
        Ull stat; /* +0000 bit15-12:LMM_SIZE, bit11-8:EMAX_DEPTH, bit7-4:LMRING, bit3-0:EXRING */
        Uint mcid; /* +0008 maximum chip-ID of IMAX (<EMAX_NCHIP>) to be chained (activated) */
        Uint dmy0;
        Uint cmd; /* +0010 host writes Ull cmd then chip# is propagated to successors */
        /*Uint cid;/** +0012 chip# ( set by write to cmd ) */
        Uint dmy1;
        Ull dmy2;
        Ull adtr; /* +0020 */
        Ull dmy3;
        Ull csel; /* +0030 */
        Ull dmrp; /* +0038 DMAREAD-PREF */
        Ull dmy4[1016];
        struct conf           conf[AMAP_DEPTH][EMAX_WIDTH]; /* +2000-3fff */
        struct {Ull br[UNIT_WIDTH];} breg[AMAP_DEPTH][EMAX_WIDTH]; /* +4000-5fff */
        struct {
            Uint ea0b ; /* ea0 base (for avoiding ld-mask-st, */
            /*Ull dmy0 :14;*/ /* should be extended to 32bits (lower 18bit is available) */
            Uint ea0o ; /* ea0 offset (for avoiding ld-mask-st, */
            /*Ull dmy1 :14;*/ /* should be extended to 32bits (lower 18bit is available) */
            Uint ea1b ; /* ea1 base (for avoiding ld-mask-st, */
            /*Ull dmy2 :14;*/ /* should be extended to 32bits (lower 18bit is available) */
            Uint ea1o ; /* ea1 offset (for avoiding ld-mask-st, */
            /*Ull dmy3 :14;*/ /* should be extended to 32bits (lower 18bit is available) */
            Uint top ; /* LMM-top virtual-address */
            /*Ull dmy4 : 1;*/
            Uint bot ; /* LMM-bot virtual-address */
            /*Ull dmy5 : 1;*/
            Ull dmy6 ;} addr[AMAP_DEPTH][EMAX_WIDTH]; /* +6000-7fff */
        struct {Ull reg[UNIT_WIDTH];} lddmrw[AMAP_DEPTH][EMAX_WIDTH];/* +8000-9fff *//*lddmq/trans-r,lddmq-w*/
        Ull dmy5[3072]; /* +a000-ffff */
    } i[EMAX_NCHIP]; /* 0000-ffff */
};
```

Figure.2.4: Control Registers

```

struct conf { /* final configuration info. for IMAX-CGRA */
    struct cdw0 { /* select EXE-in */
        Ull v      : 1; /* 0:inv, 1:insn mapped */
        Ull op1    : 6; /* alu_opcd */
        Ull op2    : 3; /* logical_opcd */
        Ull op3    : 3; /* sft_opcd */
        Ull ex1brs : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull exis   : 1; /* 0:ex1brs, 1:exdr(self-loop) */
        Ull ex1exp : 3; /* 0:H3210, 1:H1010, 2:H3232, 3:B5410, 4:B7632 */
        Ull ex2brs : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull ex2exp : 3; /* 0:H3210, 1:H1010, 2:H3232, 3:B5410, 4:B7632 */
        Ull ex3brs : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull ex3exp : 3; /* 0:H3210, 1:H1010, 2:H3232, 3:B5410, 4:B7632 */
        Ull e2is   : 2; /* 0:e2imm, 1:ex2, 2:ex3 */
#define E3IMMBITS 6
        Ull e3imm  : E3IMMBITS;
        Ull e3is   : 1; /* 0:e3imm, 1:ex3 */
        Ull init   : 2; /* bit0:activate s1+INIT0 bit1:activate s2+INIT0 */
        Ull fold   : 1; /* 0:normal, 1:load-exe-store folding */
        Ull mexOp  : 2; /* mex(sparse matrix) conditional 0:NOP, 1:AL, 2:OP_CMPA_LE, 3:GE */
        Ull mex0init: 1; /* mex(sparse matrix) 0:none, 1:INIT0? */
        Ull mex0dist: 3; /* distance 0:0, 1:1, 2:2, 3:4, 4:8, 5:16, 6:32, 7:64byte */
        Ull mex1op  : 2; /* mex(sparse matrix) conditional 0:NOP, 1:AL, 2:OP_CMPA_LE, 3:GE */
        Ull mex1init: 1; /* mex(sparse matrix) 0:none, 1:INIT0? */
        Ull mex1dist: 3; /* distance 0:0, 1:1, 2:2, 3:4, 4:8, 5:16, 6:32, 7:64byte */
        Ull mexlimit: 4; /* limit 0:0, 1:8, 2:16, ..., 10:4096, 11:8192, 12:16384, 13:32768 */
        Ull dmy00  : 1;
    } cdw0;
    struct cdw1 { /* select CEX-in and EAG-in */
        Ull cs0   : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull cs1   : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull cs2   : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull cs3   : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull cex_tab: 16; /* c3.c2.c1.c0 の組合せ (cop=NOP の場合,ffff) */
            /* 1111,1110,1101,1100,...,0001,0000 の各々に0/1を割り当てた16bitを指定 */
        Ull ea0op  : 5; /* mem_opcd */
        Ull ea0bs  : 2; /* 0:ea0br, 1:ea0dr(ea0br+self-loop), 2:ebbrs, 3:ea0dr(eabbs+self-loop) */
        Ull ea0os  : 1; /* 0:ea0or, 1:eaobrs */
        Ull ea0msk : 4; /* 14:64bit, 13:word1, 12:word0, 11-8:half3-0, 7-0:byte7-0 of offset */
        Ull ea1op  : 5; /* mem_opcd */
        Ull ea1bs  : 2; /* 0:ea1br, 1:ea1dr(ea1br+self-loop), 2:ebbrs, 3:ea1dr(self-loop) */
        Ull ea0ls  : 1; /* 0:ea1or, 1:eaobrs */
        Ull ea1msk : 4; /* 14:64bit, 13:word1, 12:word0, 11-8:half3-0, 7-0:byte7-0 of offset */
        Ull eabbs  : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
        Ull eaobrs : 4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    } cdw1;
    struct cdw2 { /* select TR/BR-in */
        Ull ts0   : 4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
        Ull ts1   : 4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
        Ull ts2   : 4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
        Ull ts3   : 4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
        Ull trs0  : 2; /* 0:lmwd0, 1:exdr, 2:ts0 */ /* 0:TR 外部書き込み用, 1,2:EX/TS 書き込み用 */
        Ull trs1  : 2; /* 0:lmwd1, 1:exdr, 2:ts1 */
        Ull trs2  : 2; /* 0:lmwd2, 1:exdr, 2:ts2 */
        Ull trs3  : 2; /* 0:lmwd3, 1:exdr, 2:ts3 */
        Ull mwsa  : 1; /* 0:lmwa, 1:ea0d */ /* 0:常時 lmwd 可能, 1,2:EXEC 時以外は強制 lmwd 可能 */
        Ull mws0  : 2; /* 0:lmwd0, 1:exdr, 2:ts0 */ /* 0:常時 lmwd 可能, 1,2:EXEC 時以外は強制 lmwd 可能 */
        Ull mws1  : 2; /* 0:lmwd1, 1:exdr, 2:ts1 */
        Ull mws2  : 2; /* 0:lmwd2, 1:exdr, 2:ts2 */
        Ull mws3  : 2; /* 0:lmwd3, 1:exdr, 2:ts3 */
        Ull brs0  : 2; /* 0:off, 1:mr10, 2:tr0, 3:mr0 */
        Ull brs1  : 2; /* 0:off, 1:mr11, 2:tr1, 3:mr1 */
        Ull brs2  : 2; /* 0:off, 1:mr12, 2:tr2, 3:exdr */
        Ull brs3  : 2; /* 0:off, 1:mr13, 2:tr3 */
        Ull mapdist: 6; /* 論理 UNIT 每にあるが, 本来は物理 UNIT に1つでよい */
        Ull lmm_mode: 2; /* 論理 LMM 每にセット 0:無効, 1:分割無, 2:2分割, 3:4分割 */
        Ull lmm_axiw: 1; /* AXI->LMM write 対象 (lmp/lmr/lmf/lmx の場合 1) */
        Ull lmm_axir: 1; /* AXI-<-LMM read 対象 (lmd/lmw/lmx の場合 1) */
        Ull dmy20  : 13;
    } cdw2;
    struct cdw3 { /* e2 immediate */
        Ull e2imm  : 64;
    } cdw3;
} conf[EMAX_DEPTH][EMAX_WIDTH]; /* 4dwords/unit costs 1cycle/unit: 4-parallel conf costs 1cycle/stage */

```

2.3 Programming model

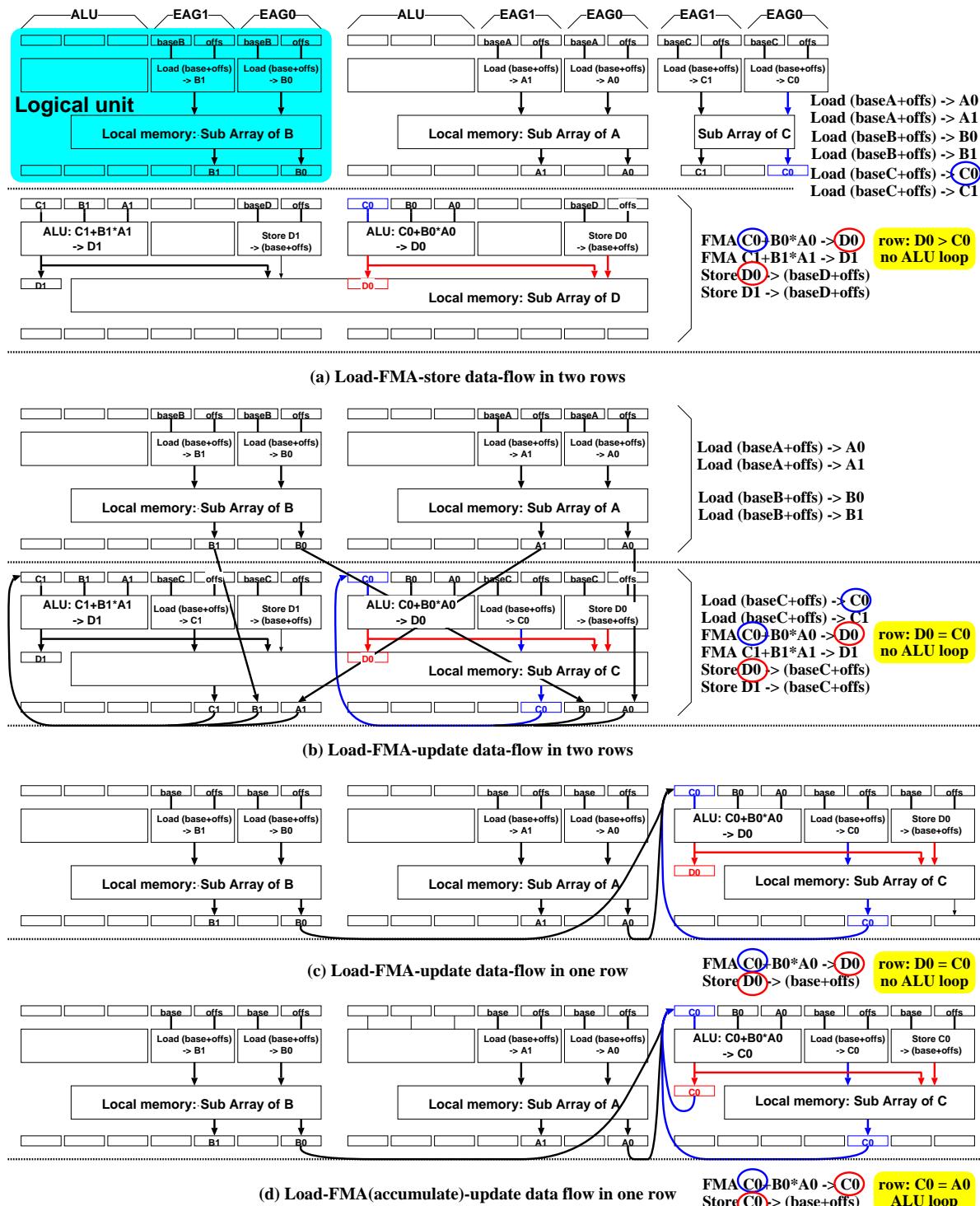


Figure 2.6: ロード-演算-ストアを基本とする IMAX のプログラミングモデル

アルゴリズムが確定したプログラムの最適化には、コンパイラの最適化オプションを用いるのが一般的である。ただし、ノイマン型コンピュータの場合、高効率を達成する究極の手法は、現在でもアセンブリ言語によるプログラミングである。機械語命令は論理的に逐次実行されるため、知識さえあれば、デバッグも比較的容易である。CGRA も同様に、機械語命令レベルの機能を多数の演算器に高密度に写像することにより高効率化を図る。しかし、演算器がメッシュ状に相互接続される従来型 CGRA は、ノイマン型のような逐次実行モデルにより動作を正確に表現することが難しく、アセンブリ言語レベルの最適化やデバッグは

極めて困難である。また、演算器群と外部メモリを分離しているため、SIMD を基本とする広いメモリバス幅と演算幅が必要である。以上の点を改善するために、IMAX は、演算器とメモリの組を基本構成要素とし、演算器間ネットワークには、メッシュ接続ではなくリング接続を採用している。このため、SIMD 構成の制約を受けず、また、ロード-演算-ストアの写像位置に関する自由度が高い。プログラミング手順は、まず、ロード-演算-ストアのデータフロー記述、次に、適切なローカルメモリ位置の選択とアドレスの写像、最後に、ローカルメモリ再利用のためのチューニングとなる。

図 2.6 は、IMAX の典型的なプログラミングモデルである。1つの論理 UNIT（水色背景部分）は、ALU、2 個のアドレス生成器 (EAG)，および、デュアルポートローカルメモリを備え、4つの論理 UNIT が CGRA の 1 行 4 列を構成する。ソースコードの各行は、行列位置情報を含むこともできる。コンパイラは、位置情報の有無や相互関係により、(a)-(d) の写像パターンのいずれかを選択し、適切な論理 UNIT に写像する。(a) は、位置情報を記述せず、配列 A, B, C, D を用いて 2 要素毎に " $D=C+B*A$ " を計算する単純なケースである。まず、CGRA の 1 行目に、Load A, Load B, および、Load C が写像され（4 列目は省略）、A, B, C を入力とする積和演算 FMA および Store D が CGRA の 2 行目に写像されている（3 および 4 列目は省略）。位置情報を記述しない場合、このように、上から下への単純なデータフローが形成される。

一方、(b) は、配列 A, B, C を用いて 2 要素毎に " $D=C+B*A$ " を計算し、結果を C に書き戻すケースである。Load の格納先 C, および、FMA の結果一時格納変数 D に位置情報を付加して、配列 C への Load と Store を同時に使う。CGRA の 1 行目に Load A, Load B が写像され、2 行目に Load C が写像される。積和演算 FMA の結果一時格納変数 D が C と同一行に指定された場合、演算器の入力 A, B, C は、一度、当該行の終端レジスタに送られた後、演算器入力ヘフィードバックされる。配列 C に対する D の Store も同一行に写像されるため、同一ローカルメモリの配列 C に対するロード-演算-ストアとなる。

(c) は、(b) を 1 行に収容するケースである。デュアルポートローカルメモリは、物理的には CGRA の 1 行に 1 個のみ装備されており、時分割多重により、最大 4 列分の独立した論理メモリに見せている。このため、(b) では 1 つの物理ローカルメモリを配列 A と B で共有するのに対し、(c) では配列 C も共有するため配列あたり使用可能なメモリ空間は減少する。それでも 1 行に収容するメリットがある場合は、Load の格納先 C には位置情報を付加せず、FMA の結果一時格納変数 D のみに位置情報を付加する。まず、Load A, Load B, および、Load C が写像され、積和演算 FMA および配列 C に対する D の Store も同一行に写像される。

(d) は、一時格納変数 D を使用せず、積和演算 FMA を " $C=C+B*A$ " と記述したケースである。Store C には位置情報を付加しない。ソースコード上は、ローカルメモリにストアしたデータを再度ロードする積和演算の繰り返しである。しかし、ストア後に同一アドレスからロードして演算に回す計算はオーバヘッドが大きく、CPU でも回避すべきハードウェア動作である。一方、CGRAs の場合、この動作は、演算器内アキュムレートにより容易に吸収できる。(c) との違いは、FMA が単純なロード-演算-ストアではなく、入力 C0 が、LOOP の初回のみ配列 C のロード結果、次回以降は演算結果に接続されるロード-アキュムレート-ストアとなる点のみである。もちろん、パイプライン浮動小数点演算器の場合、演算器内アキュムレートは毎サイクル実行が不可能である。IMAX は、前述の通り、4 列マルチスレッディングにより、アキュムレートがあっても論理 UNIT の動作にオーバヘッドが生じない特長があるため、性能低下の懸念なく (d) を利用できる。

2.4 Templates of instructions

```
cex(OP_CEXE, &ex0-9, c3, c2, c1, c0, 16bit-pattern)
```

c3, c2, c1 は各々 64bit 値であり、bit32 を連結した 4bit と、bit0 を連結した 4bit により、各々、16bit-pattern の bit 位置を取り出した結果が、ex[0-9] の bit1 および bit0 に格納される。ex0-9 の bit1 は条件付きストアの上位 32bit、bit0 は条件付きストアの下位 32bit に対応する。

```
exe(OP_X, &var|&AR[0-63][0-3], s1, e1, s2, e2, s3, e3, OP_Y, s4, OP_Z, s5)
ex4(OP_X, &var|&AR[0-63], s1, e1, s2, e2, s3, e3, OP_Y, s4, OP_Z, s5)
```

var または AR[0-63][0-3] は ALU の演算結果格納先であり、前者は位置情報無し、後者は行列位置情報に対応する。s1 から s3 の各 64bit 値は、各々、e1 から e3 による修飾後に演算器に入力される。修飾子は以下の通り。

EXP_H3210: 無加工

EXP_H1010: 下位 32bit を上位+下位 32bit にコピー

EXP_H3232: 上位 32bit を上位+下位 32bit にコピー

EXP_B5410: byte5,4,1,0 を各々 16bit に拡張し連結

EXP_B7632: byte7,6,3,2 を各々 16bit に拡張し連結

OP_X には主に算術演算、OP_Y には主に論理演算、OP_Z には主にシフト演算を記述できる。各演算は SIMD 型であり、上位 32bit と下位 32bit を独立に扱う。exe() は倍幅、ex4() は 8 倍幅の SIMD である。

```
exe(OP_X, &var, INIT0?var:var, e1, s2, e2, s3, e3, OP_Y, s4, OP_Z, s5)
```

INIT0?var:var は、C 言語としては常に var であるため冗長である。これは、CGRAにおいてホストの介入無しに多重ループに対応する工夫である。多重ループ起動前に、ホストは、内側ループに関する各初期値を CGRA 内にセットする。通常、内側ループ完了時にホストが介入して内部レジスタの再初期化を行う必要がある。しかし、IMAX では、INIT0?var:var が記述された変数は、CGRA が自ら再初期化を行い、ホスト介入のオーバヘッドを排除している。具体的には、var に初期値がセットされていることを前提に、LOOP0 の初回 (INIT0=1) は、ホストが予めセットした当該初期値を演算器の第 1 入力とし、次回以降は演算器出力を第 1 入力とするようデータパスを切替える。

```
exe(OP_X, &var, var, e1, INIT0?s2:0, e2, s3, e3, OP_Y, s4, OP_Z, s5)
```

これも、CGRAにおいて、ホストの介入無しに多重ループに対応する工夫である。INIT0?s2:0 が記述されている場合、LOOP0 の初回 (INIT0=1) は、s2 を演算器の第 2 入力とし、次回以降は 0 を第 2 入力とするようデータパスを切替える。2 次元サブアレイの先頭アドレス計算に利用できる。

```
mex(OP_MEX2, &s2, INIT0?s20:s2, INIT0?0:expr, OP_MEX1, &s1, INIT0?s10:s1
INIT0?0:expr, limit, BR[0-63][0-3][1], BR[0-63][0-3][0])
```

ホストの介入無しに多重ループ疎行列計算またはマージソートを行うためのアドレス計算補助記述である。INIT0?s20:s2 および INIT0?s10:s1 はベースアドレス、INIT0?0:expr は加算するオフセットに対応する。LOOP0 の初回 (INIT0=1) は、s2 と s1 に、s20 と s10 (初期値) が格納され、次回以降は、前回ローカルメモリから読み出した 2 つの 64bit データの上位 32bit を比較し、大小関係によって、s2 または s1、すなわちアドレス計算器出力に、0 または expr を加算した結果が格納される。また、limit はマージソート用の比較対象間距離である。使用方法はプログラム例を参照のこと。OP_MEX は以下の通り。

OP_NOP: 常に base

OP_ALWAYS: 常に base+offset

OP_CMPA_LT: 上位 32bit(BR[][],1] ≤ 上位 32bit(BR[],0]) なら base+offset。それ以外は base

OP_CMPA_GE: 上位 32bit(BR[],1] ≥ 上位 32bit(BR[],0]) なら base+offset。それ以外は base

```
mop(OP_X, ex9-0, &src|&dst, base, offset, mask, top, len, block, force, ptop, plen)
mo4(OP_X, ex9-0, &src|&dst, base, offset, mask, top, len, block, force, ptop, plen)
```

ローカルメモリに対するロードまたはストアを記述する。各項目は以下の通り。

OP_X: データ幅に応じた倍幅 SIMD 型ロード命令またはストア命令。mo4 は 8 倍幅 SIMD である

ex9-0: 無条件ストアの場合は定数 3, 前述の条件付きストアの場合は変数

src|dst: ロードの場合は格納先レジスタ, ストアの場合はストアデータ

base: 参照するメモリアドレス base+mask(offset) の base 部分。ホストの主記憶アドレスをそのまま使える。ローカルメモリ内アドレスへはコンパイラとハードウェアの連携により自動変換するので、アドレス変換を意識する必要はない

offset: 同じく offset 部分。offset の単位は 1byte である点に注意

mask: 最終的なアドレスは、base に、offset レジスタを mask により修飾した値を加えたもの。mask は以下の通り

MSK_B0: offset の bit7-0 を符号無し 64bit 拡張

MSK_B1: offset の bit15-8 を符号無し 64bit 拡張

MSK_B2: offset の bit23-16 を符号無し 64bit 拡張

MSK_B3: offset の bit31-24 を符号無し 64bit 拡張

MSK_B4: offset の bit39-32 を符号無し 64bit 拡張

MSK_B5: offset の bit47-40 を符号無し 64bit 拡張

MSK_B6: offset の bit55-48 を符号無し 64bit 拡張

MSK_B7: offset の bit63-56 を符号無し 64bit 拡張

MSK_H0: offset の bit15-0 を符号無し 64bit 拡張

MSK_H1: offset の bit31-16 を符号無し 64bit 拡張

MSK_H2: offset の bit47-32 を符号無し 64bit 拡張

MSK_H3: offset の bit63-48 を符号無し 64bit 拡張

MSK_W0: offset の bit31-0 を符号無し 64bit 拡張

MSK_W1: offset の bit63-32 を符号無し 64bit 拡張

MSK_D0: offset の bit63-0 をそのまま使用

top: バースト演算が参照するホスト主記憶領域の先頭アドレス。先頭アドレスと長さを用いてホスト DMA コントローラがホスト主記憶-LMM 間転送を行う。

len: 同じく長さ。単位は 1word (4byte) である。なお、DMA 転送速度は 256bit/cycle である。len に定数 0 を指定した場合、top により指定された領域が確保されるものの、DMA は抑止される。この指定は、同一 LMM によるダブルバッファリングを行う際に使用する。

block: IMAX では使用しない (EMAX5 では DMA ギャザ機能のパラメタ)

force: ロードとストアとで動作が異なる

0: ロードの場合、前回 DMA の先頭アドレスおよび長さが同一であれば DMA を起動せず LMM を再利用する。ストアの場合、バースト演算後に LMM からホスト主記憶にデータ転送する。ただし、次回バースト演算と同時の遅延ドレインが指定されている場合は DMA を抑止する。

1: ロードの場合、必ずホスト主記憶から LMM にデータ転送する。外部機器入力のように、アドレスが同じでも内容が異なる場合に使用する。ストアの場合、パーシャルストア（一部アドレスのみストア）を想定し、バースト演算前にホスト主記憶から一旦 LMM にデータ転送する。ただし、前回ストアとアドレス範囲が同じ場合は DMA を抑止する。

ptop: バースト演算中に使う DMA の先頭アドレス。ロードの場合、バースト演算中に次回バースト演算に必要な入力がホスト主記憶から LMM に転送される (プリフェッч). ストアの場合、バースト演算中に前回バースト演算結果が LMM からホスト主記憶に転送される (遅延ドレイン)。なお、mapdist=0 の場合、同一 LMM を対象に、LD/ST とプリフェッч/ドレインが同時動作する。

plen: 同じく長さ. 単位は 1word (4byte) である.

2.5 List of instructions

Table.2.2: Memory operations

unit	usage	mode	description
MEX	_ALWAYS,CMPA,_LE,_GE (&base, INIT0?init:base, INIT0?0:[0-64], s2, s1)	sparse matrix	LMM indexed-access
LD	LDR (-, (Ull*)d, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	64bit lmm	LMM rand-access
	LDWR (-, (Ull*)d, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	u32bit lmm	LMM rand-access
	LDBR (-, (Ull*)d, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	u8bit lmm	LMM rand-access
	LDRQ (-, (Ull*)d, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	64bit*4 lmm	LMM rand-access
	LDDMQ (ex(b0), (Ull*)d, base(++) , off, msk, -, -, -, -, -, -)	64bit*4 mem	Direct access to MM
ST	STR (ex, s, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	64bit lmm	LMM rand-access
	STWR (ex(b0), s, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	32bit lmm	LMM rand-access
	STBR (ex(b0), s, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	8bit lmm	LMM rand-access
	STRQ (-, (Ull*)s, base(++) , off, msk, top, len, blk, force-read, ptop, plen)	64bit*4 lmm	LMM rand-access
	TR (ex(b0), (Ull*)s, -, -, -, func(), -, -, -, -, -)	64bit*4 exec	Send transaction
<ul style="list-style-type: none"> - pair of LDR(BR[0][1], adr+8) and LDR(BR[0][0], adr) w/ 8B-aligned adr can load two 64bit data to BR[0][1/0] from LMM. - pair of LDR(BR[0][1], adr+8) and LDR(BR[0][0], adr) w/ 8B-unaligned adr can load 64bit data to BR[0][0] from LMM. - ex is 2bit (b1 controls word1, b0 controls word0) - msk is 14:64bit, 13:word1, 12:word0, 11-8:half3-0, 7-0:byte7-0 of offset - base++ increments address by the size of element. 			
<ul style="list-style-type: none"> - LD with force-read=0 and ptop==NULL generates current(lmr) and reuse LMM 実行前に主記憶から LMM データ転送。先頭アドレスが前回と同一の場合は再利用 - LD with force-read=1 and ptop==NULL generates current(lmf) and !reuse LMM 実行前に必ず主記憶から LMM データ転送。動画データ等、先頭アドレスが同一でも内容が異なる場合に使用 - LD with force-read=0 and ptop!=NULL generates current(lmr) and next(lmp) 実行前の主記憶から LMM へのデータ転送はせず、次回実行中にプリフェッч mapdist=0 の場合、同一 LMM にて、LD と実行中プリフェッチを同時実行 - LDDMQ set f=1 and p=1 in lmmc automatically 			
<ul style="list-style-type: none"> - ST with force-read=0 and ptop==NULL generates current(lmw) and reuse+wback LMM 実行後に LMM から主記憶 M データ転送 - ST with force-read=1 and ptop==NULL generates current(lmx) and !reuse+wback LMM 先頭アドレスが前回と同一の場合、実行後の LMM から主記憶へのデータ転送はない 先頭アドレスが変化しない LMM への追加的累算の場合、初回のみ主記憶から LMM へ初期値を転送し、 途中の主記憶-LMM 間転送はしない、最後に先頭アドレスが変化した際に主記憶へ書き戻す - ST with force-read=0 and ptop!=NULL generates current(lmw) and prev(lmd) 実行後の LMM から主記憶へのデータ転送はせず、次回実行中にドレイン mapdist=0 の場合、同一 LMM にて、ST と実行中ドレインを同時実行 - TR set f=1 and p=1 in lmmc automatically 			

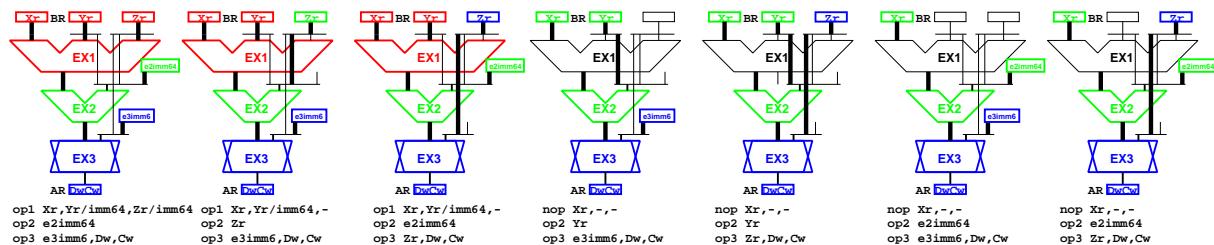


Figure.2.7: レジスタと演算器の接続ネットワーク

表 2.2 に、CGRA に写像可能なロード/ストア記述様式を示す。LDDMQ は外部メモリ直接参照、TR はトランザクション発行、残りは LMM のランダム参照用である。プログラミングの際には、まず、ストアデータに着目し、LMM に対する単一ストアが、必ずストア幅にアラインされるよう留意する。例えば 32bit 整数データを 2 つ連結した 64bit データをストアする際には、64bit にアラインしなければならない。一方、ロードデータに関しては、1 つの LMM から 64bit データを 4 つ同時に読み出す際には 256bit 境界を意識しなければならないものの、2 ポートを利用して 2 箇所から 32bit データを読み出す際には 32bit 境界のみを意識すればよい。幅方向の 4 つの LMM に同じデータを格納することにより、一度に 8 箇所からのランダム読み出しが可能であり、ストアよりも境界に関する自由度が高い。この性質を利用すれば様々なステンシル計算を無理なく写像することができる。

表 2.3 に、CGRA に写像可能な演算記述様式を示す。レジスタ幅は 64bit であり、基本データ型は、浮動小数点演算では单精度演算の 2 要素連結、整数演算では 32bit 幅の 2 要素連結、メディア演算では 16bit 幅

Table.2.3: ALU operations

unit	usage	mode	description
CEX	CEXE ((char*)ex, c3, c2, c1, c0, imm)	2bit 4in 16bit	word-wise(w1/w0) conditional execution
EX 1,2,3	ex4.SFMA (Ull*)d s1, (s1, (Ull*)s2, (Ull*)s3, s4)	8bit*32 3in	stochastic s1+s2*s3 div=s4 -l 8bit
	ex4.FMA (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	32bit*2*4 3in	floating-point s1+s2*s3
	ex4.FMS (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	32bit*2*4 3in	floating-point s1-s2*s3
	ex4.FAD (Ull*)d s1, ((Ull*)s1, (Ull*)s2, -)	32bit*2*4 2in	floating-point s1+s2
	ex4.FML (Ull*)d s1, ((Ull*)s1, (Ull*)s2, -)	32bit*2*4 2in	floating-point s1*s2
EX1	CFMA (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*1 3in	floating-point s1+s2*s3 if hi(s2)==hi(s3)
	FMA (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*2 3in	floating-point s1+s2*s3
	FMS (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*2 3in	floating-point s1-s2*s3
	FAD (Ull*)d &s1, (s1, exp, s2, exp, -, -)	32bit*2 2in	floating-point s1+s2
	FML (Ull*)d &s1, (s1, exp, s2, exp, -, -)	32bit*2 2in	floating-point s1*s2
EX	ex4.ADD3 (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	32bit*2*4 3in	integer add s1+s2+s3
	ex4.SUB3 (Ull*)d s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)	32bit*2*4 3in	integer subtract s1-(s2+s3)
	ex4.ADD (Ull*)d s1, ((Ull*)s1, (Ull*)s2, -)	32bit*2*4 2in	integer add s1+s2
	ex4.SUB (Ull*)d s1, ((Ull*)s1, (Ull*)s2, -)	32bit*2*4 2in	integer subtract s1-s2
	ADD3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*2 3in	integer add s1+(s2+s3)
	SUB3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*2 3in	integer subtract s1-(s2+s3)
	ADD (Ull*)d &s1, (s1, exp, s2, exp, -, -)	32bit*2 2in	integer add s1+s2
	SUB (Ull*)d &s1, (s1, exp, s2, exp, -, -)	32bit*2 2in	integer subtract s1-s2
	CMP_{EQ NE LT LE GT GE} ((Ull*)d, s1, s2)	32bit*2 2in	word-wise(w1/w0) compare and set 1*2bit-CC
	CMOV (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	2,32bit*2 2in	word-wise(w1/w0) conditional move
EX2	MAUH3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	16bit*4 3in	s1.exp+(s2.exp+s3.exp)
	MAUH (Ull*)d &s1, (s1, exp, s2, exp, -, -)	16bit*4 2in	s1.exp+s2.exp
	MSUH3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	16bit*4 3in	s1.exp-(s2.exp+s3.exp)
	MSUH (Ull*)d &s1, (s1, exp, s2, exp, -, -)	16bit*4 2in	s1.exp-s2.exp
	MLUH (Ull*)d &s1, (s1, exp, s2, exp, -, -)	(11bit*4)*9bit	s1.exp*s2.exp
	MMRG (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	8bit*2 3in	s1.b4 s2.b4 s3.b4 0→w1,s1.b0 s2.b0 s3.b0 0→w0
	MSSAD (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	16bit*4	s1.h3+df(s2.b7,s3.b7)+df(s2.b6,s3.b6)→d.h3
		8bit*8 2in	s1.h2+df(s2.b5,s3.b5)+df(s2.b4,s3.b4)→d.h2
			s1.h1+df(s2.b3,s3.b3)+df(s2.b2,s3.b2)→d.h1
			s1.h0+df(s2.b1,s3.b1)+df(s2.b0,s3.b0)→d.h0
EX3	MSAD (Ull*)d &s1, (s1, exp, s2, exp, -, -)	16bit*4 8bit*8 2in	df(s1.b7,s2.b7)+df(s1.b6,s2.b6)→d.h3 df(s1.b5,s2.b5)+df(s1.b4,s2.b4)→d.h2 df(s1.b3,s2.b3)+df(s1.b2,s2.b2)→d.h1 df(s1.b1,s2.b1)+df(s1.b0,s2.b0)→d.h0
	MINL3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	16bit*4 3in	(s3.h3< s3.h2)?s1.h3 s3.h3:s2.h3 s3.h2→d.w1 (s3.h1< s3.h0)?s1.h1 s3.h1:s2.h1 s3.h0→d.w0
	MINL (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	16bit*4 2in	(s1.h2< s2.h2)?s1.w1:s2.w1→d.w1 (s1.h0< s2.h0)?s1.w0:s2.w0→d.w0
	MH2BW (Ull*)d &s1, (s1, exp, s2, exp, -, -)	16bit*4 2in	s1.b6 s1.b4 s2.b6 s2.b4 s1.b2 s1.b0 s2.b2 s2.b0
	MCAS (Ull*)d &s1, (s1, exp, s2, exp, -, -)	16bit*2 2in	(s1.h2< s2.h2)?0:0xff→d.b4 (s1.h0< s2.h0)?0:0xff→d.b0
	MMID3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	8bit*8 3in	bytewise compare and output middle
	MAXM3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	8bit*8 3in	bytewise compare and output maximum
	MMIN3 (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	8bit*8 3in	bytewise compare and output minimum
	MAX (Ull*)d &s1, (s1, exp, s2, exp, -, -)	8bit*8 2in	bytewise compare and output maximum
	MMIN (Ull*)d &s1, (s1, exp, s2, exp, -, -)	8bit*8 2in	bytewise compare and output minimum
EX2	MAJ (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*2 3in	(r1&0xffffffff0..0LL) (((r1&r2)^(r1&r3)^(r2&r3))&0xffffffffLL) (r1&0xffffffff0..0LL) (((r1&r2)^(~r1&r3))&0xffffffffLL)
	CH (Ull*)d &s1, (s1, exp, s2, exp, s3, exp)	32bit*2 3in	
	AND (Ull*)d, (s4)	cascadable	64bit 2in
	OR (Ull*)d, (s4)	cascadable	64bit 2in
	XOR (Ull*)d, (s4)	cascadable	64bit 2in
EX3	SUMHH (Ull*)d, (-)	cascadable	16bit*4 1in
	SUMHL (Ull*)d, (-)	cascadable	16bit*4 1in
	ROTS (Ull*)d, (s4)	cascadable	32bit*2 2in
	SLL (Ull*)d, (s5)	cascadable	32bit*2 2in
	SRL (Ull*)d, (s5)	cascadable	32bit*2 2in
EX3	SRAA (Ull*)d, (s5)	cascadable	32bit*2 2in
	SRAB (Ull*)d, (s5)	cascadable	32bit*2 2in
	SRLM (Ull*)d, (s5)	cascadable	16bit*4 2in
- exp is 0:64bit→16bit[4], 1:low32bit→32bit[2], 2:high32bit→32bit[2] - exp is 3:byte5,4,1,0→16bit[4], 4:byte7,6,3,2→16bit[4] - (Ull*)s1,(Ull*)s2,(Ull*)s3,(Ull*)d are pointers and s1, s2, s3 are values - c3, c2, c1, c0 are 1*2bit-values, and (char*)ex is a pointer - .b[7:0] is 8bit portion, .h[3:0] is 16bit portion and .w1 w0 is 32bit portion in 64bit respectively - d s1 is d:normal operation, s1:accumulation. succeeding operations with s1 reffer the result.			

の4要素連結または8bit幅の8要素連結である。AND, OR, XOR, SUMHH, SUMHLは、単独で前段の出力レジスタを入力とできることに加え、integer-add/subの出力dを入力に指定することにより、UNIT内カスケード演算が可能である。さらに、SLL, SRL, SRAA, SRAB, SRLMは、単独で前段の出力レジスタを入力とできることに加え、integer-add/subの出力dや、AND, OR, XOR, SUMHH, SUMHLの出力Dを入力に指定することにより、UNIT内カスケード演算が可能である。すなわち、UNIT内で最大、integer-add/sub、論理演算、シフト演算の順に3つの演算をカスケード実行できる。CMP, CESE, CMOVは、条件付き実行のための記述である。CMPにより条件コードを生成し、CESEにより複数の条件コードを組合せるロードストア用実行条件を生成でき、CMOVにより单一の条件コードによる条件付き代入を行う。CEXの出力は前述のロード/ストア記述の実行条件exに使用できる。図2.7に、演算記述と演算器写像の関係を示す。

2.6 Overlapping of prefetch and drain

ステンシル計算では、(1)直前のバースト演算がLMMに格納した演算結果の主記憶への書き戻し、(2)LMMに揃っている入力データを利用した演算器のバースト動作、および、(3)次のバースト演算に必要なデータの主記憶からLMMへのプリフェッチの全てを同時動作させることにより、高効率を達成できる。本動作を利用する際の記述方法として、プログラマが(1)～(3)の全てを明示的に記述する方法と、(2)のみを記述し(1)(3)はコンパイラが自動生成する方法が考えられる。前者の場合、最初のバースト動作では入力データが揃っていないため(3)のみを有効化し、同様に最終のバースト動作では(1)のみを有効化する仕組みが必要となる。後者の場合、単純なステンシル計算でなければ(2)の情報から(3)を自動生成できず、離散ステンシルに対応できない。また、最終の(2)を検出して単独の(1)を自動的に追加することも困難である。以上のことから、IMAXでは前者を採用している。図2.8は、本動作に従うkernel_cgра()の例である。(1)に関しては、kernel_top()内のpost_processing部分において、emax5_drain_dirty_lmm()を呼び出すことにより、STRQの格納先LMMに残っている演算結果を主記憶に書き出す動作を指示する。(3)に関しては、LDRQの最後の3つの引数により、LMMからの読み出しと同時に、次段(段間の距離は、map_distにより指定)のLMMにプリフェッチするための先頭アドレス、距離、長さを指示する。

```
#define XSIZE 1024
#define YSIZE 1024
double A[YSIZE][XSIZE];
double B[YSIZE][XSIZE];
double C[YSIZE][XSIZE];
double D[YSIZE][XSIZE];

kernel_top(status) int *status; /* CPU always starts from kernel_top() */
{
    pre_processing;
    for (y=0; y<YSIZE; y++)
        kernel_cgра(y);
    //EMAX5A drain_dirty_lmm ← (1) ★
    return (status);
}
kernel_cgра(y) int y; /* If CGRA is not available, CPU executes. */
{ /* D = A * B + C */
//EMAX5A begin map_dist=1
    while (loop--) { /* mapped to while() on BR[15][0][0] */
        mo4(LDRQ, 1, src1, a++, 0, msk, A[y], XSIZE, 0, 0, A[y+1]); /* prefetch */← (3) ★
        mo4(LDRQ, 1, src2, b++, 0, msk, B[y], XSIZE, 0, 0, B[y+1]); /* prefetch */← (3) ★
        mo4(LDRQ, 1, src3, c++, 0, msk, C[y], XSIZE, 0, 0, C[y+1]); /* prefetch */← (3) ★
        ex4(FMA, dst1, src1, src2, src3);
        mo4(STRQ, 1, dst1, d++, 0, msk, NULL, XSIZE, 0, 0, D[y-1]); /* late drain */← (1) ★
    }
//EMAX5A end
}
```

Figure.2.8: プリフェッチ/ドレインとのオーバラップ実行例

2.7 マルチチップ対応

C 言語により記述するアプリケーションプログラムは、アクセラレーション対象区間がIMAX コンパイラにより ARMv8 バイナリとして埋め込まれ、ARM がアクセラレーション対象範囲に入ると、IMAX の写像情報を所定の主記憶アドレスに対して書き込みを行う。ARM は、必要なデータを DMA により IMAX の LMM 群に転送した後、IMAX を起動する。動作するアプリケーションには、ステレオマッチング・Light field レンダリング・超解像化等画像処理 11 種、科学技術ステンシル計算 5 種 (grapes, jacobi, fd6, resid, wave2d), 逆行列求解カーネル 3 種、畳み込み・誤差逆伝搬等深層学習テンソルカーネル 6 種、VBGMM カーネル 3 種、文字列検索などがあり、IMAX の多重ループ実行機能およびマルチチップ機能を利用するため、記述能力の向上とコンパイラの機能拡張を行った。記述方法の決定に際して重視したのは、同一ソースプログラムから、CPU 向けの通常コンパイラでも同一の結果を出力するバイナリを生成できる点である。図 2.9 に拡張後の記述例を示す。//EMAX5A に始まる 3 重ループ記述が、IMAX の 1 回の起動においてシステム全体に写像される機能である。最外ループが複数チップ、中間および最内ループが各チップにおける多重ループ実行機能に対応付けられる。このソースプログラムは、各 unit の機能を定義する exe(演算) や mop(メモリ参照) をライブラリ化しておくことにより、通常コンパイラでもバイナリを生成でき、また、通常のデバッガによりアルゴリズムの検証を行うことができる。アルゴリズムを確認した後に、同様に多重ループ実行機能およびマルチチップ機能に対応させた IMAX コンパイラ (C 言語等にて 9K 行の規模) を用いて IMAX 用のバイナリを生成する。このように、プログラムが意図した動作をしない場合、原因がアルゴリズムにあるのか実装にあるのかを切り分ける手段が用意されていることは、CGRA のプログラム開発環境として必須条件である。このような手段を備えることができる原因是、IMAX の全 unit が、通常の CGRA に見られる二次元配置ではなく、線形 (リニアアレイ) に配置されているためである。

```
//EMAX5A begin x1 mapdist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) {
/*2*/ for (INIT1=1,LOOP1=RMGRP,rofs=0-M*4; LOOP1--; INIT1=0)
/*1*/ for (INIT0=1,LOOP0=M-2,cofs=(0-4)&0x00000000ffffffffLL; LOOP0--; INIT0=0) {
    exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?M*4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    exe(OP_ADD, &oofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    mop(OP_LDWR, 1, &BR[2][0][1], (U11)kp00[CHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K);
    mop(OP_LDWR, 1, &BR[2][0][0], (U11)kp01[CHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K);
    mop(OP_LDWR, 1, &BR[2][1][1], (U11)kp02[CHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K);
    mop(OP_LDWR, 1, &BR[2][1][0], (U11)kp03[CHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K);
    mop(OP_LDWR, 1, &BR[2][2][1], (U11)ip00, oofs, MSK_DO, (U11)it00, M*RMGRP, 0, 0, (U11)NULL, M*RMGRP);
```

Figure.2.9: for 文によるマルチチップと多重ループの記述例

2.8 動的 DMA 連結と DMA 起動タイミング

通常、mop や mo4 ごとに、1 つの DMA が起動される。ただし、top および len が、文字列として同一である場合、コンパイラにより、同一論理 UNIT 内、同一行内、同一 IMAX 内、全 IMAX の順に、全ての DMA が 1 つにまとめられる。一方、動的 DMA 連結は、文字列として同一でなくとも、実行時に、現行の top+len が、次行の top に等しいかどうかを検査し、同じである場合、すなわち、複数の LMM が、全体として連続メモリ領域である場合に、DMA を 1 つにまとめる機能である。IMAX2 の各 LMM は、自己の担当範囲を知っており、AXI インタフェースから流れてくるアドレス情報を常時検査し、該当すれば READ/WRITE を行う。動的 DMA 連結により DMA の起動回数を削減すれば、機能を損なうことなく、全体を高速化できる。動的 DMA 連結を含む、force_read とプリフェッчの組み合わせに対する実際の DMA 起動条件は図 2.10 の通りである。

```

emax6_check_lmmi_and_dma(int mode, int phase, int lastdist, int c, int i, int j)
{
    int k, m = (i+lastdist)%EMAX_DEPTH; /* lmmo-index */
    int lmmc_topz;
    int lmmc_ofsz;
    int lmmo_stat;
    int lmmc_stat;
    int lmm_ready;
    int lmm_readz;
    int mark;
    struct lmmi *lmmiop = &emax6.lmmi[c][m][j][emax6.lmmio];
    struct lmmi *lmmicp = &emax6.lmmi[c][i][j][emax6.lmmic];
    struct lmmi *lmmiop1 = &emax6.lmmi[c][(m+i)%EMAX_DEPTH][j][emax6.lmmio];
    struct lmmi *lmmicp1 = &emax6.lmmi[c][(i+1)%EMAX_DEPTH][j][emax6.lmmic];
    U11 dmadr;
    int dmlen;
    U11 dmnxz;
    int dmrw; /* 0:mem->lmm 1:lmm->mem */
    static U11 concat_adr[EMAX_NCHIP]; /* NULL:invalid, !NULL:top_addr */
    static int concat_len[EMAX_NCHIP]; /* byte-len */
    /* check_lmmi */
    if ((phase == 1 && mode == 0) || phase == 2 || phase == 3) { /* (drain && array) || load || exec */
        lmmc_topz = (lmmicp->top == 0);
        lmmc_ofsz = (lmmicp->ofs == 0);
        lmmo_stat = ((lmmiop->rw<<2)|(lmmiop->f<<1)|(lmmiop->p)); /* v/rw/f/p */
        lmmc_stat = ((lmmicp->v & ~lmmicp->hcropy & ~lmmicp->vcopy & ((lmmicp->f&lmmicp->p) | !lmmc_topz))<<3)|(lmmicp->rw<<2)|(lmmicp->f<<1)|(lmmicp->p);
        lmm_ready = (lmmiop->v && lmmiop->blk == lmmicp->blk && lmmiop->len == lmmiop->len && lmmiop->top == lmmicp->top);
        lmm_readz = (lmmiop->v && lmmiop->blk == lmmicp->blk && lmmiop->len == lmmiop->len && (lmmiop->top+(S11)(int)lmmiop->ofs) == lmmicp->top);
    }
    /* lmx: bitmap を検査し、現 addr+len と次 addr を比べ、連続なら連結した次 addr/len を保存。最終または不連続なら保存 addr/len または現 addr/len を使つて DMA */
    if (phase == 1) { /* drain */
        if ((mode==0 && lmmo_stat!=13 && (emax6.lmmd[m][j]&=(1<<c)); /* 2 lmv&!lmd */
            { mark=1; emax6.lmmd[m][j]&=(1<<c); dmadr=lmmiop->top; dmlen=lmmiop->len; dmnxz=lmmiop1->top; dmrw=1; }/* ● 2 lmv&!lmd drain */
        else if (mode==0 && lmmo_stat!=14 && !lmm_ready && (emax6.lmmd[m][j]&=(1<<c));
            { mark=1; emax6.lmmd[m][j]&=(1<<c); dmadr=lmmiop->top; dmlen=lmmiop->len; dmnxz=lmmiop1->top; dmrw=1; }/* ● 4 lmx drain */
        else if (mode==1 &&
            { mark=1; emax6.lmmd[i][j]&=(1<<c); dmadr=lmmiop->top; dmlen=lmmiop->len; dmnxz=lmmiop1->top; dmrw=1; }/* ☆ drain_dirty_lmm */
        else
            { mark=0; }
        }
        else if (phase == 2) { /* load */
            if ((lmmc_stat== 8 && !lmm_ready) /* ● 1 lmr & !ready */
                || (lmmc_stat== 9 && !lmm_readz) /* ● 7 lmp & !readz */
                || (lmmc_stat==10 ) /* ● 3 lmf always load */
                || (lmmc_stat==14 && !lmm_ready)) { mark=1; dmadr=lmmicp->top; dmlen=lmmicp->len; dmnxz=lmmicp1->top; dmrw=0; }/* ● 3 lmx always load */
            else
                { mark=0; }
        }
        else if (phase == 3) { /* exec */
            if ((lmmc_stat== 9 && (!lastdist||!lmmc_ofsz)) { mark=1; dmadr=lmmicp->top; dmlen=lmmicp->len; dmrw=0; }/* ● 5 lmp */
            else if (lmmc_stat==12||lmmc_stat==14){mark=0; emax6.lmmd[i][j]&=(1<<c); /* 6 lmv/lmx */
            else if (lmmc_stat==13 ){mark=emax6.lmmd[m][j]&=(1<<c); emax6.lmmd[m][j]&=((!lastdist)<<c); dmadr=lmmicp->top; dmlen=lmmicp->len; dmnxz=lmmiop1->top; dmrw=1; }/* ● 6 lmd & dirty */
            else
                { mark=0; }
        }
        if (mark) {
            if (phase == 1) { /* drain */
                /* concat_adr=0 adr0,L=0 | adr1,L=0 | adr2,L=0 */
                /* concat_adr=adr0,L=0 adr0,L=0,mark=0 | adr1,L=0 | adr2,L=0 */
                /* concat_adr=adr0,L=1 mark=0 | adr1,L=0,mark=0 | adr2,L=0 */
                /* concat_adr=adr0,L=2 mark=0 | mark=0 | adr2,L=0,mark=1 */
                if ((emax6.lmmd[(m+i)%EMAX_DEPTH][j]&(1<<c)) && (dmadr+(dmlen+1)*sizeof(UInt)) == dmnxz) {
                    if (!concat_adr[c]) { concat_adr[c] = dmadr; concat_len[c] = dmlen; }
                    else
                        { concat_len[c] += dmlen+1; }
                    if (concat_len[c] < 8192) mark = 0;
                }
                else {
                    if (concat_adr[c])
                        { concat_len[c] += dmlen+1; }
                }
            }
            else if (phase == 2) { /* load */
                if (lmmiop->v && (dmadr*(dmlen+1)*sizeof(UInt)) == dmnxz) {
                    if (!concat_adr[c]) { concat_adr[c] = dmadr; concat_len[c] = dmlen; }
                    else
                        { concat_len[c] += dmlen+1; }
                    if (concat_len[c] < 8192) mark = 0;
                }
                else {
                    if (concat_adr[c])
                        { concat_len[c] += dmlen+1; }
                }
            }
        }
        /* dma */
        if (mark) {
            emax6.rw = dmrw;
            if (phase == 1) { /* drain */
                emax6.ddraddr = (concat_adr[c])?concat_adr[c]:dmadr; /* address should be 4B-aligned */
                emax6.lmmaddr = emax6.ddraddr;
                emax6.dmalen = (concat_adr[c])?concat_len[c]:dmlen; /* length should be # of words */
            }
            else if (phase == 3 && dmrw==1) { /* pdrain */
                emax6.ddraddr = dmadr+(S11)(int)lmmicp->ofs; /* ★★★ PDRAIN address should be 4B-aligned */
                emax6.lmmaddr = emax6.ddraddr;
                emax6.dmalen = dmlen; /* length should be # of words */
            }
            else if (phase == 2 /* load */ ||(phase == 3 && dmrw==0)) { /* inf */
                if (lmmiop->blk==0) { /* inf */
                    if (phase == 2) { /* load */
                        emax6.ddraddr = (concat_adr[c])?concat_adr[c]:dmadr; /* address should be 4B-aligned */
                        emax6.lmmaddr = emax6.ddraddr;
                        emax6.dmalen = (concat_adr[c])?concat_len[c]:dmlen; /* length should be # of words */
                    }
                    else {
                        emax6.ddraddr = dmadr+(S11)(int)lmmicp->ofs; /* ★★★ PLOAD address should be 4B-aligned */
                        emax6.lmmaddr = emax6.ddraddr;
                        emax6.dmalen = dmlen; /* length should be # of words */
                    }
                }
                concat_adr[c] = 0;
                emax6_kick_dma(j);
            }
        }
    }
}

```

Figure.2.10: 動的 DMA 連結と DMA 起動条件

Chapter 3

Examples

3.1 Tuning of applications

Tone_curve を対象に、チューニング方法を例示する。



Figure 3.1: Tone curve

3.1.1 C によるアルゴリズムの記述

```
/* CPU */
for (row=0; row<HT; row++) {
    for (col=0; col<WD; col++) {
        Uint pix = hin[row*WD+col];
        hout0[row*WD+col]
            = ((ht[pix>>24])<<24 | (ht[256+((pix>>16)&255])<<16 | (ht[512+((pix>>8)&255)])<<8;
    }
}
```

3.1.2 CUDA によるアルゴリズムの記述

```
/* GPU */
if (cudaSuccess != cudaMemcpy(din, hin, sizeof(Uint)*WD*HT, cudaMemcpyHostToDevice))
    { printf("can't cudaMemcpy\n"); exit(1); }
if (cudaSuccess != cudaMemcpy(dt, ht, sizeof(Uint)*256*3, cudaMemcpyHostToDevice))
    { printf("can't cudaMemcpy\n"); exit(1); }
dim3 Thread = dim3(THREADX, THREADY, 1);
dim3 Block = dim3(BLOCKX, BLOCKY, 1);
tone_curve<<<Block,Thread>>>(dout, din, dt); /* search triangle in {frontier,next} */
if (cudaSuccess != cudaMemcpy(hout1, dout, sizeof(Uint)*WD*HT, cudaMemcpyDeviceToHost))
    { printf("can't cudaMemcpy\n"); exit(1); }
__global__ void tone_curve(Uint *out, Uint *in, Uchar *t)
{
    int row, col;
    row = blockIdx.y*blockDim.y + threadIdx.y;
    col = blockIdx.x*blockDim.x + threadIdx.x;
    Uint pix = in[row*WD+col];
    out[row*WD+col] = ((t[pix>>24])<<24 | (t[256+((pix>>16)&255])<<16 | (t[512+((pix>>8)&255)])<<8;
    __syncthreads();
}
```

3.1.3 IMAX 向け C 言語記述（毎サイクル 1 画素を出力）

```

void tone_curve(r, d, t)
    unsigned int *r, *d;
    unsigned char *t;
{
    U11 t1 = t;
    U11 t2 = t+256;
    U11 t3 = t+512;
    U11 BR[16][4][4]; /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15,
        r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    int loop=WD;
//EMAX5A begin tone_curve mapdist=0
    while (loop--) {
        mop(OP_LDWR, 1, &BR[0][1][1], (U11)(r++), OLL, MSK_D0, (U11)r, 320, 0,0, (U11)NULL, 320); /* stage#0 */
        mop(OP_LDBR, 1, &BR[1][1][1], (U11)t1, BR[0][1][1], MSK_B3, (U11)t1, 64, 0,0, (U11)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][2][1], (U11)t2, BR[0][1][1], MSK_B2, (U11)t2, 64, 0,0, (U11)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][3][1], (U11)t3, BR[0][1][1], MSK_B1, (U11)t3, 64, 0,0, (U11)NULL, 64); /* stage#1 */
        exe(OP_MMRG, &r1, BR[1][1][1], EXP_H3210, BR[1][2][1], EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, 0, OP_NOP, 0);
        mop(OP_STWR, 3, &r1, (U11)(d++), OLL, MSK_D0, (U11)d, 320, 0,0, (U11)NULL, 320); /* stage#2 */
    }
//EMAX5A end
}

```

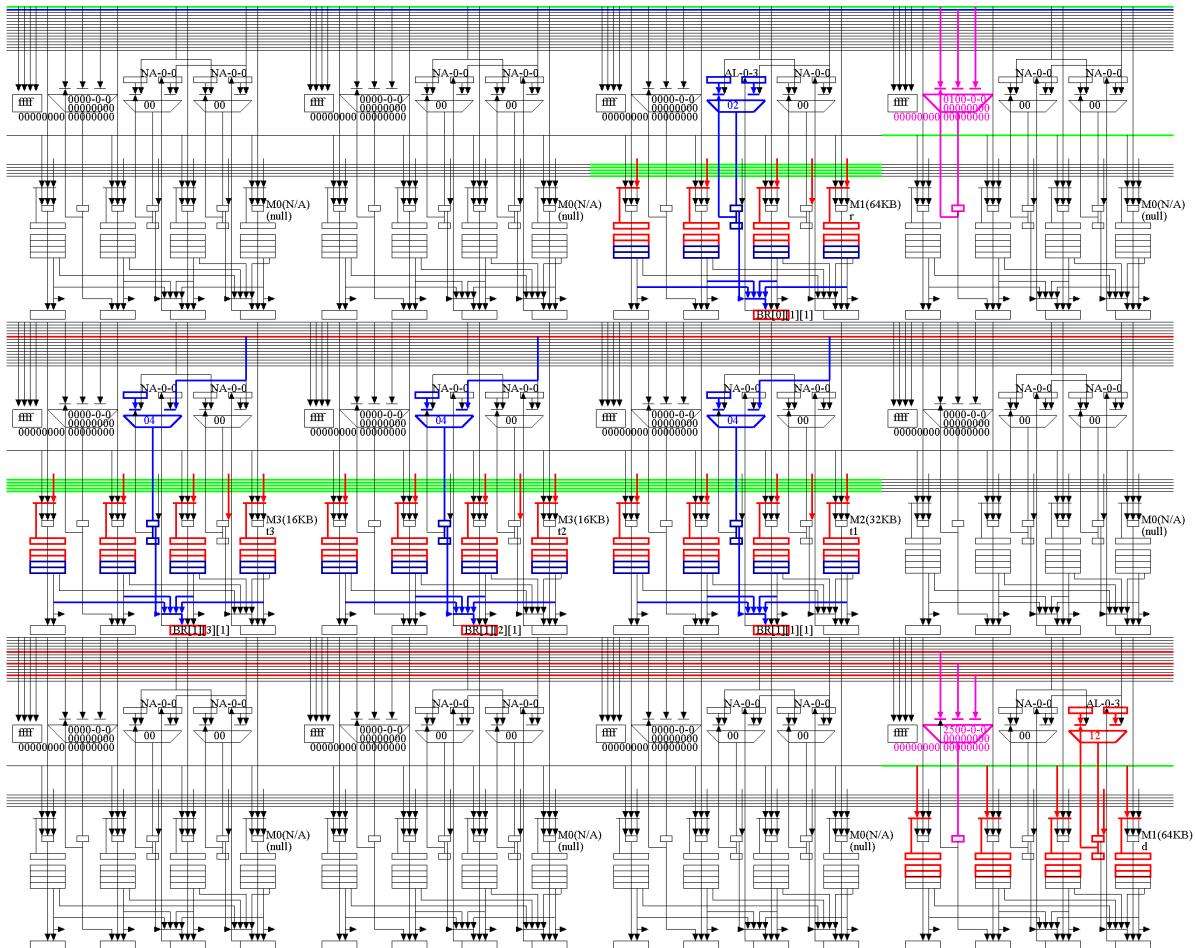


Figure.3.2: Tone curve 1 pixel per cycle

3.1.4 IMAX 向け C 言語記述（毎サイクル 2 画素を出力）

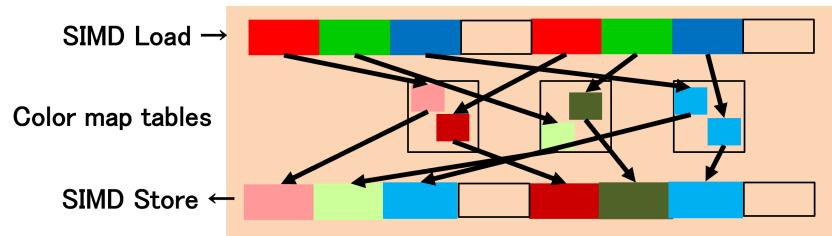


Figure.3.3: Dual tone curve

```

void tone_curve(r, d, t)
    unsigned int *r, *d;
    unsigned char *t;
{
    Ull  t1 = t;
    Ull  t2 = t+256;
    Ull  t3 = t+512;
    Ull  BR[16][4][4]; /* output registers in each unit */
    Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, , r31;
    int loop=WD/2;
//EMAX5A begin tone_curve mapdist=0
    while (loop--) {
        mop(OP_LDR, 1, &BR[0][1][1], (Ull)(rr++), OLL, MSK_D0, (Ull)r, 320, 0, 0, (Ull)NULL,320); /* stage#0 */
        mop(OP_LDBR, 1, &BR[1][1][1], (Ull)t1, BR[0][1][1], MSK_B3, (Ull)t1, 64, 0, 0, (Ull)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][1][0], (Ull)t1, BR[0][1][1], MSK_B7, (Ull)t1, 64, 0, 0, (Ull)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][2][1], (Ull)t2, BR[0][1][1], MSK_B2, (Ull)t2, 64, 0, 0, (Ull)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][2][0], (Ull)t2, BR[0][1][1], MSK_B6, (Ull)t2, 64, 0, 0, (Ull)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][3][1], (Ull)t3, BR[0][1][1], MSK_B1, (Ull)t3, 64, 0, 0, (Ull)NULL, 64); /* stage#1 */
        mop(OP_LDBR, 1, &BR[1][3][0], (Ull)t3, BR[0][1][1], MSK_B5, (Ull)t3, 64, 0, 0, (Ull)NULL, 64); /* stage#1 */
        exe(OP_CCAT, &r1, BR[1][1][0], EXP_H3210, BR[1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_CCAT, &r2, BR[1][2][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_CCAT, &r3, BR[1][3][0], EXP_H3210, BR[1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_MMRG, &r0, r1, EXP_H3210, r2, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        mop(OP_STR, 3, &r0, (Ull)(dd++), OLL, MSK_D0, (Ull)d, 320, 0, 0, (Ull)NULL,320); /* stage#2 */
    }
//EMAX5A end
}

```

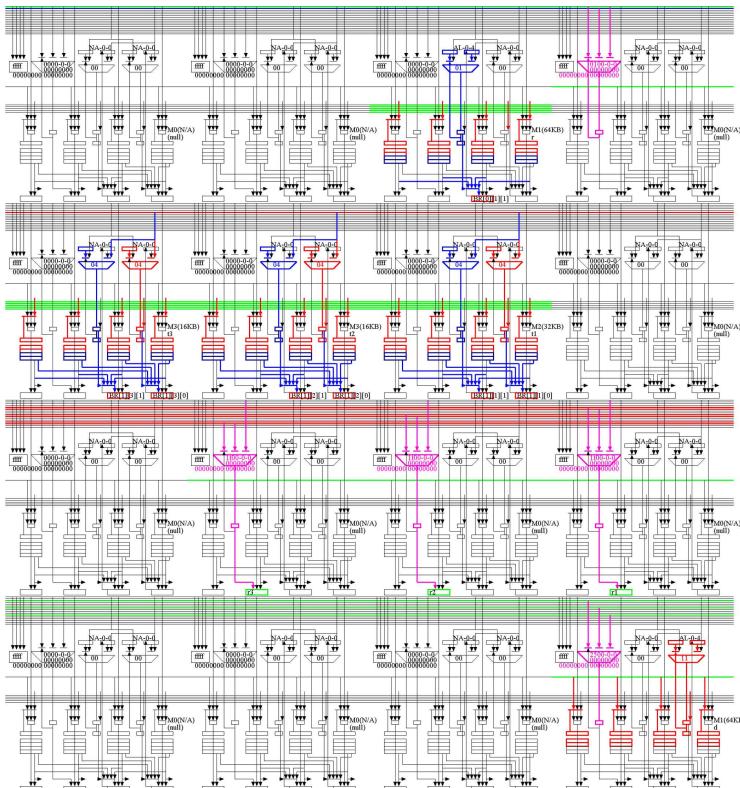


Figure.3.4: Tone curve 2 pixels per cycle

3.1.5 IMAX 向け C 言語記述（複数段利用，2重ループ一括実行，マルチチップ実行）

```

void tone_curve(Uint *r, Uint *d, Uchar *t) /* R, D, lut */
{
#define NCHIP      1
#define RMGRP      6
#define OMAP       10
#define PAD        0
#define RRANGE   ((HT-PAD*2)/NCHIP/OMAP)

    int i;
    for(i=0; i<256; i++) {
        t[i+ 0] = 0xff-i;
        t[i+256] = 0xff-i;
        t[i+512] = 0xff-i;
    }

    U11 top, rofs, cofs, oc, pofs;
    U11 t1 = t;
    U11 t2 = t+256;
    U11 t3 = t+512;
    U11 CHIP;
    U11 LOOP1, LOOP0;
    U11 INIT1, INIT0;
    U11 AR[64][4];           /* output of EX      in each unit */
    U11 BR[64][4][4];        /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 cc0, cc1, cc2, cc3, ex0, ex1;
    for (top=0; top<RRANGE; top+=RMGRP) {
        U11 rtop0[NCHIP], dtop0[NCHIP];
        U11 rtop1[NCHIP], dtop1[NCHIP];
        U11 rtop2[NCHIP], dtop2[NCHIP];
        U11 rtop3[NCHIP], dtop3[NCHIP];
        U11 rtop4[NCHIP], dtop4[NCHIP];
        U11 rtop5[NCHIP], dtop5[NCHIP];
        U11 rtop6[NCHIP], dtop6[NCHIP];
        U11 rtop7[NCHIP], dtop7[NCHIP];
        U11 rtop8[NCHIP], dtop8[NCHIP];
        U11 rtop9[NCHIP], dtop9[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
            rtop0[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*0+top)*WD; dtop0[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*0+top)*WD;
            rtop1[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*1+top)*WD; dtop1[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*1+top)*WD;
            rtop2[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*2+top)*WD; dtop2[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*2+top)*WD;
            rtop3[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*3+top)*WD; dtop3[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*3+top)*WD;
            rtop4[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*4+top)*WD; dtop4[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*4+top)*WD;
            rtop5[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*5+top)*WD; dtop5[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*5+top)*WD;
            rtop6[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*6+top)*WD; dtop6[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*6+top)*WD;
            rtop7[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*7+top)*WD; dtop7[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*7+top)*WD;
            rtop8[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*8+top)*WD; dtop8[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*8+top)*WD;
            rtop9[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*9+top)*WD; dtop9[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*9+top)*WD;
        }
    }

    //EMAX5A begin tone_curve mapdist=0
    for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
        /*2*/for (INIT1=1,LOOP1=RMGRP,rofs=0-WD*4; LOOP1--; INIT1=0) { /* stage#0 /** mapped to FOR() on BR[63][1][0] */
        /*1*/for (INIT0=1,LOOP0=WD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 /** mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, 0, EXP_H3210, OP_AND, 0xffffffffLL, OP_NOP, 0); /* s#0*/
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?WD*4:0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); /* s#0*/
            exe(OP_ADD, &pofs, rofs, EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0x000000ffffffffLL, OP_NOP, 0); /* s#1*/
            /*map0*/
            mop(OP_LDWR, 1, &BR[2][1][1], rtop0[CHIP], pofs, MSK_D0, rtop0[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* s#2*/
            mop(OP_LDBR, 1, &BR[3][1][1], t1, BR[2][1][1], MSK_B3, t1, 256/4, 0, 0, NULL, 256/4); /* s#3*/
            mop(OP_LDBR, 1, &BR[3][2][1], t2, BR[2][1][1], MSK_B2, t2, 256/4, 0, 0, NULL, 256/4); /* s#3*/
            mop(OP_LDBR, 1, &BR[3][3][1], t3, BR[2][1][1], MSK_B1, t3, 256/4, 0, 0, NULL, 256/4); /* s#3*/
            exe(OP_MMRC, &r1, BR[3][1][1], EXP_H3210, BR[3][2][1], EXP_H3210, BR[3][3][1], EXP_H3210, OP_NOP, 0, OP_NOP, 0); /* s#3*/
            mop(OP_STWR, 3, &r1, dtop0[CHIP], pofs, MSK_D0, dtop0[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* s#3*/
            :
            /*map9*/
            mop(OP_LDWR, 1, &BR[20][1][1], rtop9[CHIP], pofs, MSK_D0, rtop9[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* s#20*/
            mop(OP_LDBR, 1, &BR[21][1][1], t1, BR[20][1][1], MSK_B3, t1, 256/4, 0, 0, NULL, 256/4); /* s#21*/
            mop(OP_LDBR, 1, &BR[21][2][1], t2, BR[20][1][1], MSK_B2, t2, 256/4, 0, 0, NULL, 256/4); /* s#21*/
            mop(OP_LDBR, 1, &BR[21][3][1], t3, BR[20][1][1], MSK_B1, t3, 256/4, 0, 0, NULL, 256/4); /* s#21*/
            exe(OP_MMRC, &r1, BR[21][1][1], EXP_H3210, BR[21][2][1], EXP_H3210, BR[21][3][1], EXP_H3210, OP_NOP, 0, OP_NOP, 0); /* s#21*/
            mop(OP_STWR, 3, &r1, dtop9[CHIP], pofs, MSK_D0, dtop9[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* s#21*/
        }
    }
}

//EMAX5A end
}
//EMAX5A drain_dirty_lmm
}

```

3.2 Basics

3.2.1 FFT

```
cent% make -f Makefile-csim.emax6+dma all clean
cent% ../../src/csim/csim -x fft-csim.emax6+dma 4 4096
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean
zynq% ./fft-zynq.emax6+dma 4 4096
```

Simple implementation

最大 8192 (LMM=8K*4byte*2=64KB) 要素の FFT. なお, IMAX の空き場所に展開すれば, 大容量・高速化可能.

```
printf("<<<ORIG>>>\n");
reset_nanosec();
BlockEnd = 1;
for (BlockSize=2; BlockSize<=NumSamples; BlockSize<=1) {
    for (i=0; i<NumSamples; i+=BlockSize) {
        for (j=i,n=0; n<BlockEnd; j++,n++) {
            k = j + BlockEnd;
            idx = n + BlockEnd;
            tr = art[idx]*RealOut[k] - ait[idx]*ImagOut[k];
            ti = art[idx]*ImagOut[k] + ait[idx]*RealOut[k];
            RealOut[k] = RealOut[j] - tr;
            ImagOut[k] = ImagOut[j] - ti;
            RealOut[j] += tr;
            ImagOut[j] += ti;
        }
    }
    BlockEnd = BlockSize;
}
```

```
U11 CHIP;
U11 LOOP1, LOOPO;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
U11 ar, ai, rok, iok, roj, ioj;
U11 tr0, tio, tr1, tii, roj1, ioj1;

printf("<<<IMAX>>> NumSamples=%d (LMM should be >= %dB)\n", NumSamples, NumSamples*4*2);
reset_nanosec();

#define fft_core0(r) \
    exe(OP_ADD,     &j,           i,      EXP_H3210, OLL,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#1 */\
    exe(OP_ADDD3,   &k,           i,      EXP_H3210, n,       EXP_H3210, BlockEnd, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#1 */\
    exe(OP_ADDD3,   &idx,         OLL,    EXP_H3210, n,       EXP_H3210, BlockEnd, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#1 */\
    exe(OP_NOP,     &AR[r][0],   OLL,    EXP_H3210, OLL,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 [2][0] */\
    mop(OP_LDWR,   3, &r0k,       RealOut, k,      MSK_DO, RealOut, NumSamples, 0, 1, NULL, NumSamples); /* stage#2 RealOut[k] */\
    mop(OP_LDWR,   3, &r0j,       RealOut, j,      MSK_DO, RealOut, NumSamples, 0, 1, NULL, NumSamples); /* stage#2 RealOut[j] */\
    exe(OP_NOP,     &AR[r][2],   OLL,    EXP_H3210, OLL,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 [2][2] */\
    mop(OP_LDWR,   3, &iok,       ImagOut, k,      MSK_DO, ImagOut, NumSamples, 0, 1, NULL, NumSamples); /* stage#2 ImagOut[k] */\
    mop(OP_LDWR,   3, &ioj,       ImagOut, j,      MSK_DO, ImagOut, NumSamples, 0, 1, NULL, NumSamples); /* stage#2 ImagOut[j] */\
#define fft_core1(r) \
    exe(OP_NOP,     &AR[r][0],   OLL,    EXP_H3210, OLL,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 [3][0] */\
    mop(OP_LDWR,   3, &r1k,       art,    idx,    MSK_DO, art,    NumSamples, 0, 0, NULL, NumSamples); /* stage#3 art[idx] */\
    exe(OP_NOP,     &AR[r][2],   OLL,    EXP_H3210, OLL,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 [3][2] */\
    mop(OP_LDWR,   3, &r1i,       ait,    idx,    MSK_DO, ait,    NumSamples, 0, 0, NULL, NumSamples); /* stage#3 ait[idx] */\
    exe(OP_FML,    &tr0,        ar,    EXP_H3210, rok,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */\
    exe(OP_FML,    &tio,        ar,    EXP_H3210, iok,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */\
    exe(OP_FMS,    &tr1,        ar,    EXP_H3210, tii,    EXP_H3210, ai,      EXP_H3210, iok,    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */\
    exe(OP_FMA,    &tr2,        ai,    EXP_H3210, tio,    EXP_H3210, rok,    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */\
#define fft_final(r) \
    exe(OP_FMS,    &AR[r][0],   roj,   EXP_H3210, tr1,   EXP_H3210, 0x3f800000LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */\
    mop(OP_STWR,   3, &AR[r][0], RealOut, k,      MSK_DO, RealOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#6 */\
    exe(OP_FAD,    &AR[r][1],   roj,   EXP_H3210, tr1,   EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */\
    mop(OP_STWR,   3, &AR[r][1], RealOut, j,      MSK_DO, RealOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#6 */\
    exe(OP_FMS,    &AR[r][2],   roj,   EXP_H3210, tii,   EXP_H3210, 0x3f800000LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */\
    mop(OP_STWR,   3, &AR[r][2], ImagOut, k,      MSK_DO, ImagOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#6 */\
    exe(OP_FAD,    &AR[r][3],   roj,   EXP_H3210, tii,   EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */\
    mop(OP_STWR,   3, &AR[r][3], ImagOut, j,      MSK_DO, ImagOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#6 */\
BlockEnd = 1;
for (BlockSize=2; BlockSize<=NumSamples; BlockSize<=1) {
//with-prefetch/post-drain
//EMAXSA begin imax.mapdist=0
/*3*/for (CHIP=0; CHIP<CHIP; CHIP++) {
/*2*/for (INIT1=1,LOOP1=NumSamples/BlockSize,i=0LL<<32|(0-BlockSize)&0xffffffff; LOOP1--; INIT1=0) {
/*1*/for (INIT0=1,LOOP0=BlockEnd,n=0LL<<32|(0-1LL)&0xffffffff; LOOP0--; INIT0=0) {
    exe(OP_ADD,     &i,           i,      EXP_H3210, INIT0?BlockSize:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */\
    exe(OP_ADD,     &n,           INIT0?n:n, EXP_H3210, OLL<<32|1LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */\
    fft_core0(2); /* stage#2 */\
    fft_core1(3); /* stage#3-5 */\
    fft_final(6); /* stage#6 */\
}
}
//EMAXSA end
//EMAXSA drain_dirty_lmm
BlockEnd = BlockSize;
}
```

fourierf-imax-emax6.obj

BR/row: max=9 min=4 ave=5 EA/row: max=4 min=0 ave=1 ARpass/row: max=0

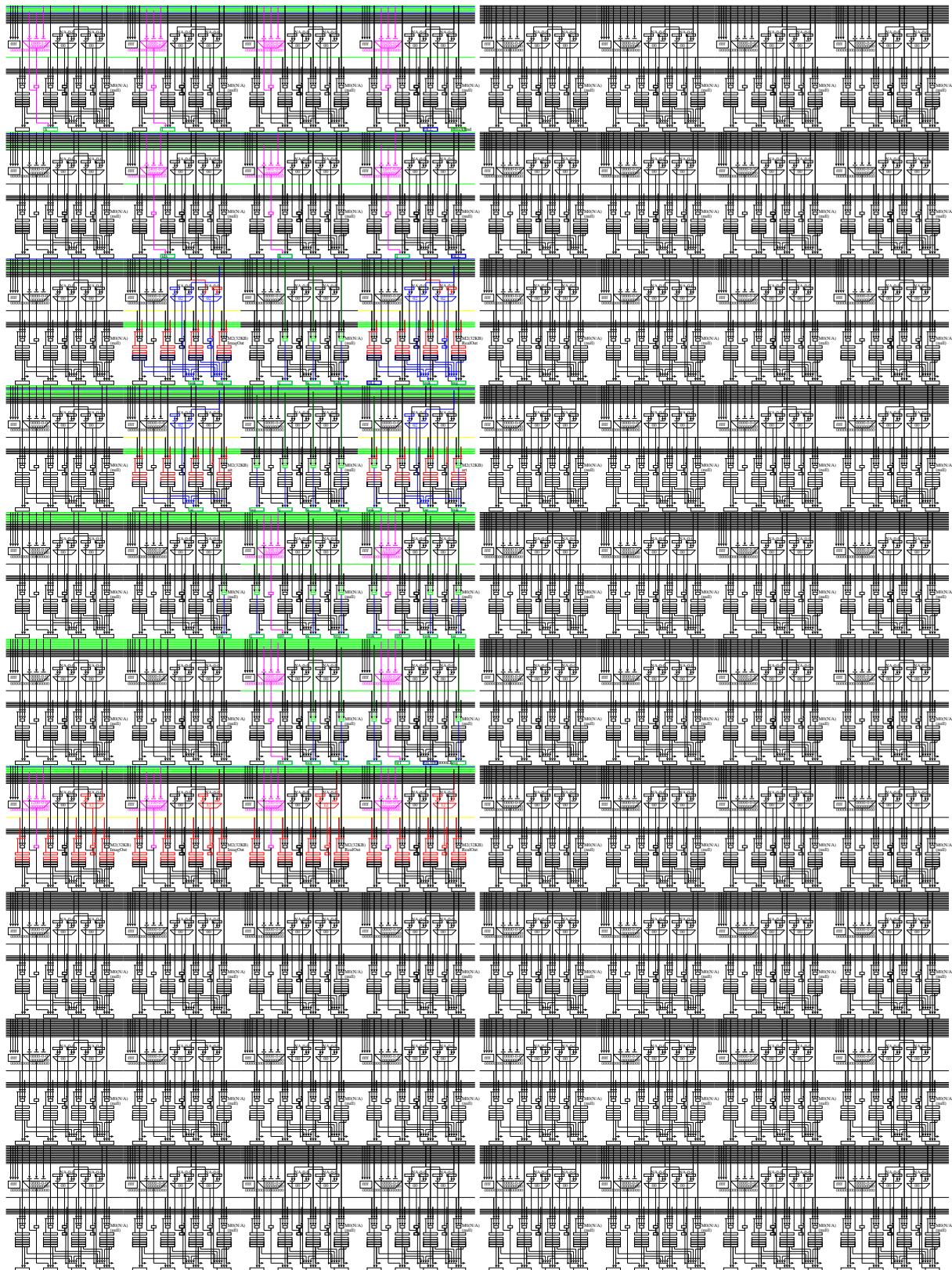


Figure.3.5: FFT

Pipelined implementation

FFT のパイプライン版。最大 4096 (LMM=4K*4byte*2*2=64KB) 要素の FFT。中間の LMM をダブルバッファとして使用している。

```

U11 CHIP;
U11 LOOP1, LOOPO, L;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc1, cc2, cc3, ex0, ex1;
U11 J[16], K[16], IDX[16]; /* log2(NumSamples=65536)=16 まで対応可 */
U11 BufReal[16], BufImag[16];
U11 ar, ai, rok, iok, roj, ioj, tr0, tio, tr1, ti1;
U11 Pipeline, Lmmrotate; /* log2(NumSamples=65536)=16 回繰り返すと、最終段の LMM に、最初の RealOut/ImagOut が格納される */
printf("<<<IMAX>> NumSamples=%d (LMM should be > %dB)\n", NumSamples, NumSamples*4*2);
reset_nanosec();
#define fft_core0(r, x, MASK_M, MASK_N, BS) \
    exe(OP_ADD, &i, L, EXP_H3210, L, EXP_H3210, OLL, EXP_H3210, OP_AND, MASK_M, OP_NOP, OLL); /* stage#1 i = (L*2)&M */ \
    exe(OP_ADD, &n, L, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, MASK_N, OP_NOP, OLL); /* stage#1 n = L &N */ \
    exe(OP_ADD, &J[x], i, EXP_H3210, n, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#2 j = i+n */ \
    exe(OP_ADD3, &K[x], i, EXP_H3210, n, EXP_H3210, BS, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#2 k = i+n+BS(2) */ \
    exe(OP_ADD3, &IDX[x], OLL, EXP_H3210, n, EXP_H3210, BS, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL) /* stage#2 idx = n+BS(2) */ \
#define fft_core1(r, x) \
    exe(OP_NOP, &AR[r][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#3 [3][0] */ \
    mop(OP_LDWR, 3, &rok, RealIn, K[x], MSK_DO, RealIn, NumSamples, 0, 1, NULL, NumSamples); /* stage#3 RealIn[k] */ \
    mop(OP_LDWR, 3, &rok, J[x], MSK_DO, RealIn, NumSamples, 0, 1, NULL, NumSamples); /* stage#3 RealIn[j] */ \
    exe(OP_NOP, &AR[r][2], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#3 [3][2] */ \
    mop(OP_LDWR, 3, &iok, ImagIn, K[x], MSK_DO, ImagIn, NumSamples, 0, 1, NULL, NumSamples); /* stage#3 ImagIn[k] */ \
    mop(OP_LDWR, 3, &iok, J[x], ImagIn, MSK_DO, ImagIn, NumSamples, 0, 1, NULL, NumSamples); /* stage#3 ImagIn[j] */ \
#define fft_core2(r, x, y, MASK_M, MASK_N, BS) \
    exe(OP_NOP, &AR[r][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#4 [4][0] */ \
    exe(OP_ADD, &i, L, EXP_H3210, L, EXP_H3210, OLL, EXP_H3210, OP_AND, MASK_M, OP_NOP, OLL); /* stage#4 i = (L*2)&M */ \
    mop(OP_LDWR, 3, &kar, art, IDX[x], MSK_DO, art, NumSamples, 0, 0, NULL, NumSamples); /* stage#4 art[Idx] */ \
    exe(OP_NOP, &AR[r][2], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#4 [4][2] */ \
    exe(OP_ADD, &n, L, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, MASK_N, OP_NOP, OLL); /* stage#4 n = L &N */ \
    mop(OP_LDWR, 3, &ait, ait, IDX[x], MSK_DO, ait, NumSamples, 0, 0, NULL, NumSamples); /* stage#4 ait[Idx] */ \
    \
    exe(OP_ADD, &J[y], i, EXP_H3210, n, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#5 j = i+n */ \
    exe(OP_ADD3, &K[y], i, EXP_H3210, n, EXP_H3210, BS, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#5 k = i+n+BS(2) */ \
    exe(OP_FML, &tr0, ar, EXP_H3210, rok, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */ \
    exe(OP_FML, &tio, ar, EXP_H3210, iok, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */ \
    \
    exe(OP_ADD3, &IDX[y], OLL, EXP_H3210, n, EXP_H3210, BS, EXP_H3210, OP_NOP, OLL, OP_SLL, 2LL); /* stage#6 idx = n+BS(2) */ \
    exe(OP_FMS, &t1r, tr0, EXP_H3210, ait, EXP_H3210, iok, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */ \
    exe(OP_FMS, &t1i, tio, EXP_H3210, ait, EXP_H3210, rok, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */ \
#define fft_core3(r, x, y) \
    exe(OP_FMS, &AR[r][0], roj, EXP_H3210, tr1, EXP_H3210, 0x3f800000LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][0], BufReal[x], K[x], MSK_DO, BufReal[x], OLL, 0, 0, NULL, OLL); /* stage#7 */ \
    mop(OP_LDWR, 3, &rok, K[y], BufReal[y], MSK_DO, BufReal[y], OLL, 0, 0, NULL, OLL); /* stage#7 BufReal[k] */ \
    exe(OP_FAD, &AR[r][1], roj, EXP_H3210, tr1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][1], BufReal[x], J[x], MSK_DO, BufReal[x], OLL, 0, 0, NULL, OLL); /* stage#7 */ \
    mop(OP_LDWR, 3, &rok, J[y], BufReal[y], MSK_DO, BufReal[y], OLL, 0, 0, NULL, OLL); /* stage#7 BufReal[j] */ \
    exe(OP_FMS, &AR[r][2], roj, EXP_H3210, t1i, EXP_H3210, 0x3f800000LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][2], BufImag[x], K[x], MSK_DO, BufImag[x], OLL, 0, 0, NULL, OLL); /* stage#7 */ \
    mop(OP_LDWR, 3, &iok, K[y], BufImag[y], MSK_DO, BufImag[y], OLL, 0, 0, NULL, OLL); /* stage#7 BufImag[k] */ \
    exe(OP_FAD, &AR[r][3], roj, EXP_H3210, t1i, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][3], BufImag[x], J[x], MSK_DO, BufImag[x], OLL, 0, 0, NULL, OLL); /* stage#7 */ \
    mop(OP_LDWR, 3, &iok, J[y], BufImag[y], MSK_DO, BufImag[y], OLL, 0, 0, NULL, OLL); /* stage#7 BufImag[j] */ \
#define fft_final(r, x) \
    exe(OP_FMS, &AR[r][0], roj, EXP_H3210, tr1, EXP_H3210, 0x3f800000LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][0], RealOut, K[x], MSK_DO, RealOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#7 */ \
    exe(OP_FAD, &AR[r][1], roj, EXP_H3210, tr1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][1], RealOut, J[x], MSK_DO, RealOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#7 */ \
    exe(OP_FMS, &AR[r][2], roj, EXP_H3210, t1i, EXP_H3210, 0x3f800000LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][2], ImagOut, K[x], MSK_DO, ImagOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#7 */ \
    exe(OP_FAD, &AR[r][3], roj, EXP_H3210, t1i, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
    mop(OP_STWR, 3, &AR[r][3], ImagOut, J[x], MSK_DO, ImagOut, NumSamples, 0, 0, NULL, NumSamples); /* stage#7 */ \
for (Pipeline=0; Pipeline<NumPipes; Pipeline++) {
/* 0: buf[0]=(1<<4)-0%4 : buf[1]=[(1<<4)-0%4]: buf[2]=[(2<<4)-0%4]: buf[3]=[(3<<4)-0%4]:3 */
/* 1: buf[0]=(1<<4)-1%4 : buf[1]=[(1<<4)-1%4]: buf[2]=[(2<<4)-1%4]: buf[3]=[(3<<4)-1%4]:2 */
/* 2: buf[0]=(1<<4)-2%4 : buf[1]=[(1<<4)-2%4]:3 buf[2]=[(2<<4)-2%4]: buf[3]=[(3<<4)-2%4]:1 */
for (Lmmrotate=0; Lmmrotate<NumBits; Lmmrotate+=4) {
    BufReal[Lmmrotate] = &pseudoLMM[NumSamples*((Lmmrotate+NumBits+1)/Pipeline)%((NumBits+1)*2)];
    BufImag[Lmmrotate] = &pseudoLMM[NumSamples*((Lmmrotate+NumBits+1)/Pipeline)%((NumBits+1)*2+1)];
}
}

//EMAXSA begin pipeline mapdist=0
/*3*/for (CHIP=0; CHIP<CHIP; CHIP++) {
/*1*/for ((INIT0=1,LOOP1=NumSamples/2,L=0LL<<32|0-1LL)&0xffffffff; LOOP0--; INIT0=0) { /* NumSamples<=4096 */
    exe(OP_ADD, &L, EXP_H3210, 0LL<<32|1LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    #if (H==2)
        fft_core0( 1, 0, 0xffffeLL, 0x0000LL, 1LL); /* stage#1-2 */
        fft_core1( 3, 0); /* stage#3 */
        fft_core2( 4, 0, 1, 0xffffcLL, 0x0001LL, 2LL); /* stage#4-6 */
        fft_core3( 7, 0, 1); /* stage#7 */
        fft_core2( 8, 1, 2, 0xffff8LL, 0x0003LL, 4LL); /* stage#8-10 */
        fft_core3(11, 1, 2); /* stage#11 */
        fft_core2(12, 2, 3, 0xffff0LL, 0x0007LL, 8LL); /* stage#12-14 */
        fft_core3(15, 2, 3); /* stage#15 */
        fft_core2(16, 3, 4, 0xffeOLL, 0x000fLL, 16LL); /* stage#16-18 */
        fft_core3(19, 3, 4); /* stage#19 */
        fft_core2(20, 4, 5, 0xffffcOLL, 0x001fLL, 32LL); /* stage#20-22 */
        fft_core3(23, 4, 5); /* stage#23 */
        fft_core2(24, 5, 6, 0xffff80LL, 0x003fLL, 64LL); /* stage#24-26 */
        fft_core3(27, 5, 6); /* stage#27 */
    :
        fft_core2(44, 10, 11, 0xffff00LL, 0x07ffffL, 2048LL); /* stage#44-46 */
        fft_core3(47, 10, 11); /* stage#47 */
        fft_core2(48, 11, 12, 0xe000LL, 0x0ffffL, 4096LL); /* stage#48-50 */
        fft_final(51, 11); /* stage#51 */
    #endif
}
}
//EMAXSA end
//EMAXSA drain_dirty_lmm
}

```

fourierf-pipeline-emax6.obj

BR/row: max=15 min=2 ave=10 EA/row: max=8 min=0 ave=2 ARpass/row: max=0

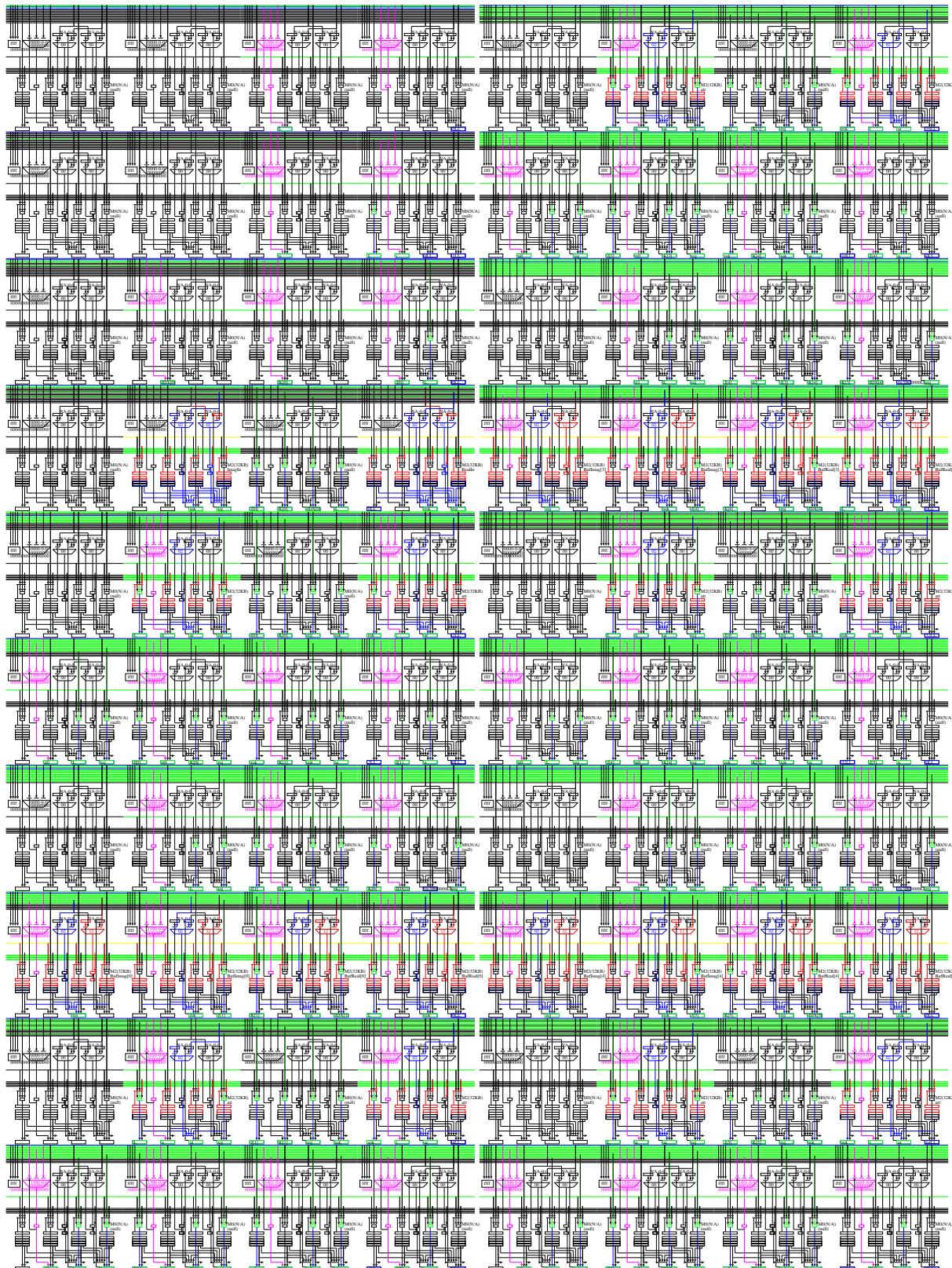


Figure.3.6: FFT

3.2.2 Merge sort

```
cent% make -f Makefile-csim.emax6+dma all clean
cent% ../../src/csim/csim -x sort-merge-csim.emax6+dma 4096
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean
zynq% ./sort-merge-zynq.emax6+dma 4096
```

最大 4096 (LMM=4K*8byte*2=64KB) 要素のパイプライン版 merge sort. 中間の LMM をダブルバッファとして使用している. なお, 入力データは各要素 8 バイトである. 上位 4 バイトが大小比較対象, 下位 4 バイトはポインタや属性など任意の値に使用できる.

```
printf("<<<ORIG>>>\n");
BlockEnd = 1;
for (BlockSize<2; BlockSize<=NumSamples; BlockSize<=1) {
    for (i=0; i<NumSamples; i+=BlockSize) {
        for (j=i, k=i+BlockEnd, t=i; t<i+BlockSize; t++) {
            int cc0 = j+i+BlockEnd;
            int cc1 = k+i+BlockSize;
            int cc2 = In[j].val < In[k].val;
            if (((cc2 && cc1) || !cc1) && cc0) /* 7,5,1 0x00a2 */ Out[t] = In[j++];
            if (((!cc2 && cc0) || !cc0) && cc1) /* 6,3,2 0x004c */ Out[t] = In[k++];
        }
    }
    BlockEnd = BlockSize;
    for (i=0; i<NumSamples; i++)
        In[i] = Out[i];
}

#define NCHIP 1
#define H 4096
U11 CHIP;
U11 LOOP1, LOOP0, L8;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, cc1, cc2, cc3, ex0, ex1, ex2, ex3;
U11 Buf[32];
U11 base, J[16], K[16]; /* log2(NumSamples=65536)=16 まで対応可 */
U11 Pipeline, Lmmrotate; /* log2(NumSamples=65536)=16 回繰り返すと, 最終段の LMM に, 最初の Out が格納される */
printf("<<<IMAX>>> NumSamples=%d (LMM should be %dB)\n", NumSamples, NumSamples*2*4);

for (i=0; i<NumBits; i++) { J[i]=0; K[i]=0; }

#define sort_core0(r, rpl, x, MASK_M, In, BE8) \
exe(OP_NOP, &AR[r][1], In, EXP_H3210, OLL, EXP_H3210, OLL, OP_NOP, OLL); /*stage#1 dmy */ \
exe(OP_ADD, &i, L8, EXP_H3210, OLL, EXP_H3210, OLL, OP_AND, MASK_M, OP_NOP, OLL); /*stage#1 i = L8&M */ \
exe(OP_NOP, &AR[rpl][3], OLL, EXP_H3210, OLL, EXP_H3210, OLL, OP_NOP, OLL, OP_NOP, OLL); /*stage#2 dmy */ \
exe(OP_ADD, &base, In, EXP_H3210, i, EXP_H3210, OLL, EXP_H3210, OLL, OP_NOP, OLL); /*stage#2 baseJ=&In[i],baseK=&In[i+BE]*

#define sort_core1(r, rpl, x, MASK_M, In, BE8, Buf, Out, Olen) \
mex(OP_CMPA_LE, &J[x], INIT0?OLL:J[x], INIT0?OLL:8LL, OP_CMPA_GE, &K[x], INIT0?BE8:X[x], INIT0?OLL:8LL, BE8, BR[r][3][1], BR[r][3][0]); /* stage#3 */ \
mop(OP_LDR, 3, &BR[r][3][1], J[x], base, MSK_D0, In, Ilen, 0, 0, NULL, Ilen); /*LMM[2] 確定 LD 実行は col2/* stage#3 */ \
mop(OP_LDR, 3, &BR[r][3][0], J[x], base, MSK_D0, In, Ilen, 0, 0, NULL, Ilen); /*LMM[1] 空借り LD 実行は col2/* stage#3 */ \
exe(OP_CMP_LT, &cc0, J[x], EXP_H1010, BE8, EXP_H1010, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* J[x]<BE8 */ /* stage#4 */ \
exe(OP_CMP_LT, &cc1, J[x], EXP_H1010, BE8+2, EXP_H1010, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* K[x]<BE8 */ /* stage#4 */ \
exe(OP_CMP_LT, &cc2, BR[r][3][1], EXP_H3232, BR[r][3][0], EXP_H3232, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* J<K */ /* stage#4 */ \
cex(OP_CEXE, &ex0, 0, cc2, cc1, cc0, 0x00a2); /* if ((cc2 && cc1 && cc0) || (!cc1 && cc0)) 7,5,1 0x00a2 */ /* stage#5 */ \
mop(OP_STR, ex0, &BR[r][3][1], Buf, L8, MSK_W0, Out, Olen, 0, 0, NULL, Olen); /*LMM[1] 空借り LD 実行は col2/* stage#5 */ \
cex(OP_CEXE, &ex1, 0, cc2, cc1, cc0, 0x004c); /* if ((!cc2 && cc1 && cc0) || (!cc1 && cc0)) 6,3,2 0x004c */ /* stage#5 */ \
mop(OP_STR, ex1, &BR[r][3][0], Buf, L8, MSK_W0, Out, Olen, 0, 0, NULL, Olen); /*LMM[1] 空借り LD 実行は col2/* stage#5 */

for (Pipeline=0; Pipeline<NumBins; Pipeline++) {
    for (Lmmrotate=0; Lmmrotate<NumBins*2; Lmmrotate++) {
        Buf[Lmmrotate] = &pseudolLMMNumSamples*((Lmmrotate+NumBits*2-Pipeline)% (NumBits*2)));
    //EMAXSA begin pipeline mapdst=0
    /*$*/ for (CHIP=0; CHIP<NCHIP; CHIP++) {
    /*$*/ for (INIT0=1,LOOP0=NumSamples,LB=OLL<<32|0-8LL)&0xffffffff; LOOP0--; INIT0=0) { /* NumSamples<=4096 */
        exe(OP_ADD, &L8, L8, EXP_H3210, OLL<<32|8LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#0 */
        #if (H==2)
            sort_core0(1, 2, 0, 0xfffffffffffffff0LL, In, 8LL); /* stage#1-2 */
            sort_core1(3, 4, 0, In, NumSamples2, 8LL, Out, Out, NumSamples2); /* stage#3-5 */
        #endif
        #if (H==4096)
            sort_core0(1, 2, 0, 0xfffffffffffffff0LL, In, 8LL);
            sort_core1(3, 4, 0, In, NumSamples2, 8LL, Buf[0], Buf[1], OLL); /* stage#3-5 */
            sort_core0(3, 4, 1, 0xfffffffffffffff0LL, Buf[1], 16LL);
            sort_core1(5, 6, 1, Buf[1], OLL, 16LL, Buf[2], Buf[3], OLL); /* stage#5-7 */
            sort_core0(5, 6, 2, 0xfffffffffffffff0LL, Buf[3], 32LL);
            sort_core1(7, 8, 2, Buf[3], OLL, 32LL, Buf[4], Buf[5], OLL); /* stage#7-9 */
            sort_core0(7, 8, 3, 0xfffffffffffffff80LL, Buf[5], 64LL);
            sort_core1(9, 10, 3, Buf[5], OLL, 64LL, Buf[6], Buf[7], OLL); /* stage#9-11 */
            sort_core0(9, 10, 4, 0xfffffffffffffff0LL, Buf[7], 128LL);
            sort_core1(11, 12, 4, Buf[7], OLL, 128LL, Buf[8], Buf[9], OLL); /* stage#11-13 */
            sort_core0(11, 12, 5, 0xfffffffffffffff0LL, Buf[9], 256LL);
            sort_core1(13, 14, 5, Buf[9], OLL, 256LL, Buf[10], Buf[11], OLL); /* stage#13-15 */
            sort_core0(13, 14, 6, 0xfffffffffffffff0LL, Buf[11], 512LL);
        :
            sort_core1(21, 22, 9, Buf[17], OLL, 4096LL, Buf[18], Buf[19], OLL); /* stage#21-23 */
            sort_core0(21, 22, 10, 0xfffffffffffffff000LL, Buf[19], 8192LL);
            sort_core1(23, 24, 10, Buf[19], OLL, 8192LL, Buf[20], Buf[21], OLL); /* stage#23-25 */
            sort_core0(23, 24, 11, 0xfffffffffffffff8000LL, Buf[21], 16384LL);
            sort_core1(25, 26, 11, Buf[21], OLL, 16384LL, Out, Out, NumSamples2); /* stage#25-27 */
        #endif
    }
    //EMAXSA end
    //EMAXSA drain_dirty_lmm
}
```

sort-merge-pipeline-emax6.obj

BR/row: max=10 min=2 ave=8 EA/row: max=4 min=0 ave=1 ARpass/row: max=0

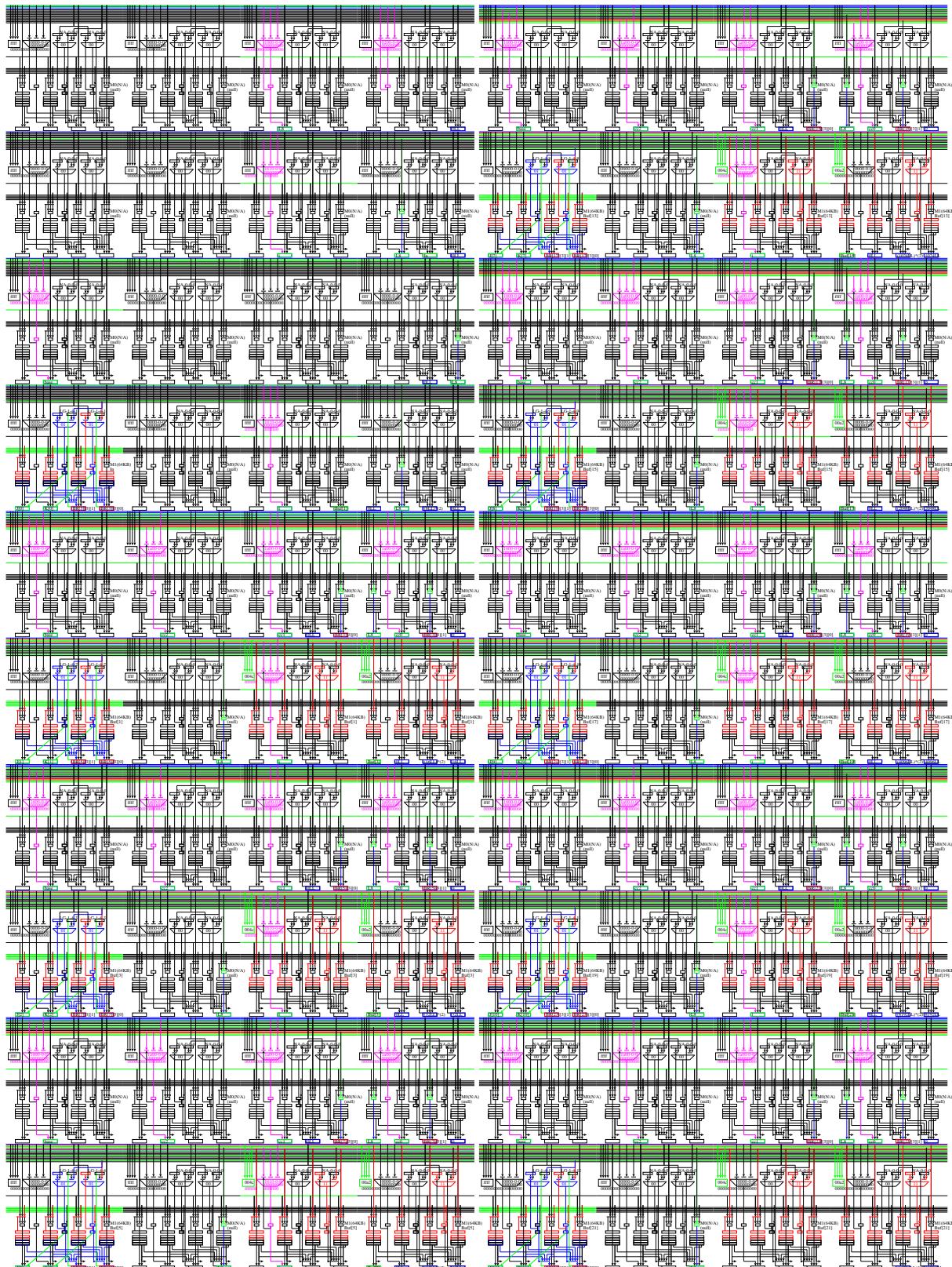


Figure.3.7: FFT

3.2.3 文字列検索

```
cent% make -f Makefile-csim.emax6+dma all clean
cent% ../../src/csim/csim -x search-csim.emax6+dma search.txt target.txt
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean
zynq% ./search-zynq.emax6+dma search.txt target.txt
```

最大 8 文字までの文字列検索である。一度のバースト演算により 9 個の文字列を検索する。

```

even if you have nowhere to do it but your living room Read the directions
even if you dont follow them Do not read beauty magazines They will only
make you feel ugly Get to know your parents You never know when they'll be
gone for good Be nice to your siblings They're your best link to your past
and the people most likely to stick with you in the future Understand that
friends come and go but with a precious few you should hold on Work hard to
bridge the gaps in geography and lifestyle because the older you get the
more you need the people who knew you when you were young Live in New York
City once but leave before it makes you hard Live in Northern California
once but leave before it makes you soft Travel Accept certain inalienable
truths Prices will rise Politicians will philander You too will get old And
when you do you'll fantasize that when you were young prices were reasonable
politicians were noble and children respected their elders Respect your
elders Dont expect anyone else to support you Maybe you have a trust fund
Maybe you'll have a wealthy spouse But you never know when either one might
run out Dont mess too much with your hair or by the time you're 40 it will
look 85 Be careful whose advice you buy but be patient with those who supply
it Advice is a form of nostalgia Dispensing it is a way of fishing the past
from the disposal wiping it off painting over the ugly parts and recycling
it for more than its worth But trust me on the sunscreen book end

Results are equal

```

Figure.3.8: String search

```

orig()
{
    int i;
    printf("<<<ORIG>>>\n");
    for (i=0; i<sum; i++) {
        init_search(i);
        strsearch(i);
    }
    return 0;
}
init_search(int i)/* for ARM */
{
    char *str = sstr[i];
    int len = slen[i];
    int j;
    for (j = 0; j <= UCHAR_MAX; j++)
        table[j] = len;
    for (j = 0; j < len; j++)
        table[(Uchar)str[j]] = len - j - 1;
    for (j = 0; j < clen; j++)
        *(out0+clen*i+j) = 0;
}
strsearch(int i)
{
    char *str = sstr[i];
    int len = slen[i];
    register size_t shift;
    register size_t pos = len - 1;
    char *found;
    while (pos < clen) {
        while (pos < clen && (shift = table[(unsigned char)target[pos]]) > 0)
            pos += shift;
        if (!shift) {
            if (!strcmp(str, &target[pos-len+1], len))
                out0[i*clen+(pos-len+1)] = 0xff;
            pos++;
        }
    }
}

```

```

imax()
{
    int i, j;
    printf("<<<IMAX>>>\n");
    for (i=0; i<sum; i++) {
        for (j=0; j<clen; j++) {
            if (!strcmp(sstr[i], &target[j], slen[i]))
                out1[i*clen+j] = 0xff;
            else
                out1[i*clen+j] = 0;
        }
    }
}

```

```

imax()
{
    U11 CHIP;
    U11 LOOP1, LOOP0;
    U11 INTI1, INIT0;
    U11 AR[64] [4]; /* output of EX in each unit */
    U11 BR[64] [4] [4]; /* output registers in each unit */
    U11 r00, r01, r02, r03, r04, r05, r06, r07, r08, r09, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 c0, c1, c2, c3, ex0, ex1;
    U11 t0[NCHIP], t1[NCHIP], t2[NCHIP], t3[NCHIP], t4[NCHIP], t5[NCHIP], t6[NCHIP], t7[NCHIP], t8[NCHIP];
    U11 tot[NCHIP], tit[NCHIP], t2t[NCHIP], t3t[NCHIP], t4t[NCHIP], t5t[NCHIP], t6t[NCHIP], t7t[NCHIP], t8t[NCHIP];
    U11 r0[NCHIP], r1[NCHIP], r2[NCHIP], r3[NCHIP], r4[NCHIP], r5[NCHIP], r6[NCHIP], r7[NCHIP], r8[NCHIP];
    U11 r0t[NCHIP], rit[NCHIP], r2t[NCHIP], r3t[NCHIP], r4t[NCHIP], r5t[NCHIP], r6t[NCHIP], r7t[NCHIP], r8t[NCHIP];
    U11 i, day, loop=clen/NCHIP;
    U11 dwi = clen/NCHIP/4+1; /* dwords */
    U11 dwo = clen/NCHIP/4 ; /* dwords */

    printf("<<<IMAX>>\n");
    for (CHIP=0; CHIP<NCHIP; CHIP++) {
        tot[NCHIP]=target+(clen/NCHIP*NCHIP);
        tit[NCHIP]=target+(clen/NCHIP*NCHIP);
        t2t[NCHIP]=target+(clen/NCHIP*NCHIP);
        t3t[NCHIP]=target+(clen/NCHIP*NCHIP);
        t4t[NCHIP]=target+(clen/NCHIP*NCHIP);
        t5t[NCHIP]=target+(clen/NCHIP*NCHIP);
        t6t[NCHIP]=target+(clen/NCHIP*NCHIP);
        t7t[NCHIP]=target+(clen/NCHIP*NCHIP);
        t8t[NCHIP]=target+(clen/NCHIP*NCHIP);
    }
    for (i=0; i<sum; i+=DMAP) {
        U11 c00=sstr[i+0][0], c01=sstr[i+0][1], c02=sstr[i+0][2], c03=sstr[i+0][3], c04=sstr[i+0][4], c05=sstr[i+0][5], c06=sstr[i+0][6], c07=sstr[i+0][7];
        U11 c10=sstr[i+1][0], c11=sstr[i+1][1], c12=sstr[i+1][2], c13=sstr[i+1][3], c14=sstr[i+1][4], c15=sstr[i+1][5], c16=sstr[i+1][6], c17=sstr[i+1][7];
        :
        U11 c80=sstr[i+8][0], c81=sstr[i+8][1], c82=sstr[i+8][2], c83=sstr[i+8][3], c84=sstr[i+8][4], c85=sstr[i+8][5], c86=sstr[i+8][6], c87=sstr[i+8][7];
        U11 slen0=slen[i+0], selen1=slen[i+1], selen2=slen[i+2], selen3=slen[i+3], selen4=slen[i+4], selen5=slen[i+5], selen6=slen[i+6], selen7=slen[i+7], selen8=slen[i+8];
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            t0[CHIP] = tot[CHIP]-1;
            t1[CHIP] = tit[CHIP]-1;
            :
            t8[CHIP] = t8t[CHIP]-1;
            r0[CHIP] = r0t[CHIP] = out1+(i+0)*clen+(clen/NCHIP*NCHIP);
            r1[CHIP] = rit[CHIP] = out1+(i+1)*clen+(clen/NCHIP*NCHIP);
            :
            r8[CHIP] = r8t[CHIP] = out1+(i+8)*clen+(clen/NCHIP*NCHIP);
        }
        //EMAX5A begin search mapdist=0
        for (CHIP=0; CHIP<NCHIP; CHIP++) { //チップ方向は検索対象文書の大きさ方向
            for (INIT0=1,LOOP0=loop,dmy0=0; LOOP0--; INIT0=0) { //長さは 64KB 文字まで
                /* map#0 */
                /*@01,*/ exe(OP_ADD, &t0[CHIP], t0[CHIP], EXP_H3210, 1LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL);
                /*@01,*/ exe(OP_MCAS, &r00, slen0, EXP_H3210, 1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@01,*/ exe(OP_MCAS, &r01, slen0, EXP_H3210, 2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@01,*/ exe(OP_MCAS, &r02, slen0, EXP_H3210, 3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@01,*/ exe(OP_MCAS, &r03, slen0, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@01,*/ mop(OP_LDBR, 1, &r1[1][0][1], t0[CHIP], 0, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ mop(OP_LDBR, 1, &r1[1][0][0], t0[CHIP], 1, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ exe(OP_LDBR, 1, &r1[1][1][1], t0[CHIP], 2, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ exe(OP_LDBR, 1, &r1[1][1][0], t0[CHIP], 3, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ mop(OP_LDBR, 1, &r1[1][2][1], t0[CHIP], 4, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ exe(OP_LDBR, 1, &r1[1][2][0], t0[CHIP], 5, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ mop(OP_LDBR, 1, &r1[1][3][1], t0[CHIP], 6, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@01,*/ exe(OP_LDBR, 1, &r1[1][3][0], t0[CHIP], 7, MSK_DO, t0t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@02,*/ exe(OP_CMP_NE, &x16, c00, EXP_H3210, BR[1][0][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r00, OP_NOP, OLL); // 1 if unmatch
                /*@02,*/ exe(OP_CMP_NE, &x17, c01, EXP_H3210, BR[1][0][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r01, OP_NOP, OLL); // 1 if unmatch
                /*@02,*/ exe(OP_CMP_NE, &x18, c02, EXP_H3210, BR[1][1][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r02, OP_NOP, OLL); // 1 if unmatch
                /*@02,*/ exe(OP_CMP_NE, &x19, c03, EXP_H3210, BR[1][1][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r03, OP_NOP, OLL); // 1 if unmatch
                /*@03,*/ exe(OP_MCAS, &r04, slen0, EXP_H3210, 5, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@03,*/ exe(OP_MCAS, &r05, slen0, EXP_H3210, 6, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@03,*/ exe(OP_MCAS, &r06, slen0, EXP_H3210, 7, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@03,*/ exe(OP_MCAS, &r07, slen0, EXP_H3210, 8, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@04,*/ exe(OP_CMP_NE, &x20, c04, EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r04, OP_NOP, OLL); // 1 if unmatch
                /*@04,*/ exe(OP_CMP_NE, &x21, c05, EXP_H3210, BR[1][2][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r05, OP_NOP, OLL); // 1 if unmatch
                /*@04,*/ exe(OP_CMP_NE, &x22, c06, EXP_H3210, BR[1][3][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r06, OP_NOP, OLL); // 1 if unmatch
                /*@04,*/ exe(OP_CMP_NE, &x23, c07, EXP_H3210, BR[1][3][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r07, OP_NOP, OLL); // 1 if unmatch
                /*@05,*/ exe(OP_ADD3, &r10, r16, EXP_H3210, r17, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@05,*/ exe(OP_ADD3, &r11, r19, EXP_H3210, r20, EXP_H3210, r21, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@05,*/ exe(OP_ADD3, &r12, r22, EXP_H3210, r23, EXP_H3210, r24, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@06,*/ exe(OP_MCAS, &r00, EXP_H3210, r0, EXP_H3210, r00, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@07,*/ exe(OP_MCAS, &r31, OLL, EXP_H3210, r00, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); // FF if match
                /*@07,*/ mop(OP_STBR, 3, &r31, r0[CHIP]++, 0, MSK_DO, r0t[CHIP], dw0, 0, 0, (U11)NULL, dw0);

                /* map#8 */
                /*@056,*/ exe(OP_ADD, &t8[CHIP], t8[CHIP], EXP_H3210, 1LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL);
                /*@057,*/ exe(OP_MCAS, &r00, slen8, EXP_H3210, 1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@057,*/ exe(OP_MCAS, &r01, slen8, EXP_H3210, 2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@057,*/ exe(OP_MCAS, &r02, slen8, EXP_H3210, 3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@057,*/ exe(OP_MCAS, &r03, slen8, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@057,*/ mop(OP_LDBR, 1, &r57[5][0][1], t8[CHIP], 0, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ mop(OP_LDBR, 1, &r57[5][0][0], t8[CHIP], 1, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ exe(OP_LDBR, 1, &r57[5][1][1], t8[CHIP], 2, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ exe(OP_LDBR, 1, &r57[5][1][0], t8[CHIP], 3, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ exe(OP_LDBR, 1, &r57[5][2][1], t8[CHIP], 4, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ exe(OP_LDBR, 1, &r57[5][2][0], t8[CHIP], 5, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ exe(OP_LDBR, 1, &r57[5][3][1], t8[CHIP], 6, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@057,*/ exe(OP_LDBR, 1, &r57[5][3][0], t8[CHIP], 7, MSK_DO, t7t[CHIP], dwi, 0, 0, (U11)NULL, dwi);
                /*@058,*/ exe(OP_CMP_NE, &x16, c80, EXP_H3210, BR[57][0][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r00, OP_NOP, OLL); // 1 if unmatch
                /*@058,*/ exe(OP_CMP_NE, &x17, c81, EXP_H3210, BR[57][0][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r01, OP_NOP, OLL); // 1 if unmatch
                /*@058,*/ exe(OP_CMP_NE, &x18, c82, EXP_H3210, BR[57][1][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r02, OP_NOP, OLL); // 1 if unmatch
                /*@058,*/ exe(OP_CMP_NE, &x19, c83, EXP_H3210, BR[57][1][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r03, OP_NOP, OLL); // 1 if unmatch
                /*@059,*/ exe(OP_MCAS, &r04, slen8, EXP_H3210, 5, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@059,*/ exe(OP_MCAS, &r05, slen8, EXP_H3210, 6, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@059,*/ exe(OP_MCAS, &r06, slen8, EXP_H3210, 7, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@059,*/ exe(OP_MCAS, &r07, slen8, EXP_H3210, 8, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
                /*@060,*/ exe(OP_CMP_NE, &x20, c84, EXP_H3210, BR[57][2][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r04, OP_NOP, OLL); // 1 if unmatch
                /*@060,*/ exe(OP_CMP_NE, &x21, c85, EXP_H3210, BR[57][2][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r05, OP_NOP, OLL); // 1 if unmatch
                /*@060,*/ exe(OP_CMP_NE, &x22, c86, EXP_H3210, BR[57][3][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r06, OP_NOP, OLL); // 1 if unmatch
                /*@060,*/ exe(OP_CMP_NE, &x23, c87, EXP_H3210, BR[57][3][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r07, OP_NOP, OLL); // 1 if unmatch
                /*@061,*/ exe(OP_ADD3, &r10, r16, EXP_H3210, r17, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@061,*/ exe(OP_ADD3, &r11, r19, EXP_H3210, r20, EXP_H3210, r21, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@061,*/ exe(OP_ADD3, &r12, r22, EXP_H3210, r23, EXP_H3210, r24, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@062,*/ exe(OP_MCAS, &r00, EXP_H3210, r0, EXP_H3210, r00, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
                /*@063,*/ exe(OP_MCAS, &r31, OLL, EXP_H3210, r00, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); // FF if match
                /*@063,*/ mop(OP_STBR, 3, &r31, r8[CHIP]++, 0, MSK_DO, r8t[CHIP], dw0, 0, 0, (U11)NULL, dw0);
            }
        }
        //EMAX5A end
        //EMAX5A drain_dirty_lmm
    }
}

```

pbmsrch+rmm-search-emax6.obj

BR/row: max=16 min=2 ave=9 EA/row: max=8 min=0 ave=1 ARpass/row: max=0

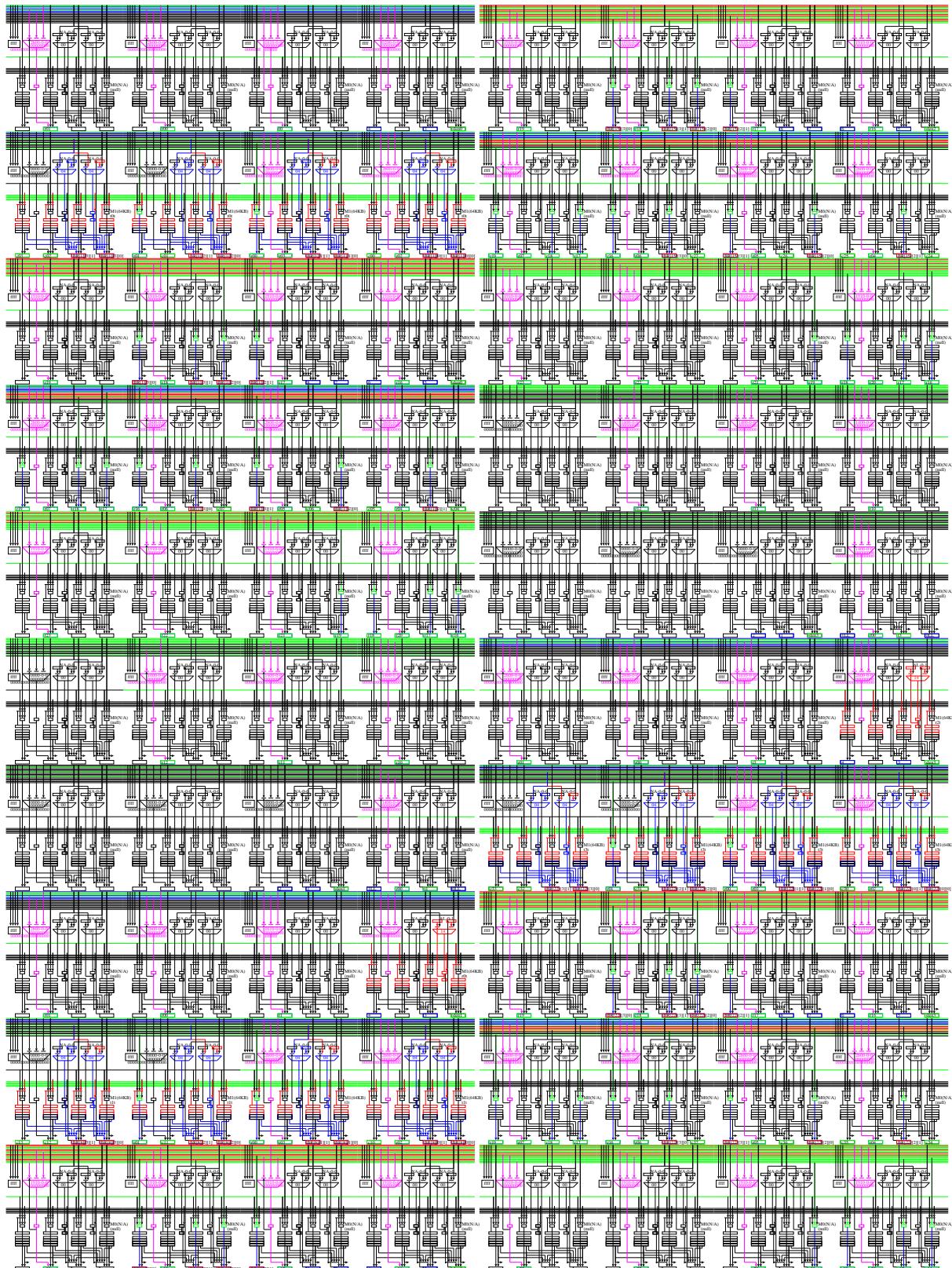


Figure.3.9: 文字列検索

3.2.4 16x16 置み込み

```
cent% make -f Makefile-csim.emax6+dma all clean
cent% ../../src/csim/csim -x conv16-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean
zynq% ./conv16-zynq.emax6+dma
```

16x16 の置み込み演算である。一度のバースト演算により 1 行分のみ計算する。ステンシル計算であり mapdist=2 である。

```
conv16_x1(float *yi, float *yo)
{
    U11 loop = image_WD/2-8;
    U11 x = 8;
    #if !defined(EMAX5) & !defined(EMAX6)
    while (loop--) {
        yo[x] = yim8[x-8]*SCON[ 0]+yim8[x-7]*SCON[ 1]+yim8[x-6]*SCON[ 2]+yim8[x-5]*SCON[ 3]+yim8[x-4]*SCON[ 4]+yim8[x-3]*SCON[ 5]+yim8[x-2]*SCON[ 6]+yim8[x-1]*SCON[ 7];
        +yim8[x+0]*SCON[ 8]+yim8[x+1]*SCON[ 9]+yim8[x+2]*SCON[ 10]+yim8[x+3]*SCON[ 11]+yim8[x+4]*SCON[ 12]+yim8[x+5]*SCON[ 13]+yim8[x+6]*SCON[ 14]+yim8[x+7]*SCON[ 15];
        +yim7[x-8]*SCON[ 16]+yim7[x-7]*SCON[ 17]+yim7[x-6]*SCON[ 18]+yim7[x-5]*SCON[ 19]+yim7[x-4]*SCON[ 20]+yim7[x-3]*SCON[ 21]+yim7[x-2]*SCON[ 22]+yim7[x-1]*SCON[ 23];
        +yim7[x+0]*SCON[ 24]+yim7[x+1]*SCON[ 25]+yim7[x+2]*SCON[ 26]+yim7[x+3]*SCON[ 27]+yim7[x+4]*SCON[ 28]+yim7[x+5]*SCON[ 29]+yim7[x+6]*SCON[ 30]+yim7[x+7]*SCON[ 31];
        +yim6[x-8]*SCON[ 32]+yim6[x-7]*SCON[ 33]+yim6[x-6]*SCON[ 34]+yim6[x-5]*SCON[ 35]+yim6[x-4]*SCON[ 36]+yim6[x-3]*SCON[ 37]+yim6[x-2]*SCON[ 38]+yim6[x-1]*SCON[ 39];
        :
        +yip1[x+0]*SCON[162]+yip1[x+1]*SCON[163]+yip1[x+2]*SCON[164]+yip1[x+3]*SCON[155]+yip1[x+4]*SCON[156]+yip1[x+5]*SCON[157]+yip1[x+6]*SCON[158]+yip1[x+7]*SCON[159];
        +yip2[x-8]*SCON[160]+yip2[x-7]*SCON[161]+yip2[x-6]*SCON[162]+yip2[x-5]*SCON[163]+yip2[x-4]*SCON[164]+yip2[x-3]*SCON[165]+yip2[x-2]*SCON[166]+yip2[x-1]*SCON[167];
        +yip2[x+0]*SCON[168]+yip2[x+1]*SCON[169]+yip2[x+2]*SCON[170]+yip2[x+3]*SCON[171]+yip2[x+4]*SCON[172]+yip2[x+5]*SCON[173]+yip2[x+6]*SCON[174]+yip2[x+7]*SCON[175];
        +yip3[x-8]*SCON[176]+yip3[x-7]*SCON[177]+yip3[x-6]*SCON[178]+yip3[x-5]*SCON[179]+yip3[x-4]*SCON[180]+yip3[x-3]*SCON[181]+yip3[x-2]*SCON[182]+yip3[x-1]*SCON[183];
        +yip3[x+0]*SCON[184]+yip3[x+1]*SCON[185]+yip3[x+2]*SCON[186]+yip3[x+3]*SCON[187]+yip3[x+4]*SCON[188]+yip3[x+5]*SCON[189]+yip3[x+6]*SCON[190]+yip3[x+7]*SCON[191];
        +yip4[x-8]*SCON[192]+yip4[x-7]*SCON[193]+yip4[x-6]*SCON[194]+yip4[x-5]*SCON[195]+yip4[x-4]*SCON[196]+yip4[x-3]*SCON[197]+yip4[x-2]*SCON[198]+yip4[x-1]*SCON[199];
        +yip4[x+0]*SCON[200]+yip4[x+1]*SCON[201]+yip4[x+2]*SCON[202]+yip4[x+3]*SCON[203]+yip4[x+4]*SCON[204]+yip4[x+5]*SCON[205]+yip4[x+6]*SCON[206]+yip4[x+7]*SCON[207];
        +yip5[x-8]*SCON[208]+yip5[x-7]*SCON[209]+yip5[x-6]*SCON[210]+yip5[x-5]*SCON[211]+yip5[x-4]*SCON[212]+yip5[x-3]*SCON[213]+yip5[x-2]*SCON[214]+yip5[x-1]*SCON[215];
        +yip5[x+0]*SCON[216]+yip5[x+1]*SCON[217]+yip5[x+2]*SCON[218]+yip5[x+3]*SCON[219]+yip5[x+4]*SCON[220]+yip5[x+5]*SCON[221]+yip5[x+6]*SCON[222]+yip5[x+7]*SCON[223];
        +yip6[x-8]*SCON[224]+yip6[x-7]*SCON[225]+yip6[x-6]*SCON[226]+yip6[x-5]*SCON[227]+yip6[x-4]*SCON[228]+yip6[x-3]*SCON[229]+yip6[x-2]*SCON[230]+yip6[x-1]*SCON[231];
        +yip6[x+0]*SCON[232]+yip6[x+1]*SCON[233]+yip6[x+2]*SCON[234]+yip6[x+3]*SCON[235]+yip6[x+4]*SCON[236]+yip6[x+5]*SCON[237]+yip6[x+6]*SCON[238]+yip6[x+7]*SCON[239];
        +yip7[x-8]*SCON[240]+yip7[x-7]*SCON[241]+yip7[x-6]*SCON[242]+yip7[x-5]*SCON[243]+yip7[x-4]*SCON[244]+yip7[x-3]*SCON[245]+yip7[x-2]*SCON[246]+yip7[x-1]*SCON[247];
        +yip7[x+0]*SCON[248]+yip7[x+1]*SCON[249]+yip7[x+2]*SCON[250]+yip7[x+3]*SCON[251]+yip7[x+4]*SCON[252]+yip7[x+5]*SCON[253]+yip7[x+6]*SCON[254]+yip7[x+7]*SCON[255];
        x += 2;
    }
    #endif
}
```

```
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 c000=DCON[ 0], c004=DCON[ 1], c006=DCON[ 3], c008=DCON[ 4], c010=DCON[ 5], c012=DCON[ 6], c014=DCON[ 7];
U11 c016=DCON[ 8], c018=DCON[ 9], c020=DCON[ 10], c022=DCON[ 11], c024=DCON[ 13], c030=DCON[ 15];
U11 c032=DCON[ 16], c034=DCON[ 17], c036=DCON[ 18], c038=DCON[ 19], c040=DCON[ 20], c042=DCON[ 21], c044=DCON[ 22], c046=DCON[ 23];
U11 c048=DCON[ 24], c050=DCON[ 25], c052=DCON[ 26], c054=DCON[ 27], c056=DCON[ 28], c058=DCON[ 29], c060=DCON[ 30], c062=DCON[ 31];
U11 c064=DCON[ 32], c066=DCON[ 33], c068=DCON[ 34], c070=DCON[ 35], c072=DCON[ 36], c074=DCON[ 37], c076=DCON[ 38], c078=DCON[ 39];
:
U11 c240=DCON[120], c242=DCON[121], c244=DCON[122], c246=DCON[123], c248=DCON[124], c250=DCON[125], c252=DCON[126], c254=DCON[127];
//EMAX5A begin x1 mapdist=2
while (loop--) { /* mapped to WHILE() on BR[15][0][0] stage#0 */
    mop(OP_LDR, 3, &BR[0][0][1], yim80++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr1, yim81++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr2, yim82++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr3, yim83++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr4, yim84++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr5, yim85++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr6, yim86++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    mop(OP_LDR, 3, kr7, yim87++, 0, MSK_D0, yim80, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
    exe(OP_FMA, kr10, OLL, EXP_H3210, BR[0][1], EXP_H3210, c000, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_FMA, kr11, OLL, EXP_H3210, r1, EXP_H3210, c002, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_FMA, kr12, OLL, EXP_H3210, r2, EXP_H3210, c004, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_FMA, kr13, OLL, EXP_H3210, r3, EXP_H3210, c006, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_FMA, kr20, r10, EXP_H3210, r4, EXP_H3210, c008, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
    exe(OP_FMA, kr21, r11, EXP_H3210, r5, EXP_H3210, c010, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
    exe(OP_FMA, kr22, r12, EXP_H3210, r6, EXP_H3210, c012, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
    exe(OP_FMA, kr23, r13, EXP_H3210, r7, EXP_H3210, c014, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
    mop(OP_LDR, 3, &BR[2][0][1], yim70++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr1, yim71++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr2, yim72++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr3, yim73++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr4, yim74++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr5, yim75++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr6, yim76++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    mop(OP_LDR, 3, kr7, yim77++, 0, MSK_D0, yim70, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    :
    exe(OP_FMA, kr10, r20, EXP_H3210, BR[30][1], EXP_H3210, c240, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#31 */
    exe(OP_FMA, kr11, r21, EXP_H3210, r1, EXP_H3210, c242, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#31 */
    exe(OP_FMA, kr12, r22, EXP_H3210, r2, EXP_H3210, c244, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#31 */
    exe(OP_FMA, kr13, r23, EXP_H3210, r3, EXP_H3210, c246, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#31 */
    exe(OP_FMA, kr20, r10, EXP_H3210, r4, EXP_H3210, c248, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#32 */
    exe(OP_FMA, kr21, r11, EXP_H3210, r5, EXP_H3210, c250, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#32 */
    exe(OP_FMA, kr22, r12, EXP_H3210, r6, EXP_H3210, c252, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#32 */
    exe(OP_FMA, kr23, r13, EXP_H3210, r7, EXP_H3210, c254, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#32 */
    exe(OP_FAD, kr10, r20, EXP_H3210, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#33 */
    exe(OP_FAD, kr12, r22, EXP_H3210, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#33 */
    exe(OP_FAD, &AR[35][0], r10, EXP_H3210, r12, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#35 */
    mop(OP_STR, 3, &AR[35][0], yoo++, OLL, MSK_D0, (U11)yoo, 152, 0, 0, yop, 152); /* stage#35 */
}
//EMAX5A end
#endif
}
```

conv16-x1-emax6.obj

BR/row: max=16 min=2 ave=12 EA/row: max=16 min=0 ave=7 ARpass/row: max=0

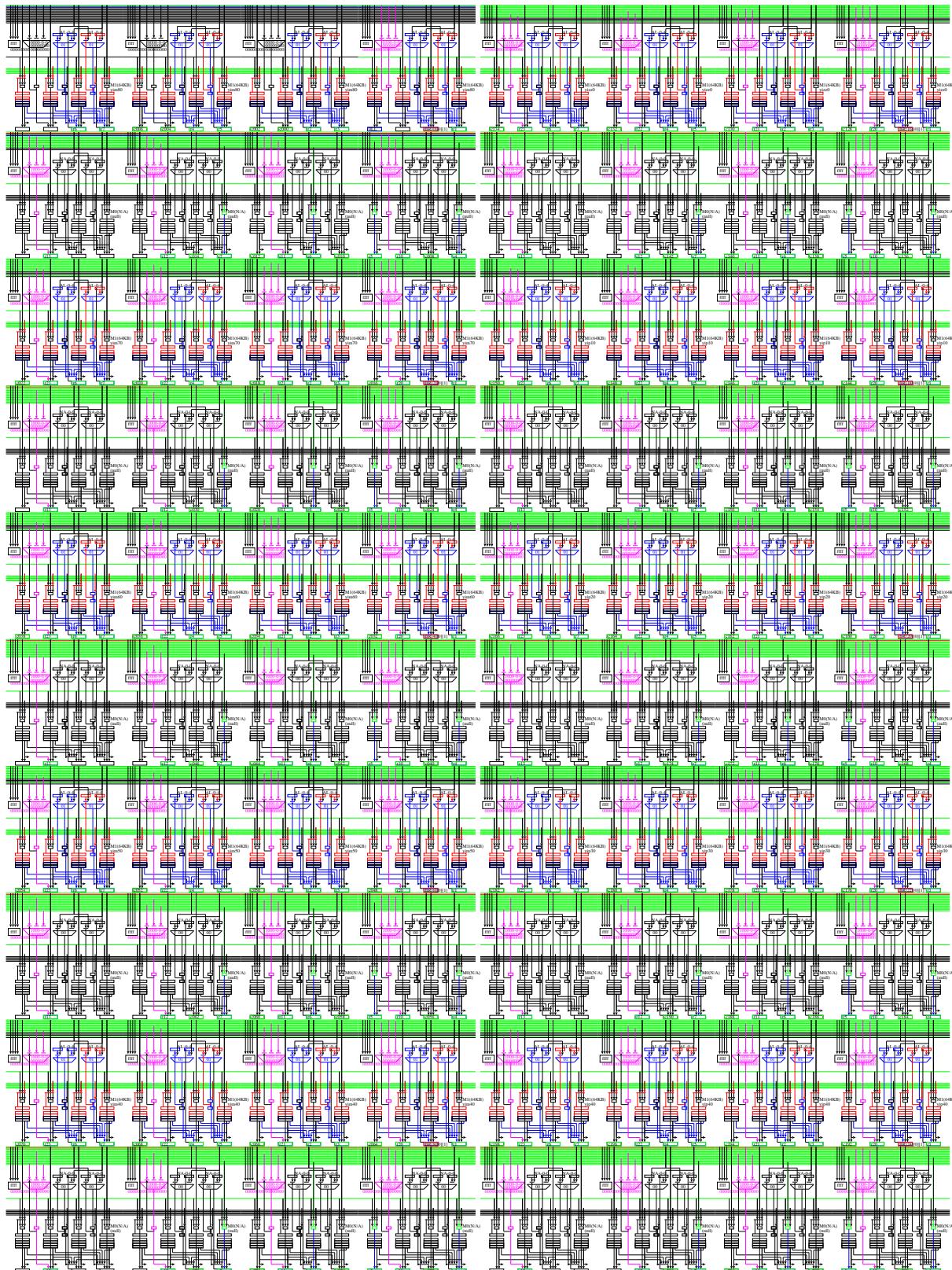


Figure.3.10: 16x16 畳み込み

3.2.5 VBGMM_Logsum

```
cent% make -f Makefile-csim.emax6+dma test016-csim.emax6+dma clean
cent% ../../src/csim/csim -x test016-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma test016-zynq.emax6+dma clean
zynq% ./test016-zynq.emax6+dma
```

```
for (chip=0; chip<NCHIP; chip++) { /* will be parallelized by multi-chip (M/#chip) */
    for (grp=M/NCHIP*chip; grp<M/NCHIP*(chip+1); grp+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
        typedef struct {Uint i[4];} Ui4;
        Ui4 *c60 = C1+grp*M, *c600 = c60;
        U11 row, bofs, rofs;
        U11 b00;
        U11 PARAM = 0x0000000100000001LL; /* 1 */
//EMAXSA begin x1 mapdist=0
/*2*/ for (INITi=1,LOOP1=RMGRP, row=0-M*4; LOOP1--; INITi--) { /* stage#0 */
    /*1*/ for (INIT0=1,LOOP0=M/W,bofs=0-W*4; LOOP0--; INIT0--) { /* stage#0 */
        exe(OP_ADD,      &bofs, INIT0?bofs:bofs, EXP_H3210, W*4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000000ffffffffffLL, OP_NOP, OLL);/* stage#0 */
        exe(OP_ADD,      &row,  row, EXP_H3210, INIT0?W*4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD,      &rofs, row, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000000ffffffffffLL, OP_NOP, OLL);/* stage#1 */
        mop(OP_LDWR,     1, &b00, (U11)c600, (U11)rofs, MSK_WO, (U11)c60, M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#2 */
        exe(OP_ADD,      &b00, INIT0?b00:b00, EXP_H3210, PARAM, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        mop(OP_STWR,     1, &b00, (U11)rofs, (U11)c600, MSK_D0, (U11)c60, M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#2 */
    }
}
//EMAXSA end
//EMAXSA drain_dirty_lmm
}
```

3.2.6 Stochastic sgemm00

```
cent% make -f Makefile-csim.emax6+dma test021-csim.emax6+dma clean
cent% ../../src/csim/csim -x test021-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma test021-zynq.emax6+dma clean
zynq% ./test021-zynq.emax6+dma
```

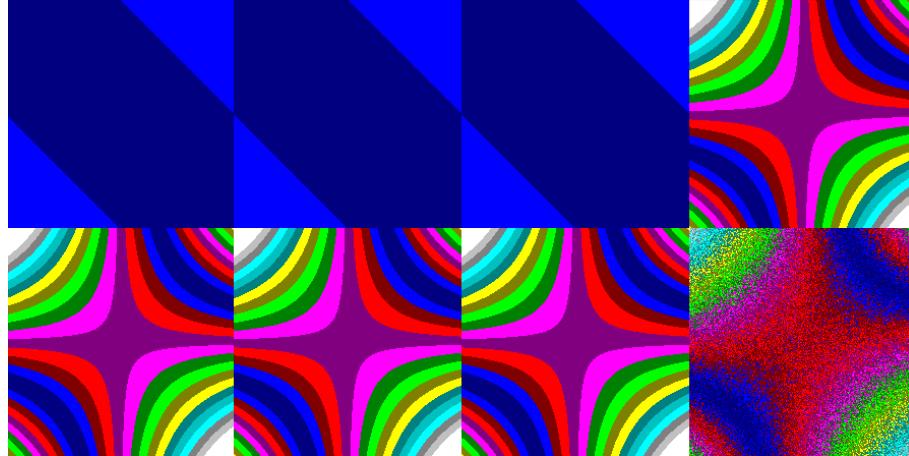


Figure 3.11: Stochastic sgemm00

```
for (top=0; top<BS/NCHIP; top+=H) { /* will be parallelized by multi-chip (M/#chip) */
    for (blk=0; blk<OC4; blk+=RMGRP) { /* 3 重ループ展開の外側対象 */
        char *a[H][NCHIP];
        char *b, *b0;
        char *c[H][NCHIP], *c0[H][NCHIP];
        b = (Uchar*)i_m0B+blk*IC32; b0 = b+IC32*0;
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            for (k=0; k<H; k++) {
                a[k][CHIP] = (Uchar*)i_mOA+(CHIP*BS/NCHIP+top+k)*IC32;
                c[k][CHIP] = (Uchar*)i_mOC+(CHIP*BS/NCHIP+top+k)*OC4+blk;
                c0[k][CHIP]= c[k][CHIP]+0;
            }
        }
    }
}

#define spike01_core1(r, s) \
    mod(OP_LDRQ, 1, BR[r][2], (U11)b0, (U11)bfs, MSK_W1, (U11)b, IC32D4RMRGP, 0, 0, (U11)NULL, IC32D4RMRGP); /* stage#2 */ \
    mod(OP_LDRQ, 1, BR[r][1], (U11)a[s][CHIP], (U11)cofs, MSK_W1, (U11)a[s][CHIP], IC32D4, 0, 0, (U11)NULL, IC32D4); /* stage#2 */ \
    exe(OP_NOP, &AR[r][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */ \
    mod(OP_LDBR, 1, &b0, (U11)c0[s][CHIP], (U11)oofs, MSK_W0, (U11)c[s][CHIP], RMGRPD4, 0, 1, (U11)NULL, RMGRPD4); /* stage#2 */ \
    ex4(OP_SFMA, &b0, INIT0?b00:b0, EXP_H3210, BR[r][1], EXP_H3210, BR[r][2], EXP_H3210, OP_NOP, 3LL, OP_NOP, OLL); /* stage#2 */ \
    mod(OP_STBR, 1, &b0, (U11)oofs, (U11)c0[s][CHIP], MSK_D0, (U11)c[s][CHIP], RMGRPD4, 0, 1, (U11)NULL, RMGRPD4); /* stage#2 */

//EMAX5A begin smax2 mapdist0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/*3*/ for (INIT1=1,LOOP1=RMRGP,rofs=(0-IC32)<<32|(0-1LL)&0xffffffff; LOOP1--; INIT1=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
/*3*/ /*3*/ for (INIT0=1,LOOP0=IC32/32,cofs=(0-32LL)<<32|(0)&0xffffffff; LOOP0--; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
/*3*/ /*3*/ /*3*/ for (INIT0=1,LOOP0=IC32/32,cofs=(0-32LL)<<32|(0)&0xffffffff; LOOP0--; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, (32LL)<<32|(0), EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffffLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?IC521:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD, &bofs, rofs, EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0xfffffffffffffLL, OP_NOP, OLL); /* stage#1 */
        exe(OP_ADD, &oofs, rofs, EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL); /* stage#1 */
    }

    spike01_core1( 2, 0);
    spike01_core1( 3, 1);
    spike01_core1( 4, 2);
    :
    spike01_core1(18, 18);
    spike01_core1(19, 19); /* H=20 */
    spike01_core1(20, 20);
    spike01_core1(21, 21);
    spike01_core1(22, 22);
    spike01_core1(23, 23);
    spike01_core1(24, 24); /* H=25 */
    :
    spike01_core1(50, 48);
    spike01_core1(51, 49); /* H=50 */
}
}
//EMAX5A end
}
//EMAX5A drain_dirty_lmm
```

smax-smax2-emax6.obj

BR/row: max=13 min=2 ave=12 EA/row: max=4 min=0 ave=3 ARpass/row: max=0

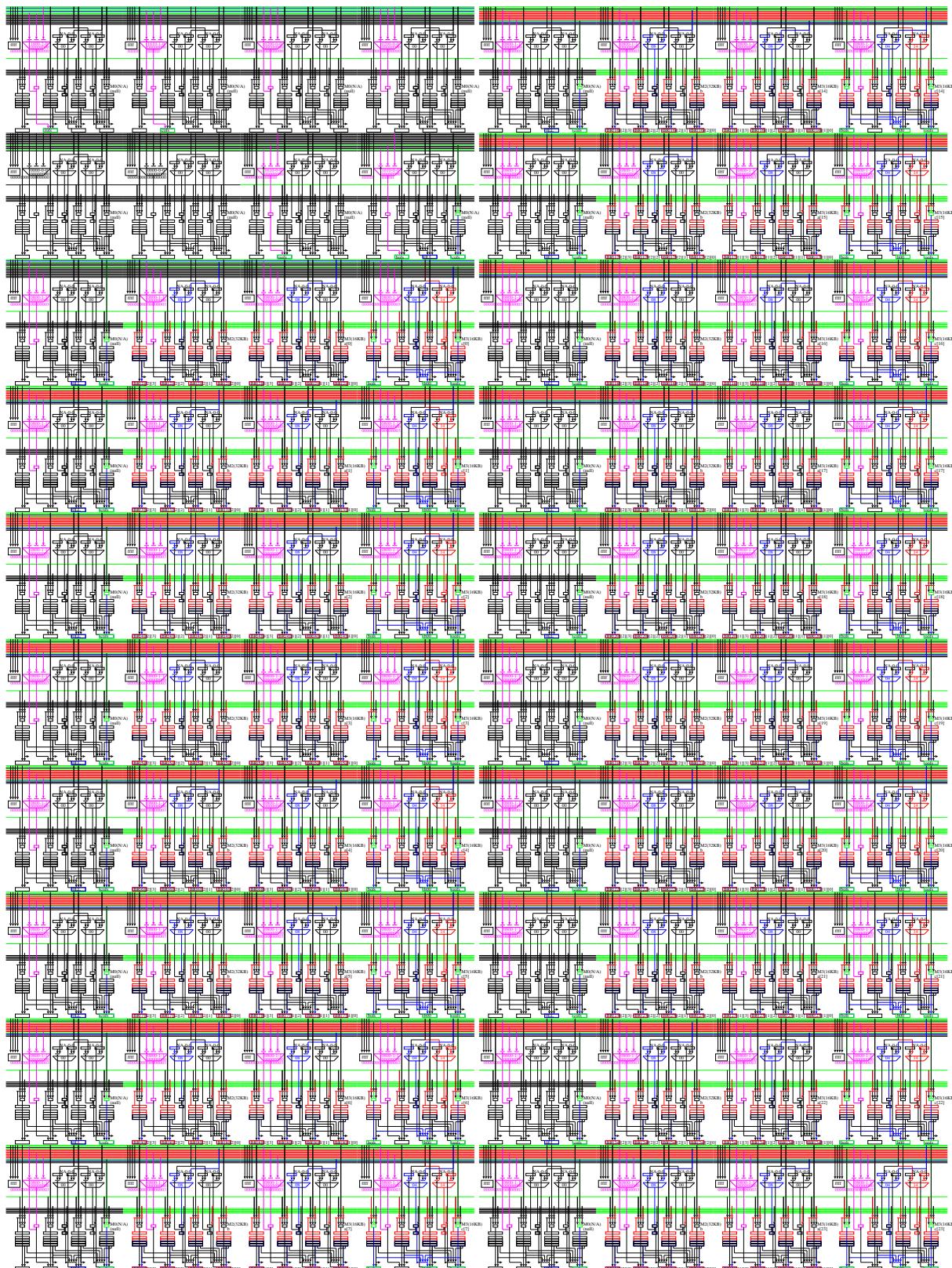


Figure.3.12: Stochastic sgemm00

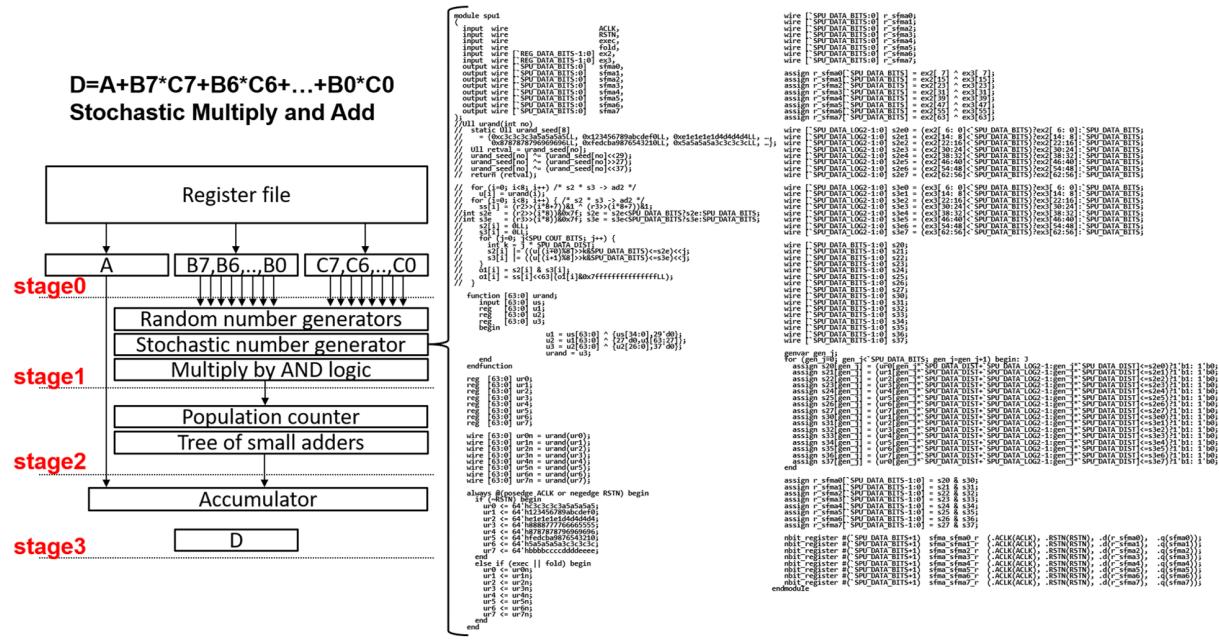


Figure 3.13: SFMA 1st stage

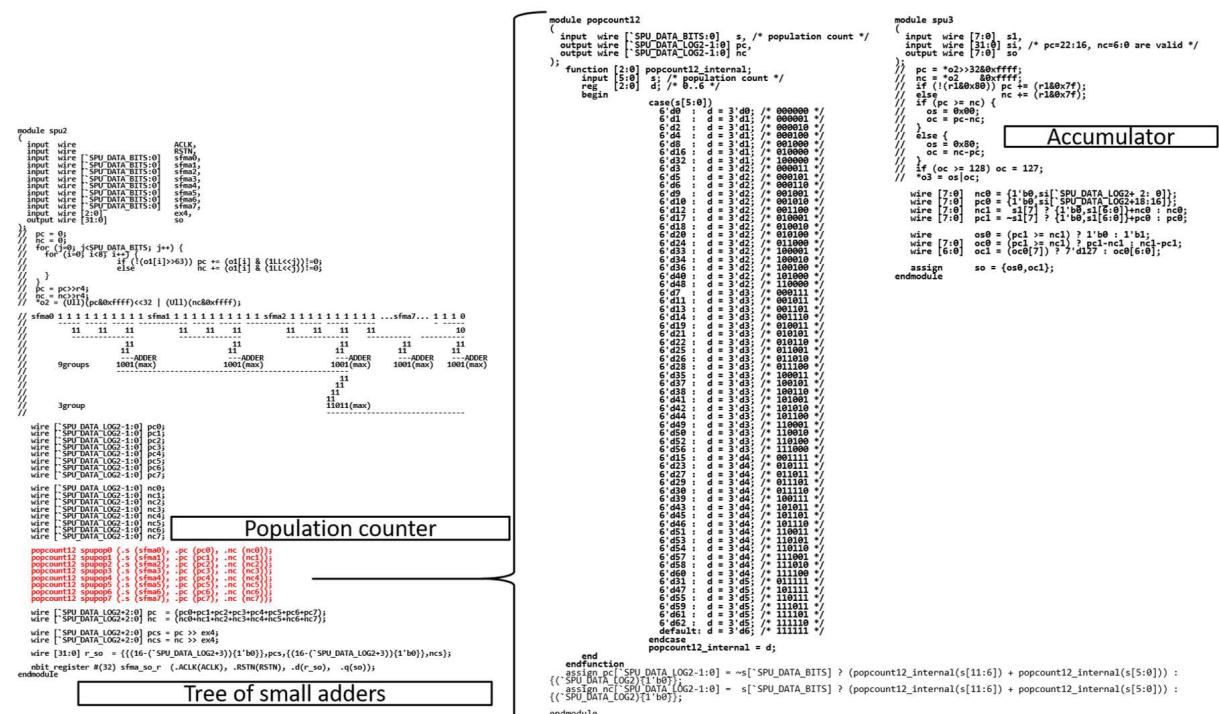


Figure 3.14: SFMA 2nd and 3rd stage

3.2.7 Sparse matrix multiplication

```
cent% make -f Makefile-csim.emax6+dma test022-csim.emax6+dma clean
cent% ../../src/csim/csim -x test022-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma test022-zynq.emax6+dma clean
zynq% ./test022-zynq.emax6+dma
```

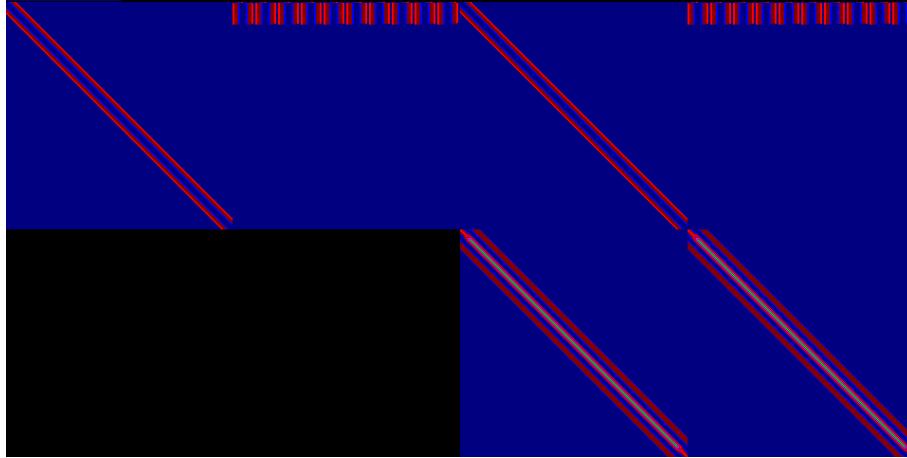


Figure.3.15: Sparse matrix multiplication

Single flow

```
for (blk=0; blk<M2; blk+=RMGRP) { /* 3 重ループ展開の外側対象 */
    for (top=0; top<M1/NCHIP; top+=H) { /* will be parallelized by multi-chip (M/#chip) */
        packed *a[H][NCHIP], *a0[H][NCHIP];
        packed *b, *b0[H];
        float *c[H][NCHIP], *c0[H][NCHIP];
        b = B32_P+blk*LP;
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            for (k=0; k<H; k++) {
                a[K][CHIP] = A32_P+(CHIP*M1/NCHIP+top+k)*LP;
                c[K][CHIP] = C32_1+(CHIP*M1/NCHIP+top+k)*M2+blk;
                c0[K][CHIP] = c[K][CHIP]+0;
            }
        }
#define sparse_core1(r, h) \
    mex(OP_CMPA_LE, &b0[h], INIT0?b:b0[h], INIT0?0:8, OP_CMPA_GE, &a0[h][CHIP], INIT0?a[h][CHIP]:a0[h][CHIP], INIT0?0:8, OLL, BR[r][2][1], BR[r][2][0]);\
    mop(OP_LDR, 3, &BR[r][2][1], b0[h], bofs, MSK_W1, b, 2*LP*RMGRP, 0, 0, NULL, 2*LP*RMGRP);/*LMM[2] col2*/\
    mop(OP_LDR, 3, &BR[r][2][0], a0[h][CHIP], bofs, MSK_W0, a[h][CHIP], 2*LP, 0, 0, NULL, 2*LP);/*LMM[1] col2*/\
    exe(OP_NOP, 0, OLL, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0);\
    mop(OP_LDWR, 1, &c00, c0[h][CHIP], oofs, MSK_W0, c[h][CHIP], RMGRP, 0, 1, NULL, RMGRP);\
    exe(OP_CFMA, &c00, INIT0?c00:c00, EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][2][0], EXP_H3210, OP_NOP, 0, OP_NOP, 0);\
    mop(OP_STWR, 1, &c00, oofs, c0[h][CHIP], MSK_D0, c[h][CHIP], RMGRP, 0, 1, NULL, RMGRP);

//EMAX5A begin imax mapdist=0
/**/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/**/ for (INIT1=1,LOOP1=RMGRP,rofs=(0-LP*8)<<32|((0-4LL)&0xffffffff); LOOP1--; INIT1=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
/**/ /*1*/ for (INIT0=1,LOOP0=LP,cofs=(0LL)<<32|(0LL)&0xffffffff; LOOP0--; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?(LP*8)<<32|(4LL):0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &bofs, rofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &oofs, rofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffffL, OP_NOP, OLL); /* stage#1 */

            sparse_core1(2, 0);
            sparse_core1(3, 1);
            sparse_core1(4, 2);
            sparse_core1(5, 3);
            sparse_core1(6, 4);
            sparse_core1(7, 5);
            sparse_core1(8, 6);
            sparse_core1(9, 7);
            sparse_core1(10, 8);
            sparse_core1(11, 9);
            sparse_core1(12, 10);
            sparse_core1(13, 11);
            sparse_core1(14, 12);
            sparse_core1(15, 13);
            sparse_core1(16, 14);
            sparse_core1(17, 15); /* H=16 */
        }
    }
}
//EMAX5A end
}
//EMAX5A drain_dirty_lmm
```

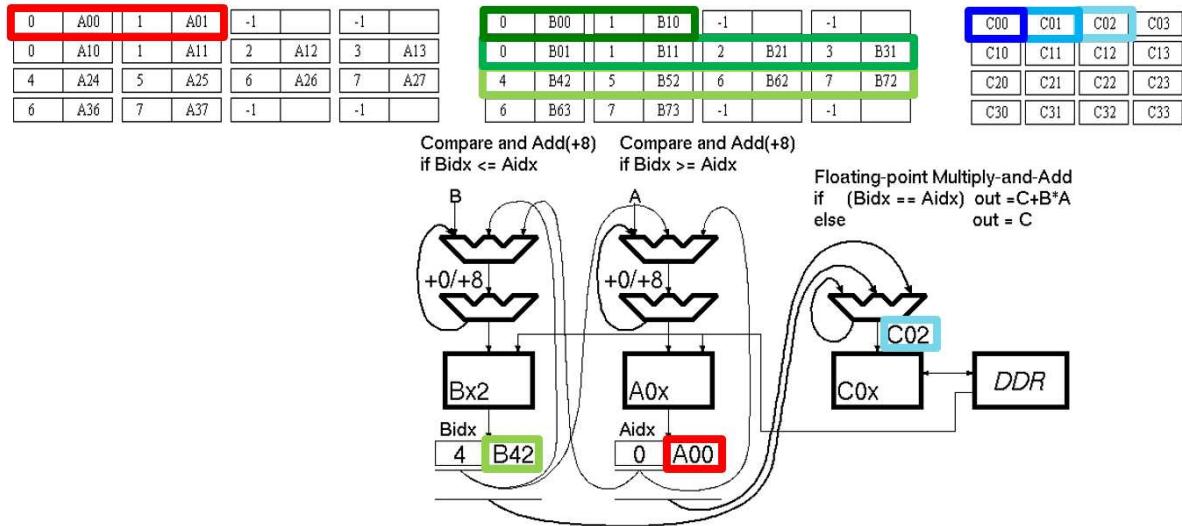


Figure 3.16: Data path for sparse matrix multiplication

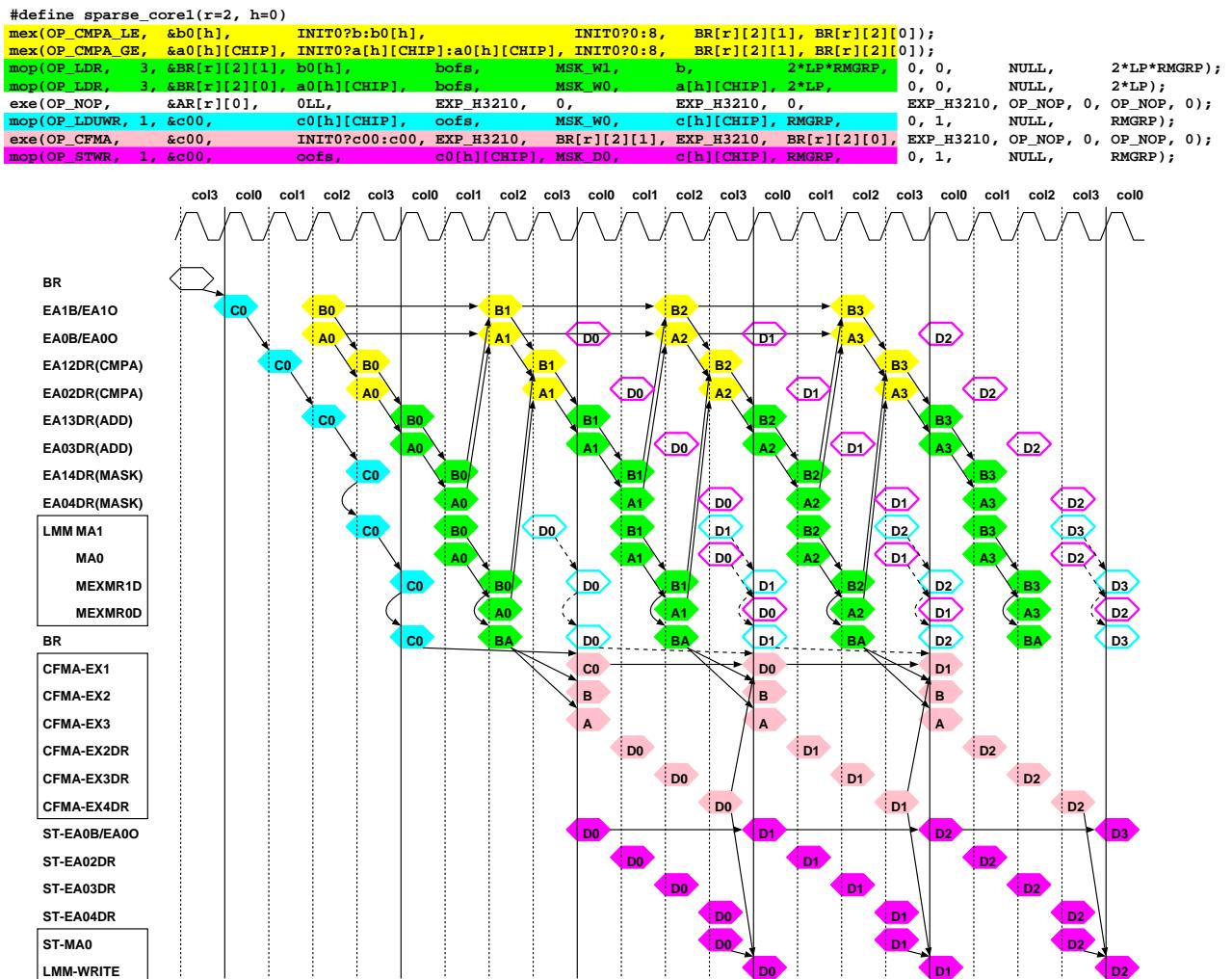


Figure 3.17: Timing chart (single flow)

test022-imax-emax6.obj

BR/row: max=6 min=2 ave=5 EA/row: max=4 min=0 ave=3 ARpass/row: max=0

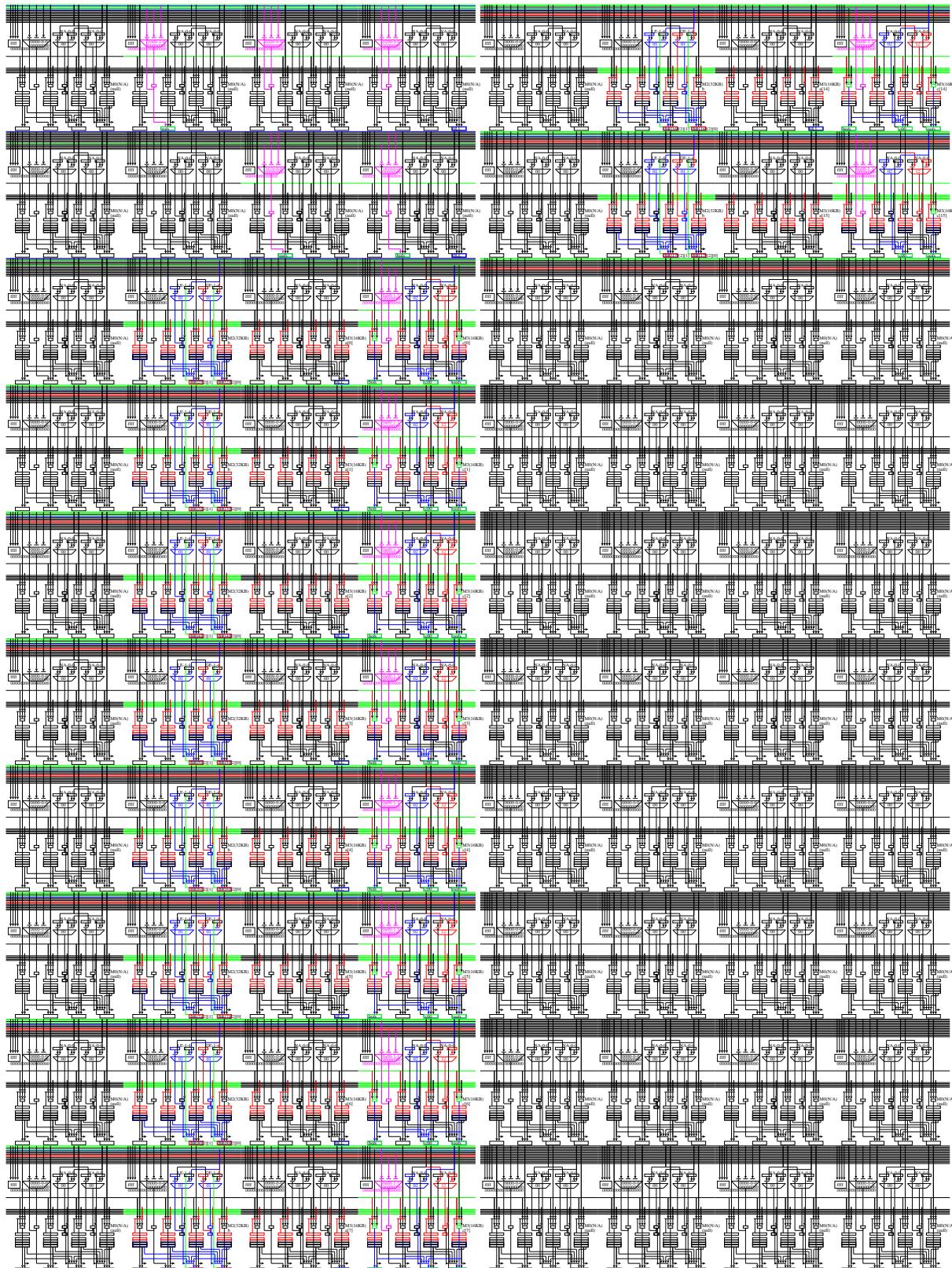


Figure.3.18: Sparse matrix multiplication (single flow)

Dual flow

```

for (blk=0; blk<M2; blk+=RMGRP) { /* 3 重ループ展開の外側対象 */
    for (top=0; top<M1/NCHIP; top+=H) { /* will be parallelized by multi-chip (M/#chip) */
        packed *a[H] [NCHIP], *a0[H] [2] [NCHIP]; /* a は共通,a0 は同一行を参照 (移動パターンが違うので 2 つ必要) */
        packed *b[2], *b0[H] [2]; /* b は共通,b0 の 2 行を 1unit にマルチスレッド化すれば c が連続 */
        float c[H] [NCHIP], c0[H] [2] [NCHIP]; /* c は共通,c0 は連続 */
        b[0] = B32_P(b1k+0)*LP;
        b[1] = B32_P(b1k+1)*LP;
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            for (k=0; k<H; k++) {
                a[k] [CHIP] = A32_P((CHIP*M1/NCHIP+top+k)*LP;
                c[k] [CHIP] = C32_1+(CHIP*M1/NCHIP+top+k)*M2+blk;
                c0[k] [0] [CHIP] = c[k] [CHIP]+0;
                c0[k] [1] [CHIP] = c[k] [CHIP]+1;
            }
        }
#define sparse_core1(r, h) \
    mex(OP_CMPA_LE, &b0[h][0], INIT0?b[0]:b0[h][0], INIT0?0:8, OP_CMPA_GE, &a0[h][0][CHIP], INIT0?a[h][0][CHIP]:a0[h][0][CHIP], INIT0?0:8, OLL, BR[r][2][1], BR[r][2][0]);\
    mop(OP_LDR, 3, &BR[r][2][1], b0[h][0], bofs, MSK_W1, b[0], 2*LP*RMGRP, 0, 0, NULL, 2*LP*RMGRP);\
    mop(OP_LDR, 3, &BR[r][2][0], a0[h][0][CHIP], bofs, MSK_W0, a[h][CHIP], 2*LP, 0, 0, NULL, 2*LP);\
    mex(OP_CMPA_LE, &b0[h][1], INIT0?b[1]:b0[h][1], INIT0?0:8, OP_CMPA_GE, &a0[h][1][CHIP], INIT0?a[h][1][CHIP]:a0[h][1][CHIP], INIT0?0:8, OLL, BR[r][3][1], BR[r][3][0]);\
    mop(OP_LDR, 3, &BR[r][3][1], b0[h][1], bofs, MSK_W1, b[0], 2*LP*RMGRP, 0, 0, NULL, 2*LP*RMGRP);\
    mop(OP_LDR, 3, &BR[r][3][0], a0[h][1][CHIP], bofs, MSK_W0, a[h][CHIP], 2*LP, 0, 0, NULL, 2*LP);\
    exe(OP_LDWR, 1, &c00, c0[h][0][CHIP], EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0);\
    exe(OP_CFMA, &c00, INIT0?c00:c00, EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][2][0], EXP_H3210, OP_NOP, 0, OP_NOP, 0);\
    mop(OP_LDWR, 1, &c00, oofs, c0[h][0][CHIP], RMGRP, 0, 1, NULL, RMGRP);\
    mop(OP_CFMA, &c01, INIT0?c01:c01, EXP_H3210, BR[r][3][1], EXP_H3210, BR[r][3][0], EXP_H3210, OP_NOP, 0, OP_NOP, 0);\
    mop(OP_STWR, 1, &c01, oofs, c0[h][1][CHIP], RMGRP, 0, 1, NULL, RMGRP);
//EMAX5A begin max mapdist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/*1*/ for (INIT1=1, LOOP1=RMGRP/2, rofs=(0-8*LP*2)<<32|(0-8LL)&0xffffffff; LOOP1--; INIT1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
/*1*/ for (INIT0=1, LOOP0=LP, cofs=(0LL)<<32|(0LL)&0xffffffff; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?(8*LP*2)<<32|(8LL):0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_ADD, &rofs, rofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffff00000000LL, OP_NOP, OLL);
        exe(OP_ADD, &oofs, rofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
        sparse_core1( 2, 0);
        sparse_core1( 3, 1); /* H=2 */
        sparse_core1( 4, 2);
        sparse_core1( 5, 3); /* H=4 */
        sparse_core1( 6, 4);
        sparse_core1( 7, 5);
        sparse_core1( 8, 6);
        sparse.core1( 9, 7); /* H=8 */
        sparse.core1( 10, 8);
        sparse.core1( 11, 9);
        sparse.core1( 12, 10);
        sparse.core1( 13, 11);
        sparse.core1( 14, 12);
        sparse.core1( 15, 13);
        sparse.core1( 16, 14);
        sparse.core1( 17, 15); /* H=16 */
    }
}
//EMAX5A end
}
//EMAX5A drain_dirty_lmm

```

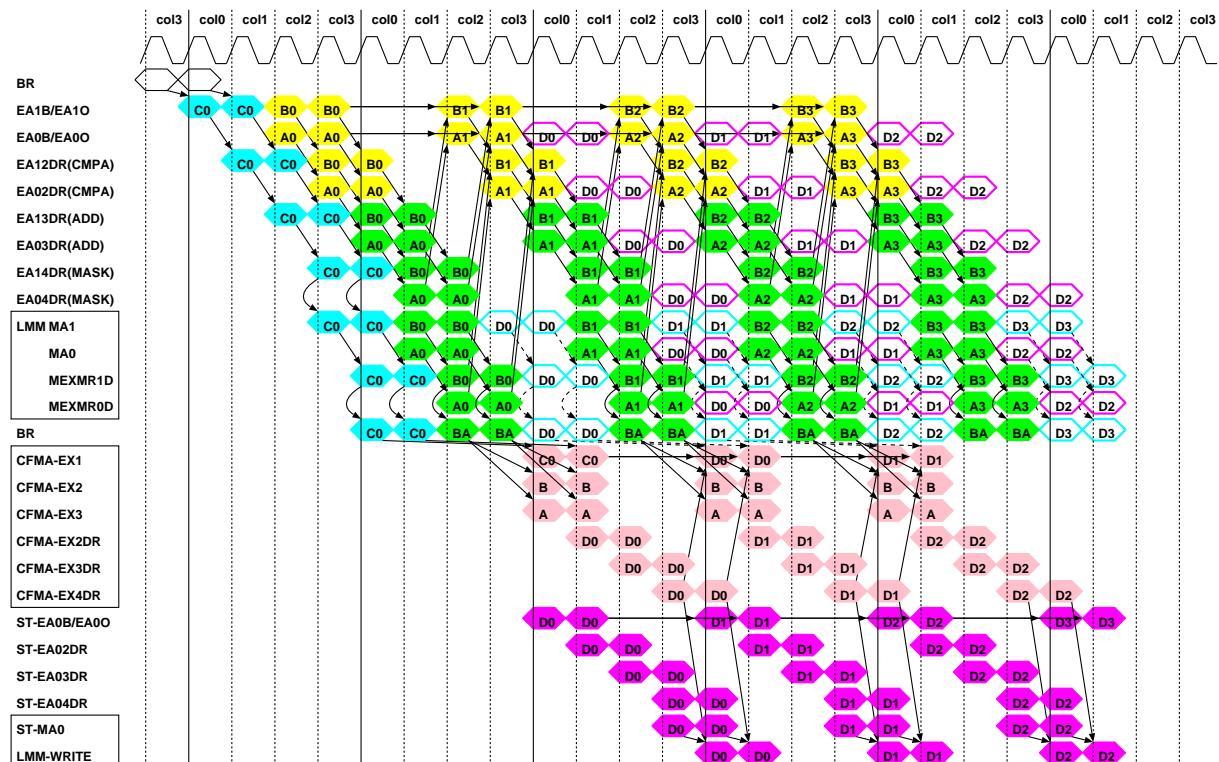


Figure 3.19: Timing chart (dual flow)

test022-imax-emax6.obj

BR/row: max=9 min=2 ave=7 EA/row: max=8 min=0 ave=7 ARpass/row: max=0

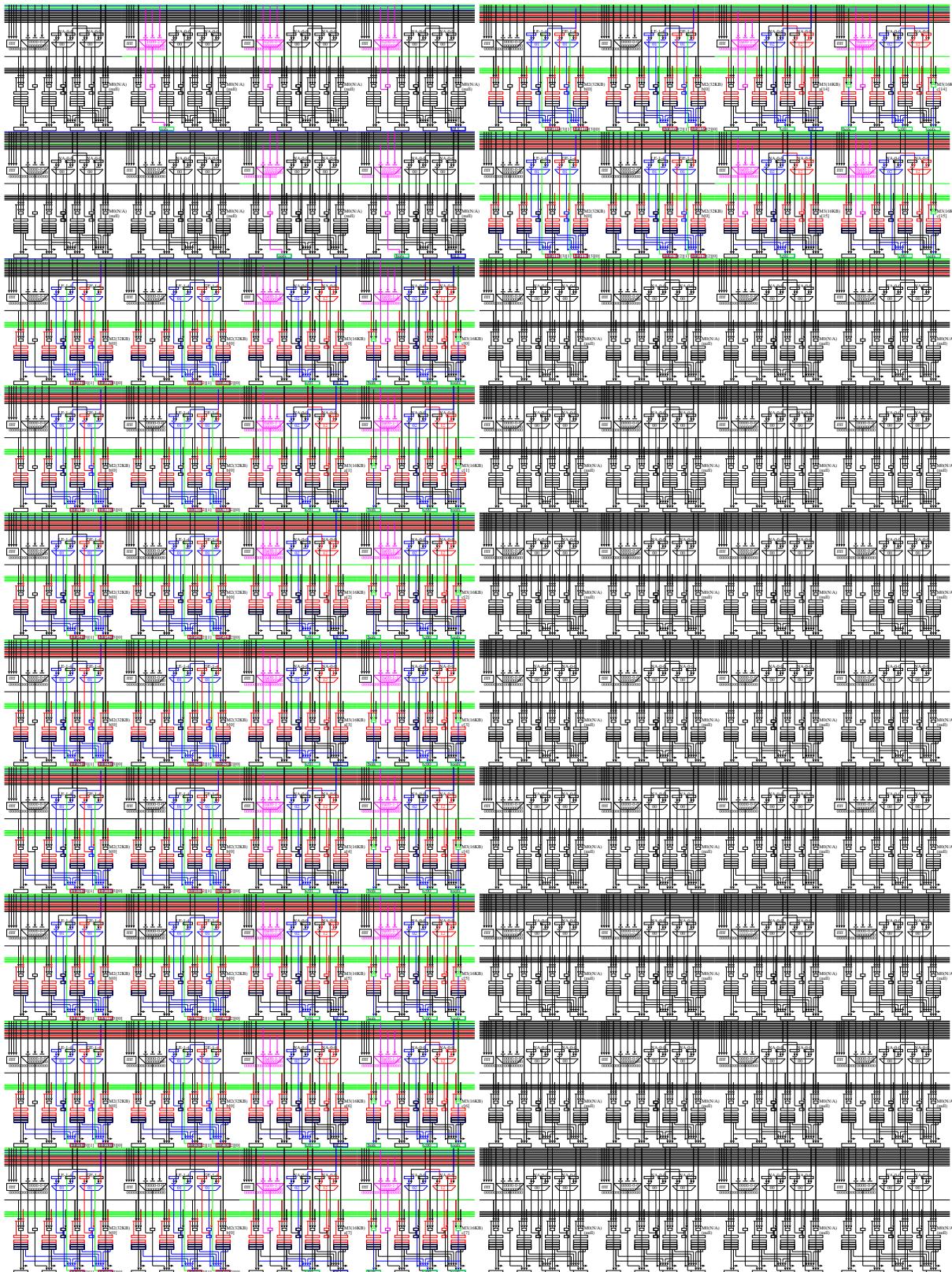


Figure.3.20: Sparse matrix multiplication (dual flow)

3.2.8 Sparse matrix compression

```
cent% make -f Makefile-csim.emax6+dma test024-csim.emax6+dma clean
cent% ../../src/csim/csim -x test024-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma test024-zynq.emax6+dma clean
zynq% ./test024-zynq.emax6+dma
```

密行列を疎行列圧縮表現に変換する。mapdist=0 のまま、同一 LMM の前半と後半を交互に使って、ダブルバッファリングする。前回実行結果の drain と、圧縮演算と、次回実行に必要な密行列のロードを同時に実行することで、高速化している。mop() 中で f=1 にすることで、BasIP の毎回の DDR への書き戻しを抑制している（最初の 1 回だけ load が増えるのは仕方ない）。

```
*Bas1P = B32_P-1; /* end of B32_0 */
for (i=0; i<M1; i+=RMGRP) {
    r1      = (U11)(i*M2-1)<<32;
    ibase0 = B32_0+i*M2;
    itop0  = B32_0+i*M2;
    itop1  = itop0+RMGRP*M2;
    obase0 = *Bas1P; /* end of B32_0 */
    otop1  = otop0;
    otop0  = *Bas1P+8; /* top of B32_P */

/*with-prefetch/post-drain
//EMAXSA begin imax mapdist=0
/*3*/for (CHIP=0; CHIP<CHIP; CHIP++) {
/*2*/for (INITI=1,LOOP1=RMGRP,rofs=0; LOOP1--; INITI=0) {
/*1*/for (INIT0=1,LOOP0=M2,cofs=0; LOOP0--; INIT0=0) {
    mop(OP_LDWR, 1, &r0,          ibase0++,           0,           MSK_D0,       itop0, M2*RMGRP, 0, 0, 0, itop1, M2*RMGRP);
    exe(OP_ADD,   &r1,          r1,           EXP_H3210, 0x100000000LL, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
    exe(OP_NOP,   &std,         r1,           EXP_H3210, 0, EXP_H3210, OP_OR, r0, OP_NOP, OLL);
    exe(OP_CMP_EQ, &c0,          r0,           EXP_H1010, 0x00000000LL, EXP_H1010, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
    exe(OP_CMP_EQ, &c1,          r0,           EXP_H1010, 0x80000000LL, EXP_H1010, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
    exe(OP_NOP,   &c2,          c0,           EXP_H3210, 0, EXP_H3210, OP_OR, cc1, OP_NOP, OLL);
    exe(OP_CMV,   &oofs,        cc2,           EXP_H3210, 0, EXP_H3210, 8, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
    exe(OP_ADD,   &obase0,      obase0, EXP_H3210, oofs, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
    mop(OP_STR,   3,  &obase0,   Bas1P, 0, MSK_D0, Bas1P, 2, 0, 0, 0, NULL, 2);
    exe(OP_NOP,   &AR[5][0], 0, EXP_H3210, 0, EXP_H3210, 0, EXP_H1010, OP_NOP, 0, OP_NOP, OLL);
    cex(OP_CEXE,  &ex0,        0, 0, 0, cc2, 0x0001);
    mop(OP_STR,   ex0, &std,     obase0, 0, MSK_D0, otop0, LP*2*RMGRP, 0, 0, otop1, LP*2*RMGRP);
}
}
//EMAXSA end
//EMAXSA drain_dirty_lmm
}
count1 = (packed*)Bas1P-(packed*)B32_P+1;
printf("Bas1P=%08.8x_%08.8x Packed=%d\n", (UInt)(*Bas1P>>32), (UInt)*Bas1P, count1);
```

test024-imax-emax6.obj

BR/row: max=5 min=3 ave=3

EA/row: max=2 min=0 ave=0

ARpass/row: max=0

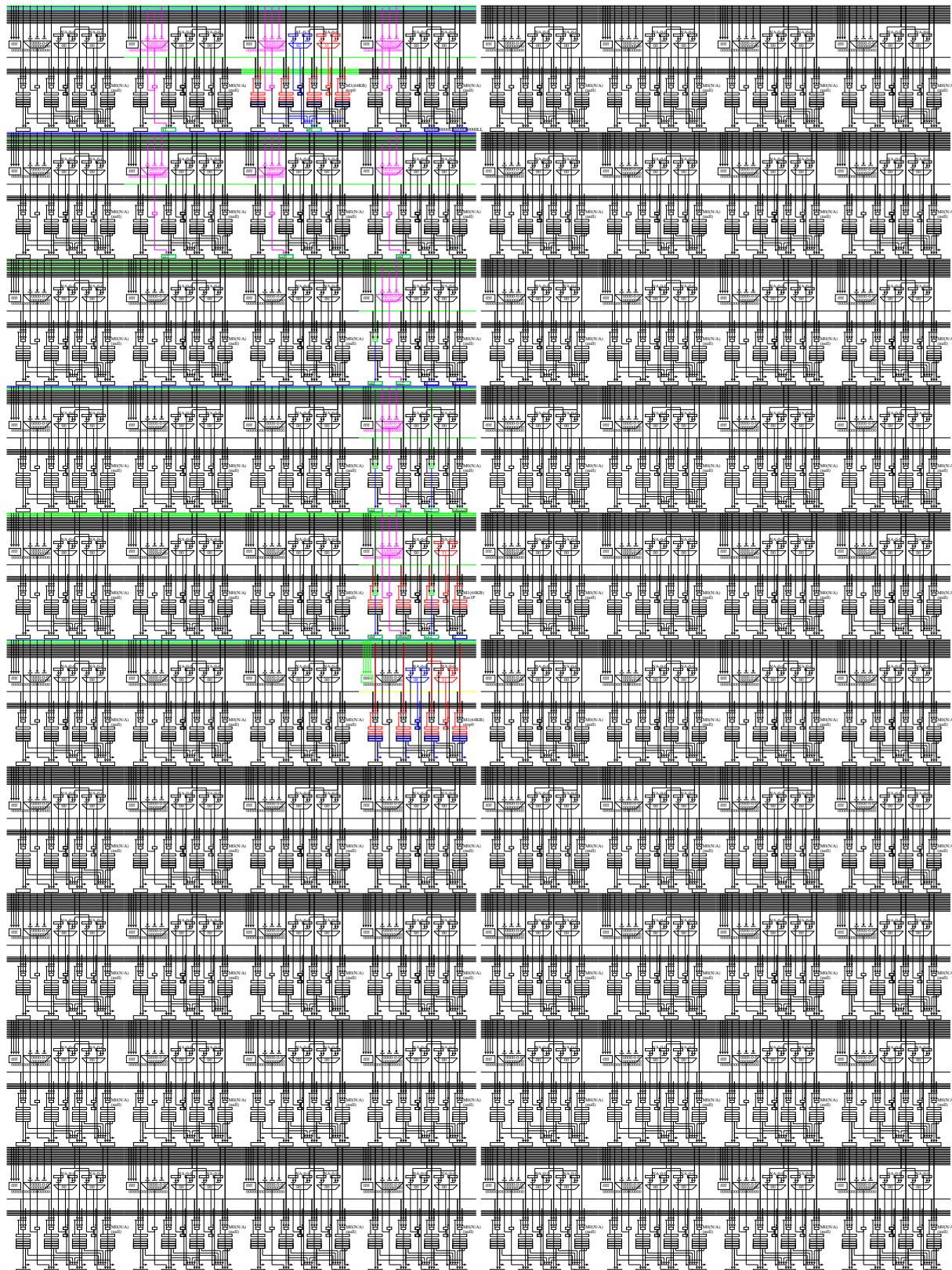


Figure.3.21: Sparse matrix compression

3.3 2D-imaging

```
cent% make -f Makefile-csim.emax6+dma all clean
cent% ../../src/csim/csim -x filter-csim.emax6+dma -f 81.ppm 82.ppm
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean
zynq% ./filter-zynq.emax6+dma -f 81.ppm 82.ppm
```

3.3.1 Tone_curve



Figure 3.22: Tone curve

RGB 各色の変換テーブルに基づき、画像を色変換する。一度のバースト演算により、10箇所 (OMAP=10) の各々について、6 行分の演算を行う (RMGRP=6)。ステンシル計算ではないため mapdist=0 である。なお、未使用 stage を使用した PLOAD により性能向上が可能である。

```
void tone_curve(Uint *r, Uint *d, Uchar *t) /* R, D, lut */
#if defined(EMAX5) && !defined(EMAX6)
for (top=PAD; top<HT-PAD; top++) { /* will be parallelized by multi-chip (M/#chip) */
    for (cofs=PAD; cofs<WD-PAD; cofs++) {
        Uint pix = *(r+top*WD+cofs);
        *(d+top*WD+cofs) = ((t[pix>>24])<<24 | (t[256+((pix>>16)&255)])<<16 | (t[512+((pix>>8)&255)])<<8);
    }
}
#endif
```

```
for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
        for (rofs=0; rofs<WD-PAD; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = (CHIP*RRANGE*OMAP+top+rofs)*WD;
            for (cofs=PAD; cofs<WD-PAD; cofs++) {
                for (oo=0; oo<OMAP; oo++) {
                    Uint pix = *(r+idx+oo*RRANGE*WD+cofs);
                    *(d+idx+oo*RRANGE*WD+cofs) = ((t[pix>>24])<<24 | (t[256+((pix>>16)&255)])<<16 | (t[512+((pix>>8)&255)])<<8);
                }
            }
        }
    }
}
```

```

U11 LOOP1, LOOPO;
U11 INIT1, INITO;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
for (top=0; top<RRANGE; top+=RMGRP) {
    U11 rtop0[NCHIP], dtop0[NCHIP];
    U11 rtop1[NCHIP], dtop1[NCHIP];
    U11 rtop2[NCHIP], dtop2[NCHIP];
    U11 rtop3[NCHIP], dtop3[NCHIP];
    U11 rtop4[NCHIP], dtop4[NCHIP];
    U11 rtop5[NCHIP], dtop5[NCHIP];
    U11 rtop6[NCHIP], dtop6[NCHIP];
    U11 rtop7[NCHIP], dtop7[NCHIP];
    U11 rtop8[NCHIP], dtop8[NCHIP];
    U11 rtop9[NCHIP], dtop9[NCHIP];
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    rtop0[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*0+top)*WD; dtop0[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*0+top)*WD;
    :
    rtop[CHIP] = r+(CHIP*RRANGE*OMAP+RRANGE*9+top)*WD; dtop9[CHIP] = d+(CHIP*RRANGE*OMAP+RRANGE*9+top)*WD;
}
//EMAXSA begin tone_curve.mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*2*/for (INIT1=1,LOOP1=RMGRP,rofs=0-WD*4; LOOP1--; INIT1=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
/*1*/for (INIT0=1,LOOP0=WD,rofs=0; LOOP0--; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000000fffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?WD*4:0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    exe(OP_ADD, &rofs, rofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000000fffffffLL, OP_NOP, OLL); /* stage#1 */
/*map0*/
mop(OP_LDWR, 1, &BR[2][1][1], (U11)rtop0[CHIP], pofs, MSK_D0, (U11)rtop0[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#2 */
mop(OP_LDBR, 1, &BR[3][1][1], (U11)t1, BR[2][1][1], MSK_B3, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#3 */
mop(OP_LDBR, 1, &BR[3][2][1], (U11)t2, BR[2][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#3 */
mop(OP_LDBR, 1, &BR[3][3][1], (U11)t3, BR[2][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#3 */
exe(OP_MMGR, &r1, BR[3][1][1], EXP_H3210, BR[3][2][1], EXP_H3210, BR[3][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
mop(OP_STWR, 3, &r1, (U11)dtop0[CHIP], pofs, MSK_D0, (U11)dtop0[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#3 */
:
/*map5*/
mop(OP_LDWR, 1, &BR[12][1][1], (U11)rtop5[CHIP], pofs, MSK_D0, (U11)rtop5[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#12 */
mop(OP_LDBR, 1, &BR[13][1][1], (U11)t1, BR[12][1][1], MSK_B3, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#13 */
mop(OP_LDBR, 1, &BR[13][2][1], (U11)t2, BR[12][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#13 */
mop(OP_LDBR, 1, &BR[13][3][1], (U11)t3, BR[12][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#13 */
exe(OP_MMGR, &r1, BR[13][1][1], EXP_H3210, BR[13][2][1], EXP_H3210, BR[13][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13 */
mop(OP_STWR, 3, &r1, (U11)dtop5[CHIP], pofs, MSK_D0, (U11)dtop5[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#13 */
/*map6*/
mop(OP_LDWR, 1, &BR[14][1][1], (U11)rtop6[CHIP], pofs, MSK_D0, (U11)rtop6[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#14 */
mop(OP_LDBR, 1, &BR[15][1][1], (U11)t1, BR[14][1][1], MSK_B3, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#15 */
mop(OP_LDBR, 1, &BR[15][2][1], (U11)t2, BR[14][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#15 */
mop(OP_LDBR, 1, &BR[15][3][1], (U11)t3, BR[14][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#15 */
exe(OP_MMGR, &r1, BR[15][1][1], EXP_H3210, BR[15][2][1], EXP_H3210, BR[15][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#15 */
mop(OP_STWR, 3, &r1, (U11)dtop6[CHIP], pofs, MSK_D0, (U11)dtop6[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#15 */
/*map7*/
mop(OP_LDWR, 1, &BR[16][1][1], (U11)rtop7[CHIP], pofs, MSK_D0, (U11)rtop7[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#16 */
mop(OP_LDBR, 1, &BR[17][1][1], (U11)t1, BR[16][1][1], MSK_B3, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#17 */
mop(OP_LDBR, 1, &BR[17][2][1], (U11)t2, BR[16][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#17 */
mop(OP_LDBR, 1, &BR[17][3][1], (U11)t3, BR[16][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#17 */
exe(OP_MMGR, &r1, BR[17][1][1], EXP_H3210, BR[17][2][1], EXP_H3210, BR[17][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#17 */
mop(OP_STWR, 3, &r1, (U11)dtop7[CHIP], pofs, MSK_D0, (U11)dtop7[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#17 */
/*map8*/
mop(OP_LDWR, 1, &BR[18][1][1], (U11)rtop8[CHIP], pofs, MSK_D0, (U11)rtop8[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#18 */
mop(OP_LDBR, 1, &BR[18][1][1], (U11)t1, BR[18][1][1], MSK_B3, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#19 */
mop(OP_LDBR, 1, &BR[18][2][1], (U11)t2, BR[18][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#19 */
mop(OP_LDBR, 1, &BR[18][3][1], (U11)t3, BR[18][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#19 */
exe(OP_MMGR, &r1, BR[18][1][1], EXP_H3210, BR[18][2][1], EXP_H3210, BR[18][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#19 */
mop(OP_STWR, 3, &r1, (U11)dtop8[CHIP], pofs, MSK_D0, (U11)dtop8[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#19 */
/*map9*/
mop(OP_LDWR, 1, &BR[20][1][1], (U11)rtop9[CHIP], pofs, MSK_D0, (U11)rtop9[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#20 */
mop(OP_LDBR, 1, &BR[21][1][1], (U11)t1, BR[20][1][1], MSK_B3, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#21 */
mop(OP_LDBR, 1, &BR[21][2][1], (U11)t2, BR[20][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#21 */
mop(OP_LDBR, 1, &BR[21][3][1], (U11)t3, BR[20][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#21 */
exe(OP_MMGR, &r1, BR[21][1][1], EXP_H3210, BR[21][2][1], EXP_H3210, BR[21][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#21 */
mop(OP_STWR, 3, &r1, (U11)dtop9[CHIP], pofs, MSK_D0, (U11)dtop9[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP); /* stage#21 */
}
}
//EMAXSA end
}
//EMAXSA drain_dirty_lmm

```

filter+rmm-tone_curve-emax6.obj

BR/row: max=7 min=1 ave=2 EA/row: max=3 min=0 ave=2 ARpass/row: max=0

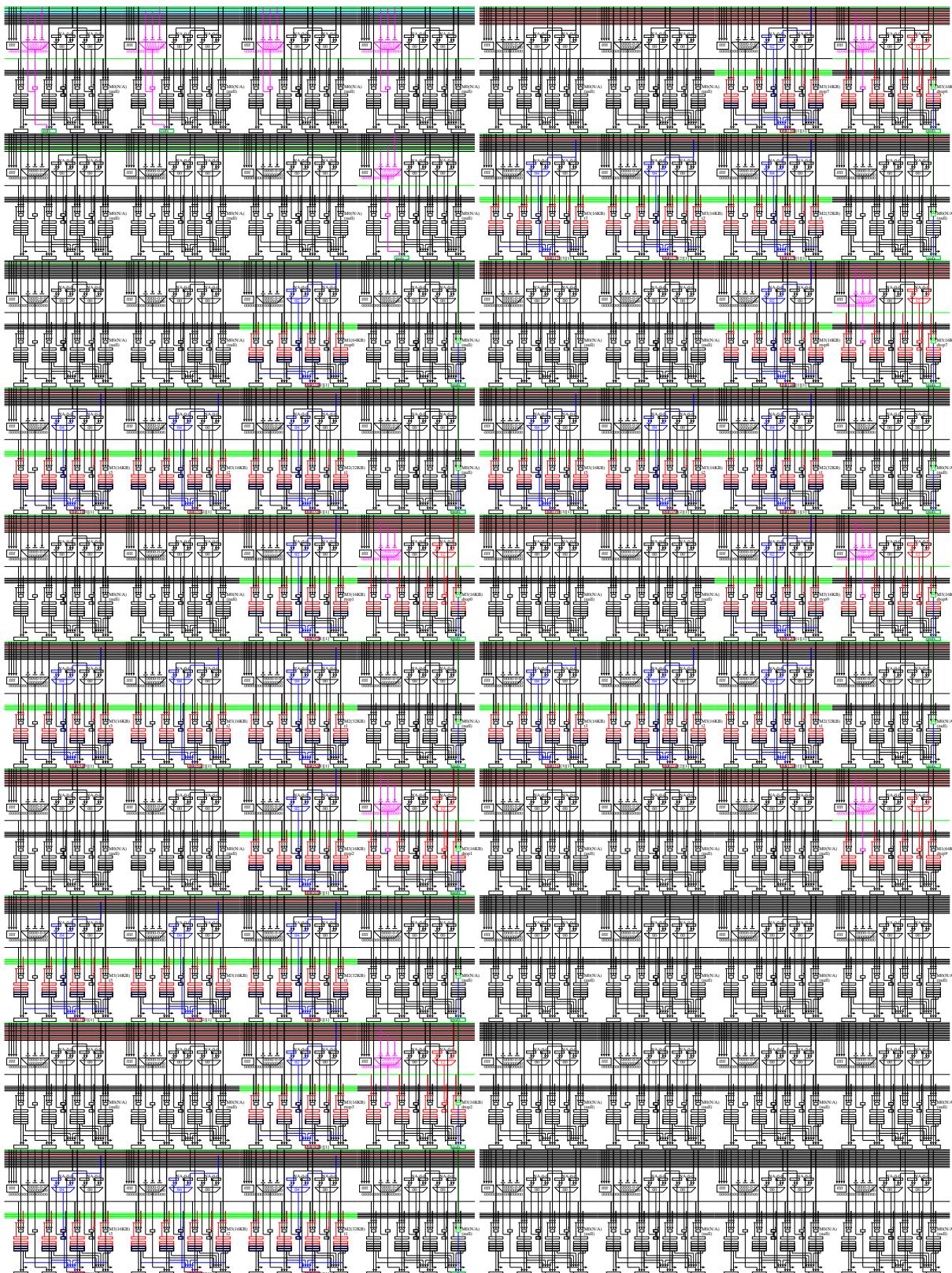


Figure.3.23: Tone curve

3.3.2 Hokan1 with stencil

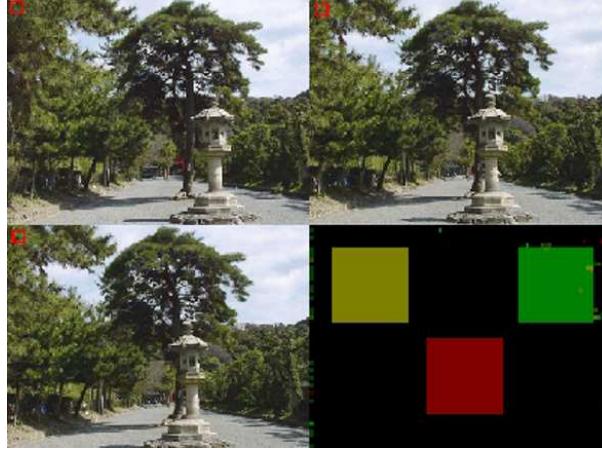


Figure 3.24: Hokan

フレーム補間の第1段階である。12x12の領域と4x4の領域のSAD値(4x4個)を求める。一度のバースト演算により8行分のSAD計算を行う(RMGRP=8)。ステンシル計算でありmapdist=7である。

```

void hokani(UInt *c, UInt *p, struct SAD1 *s) /* W, R, SAD1 */
#if !defined(EMAX5) && !defined(EMAX6)
for (topePAD; topHT-PAD; top++) { /* scan-lines */
    for (pofs=4; pofs<4; pofs++) {
        Ushort *t = s->SAD1[top/4][pofs+4];
        for (cofs=0; cofs<WD; cofs++) {
            int j = cofs/4;
            int k = cofs/4*2;
            UInt *c2 = c+top*WD;
            UInt *p2 = p+(top+pofs)*WD;
            *t += df(c2[j],p2[j+k-4]) + df(c2[j+1],p2[j+k-3]) + df(c2[j+2],p2[j+k-2]) + df(c2[j+3],p2[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
            *(t+1) += df(c2[j],p2[j+k-3]) + df(c2[j+1],p2[j+k-2]) + df(c2[j+2],p2[j+k-1]) + df(c2[j+3],p2[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
            t += 2;
        }
    }
#endif

for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+PAD+top+rofs;
            UInt *c0 = c+idx*WD;
            UInt *p0 = p+(idx-4)*WD;
            UInt *p1 = p+(idx-3)*WD;
            UInt *p2 = p+(idx-2)*WD;
            UInt *p3 = p+(idx-1)*WD;
            UInt *p4 = p+(idx+0)*WD;
            UInt *p5 = p+(idx+1)*WD;
            UInt *p6 = p+(idx+2)*WD;
            UInt *p7 = p+(idx+3)*WD;
            Ushort *t0 = s->SAD1[idx/4][0];
            Ushort *t1 = s->SAD1[idx/4][1];
            Ushort *t2 = s->SAD1[idx/4][2];
            Ushort *t3 = s->SAD1[idx/4][3];
            Ushort *t4 = s->SAD1[idx/4][4];
            Ushort *t5 = s->SAD1[idx/4][5];
            Ushort *t6 = s->SAD1[idx/4][6];
            Ushort *t7 = s->SAD1[idx/4][7];
            for (cofs=0; cofs<WD; cofs++) {
                int j = cofs/4;
                int k = cofs/4*2;
                *t0 += df(c0[j],p0[j+k-4]) + df(c0[j+1],p0[j+k-3]) + df(c0[j+2],p0[j+k-2]) + df(c0[j+3],p0[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t0+1) += df(c0[j],p0[j+k-3]) + df(c0[j+1],p0[j+k-2]) + df(c0[j+2],p0[j+k-1]) + df(c0[j+3],p0[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t0 += 2;
                *t1 += df(c0[j],p1[j+k-4]) + df(c0[j+1],p1[j+k-3]) + df(c0[j+2],p1[j+k-2]) + df(c0[j+3],p1[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t1+1) += df(c0[j],p1[j+k-3]) + df(c0[j+1],p1[j+k-2]) + df(c0[j+2],p1[j+k-1]) + df(c0[j+3],p1[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t1 += 2;
                *t2 += df(c0[j],p2[j+k-4]) + df(c0[j+1],p2[j+k-3]) + df(c0[j+2],p2[j+k-2]) + df(c0[j+3],p2[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t2+1) += df(c0[j],p2[j+k-3]) + df(c0[j+1],p2[j+k-2]) + df(c0[j+2],p2[j+k-1]) + df(c0[j+3],p2[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t2 += 2;
                *t3 += df(c0[j],p3[j+k-4]) + df(c0[j+1],p3[j+k-3]) + df(c0[j+2],p3[j+k-2]) + df(c0[j+3],p3[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t3+1) += df(c0[j],p3[j+k-3]) + df(c0[j+1],p3[j+k-2]) + df(c0[j+2],p3[j+k-1]) + df(c0[j+3],p3[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t3 += 2;
                *t4 += df(c0[j],p4[j+k-4]) + df(c0[j+1],p4[j+k-3]) + df(c0[j+2],p4[j+k-2]) + df(c0[j+3],p4[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t4+1) += df(c0[j],p4[j+k-3]) + df(c0[j+1],p4[j+k-2]) + df(c0[j+2],p4[j+k-1]) + df(c0[j+3],p4[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t4 += 2;
                *t5 += df(c0[j],p5[j+k-4]) + df(c0[j+1],p5[j+k-3]) + df(c0[j+2],p5[j+k-2]) + df(c0[j+3],p5[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t5+1) += df(c0[j],p5[j+k-3]) + df(c0[j+1],p5[j+k-2]) + df(c0[j+2],p5[j+k-1]) + df(c0[j+3],p5[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t5 += 2;
                *t6 += df(c0[j],p6[j+k-4]) + df(c0[j+1],p6[j+k-3]) + df(c0[j+2],p6[j+k-2]) + df(c0[j+3],p6[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t6+1) += df(c0[j],p6[j+k-3]) + df(c0[j+1],p6[j+k-2]) + df(c0[j+2],p6[j+k-1]) + df(c0[j+3],p6[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t6 += 2;
                *t7 += df(c0[j],p7[j+k-4]) + df(c0[j+1],p7[j+k-3]) + df(c0[j+2],p7[j+k-2]) + df(c0[j+3],p7[j+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
                *(t7+1) += df(c0[j],p7[j+k-3]) + df(c0[j+1],p7[j+k-2]) + df(c0[j+2],p7[j+k-1]) + df(c0[j+3],p7[j+k]); /* p[-3],p[-2],p[-1],p[0] -> p[-1],p[0],p[1],p[2] */
                t7 += 2;
            }
        }
    }
}

```

```

U11 LOOP1, LOOPO;
U11 INIT1, INITO;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc1, cc2, cc3, ex0, ex1;
for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs++) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        U11 jw, kw;
        Uint *c0[NCHIP];
        UInt *p0[NCHIP], *p1[NCHIP], *p2[NCHIP], *p3[NCHIP], *p4[NCHIP], *p5[NCHIP], *p6[NCHIP], *p7[NCHIP];
        Ushort *t0[NCHIP], *t1[NCHIP], *t2[NCHIP], *t3[NCHIP], *t4[NCHIP], *t5[NCHIP], *t6[NCHIP], *t7[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            int idx = CHIP*RRANGE+PAD+top+rofs;
            c0[CHIP] = c + idx;
            /*WD;
            p0[CHIP] = p+(idx-4)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p1[CHIP] = p+(idx-3)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p2[CHIP] = p+(idx-2)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p3[CHIP] = p+(idx-1)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p4[CHIP] = p+(idx+0)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p5[CHIP] = p+(idx+1)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p6[CHIP] = p+(idx+2)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            p7[CHIP] = p+(idx+3)*WD; /* jk: 0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
            t0[CHIP] = s->SAD1[idx/4][0]; /* SAD1[HT4/1][8][WD/4][8] ... [8][WD/4][8]=5120(2B) 0x1400 */
            t1[CHIP] = s->SAD1[idx/4][1];
            t2[CHIP] = s->SAD1[idx/4][2];
            t3[CHIP] = s->SAD1[idx/4][3];
            t4[CHIP] = s->SAD1[idx/4][4];
            t5[CHIP] = s->SAD1[idx/4][5];
            t6[CHIP] = s->SAD1[idx/4][6];
            t7[CHIP] = s->SAD1[idx/4][7];
        }
    }
//EMAX5A begin hokan1 mapdist7
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*1*/for ((INITO=1,LOOP0=WD,cofs=0-4; LOOP0--; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
/*@0,1*/ exe(OP_ADD, &cofs, INITO?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL); /* stage#0 */
/*int j = cofc/4 */4*4/*4*4*/
/*int k = cofc/4 */4*4/*4*/
/*@0,1*/0/* exe(OP_NOP, &jw,
/*@0,1*/1/* exe(OP_NOP, &kw,
cofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 12LL, OP_SLL, OLL);
/*k*=4*/
/*@0,2,0*/0/* exe(OP_ADD, &r12,
cofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, ~15LL, OP_SLL, OLL);
/*@0,2,1*/0/* exe(OP_ADD3, &r13,
cofs, EXP_H3210, jw, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,3,*/*0/* m0p(OP_LDWR, 1, kr0,
r12, OLL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,3,1/*0/* m0p(OP_LDWR, 1, kr1,
r12, 4LL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,3,2/*0/* m0p(OP_LDWR, 1, kr2,
r12, 8LL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,3,3/*0/* m0p(OP_LDWR, 1, kr3,
r12, 12LL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,4,0/*0/* m0p(OP_LDWR, 1, kB8[4][0][1],
r13, -16LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,4,1/*0/* m0p(OP_LDWR, 1, kr25,
r13, -12LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,4,2/*0/* m0p(OP_LDWR, 1, kr26,
r13, -8LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,4,3/*0/* m0p(OP_LDWR, 1, kr27,
r13, -4LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,4,3/*1/*0/* m0p(OP_LDWR, 1, kr28,
r13, OLL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,5,0/*0/* exe(OP_MSSAD, &r11,
OLL, EXP_H3210, r0, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,5,1/*0/* exe(OP_MSSAD, &r13,
OLL, EXP_H3210, r1, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,5,2/*0/* exe(OP_MSSAD, &r15,
OLL, EXP_H3210, r2, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,5,3/*0/* exe(OP_MSSAD, &r17,
OLL, EXP_H3210, r3, EXP_H3210, r28, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,6,0/*0/* exe(OP_MSSAD, &r10,
OLL, EXP_H3210, r0, EXP_H3210, r1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,6,1/*0/* exe(OP_MSSAD, &r12,
OLL, EXP_H3210, r1, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,6,2/*0/* exe(OP_MSSAD, &r14,
OLL, EXP_H3210, r2, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,6,3/*0/* exe(OP_MSSAD, &r16,
OLL, EXP_H3210, r3, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,7,0/*0/* exe(OP_MAUV,
r10, EXP_H3210, r12, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,7,1/*0/* exe(OP_MAUV,
r11, EXP_H3210, r13, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,7,2/*0/* exe(OP_MAUV,
r14, EXP_H3210, r16, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,7,3/*0/* exe(OP_MAUV,
r15, EXP_H3210, r17, EXP_H3210, r28, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,8,0/*0/* exe(OP_MAUV,
r10, EXP_H3210, r20, EXP_H3210, r29, EXP_H3210, OP_SUMMH, OLL, OP_NOP, OLL);
/*@0,8,1/*0/* exe(OP_MAUV,
r11, EXP_H3210, r21, EXP_H3210, r25, EXP_H3210, OLL, EXP_H3210, OP_SUMMH, OLL, OP_NOP, OLL);
/*@0,9,0/*0/* m0p(OP_LDWR, 1, kB8[9][0][1],
r10, EXP_H3210, cofs, MSK_DO, (U11)t0[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@0,9,1/*0/* exe(OP_MAUV3,
r11, EXP_H3210, r10, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,9,0/*0/* m0p(OP_STWR, 3, &R8[9][0],
cofs, t0[CHIP], MSK_DO, (U11)t0[CHIP], WD, 0, 1, (U11)NULL, WD);
/*k*=3*/
/*@0,51,2/*0/* exe(OP_ADD,
r12, OLL, EXP_H3210, jw, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,51,3/*0/* exe(OP_ADD3,
r13, OLL, EXP_H3210, jw, EXP_H3210, kw, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,52,0/*0/* m0p(OP_LDWR, 1, kr0,
r12, OLL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,52,1/*0/* m0p(OP_LDWR, 1, kr1,
r12, 4LL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,52,2/*0/* m0p(OP_LDWR, 1, kr2,
r12, 8LL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,52,3/*0/* m0p(OP_LDWR, 1, kr3,
r12, 12LL, MSK_DO, (U11)c0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,53,0/*0/* m0p(OP_LDWR, 1, kB8[53][0][1],
r13, -16LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,53,1/*0/* m0p(OP_LDWR, 1, kr25,
r13, -12LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,53,2/*0/* m0p(OP_LDWR, 1, kr26,
r13, -8LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,53,3/*0/* m0p(OP_LDWR, 1, kr27,
r13, -4LL, MSK_DO, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@0,54,0/*0/* exe(OP_MSSAD,
r11, OLL, EXP_H3210, r0, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,54,1/*0/* exe(OP_MSSAD,
r13, OLL, EXP_H3210, r1, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,54,2/*0/* exe(OP_MSSAD,
r15, OLL, EXP_H3210, r2, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,54,3/*0/* exe(OP_MSSAD,
r17, OLL, EXP_H3210, r3, EXP_H3210, r28, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,55,0/*0/* exe(OP_MSSAD,
r10, OLL, EXP_H3210, r0, EXP_H3210, r1, EXP_H3210, BR[53][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,55,1/*0/* exe(OP_MSSAD,
r12, OLL, EXP_H3210, r1, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,55,2/*0/* exe(OP_MSSAD,
r14, OLL, EXP_H3210, r2, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,55,3/*0/* exe(OP_MSSAD,
r16, OLL, EXP_H3210, r3, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,56,0/*0/* exe(OP_MAUV,
r10, EXP_H3210, r12, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,56,1/*0/* exe(OP_MAUV,
r11, EXP_H3210, r13, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,56,2/*0/* exe(OP_MAUV,
r14, EXP_H3210, r16, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,56,3/*0/* exe(OP_MAUV,
r15, EXP_H3210, r17, EXP_H3210, r28, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,57,0/*0/* exe(OP_MAUV,
r10, EXP_H3210, r20, EXP_H3210, r29, EXP_H3210, OP_SUMMH, OLL, OP_NOP, OLL);
/*@0,57,1/*0/* exe(OP_MAUV,
r11, EXP_H3210, r21, EXP_H3210, r25, EXP_H3210, OLL, EXP_H3210, OP_SUMMH, OLL, OP_NOP, OLL);
/*@0,58,0/*0/* m0p(OP_LDWR, 1, kB8[58][0][1],
r10, EXP_H3210, cofs, MSK_DO, (U11)t7[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@0,58,1/*0/* exe(OP_MAUV3,
r11, EXP_H3210, r10, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@0,58,0/*0/* m0p(OP_STWR, 3, &R8[58][0],
cofs, t7[CHIP], MSK_DO, (U11)t7[CHIP], WD, 0, 1, (U11)NULL, WD);
}
}
//EMAX5A end
}
}
//EMAX5A drain_dirty_lmm

```

filter+rmm-hokan1-emax6.obj

BR/row: max=16 min=1 ave=9 EA/row: max=5 min=0 ave=1 ARpass/row: max=0

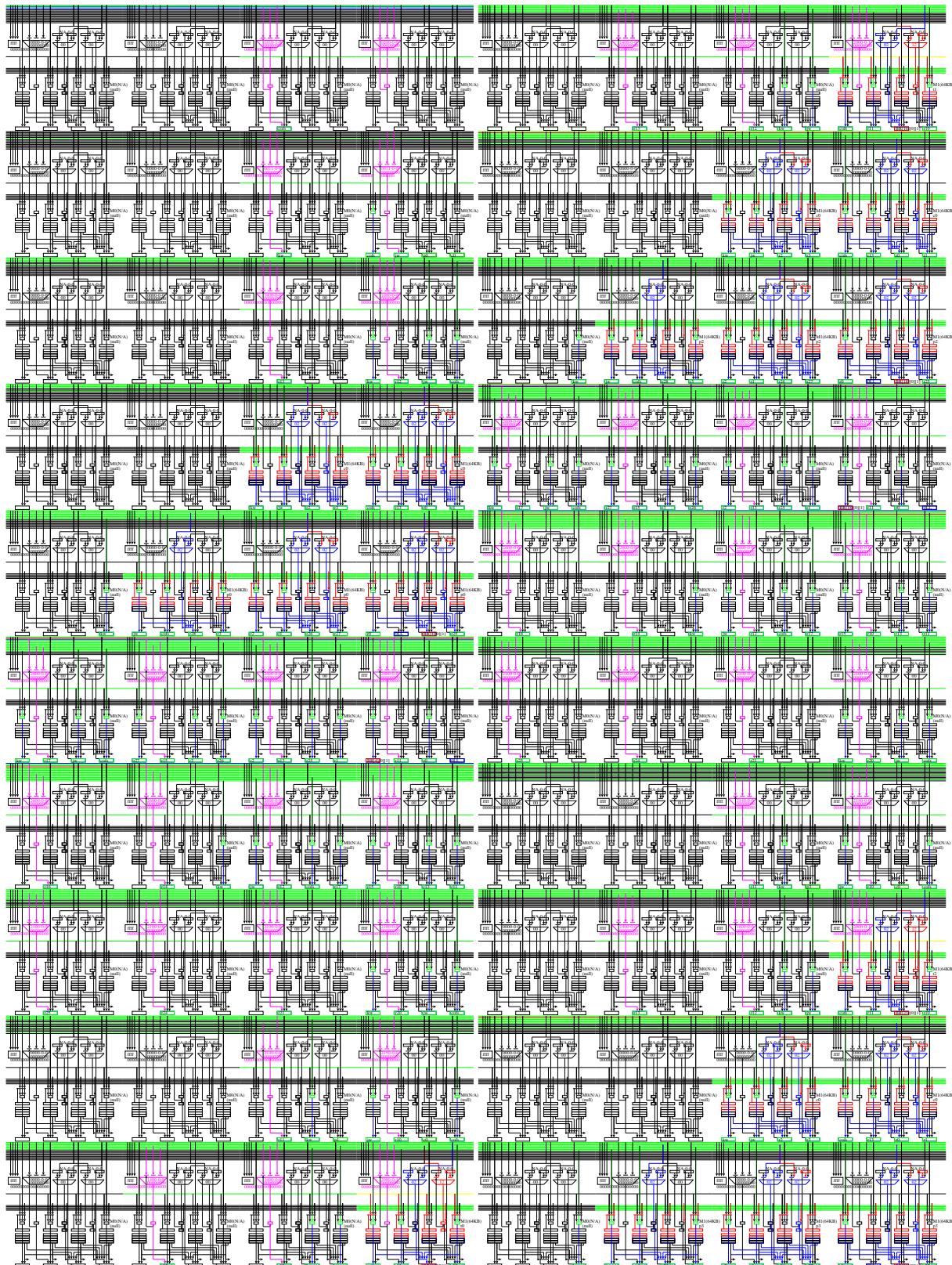


Figure.3.25: Hokan1

3.3.3 Hokan2 with stencil

フレーム補間の第2段階である。4x4個のSAD値から最も小さいSAD値を探し、対応するxy相対座標を求める。一度のバースト演算により12行分の座標計算を行う(RMGRP=12)。ステンシル計算ではなくmapdist=0である。

```
void hokan2(struct SAD1 *s, Uint *minxy) /* [WD/4][8] */
{
    #if !defined(EMAX5) && !defined(EMAX6)
        for (top=PAD; top<HT-PAD; top+=4) { /* scan-lines */
            Uint *xy = minxy+top*WD;
            for (pofs=4; pofs<4; pofs++) {
                Ushort *t = s->SAD1[top/4][pofs+4];
                int idx = ((pofs/2)&0xff)<<16;
                for (cofs=0; cofs<WD; cofs++) { /* j%4==0 の時のみ minxy[j] に有効値。他はゴミ */
                    int 11 = ((-2)<<24)|idx*(t );
                    if ((xy[cofs]&HM) > *(t )) xy[cofs] = 11;
                    int 12 = ((-1)<<24)|idx*(t+1); if ((xy[cofs]&HM) > *(t+1)) xy[cofs] = 12;
                    int 13 = ((-1)<<24)|idx*(t+2); if ((xy[cofs]&HM) > *(t+2)) xy[cofs] = 13;
                    int 14 = (( 0)<<24)|idx*(t+3); if ((xy[cofs]&HM) > *(t+3)) xy[cofs] = 14;
                    int 15 = (( 0)<<24)|idx*(t+4); if ((xy[cofs]&HM) > *(t+4)) xy[cofs] = 15;
                    int 16 = (( 0)<<24)|idx*(t+5); if ((xy[cofs]&HM) > *(t+5)) xy[cofs] = 16;
                    int 17 = (( 1)<<24)|idx*(t+6); if ((xy[cofs]&HM) > *(t+6)) xy[cofs] = 17;
                    int 18 = (( 1)<<24)|idx*(t+7); if ((xy[cofs]&HM) > *(t+7)) xy[cofs] = 18;
                    t += 2;
                }
            }
        }
    #endif
}
```

```
Uint ix0 = ((-4/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix1 = ((-3/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix2 = ((-2/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix3 = ((-1/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix4 = (( 0/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix5 = ((+1/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix6 = ((+2/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
Uint ix7 = ((+3/2)&0xff)<<16; /* -2,-1,-1,0,0,0,1,1 */
for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs+=4) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP+=4) { /* will be parallelized by multi-chip (M/#chip) */
            Ushort *t0 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][0];
            Ushort *t1 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][1];
            Ushort *t2 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][2];
            Ushort *t3 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][3];
            Ushort *t4 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][4];
            Ushort *t5 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][5];
            Ushort *t6 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][6];
            Ushort *t7 = s->SAD1[[CHIP*RRANGE+top+rofs]/4][7];
            Uint *xy = minxy+((CHIP*RRANGE+top+rofs)*WD);
            for (cofs=0; cofs<WD; cofs++) { /* j%4==0 の時のみ minxy[j] に有効値。他はゴミ */
                int 11, 12, 13, 14, 15, 16, 17, 18;
                11=(((-2)<<24)|ix0|*(t0 )); if((xy[cofs]&HM)**(t0 ))xy[cofs]=11;
                12=(((-1)<<24)|ix0|*(t0+1)); if((xy[cofs]&HM)**(t0+1))xy[cofs]=12;
                13=(((-1)<<24)|ix0|*(t0+2)); if((xy[cofs]&HM)**(t0+2))xy[cofs]=13;
                14= ix0|*(t0+3); if((xy[cofs]&HM)**(t0+3))xy[cofs]=14;
                15= ix0|*(t0+4); if((xy[cofs]&HM)**(t0+4))xy[cofs]=15;
                16= ix0|*(t0+5); if((xy[cofs]&HM)**(t0+5))xy[cofs]=16;
                17=(( 1)<<24)|ix0|*(t0+6); if((xy[cofs]&HM)**(t0+6))xy[cofs]=17;
                18=(( 1)<<24)|ix0|*(t0+7); if((xy[cofs]&HM)**(t0+7))xy[cofs]=18;
                t0 += 2;
                11=(((-2)<<24)|ix1|*(t1 )); if((xy[cofs]&HM)**(t1 ))xy[cofs]=11;
                12=(((-1)<<24)|ix1|*(t1+1)); if((xy[cofs]&HM)**(t1+1))xy[cofs]=12;
                13=(((-1)<<24)|ix1|*(t1+2)); if((xy[cofs]&HM)**(t1+2))xy[cofs]=13;
                14= ix1|*(t1+3); if((xy[cofs]&HM)**(t1+3))xy[cofs]=14;
                15= ix1|*(t1+4); if((xy[cofs]&HM)**(t1+4))xy[cofs]=15;
                16= ix1|*(t1+5); if((xy[cofs]&HM)**(t1+5))xy[cofs]=16;
                17=(( 1)<<24)|ix1|*(t1+6); if((xy[cofs]&HM)**(t1+6))xy[cofs]=17;
                18=(( 1)<<24)|ix1|*(t1+7); if((xy[cofs]&HM)**(t1+7))xy[cofs]=18;
                :
                t5 += 2;
                11=(((-2)<<24)|ix6|*(t6 )); if((xy[cofs]&HM)**(t6 ))xy[cofs]=11;
                12=(((-1)<<24)|ix6|*(t6+1)); if((xy[cofs]&HM)**(t6+1))xy[cofs]=12;
                13=(((-1)<<24)|ix6|*(t6+2)); if((xy[cofs]&HM)**(t6+2))xy[cofs]=13;
                14= ix6|*(t6+3); if((xy[cofs]&HM)**(t6+3))xy[cofs]=14;
                15= ix6|*(t6+4); if((xy[cofs]&HM)**(t6+4))xy[cofs]=15;
                16= ix6|*(t6+5); if((xy[cofs]&HM)**(t6+5))xy[cofs]=16;
                17=(( 1)<<24)|ix6|*(t6+6); if((xy[cofs]&HM)**(t6+6))xy[cofs]=17;
                18=(( 1)<<24)|ix6|*(t6+7); if((xy[cofs]&HM)**(t6+7))xy[cofs]=18;
                t6 += 2;
                11=(((-2)<<24)|ix7|*(t7 )); if((xy[cofs]&HM)**(t7 ))xy[cofs]=11;
                12=(((-1)<<24)|ix7|*(t7+1)); if((xy[cofs]&HM)**(t7+1))xy[cofs]=12;
                13=(((-1)<<24)|ix7|*(t7+2)); if((xy[cofs]&HM)**(t7+2))xy[cofs]=13;
                14= ix7|*(t7+3); if((xy[cofs]&HM)**(t7+3))xy[cofs]=14;
                15= ix7|*(t7+4); if((xy[cofs]&HM)**(t7+4))xy[cofs]=15;
                16= ix7|*(t7+5); if((xy[cofs]&HM)**(t7+5))xy[cofs]=16;
                17=(( 1)<<24)|ix7|*(t7+6); if((xy[cofs]&HM)**(t7+6))xy[cofs]=17;
                18=(( 1)<<24)|ix7|*(t7+7); if((xy[cofs]&HM)**(t7+7))xy[cofs]=18;
                t7 += 2;
            }
        }
    }
}
```

```

U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
U11 ix0 = ((-4/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix1 = ((-3/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix2 = ((-2/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix3 = ((-1/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix4 = (( 0/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix5 = ((+1/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix6 = ((+2/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
U11 ix7 = ((+3/2)&0xffff)<<16; /* -2,-1,-1,0,0,0,1,1 */
for (top0=0; top<RRANGE; top+=RMRGP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs+=4) { /* will be parallelized by multi-chip (M/#chip) */
        Uint *xy[NCHIP];
        Uint *t00[NCHIP], *t10[NCHIP], *t20[NCHIP], *t30[NCHIP], *t40[NCHIP], *t50[NCHIP], *t60[NCHIP], *t70[NCHIP];
        Uint *t01[NCHIP], *t11[NCHIP], *t21[NCHIP], *t31[NCHIP], *t41[NCHIP], *t51[NCHIP], *t61[NCHIP], *t71[NCHIP];
        Uint *t02[NCHIP], *t12[NCHIP], *t22[NCHIP], *t32[NCHIP], *t42[NCHIP], *t52[NCHIP], *t62[NCHIP], *t72[NCHIP];
        Uint *t03[NCHIP], *t13[NCHIP], *t23[NCHIP], *t33[NCHIP], *t43[NCHIP], *t53[NCHIP], *t63[NCHIP], *t73[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+PAD+top+rofs;
            t00[CHIP] = s->SAD1[idx/4][0]; t01[CHIP] = t00[CHIP]+1; t02[CHIP] = t00[CHIP]+2; t03[CHIP] = t00[CHIP]+3;
            t10[CHIP] = s->SAD1[idx/4][1]; t11[CHIP] = t10[CHIP]+1; t12[CHIP] = t10[CHIP]+2; t13[CHIP] = t10[CHIP]+3;
            t20[CHIP] = s->SAD1[idx/4][2]; t21[CHIP] = t20[CHIP]+1; t22[CHIP] = t20[CHIP]+2; t23[CHIP] = t20[CHIP]+3;
            t30[CHIP] = s->SAD1[idx/4][3]; t31[CHIP] = t30[CHIP]+1; t32[CHIP] = t30[CHIP]+2; t33[CHIP] = t30[CHIP]+3;
            t40[CHIP] = s->SAD1[idx/4][4]; t41[CHIP] = t40[CHIP]+1; t42[CHIP] = t40[CHIP]+2; t43[CHIP] = t40[CHIP]+3;
            t50[CHIP] = s->SAD1[idx/4][5]; t51[CHIP] = t50[CHIP]+1; t52[CHIP] = t50[CHIP]+2; t53[CHIP] = t50[CHIP]+3;
            t60[CHIP] = s->SAD1[idx/4][6]; t61[CHIP] = t60[CHIP]+1; t62[CHIP] = t60[CHIP]+2; t63[CHIP] = t60[CHIP]+3;
            t70[CHIP] = s->SAD1[idx/4][7]; t71[CHIP] = t70[CHIP]+1; t72[CHIP] = t70[CHIP]+2; t73[CHIP] = t70[CHIP]+3;
            xy[CHIP] = minxy+idx*WD;
        }
    }
//EMAX5A begin hokan2 mapdist=0
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    /*1*/for (INIT0=1,LOOP0=WD,cofs=0-4; LOOP0=0; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
        /*@0,1*/ exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
        /*@k=4*/
        /*@01,0*/ mop(OP_LDWR, 1, &r10, t00[CHIP], cofs, MSK_D0, t00[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@01,0*/ exe(OP_NOP, &r28, (-2LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix0, OP_NOP, OLL);
        /*@01,0*/ mop(OP_LDWR, 1, &r12, t01[CHIP], cofs, MSK_D0, t01[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@01,1*/ exe(OP_LDWR, 1, &r29, t02[CHIP], cofs, MSK_D0, t02[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@01,1*/ mop(OP_LDWR, 1, &r14, t03[CHIP], cofs, MSK_D0, t03[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@01,2*/ exe(OP_NOP, &r31, (-1LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix0, OP_NOP, OLL);
        /*@01,2*/ mop(OP_LDWR, 1, &r16, t04[CHIP], cofs, MSK_D0, t04[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@01,2*/ exe(OP_MINL3, &r16, t05[CHIP], cofs, MSK_D0, t05[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@02,0*/ exe(OP_MINL3, &r16, t06[CHIP], cofs, MSK_D0, t06[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@02,1*/ exe(OP_MINL3, &r12, t07[CHIP], cofs, MSK_D0, t07[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@02,2*/ exe(OP_MINL3, &r14, t08[CHIP], cofs, MSK_D0, t08[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@02,3*/ exe(OP_MINL3, &r16, t09[CHIP], cofs, MSK_D0, t09[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@03,0*/ exe(OP_MINL, &r20, t10[CHIP], cofs, MSK_D0, t10[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@03,1*/ exe(OP_MINL, &r24, t11[CHIP], cofs, MSK_D0, t11[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,0*/ exe(OP_MINL, &r20, t12[CHIP], cofs, MSK_D0, t12[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,1*/ exe(OP_MINL, &r24, t13[CHIP], cofs, MSK_D0, t13[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,2*/ exe(OP_MINL, &r31, t14[CHIP], cofs, MSK_D0, t14[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,3*/ exe(OP_MINL, &r31, t15[CHIP], cofs, MSK_D0, t15[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,0*/ exe(OP_MINL3, &r16, t16[CHIP], cofs, MSK_D0, t16[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,1*/ exe(OP_MINL3, &r12, t17[CHIP], cofs, MSK_D0, t17[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,2*/ exe(OP_MINL3, &r14, t18[CHIP], cofs, MSK_D0, t18[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,3*/ exe(OP_MINL3, &r16, t19[CHIP], cofs, MSK_D0, t19[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@06,0*/ exe(OP_MINL, &r20, t20[CHIP], cofs, MSK_D0, t20[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@06,1*/ exe(OP_MINL, &r24, t21[CHIP], cofs, MSK_D0, t21[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@07,0*/ exe(OP_MINL, &r24, t22[CHIP], cofs, MSK_D0, t22[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@08,0*/ exe(OP_MINL, &r20, t23[CHIP], cofs, MSK_D0, t23[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@k=2*/
        /*@024,0*/ mop(OP_LDWR, 1, &r10, t10[CHIP], cofs, MSK_D0, t10[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,1*/ exe(OP_NOP, &r28, (-2LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix1, OP_NOP, OLL);
        /*@04,1*/ mop(OP_LDWR, 1, &r12, t11[CHIP], cofs, MSK_D0, t11[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,2*/ exe(OP_NOP, &r29, t12[CHIP], cofs, MSK_D0, t12[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,2*/ mop(OP_LDWR, 1, &r14, t13[CHIP], cofs, MSK_D0, t13[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@04,3*/ mop(OP_LDWR, 1, &r16, t14[CHIP], cofs, MSK_D0, t14[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,0*/ exe(OP_MINL3, &r10, t15[CHIP], cofs, MSK_D0, t15[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,1*/ exe(OP_MINL3, &r12, t16[CHIP], cofs, MSK_D0, t16[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,2*/ exe(OP_MINL3, &r14, t17[CHIP], cofs, MSK_D0, t17[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@05,3*/ exe(OP_MINL3, &r16, t18[CHIP], cofs, MSK_D0, t18[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@06,0*/ exe(OP_MINL, &r20, t19[CHIP], cofs, MSK_D0, t19[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@06,1*/ exe(OP_MINL, &r24, t20[CHIP], cofs, MSK_D0, t20[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@07,0*/ exe(OP_MINL, &r24, t21[CHIP], cofs, MSK_D0, t21[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@08,0*/ exe(OP_MINL, &r20, t22[CHIP], cofs, MSK_D0, t22[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@k=3*/
        /*@028,0*/ mop(OP_LDWR, 1, &r10, t20[CHIP], cofs, MSK_D0, t20[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,1*/ exe(OP_NOP, &r28, (-2LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix6, OP_NOP, OLL);
        /*@028,1*/ mop(OP_LDWR, 1, &r12, t21[CHIP], cofs, MSK_D0, t21[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,2*/ exe(OP_NOP, &r29, t22[CHIP], cofs, MSK_D0, t22[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,2*/ mop(OP_LDWR, 1, &r14, t23[CHIP], cofs, MSK_D0, t23[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,3*/ exe(OP_NOP, &r31, t24[CHIP], cofs, MSK_D0, t24[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,4*/ mop(OP_MINL3, &r10, t25[CHIP], cofs, MSK_D0, t25[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@025,1*/ exe(OP_MINL3, &r12, t26[CHIP], cofs, MSK_D0, t26[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@025,2*/ exe(OP_MINL3, &r14, t27[CHIP], cofs, MSK_D0, t27[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@025,3*/ exe(OP_MINL3, &r16, t28[CHIP], cofs, MSK_D0, t28[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@026,0*/ exe(OP_MINL, &r20, t29[CHIP], cofs, MSK_D0, t29[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@026,1*/ exe(OP_MINL, &r24, t30[CHIP], cofs, MSK_D0, t30[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@027,0*/ exe(OP_MINL, &r24, t31[CHIP], cofs, MSK_D0, t31[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,0*/ exe(OP_MINL, &r20, t32[CHIP], cofs, MSK_D0, t32[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@k=2*/
        /*@028,0*/ mop(OP_LDWR, 1, &r10, t30[CHIP], cofs, MSK_D0, t30[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,1*/ exe(OP_NOP, &r28, (-2LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix7, OP_NOP, OLL);
        /*@028,1*/ mop(OP_LDWR, 1, &r12, t31[CHIP], cofs, MSK_D0, t31[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,2*/ exe(OP_NOP, &r29, t32[CHIP], cofs, MSK_D0, t32[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,2*/ mop(OP_LDWR, 1, &r14, t33[CHIP], cofs, MSK_D0, t33[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,3*/ exe(OP_NOP, &r31, t34[CHIP], cofs, MSK_D0, t34[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@028,4*/ mop(OP_MINL3, &r10, t35[CHIP], cofs, MSK_D0, t35[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@029,1*/ exe(OP_MINL3, &r12, t36[CHIP], cofs, MSK_D0, t36[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@029,2*/ exe(OP_MINL3, &r14, t37[CHIP], cofs, MSK_D0, t37[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@029,3*/ exe(OP_MINL3, &r16, t38[CHIP], cofs, MSK_D0, t38[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@030,0*/ exe(OP_MINL, &r20, t39[CHIP], cofs, MSK_D0, t39[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@030,1*/ exe(OP_MINL, &r24, t40[CHIP], cofs, MSK_D0, t40[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@031,0*/ exe(OP_MINL, &r24, t41[CHIP], cofs, MSK_D0, t41[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@032,0*/ exe(OP_MINL, &r20, t42[CHIP], cofs, MSK_D0, t42[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@033,0*/ mop(OP_LDWR, 1, &AR[33][0][1], xy[CHIP], cofs, MSK_D0, xy[CHIP], WD, 0, 1, (U11)NULL, WD);
        /*@033,0*/ exe(OP_MINL, &AR[33][0][1], r0, EXP_H3210, BR[33][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@033,0*/ mop(OP_STWR, 3, &AR[33][0], cofs, xy[CHIP], MSK_D0, xy[CHIP], WD, 0, 1, (U11)NULL, WD);
    }
}
//EMAX5A end
}
}
//EMAX5A drain_dirty_lmm

```

filter+rmm-hokan2-emax6.obj

BR/row: max=10 min=2 ave=6 EA/row: max=4 min=0 ave=1 ARpass/row: max=0

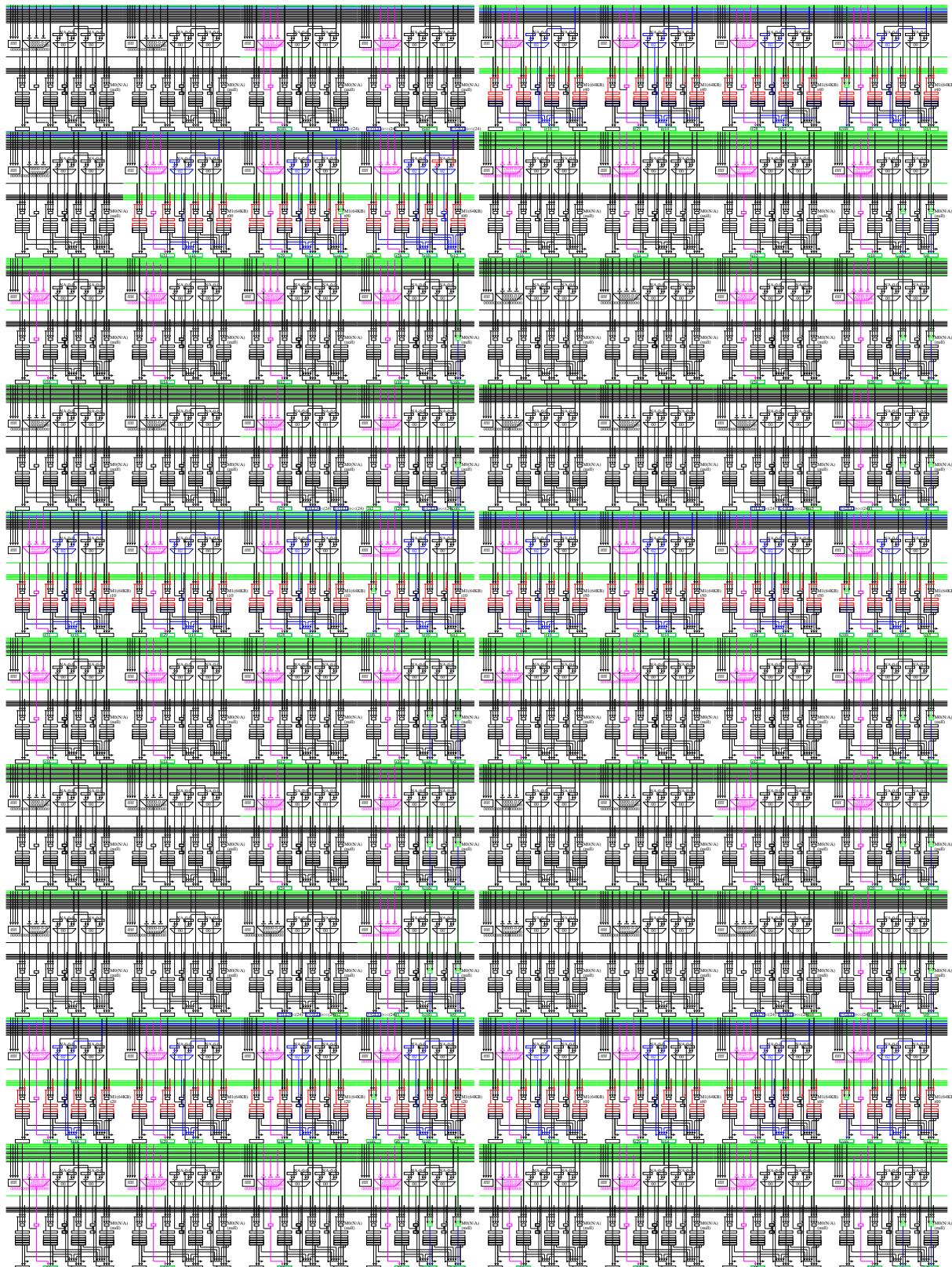


Figure.3.26: Hokan2

3.3.4 Hokan3 with stencil

フレーム補間の第3段階である。類似度が高い方向へのxy相対座標を元に、参照画像を貼り付ける。一度のバースト演算により12行分の貼り付け処理を行う(RMGRP=12)。ステンシル計算ではなくmapdist=0である。

```
void hokan3(UInt *minxy, UInt *r, UInt *d)
{
    #if !defined(EMAX5) && !defined(EMAX6)
    for (top=PAD; top<HT-PAD; top++) { /* scan-lines */
        UInt *xy = minxy+(top/4*4)*WD;
        UInt *dp = d+top*WD;
        for (pofs=-2; pofs<2; pofs++) {
            UInt *rp = r+(top+pofs)*WD;
            for (cofs=0; cofs<WD; cofs++) {
                int x = (int) xy[cofs/4*4]>>24;
                int y = (int)(xy[cofs/4*4]<<8)>>24;
                if (y == pofs) dp[cofs] = rp[cofs+x];
            }
        }
    }
    #endif
}
```

```
for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+top+rofs;
            UInt *xy = minxy+(idx/4*4)*WD;
            UInt *dp = d+idx*WD;
            UInt *rp0 = r+(idx-2)*WD;
            UInt *rp1 = r+(idx-1)*WD;
            UInt *rp2 = r+(idx+0)*WD;
            UInt *rp3 = r+(idx+1)*WD;
            for (cofs=0; cofs<WD; cofs++) {
                int x = (int) xy[cofs/4*4]>>24;
                int y = (int)(xy[cofs/4*4]<<8)>>24;
                dp[cofs] = (y == -2)?rp0[cofs+x]:
                (y == -1)?rp1[cofs+x]:
                (y == 0)?rp2[cofs+x]:
                (y == 1)?rp3[cofs+x]:0;
            }
        }
    }
}
```

```
U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        U11 jw;
        UInt *xy[NCHIP], *dp[NCHIP], *rp0[NCHIP], *rp1[NCHIP], *rp2[NCHIP], *rp3[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+top+rofs;
            xy[CHIP] = minxy+(idx/4*4)*WD;
            dp[CHIP] = d+idx*WD;
            rp0[CHIP] = r+(idx-2)*WD;
            rp1[CHIP] = r+(idx-1)*WD;
            rp2[CHIP] = r+(idx+0)*WD;
            rp3[CHIP] = r+(idx+1)*WD;
        }
    }
//EMAX5A begin hokan3 mapdist=0
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*1*/for (INIT0=1,LOOP0=WD,cofs=0-4; LOOP0--; INIT0--) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
/*@0,1*/ exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
/*@1,0*/ exe(OP_NOP, &jw, cofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, "15LL", OP_SLL, OLL);

/*@2,0*/ mop(OP_LDWR, 1, &r10, xy[CHIP], jw, MSK_D0, xy[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,0*/ exe(OP_NOP, &r10, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xf0000000LL, OP_SRRA, 22LL); /*x*/
/*@3,1*/ exe(OP_NOP, &r3, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0ff00000LL, OP_SRAB, 16LL); /*y*/
/*@4,0*/ exe(OP_ADD, &r4, r2, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

/*@5,0*/ mop(OP_LDWR, 1, &r10, rp0[CHIP], r4, MSK_D0, rp0[CHIP], WD, 0, 0, (U11)NULL, WD); /*rp0[cofs+x]*/
/*@5,1*/ exe(OP_NOP, &r2, r3, EXP_H3210, -2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /*y== -2*/
/*@6,0*/ exe(OP_CMV, &r0, r5, EXP_H3210, r10, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

/*@6,0*/ mop(OP_LDWR, 1, &r10, rp1[CHIP], r4, MSK_D0, rp1[CHIP], WD, 0, 0, (U11)NULL, WD); /*rp1[cofs+x]*/
/*@6,1*/ exe(OP_CMPEQ, &r5, r3, EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /*y== 0*/
/*@7,0*/ exe(OP_CMV, &r0, r5, EXP_H3210, r10, EXP_H3210, r0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

/*@7,0*/ mop(OP_LDWR, 1, &r10, rp2[CHIP], r4, MSK_D0, rp2[CHIP], WD, 0, 0, (U11)NULL, WD); /*rp2[cofs+x]*/
/*@7,1*/ exe(OP_CMPEQ, &r5, r3, EXP_H3210, 1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /*y== 1*/
/*@8,0*/ exe(OP_CMV, &r0, r5, EXP_H3210, r10, EXP_H3210, r0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

/*@8,0*/ mop(OP_LDWR, 1, &r10, rp3[CHIP], r4, MSK_D0, rp3[CHIP], WD, 0, 0, (U11)NULL, WD); /*rp3[cofs+x]*/
/*@8,1*/ exe(OP_CMPEQ, &r5, r3, EXP_H3210, 1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /*y== 1*/
/*@9,0*/ mop(OP_STWR, 3, &r0, dp[CHIP], cofs, MSK_D0, dp[CHIP], WD, 0, 1, (U11)NULL, WD); /*y== 1*/

}
//EMAX5A end
}
}
//EMAX5A drain_dirty_lmm
```

filter+rmm-hokan3-emax6.obj

BR/row: max=7 min=1 ave=3 EA/row: max=1 min=0 ave=0 ARpass/row: max=0

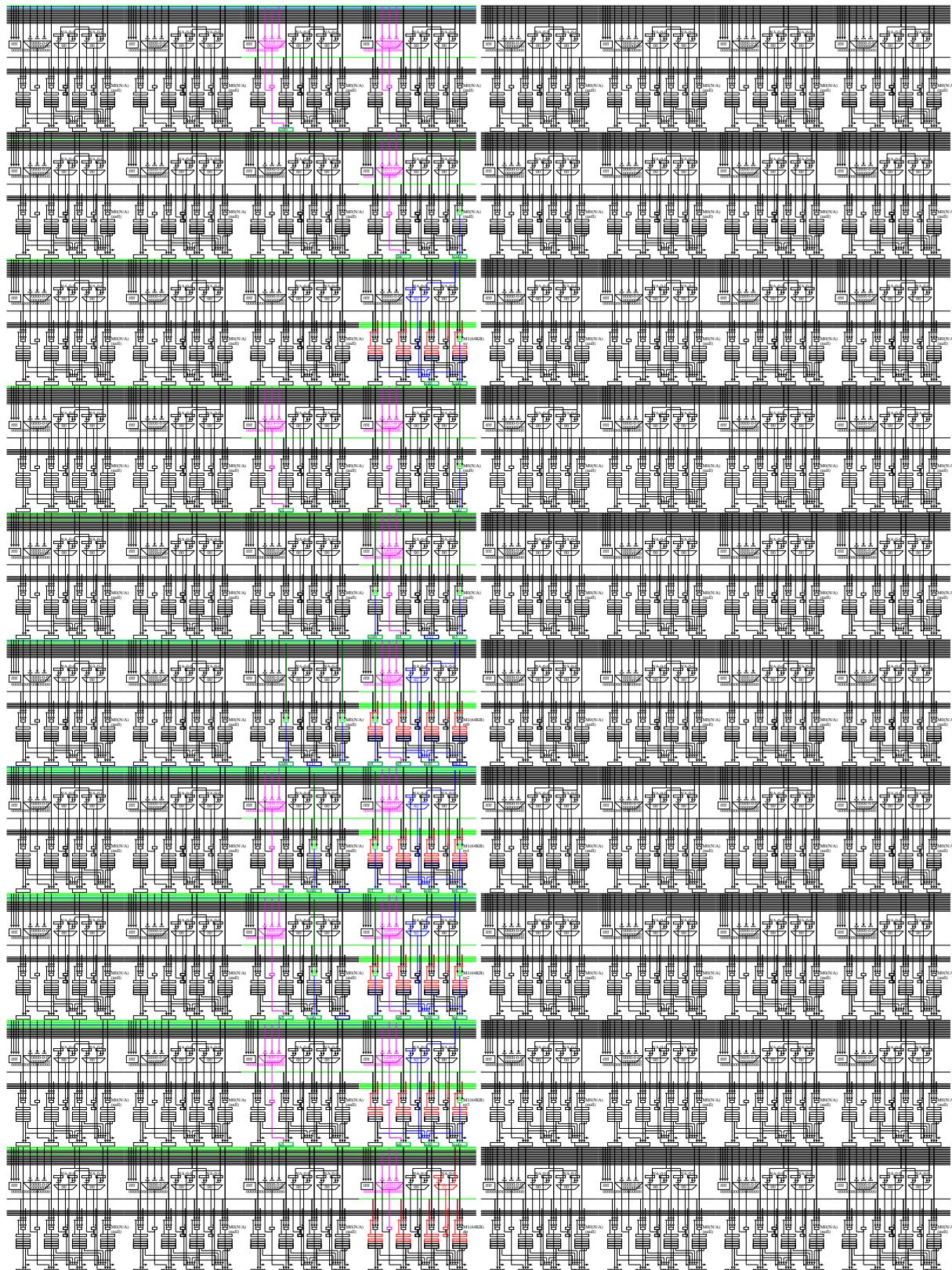


Figure.3.27: Hokan3

3.3.5 Expand4k with stencil



Figure 3.28: Expand4k

超解像処理である。低解像度画像を拡大補間して高解像度画像を生成する。一度のバースト演算により 12 行分の補間処理を行う (RMGRP=12)。入力画像の移動距離が一定ではないためステンシル計算ではなく mapdist=0 である。

```

void expand4k(UInt *p, struct X *r)
#if !defined(EMAX5) && !defined(EMAX6)
for (top=0; top<RRANGE; top+=RMGRP) { /* scan-lines */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+top+rofs;
            int k = idx*HT/768;
            int kfraq = (((idx*HT)<<4)/768)&15; /* 4bit */
            int kad = 16-ad(kfraq,8);
            int sk1 = ss(kfraq,8);
            int sk2 = ss(8,kfraq);
            Uint *pp = p+k*WD;
            Uint *rp = r->X[idx];
            for (cofs=0; cofs<1024; cofs++) { /* 本当は 4095 まで */
                int p1 = cofs*WD/1024;
                int lfracq = (((cofs*WD)<<4)/1024)&15; /* 4bit */
                int lad = 16-ad(lfracq,8);
                int s11 = ss(lfracq,8);
                int s12 = ss(8,lfracq);
                int r1 = kad*lad; /* 4bit*4bit */
                int r3 = kad*s11; /* 4bit*4bit */
                int r2 = kad*s12; /* 4bit*4bit */
                int r5 = sk1*lad; /* 4bit*4bit */
                int r9 = sk1*s11; /* 4bit*4bit */
                int r8 = sk1*s12; /* 4bit*4bit */
                int r4 = sk2*lad; /* 4bit*4bit */
                int r7 = sk2*s11; /* 4bit*4bit */
                int r6 = sk2*s12; /* 4bit*4bit */
                *rp = (unsigned int)((pp[p1]>>24&0xff)*r1
                    + (pp[p1-1]>>24&0xff)*r2 + (pp[p1+1]>>24&0xff)*r3 + (pp[p1-WD ]>>24&0xff)*r4 + (pp[p1+WD ]>>24&0xff)*r5
                    + (pp[p1-WD-1]>>24&0xff)*r6 + (pp[p1-WD+1]>>24&0xff)*r7 + (pp[p1+WD-1]>>24&0xff)*r8 + (pp[p1+WD+1]>>24&0xff)*r9)/256<<24
                | (unsigned int)((pp[p1]>>16&0xff)*r1
                    + (pp[p1-1]>>16&0xff)*r2 + (pp[p1+1]>>16&0xff)*r3 + (pp[p1-WD ]>>16&0xff)*r4 + (pp[p1+WD ]>>16&0xff)*r5
                    + (pp[p1-WD-1]>>16&0xff)*r6 + (pp[p1-WD+1]>>16&0xff)*r7 + (pp[p1+WD-1]>>16&0xff)*r8 + (pp[p1+WD+1]>>16&0xff)*r9)/256<<16
                | (unsigned int)((pp[p1]>>8&0xff)*r1
                    + (pp[p1-1]>>8&0xff)*r2 + (pp[p1+1]>>8&0xff)*r3 + (pp[p1-WD ]>>8&0xff)*r4 + (pp[p1+WD ]>>8&0xff)*r5
                    + (pp[p1-WD-1]>>8&0xff)*r6 + (pp[p1-WD+1]>>8&0xff)*r7 + (pp[p1+WD-1]>>8&0xff)*r8 + (pp[p1+WD+1]>>8&0xff)*r9)/256<<8;
                rp++;
            }
        }
    }
}
#endif

```

```

for (top=0; top<RMGRP; top+=RMGRP) { /* scan-lines */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*NRANGE+top+rofs;
            int k = idx*HT/768;
            int kfraq = (((idx*HT)<<4)/768)&15; /* 4bit */
            int kad = 16-ad(kfraq,8);
            int sk1 = ss(kfraq,8);
            int sk2 = ss(8,kfraq);
            Uint *pp = p+k*WD;
            Uint *rp = r->X[idx];
            for (cofs=0; cofs<1024; cofs++) { /* 本当は 4095 まで */
                int p1 = cofs*WD/1024;
                int lfrac = (((cofs*WD)<<4)/1024)&15; /* 4bit */
                int lad = 16-ad(lfrac,8);
                int s11 = ss(lfrac,8);
                int s12 = ss(8,lfrac);
                int r1 = kad*lad; /* 4bit*4bit */
                int r3 = kad*s11; /* 4bit*4bit */
                int r2 = kad*s12; /* 4bit*4bit */
                int r5 = sk1*lad; /* 4bit*4bit */
                int r9 = sk1*s11; /* 4bit*4bit */
                int r8 = sk1*s12; /* 4bit*4bit */
                int r4 = sk2*lad; /* 4bit*4bit */
                int r7 = sk2*s11; /* 4bit*4bit */
                int r6 = sk2*s12; /* 4bit*4bit */
                Uint ph, pl, x;
                ph = madd(mmml(b2h(pp[p1      ], 1), r1), mmul(b2h(pp[p1-1], 1), r2));
                ph = madd(mmml(b2h(pp[p1      +1], 1), r3), ph);
                ph = madd(mmml(b2h(pp[p1-WD ], 1), r4), ph);
                ph = madd(mmml(b2h(pp[p1+WD ], 1), r5), ph);
                ph = madd(mmml(b2h(pp[p1-WD-1], 1), r6), ph);
                ph = madd(mmml(b2h(pp[p1-WD+1], 1), r7), ph);
                ph = madd(mmml(b2h(pp[p1+WD-1], 1), r8), ph);
                ph = madd(mmml(b2h(pp[p1+WD+1], 1), r9), ph);
                pl = madd(mmml(b2h(pp[p1      ], 0), r1), mmul(b2h(pp[p1-1], 0), r2));
                pl = madd(mmml(b2h(pp[p1      +1], 0), r3), pl);
                pl = madd(mmml(b2h(pp[p1-WD ], 0), r4), pl);
                pl = madd(mmml(b2h(pp[p1-WD-1], 0), r5), pl);
                pl = madd(mmml(b2h(pp[p1-WD+1], 0), r6), pl);
                pl = madd(mmml(b2h(pp[p1+WD-1], 0), r7), pl);
                pl = madd(mmml(b2h(pp[p1+WD+1], 0), r8), pl);
                pl = madd(mmml(b2h(pp[p1+WD+1], 0), r9), pl);
                *rp = h2b(msrl(ph, 8), 1) | h2b(msrl(pl, 8), 0);
                rp++;
            }
        }
    }
}

```

```

U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
for (top=0; top<RRANGE; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        S11 kad[NCHIP], sk1[NCHIP], sk2[NCHIP];
        Uint *pp[NCHIP], *p0[NCHIP], *p1[NCHIP], *p2[NCHIP], *rp[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP+=4) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+top+rofs;
            int k = idx>HT?768;
            int kfraq = (((idx*HT)<(4*768))/15; /* 4bit */
            kfraq = 16-ad(kfraq,8);
            sk1[CHIP] = ss(kfraq,8);
            sk2[CHIP] = ss(8,kfraq);
            pp[CHIP] = prk+WD;
            p0[CHIP] = pp[CHIP]-WD;
            p1[CHIP] = pp[CHIP];
            p2[CHIP] = pp[CHIP]+WD;
            rp[CHIP] = r->x[idx];
        }
    }
}

//EMAX5A begin expand4k mapdist=0
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    /*1*/for ((INIT0=1,LOOP0=1024,cofs=0-WD); LOOP0>0; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        /*@0,1*/ exe(OP_ADD, &cofs, cofsl, EXP_H3210, WD, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffff, OP_NOP, OLL);
        /*@1,0*/ exe(OP_NOP, &r0, cofsl, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 1023LL, OP_SRL, 8LL);
        /*@1,1*/ exe(OP_NOP, &r4, cofsl, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x3c0LL, OP_SRL, 6LL);
        /*@2,0*/ exe(OP_ADD, &r0, pp[CHIP], EXP_H3210, r0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@2,1*/ exe(OP_MSUH, &r1, r4, EXP_H3210, 8LL, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@2,2*/ exe(OP_MSUH, &r2, 8LL, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@2,3*/ exe(OP_MSSAD, &r3, OLL, EXP_H3210, r4, EXP_H3210, 8LL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@3,0*/ exe(OP_MSUH, &r3, 16LL, EXP_H3210, r3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,1*/ exe(OP_MLUH, &r21, sk2[CHIP], EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,1*/ mop(OP_LDWR, 1, &r10, r0, -1276, MSK_D0, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@0,2*/ exe(OP_MLUH, &r22, sk2[CHIP], EXP_H3210, r2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,2*/ mop(OP_LDWR, 1, &r11, r0, -1284, MSK_D0, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@0,3*/ exe(OP_MLUH, &r23, sk2[CHIP], EXP_H3210, r3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,3*/ mop(OP_LDWR, 1, &r12, r0, -1280, MSK_D0, (U11)p0[CHIP], WD, 0, 0, (U11)NULL, WD);

        /*@0,5,1*/ exe(OP_MLUH, &r13, r10, EXP_B5410, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,5,2*/ exe(OP_MLUH, &r14, r11, EXP_B5410, r22, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,5,3*/ exe(OP_MLUH, &r15, r12, EXP_B5410, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,6,0*/ exe(OP_MAUH3, &r16, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,6,1*/ exe(OP_MLUH, &r13, r10, EXP_B7632, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,6,2*/ exe(OP_MLUH, &r14, r11, EXP_B7632, r22, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,6,3*/ exe(OP_MLUH, &r15, r12, EXP_B7632, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,7,0*/ exe(OP_MAUH3, &r17, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,7,1*/ exe(OP_MLUH, &r21, kad[CHIP], EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,7,1*/ mop(OP_LDWR, 1, &r10, r0, 4, MSK_D0, (U11)p1[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@0,7,2*/ exe(OP_MLUH, &r22, kad[CHIP], EXP_H3210, r2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,7,2*/ mop(OP_LDWR, 1, &r11, r0, -4, MSK_D0, (U11)p1[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@0,7,3*/ exe(OP_MLUH, &r23, kad[CHIP], EXP_H3210, r3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,7,3*/ mop(OP_LDWR, 1, &r12, r0, 0, MSK_D0, (U11)p1[CHIP], WD, 0, 0, (U11)NULL, WD);

        /*@0,8,1*/ exe(OP_MLUH, &r13, r10, EXP_B5410, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,8,2*/ exe(OP_MLUH, &r14, r11, EXP_B5410, r22, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,8,3*/ exe(OP_MLUH, &r15, r12, EXP_B5410, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,9,0*/ exe(OP_MAUH3, &r18, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,9,1*/ exe(OP_MLUH, &r13, r10, EXP_B7632, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,9,2*/ exe(OP_MLUH, &r14, r11, EXP_B7632, r22, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,9,3*/ exe(OP_MLUH, &r15, r12, EXP_B7632, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,10,0*/ exe(OP_MAUH3, &r19, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,10,1*/ exe(OP_MLUH, &r21, sk1[CHIP], EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,10,1*/ mop(OP_LDWR, 1, &r10, r0, 1284, MSK_D0, (U11)p2[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@0,10,2*/ exe(OP_MLUH, &r22, sk1[CHIP], EXP_H3210, r2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,10,2*/ mop(OP_LDWR, 1, &r11, r0, 1276, MSK_D0, (U11)p2[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@0,10,3*/ exe(OP_MLUH, &r23, sk1[CHIP], EXP_H3210, r3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,10,3*/ mop(OP_LDWR, 1, &r12, r0, 1280, MSK_D0, (U11)p2[CHIP], WD, 0, 0, (U11)NULL, WD);

        /*@0,11,1*/ exe(OP_MLUH, &r13, r10, EXP_B5410, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,11,2*/ exe(OP_MLUH, &r14, r11, EXP_B5410, r22, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,11,3*/ exe(OP_MLUH, &r15, r12, EXP_B5410, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,12,0*/ exe(OP_MAUH3, &r20, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,12,1*/ exe(OP_MLUH, &r13, r10, EXP_B7632, r21, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,12,2*/ exe(OP_MLUH, &r14, r11, EXP_B7632, r22, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,12,3*/ exe(OP_MLUH, &r15, r12, EXP_B7632, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

        /*@0,13,0*/ exe(OP_MAUH3, &r21, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,14,0*/ exe(OP_MAUH3, &r21, r17, EXP_H3210, r19, EXP_H3210, r21, EXP_H3210, OP_OR, OLL, OP_SRL, 8LL);
        /*@0,14,1*/ exe(OP_MAUH3, &r20, r16, EXP_H3210, r18, EXP_H3210, r20, EXP_H3210, OP_OR, OLL, OP_SRL, 8LL);

        /*@0,15,0*/ exe(OP_MH2BW, &r31, r21, EXP_H3210, r20, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@0,15,0*/ mop(OP_STWR, 3, &r31, (U11)(rp[CHIP]+), OLL, MSK_D0, (U11)rp[CHIP], 1024, 0, 0, (U11)NULL, 1024);
    }
}
}

//EMAX5A end
}
}

//EMAX5A drain_dirty_lmm

```

filter+rmm-expand4k-emax6.obj

BR/row: max=15 min=1 ave=8 EA/row: max=3 min=0 ave=0 ARpass/row: max=0

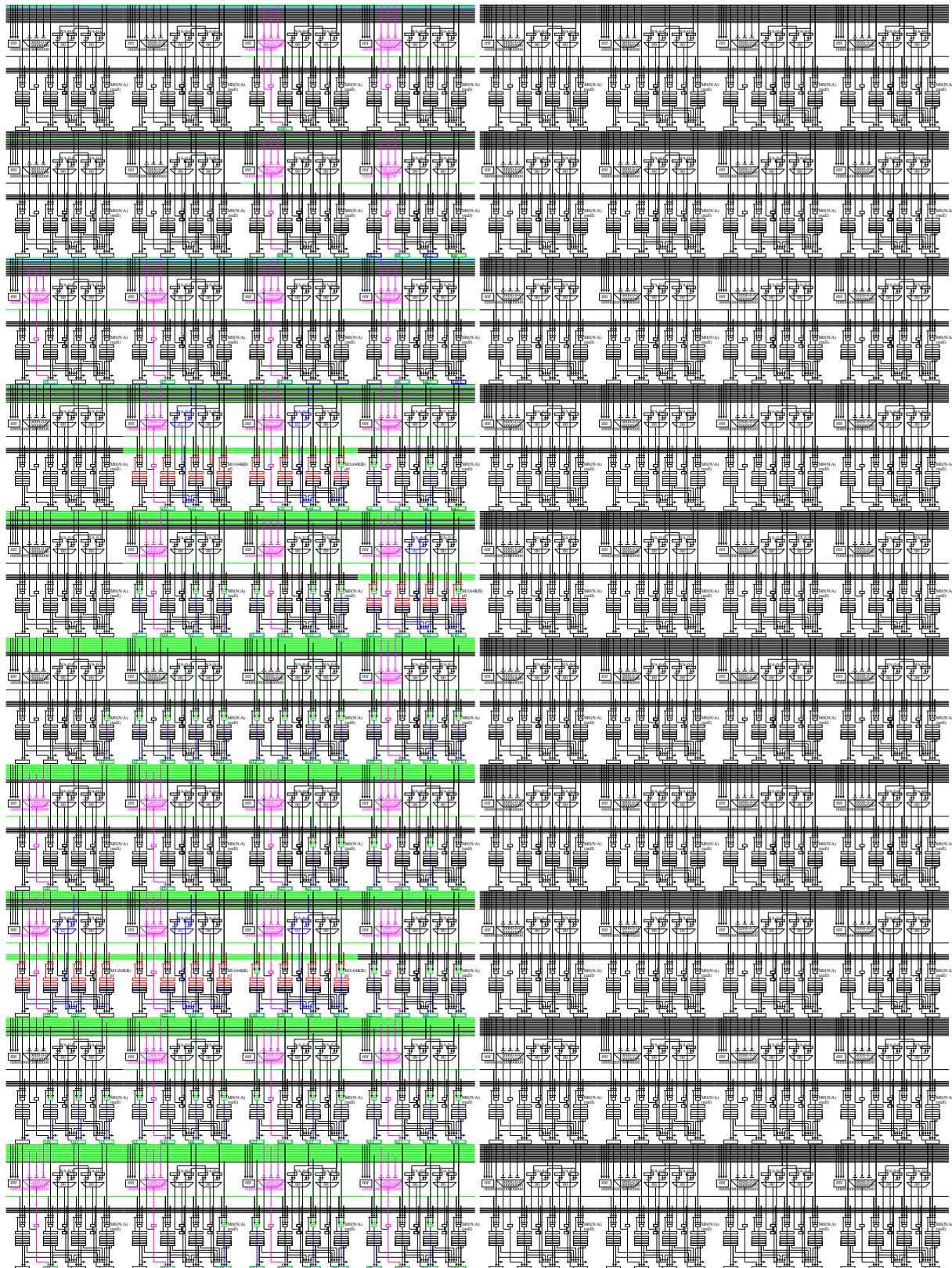


Figure.3.29: Expand4k

3.3.6 Unsharp with stencil



Figure 3.30: Unsharp

エッジを強調する 3x3 の鮮鋭化フィルタである。一度のバースト演算により、6箇所 (OMAP=6) の各々について、10 行分のフィルタ処理を行う (RMGRP=10)。ステンシル計算であり mapdist=1 である。

```
void unsharp(Uchar *p, Uchar *r)
#if !defined(EMAX5) & !defined(EMAX6)
for (top=PAD; top<HT-PAD; top++) { /* scan-lines */
    Uchar *p0 = p+((top )*WD+(0 ))*4; // p1 p5 p2
    Uchar *p1 = p+((top-1)*WD+(0-1))*4; // p6 p0 p7
    Uchar *p2 = p+((top-1)*WD+(0+1))*4; // p3 p8 p4
    Uchar *p3 = p+((top+1)*WD+(0-1))*4;
    Uchar *p4 = p+((top+1)*WD+(0+1))*4;
    Uchar *p5 = p+((top+1)*WD+(0 ))*4;
    Uchar *p6 = p+((top )*WD+(0-1))*4;
    Uchar *p7 = p+((top )*WD+(0+1))*4;
    Uchar *p8 = p+((top+1)*WD+(0 ))*4;
    Uchar *rp = r+((top )*WD+(0 ))*4;
    for (cofs=0; cofs<WD; cofs++) {
        int t0,t1,t2;
        rp[0] = 0;
        t0 = p[1]; t1 = p[1]*p2[1]+p3[1]*p4[1]; t2 = p5[1]+p6[1]+p7[1]+p8[1];
        rp[1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
        t0 = p[2]; t1 = p[2]*p2[2]+p3[2]*p4[2]; t2 = p5[2]+p6[2]+p7[2]+p8[2];
        rp[2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
        t0 = p[3]; t1 = p[3]*p2[3]+p3[3]*p4[3]; t2 = p5[3]+p6[3]+p7[3]+p8[3];
        rp[3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
        p0+=4; p1+=4; p2+=4; p3+=4; p4+=4; p5+=4; p6+=4; p7+=4; p8+=4; rp+=4;
    }
}
#endif
```

```
for (top=0; top<RRANGE; top+=RMGRP) { /* scan-lines */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = (CHIP*RRANGE+DMAP+top+rofs)*WD;
            Uchar *p0 = p+idx + (0 )*4; // p1 p5 p2
            Uchar *p1 = p+idx-WD+(0-1))*4; // p6 p0 p7
            Uchar *p2 = p+idx-WD+(0+1))*4; // p3 p8 p4
            Uchar *p3 = p+idx+WD+(0-1))*4;
            Uchar *p4 = p+idx+WD+(0+1))*4;
            Uchar *p5 = p+idx-WD+(0 ))*4;
            Uchar *p6 = p+idx -(0-1))*4;
            Uchar *p7 = p+idx -(0+1))*4;
            Uchar *p8 = p+idx+WD+(0 ))*4;
            Uchar *rp = r+idx +(0 ))*4;
            for (cofs=0; cofs<WD; cofs++) {
                for (oc=0; oc<OMAP; oc++) {
                    Uint pix0 = (oc*RRANGE*WD+cofs)*4+0;
                    Uint pix1 = (oc*RRANGE*WD+cofs)*4+1;
                    Uint pix2 = (oc*RRANGE*WD+cofs)*4+2;
                    Uint pix3 = (oc*RRANGE*WD+cofs)*4+3;
                    int t0,t1,t2;
                    rp[pix0] = 0;
                    t0 = p0[pix1]; t1 = p1[pix1]*p2[pix1]+p3[pix1]*p4[pix1]; t2 = p5[pix1]+p6[pix1]+p7[pix1]+p8[pix1];
                    rp[pix1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
                    t0 = p0[pix2]; t1 = p1[pix2]*p2[pix2]+p3[pix2]*p4[pix2]; t2 = p5[pix2]+p6[pix2]+p7[pix2]+p8[pix2];
                    rp[pix2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
                    t0 = p0[pix3]; t1 = p1[pix3]*p2[pix3]+p3[pix3]*p4[pix3]; t2 = p5[pix3]+p6[pix3]+p7[pix3]+p8[pix3];
                    rp[pix3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);
                }
            }
        }
    }
}
```

```

U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc1, cc2, cc3, ex0, ex1;
for (top0; top<RRANGE; top+=RMGRP) { /* scan-lines */
    for (rofs=0; rofs<RRANGE; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        Uchar *pp0[NCHIP], *pc0[NCHIP], *pn0[NCHIP], *rc0[NCHIP];
        Uchar *pp1[NCHIP], *pc1[NCHIP], *pn1[NCHIP], *rc1[NCHIP];
        Uchar *pp2[NCHIP], *pc2[NCHIP], *pn2[NCHIP], *rc2[NCHIP];
        Uchar *pp3[NCHIP], *pc3[NCHIP], *pn3[NCHIP], *rc3[NCHIP];
        Uchar *pp4[NCHIP], *pc4[NCHIP], *pn4[NCHIP], *rc4[NCHIP];
        Uchar *pp5[NCHIP], *pc5[NCHIP], *pn5[NCHIP], *rc5[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = (CHIP*RRANGE+MAP+top+rofs)*WD4;
            pp0[CHIP] = p+idx+RRANGE*WD* 0-1280; pc0[CHIP] = p+idx+RRANGE*WD* 0+1280; rc0[CHIP] = r+idx+RRANGE*WD* 0;
            pp1[CHIP] = p+idx+RRANGE*WD* 4-1280; pc1[CHIP] = p+idx+RRANGE*WD* 4+1280; rc1[CHIP] = r+idx+RRANGE*WD* 4;
            pp2[CHIP] = p+idx+RRANGE*WD* 8-1280; pc2[CHIP] = p+idx+RRANGE*WD* 8+1280; rc2[CHIP] = r+idx+RRANGE*WD* 8;
            pp3[CHIP] = p+idx+RRANGE*WD*12-1280; pc3[CHIP] = p+idx+RRANGE*WD*12+1280; rc3[CHIP] = r+idx+RRANGE*WD*12;
            pp4[CHIP] = p+idx+RRANGE*WD*16-1280; pc4[CHIP] = p+idx+RRANGE*WD*16+1280; rc4[CHIP] = r+idx+RRANGE*WD*16;
            pp5[CHIP] = p+idx+RRANGE*WD*20-1280; pc5[CHIP] = p+idx+RRANGE*WD*20+1280; rc5[CHIP] = r+idx+RRANGE*WD*20;
        }
    }
//EMAX5A begin unsharp mapdist=1
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*1*/for (INIT0=1,LOOP0=WD,cofs=0;LOOP0>0;INIT0=0) { /* stage#0 */ mapped to FOR() on BR[63][0][0] */
/*@0,1*/ exec(OP_ADD, &cofs, cofs, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL);
/*@map0*/
/*@1,0*/ exec(OP_ADD, &coefs, pc0[CHIP], EXP_H3210, coefs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@2,0*/ exec(OP_LDWR, 1, &r1, pofs, -1276, MSK_DO, (U11)pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,0*/ exec(OP_LDWR, 1, &r2, pofs, -1284, MSK_DO, (U11)pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,1*/ exec(OP_LDWR, 1, &r5, pofs, -1280, MSK_DO, (U11)pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,0*/ exec(OP_MAUH, &r11, r1, EXP_B5410, r2, EXP_B5410, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@3,0*/ exec(OP_LDWR, 1, &r6, pofs, 4, MSK_DO, (U11)pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,0*/ exec(OP_LDWR, 1, &r7, pofs, -4, MSK_DO, (U11)pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,1*/ exec(OP_MAUH, &r12, r1, EXP_B7632, r2, EXP_B7632, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@3,1*/ exec(OP_LDWR, 1, &r0, pofs, 0, MSK_DO, (U11)pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,1*/ exec(OP_MLUH, &r20, r0, EXP_B5410, 239, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,0*/ exec(OP_LDWR, 1, &r3, pofs, 1284, MSK_DO, (U11)pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,0*/ exec(OP_LDWR, 1, &r4, pofs, 1276, MSK_DO, (U11)pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,1*/ exec(OP_MLUH, &r21, r0, EXP_B7632, 239, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,1*/ exec(OP_LDWR, 1, &r8, pofs, 1280, MSK_DO, (U11)pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,1*/ exec(OP_MAUH, &r15, r5, EXP_B5410, r6, EXP_B5410, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,3*/ exec(OP_MAUH, &r16, r5, EXP_B7632, r6, EXP_B7632, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@5,0*/ exec(OP_MAUH3, &r11, r3, EXP_B5410, r4, EXP_B5410, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@5,1*/ exec(OP_MAUH3, &r12, r3, EXP_B7632, r4, EXP_B7632, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@6,0*/ exec(OP_MLUH, &r13, r11, EXP_H3210, 13, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@6,1*/ exec(OP_MLUH, &r14, r12, EXP_H3210, 13, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@6,2*/ exec(OP_MAUH3, &r15, r7, EXP_B5410, r15, EXP_B5410, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@6,3*/ exec(OP_MAUH3, &r16, r7, EXP_B7632, r8, EXP_B7632, r16, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@7,0*/ exec(OP_NOP, &r7, r15, EXP_H3210, 0LL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 2LL);
/*@7,1*/ exec(OP_MLUH, &r17, r15, EXP_H3210, 15, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@7,2*/ exec(OP_NOP, &r8, r16, EXP_H3210, 0LL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 2LL);
/*@7,3*/ exec(OP_MLUH, &r18, r16, EXP_H3210, 15, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@8,0*/ exec(OP_MSUH3, &r10, r20, EXP_H3210, r7, EXP_H3210, r17, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@8,1*/ exec(OP_MSUH3, &r11, r21, EXP_H3210, r8, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@8,2*/ exec(OP_MSUH3, &r20, r10, EXP_H3210, r13, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 7LL);
/*@9,1*/ exec(OP_MSUH, &r21, r11, EXP_H3210, r14, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 7LL);
/*@10,0*/ exec(OP_MH2BW, &r31, r21, EXP_H3210, r20, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@10,0*/ exec(OP_STWR, 3, &r31, rc0[CHIP], coefs, MSK_DO, rc0[CHIP], WD, 0, 0, (U11)NULL, WD);
:
/*map5*/
/*@46,1*/ exec(OP_ADD, &coefs, pc5[CHIP], EXP_H3210, coefs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@47,0*/ exec(OP_LDWR, 1, &r1, pofs, -1276, MSK_DO, (U11)pp5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@47,0*/ exec(OP_LDWR, 1, &r2, pofs, -1284, MSK_DO, (U11)pp5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@47,1*/ exec(OP_LDWR, 1, &r5, pofs, -1280, MSK_DO, (U11)pp5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@48,0*/ exec(OP_MAUH, &r11, r1, EXP_B5410, r2, EXP_B5410, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@48,0*/ exec(OP_LDWR, 1, &r6, pofs, 4, MSK_DO, (U11)pc5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@48,0*/ exec(OP_LDWR, 1, &r7, pofs, -4, MSK_DO, (U11)pc5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@48,1*/ exec(OP_MAUH, &r12, r1, EXP_B7632, r2, EXP_B7632, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@48,1*/ exec(OP_LDWR, 1, &r0, pofs, 0, MSK_DO, (U11)pc5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@49,0*/ exec(OP_MLUH, &r20, r0, EXP_B5410, 239, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@49,0*/ exec(OP_LDWR, 1, &r3, pofs, 1284, MSK_DO, (U11)pn5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@49,0*/ exec(OP_LDWR, 1, &r4, pofs, 1276, MSK_DO, (U11)pn5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@49,1*/ exec(OP_MLUH, &r21, r0, EXP_B7632, 239, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@49,1*/ exec(OP_LDWR, 1, &r8, pofs, 1280, MSK_DO, (U11)pn5[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@49,2*/ exec(OP_MAUH, &r15, r5, EXP_B5410, r6, EXP_B5410, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@49,3*/ exec(OP_MAUH, &r16, r5, EXP_B7632, r6, EXP_B7632, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@50,0*/ exec(OP_MAUH3, &r11, r3, EXP_B5410, r4, EXP_B5410, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@50,1*/ exec(OP_MAUH3, &r12, r3, EXP_B7632, r4, EXP_B7632, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@51,0*/ exec(OP_MLUH, &r13, r11, EXP_H3210, 13, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@51,1*/ exec(OP_MLUH, &r14, r12, EXP_H3210, 13, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@51,2*/ exec(OP_MAUH3, &r15, r7, EXP_B5410, r8, EXP_B5410, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@51,3*/ exec(OP_MAUH3, &r16, r7, EXP_B7632, r8, EXP_B7632, r16, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@52,0*/ exec(OP_NOP, &r7, r15, EXP_H3210, 0LL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 2LL);
/*@52,1*/ exec(OP_MLUH, &r17, r15, EXP_H3210, 15, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@52,2*/ exec(OP_NOP, &r8, r16, EXP_H3210, 0LL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 2LL);
/*@52,3*/ exec(OP_MLUH, &r18, r16, EXP_H3210, 15, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@53,0*/ exec(OP_MSUH3, &r10, r20, EXP_H3210, r7, EXP_H3210, r17, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@53,1*/ exec(OP_MSUH3, &r11, r21, EXP_H3210, r8, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@54,0*/ exec(OP_MSUH, &r20, r10, EXP_H3210, r13, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 7LL);
/*@54,1*/ exec(OP_MSUH, &r21, r11, EXP_H3210, r14, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRLM, 7LL);
/*@55,0*/ exec(OP_MH2BW, &r31, r21, EXP_H3210, r20, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@55,0*/ exec(OP_STWR, 3, &r31, rc5[CHIP], coefs, MSK_DO, rc5[CHIP], WD, 0, 0, (U11)NULL, WD);
}
}
//EMAX5A end
}
}
//EMAX5A drain_dirty_lmm

```

filter+rmm-unsharp-emax6.obj

BR/row: max=11 min=2 ave=6 EA/row: max=3 min=0 ave=1 ARpass/row: max=0



Figure.3.31: Unsharp

3.3.7 Blur with stencil



Figure 3.32: Blur

3x3 のパイプライン化メディアンフィルタである。一度のバースト演算により、4箇所 (OMAP=4) の各々について、15 行分のフィルタ処理を行う (RMGRP=15)。ステンシル計算であり mapdist=1 である。

```
void blur(Uint *p, Uint *r)
#if !defined(EMAX5) && !defined(EMAX6)
for (top>PAD; top<HT-PAD; top++) { /* scan-lines */
    Uint *p0 = p+(top )*WD ;
    Uint *p1 = p+(top )*WD ;
    Uint *p2 = p+(top )*WD+1;
    Uint *p3 = p+(top )*WD+1;
    Uint *p4 = p+(top-1)*WD ;
    Uint *p5 = p+(top+1)*WD ;
    Uint *p6 = p+(top-1)*WD-1;
    Uint *p7 = p+(top-1)*WD+1;
    Uint *p8 = p+(top+1)*WD-1;
    Uint *p9 = p+(top+1)*WD+1;
    Uint *rp = r+(top )*WD ;
    for (cofs=0; cofs<WD; cofs++) {
        *rp = (Uint)((*p1>>24&0xff)*20
            + (*p2>>24&0xff)*12 + (*p3>>24&0xff)*12 + (*p4>>24&0xff)*12
            + (*p5>>24&0xff)*8 + (*p7>>24&0xff)*8 + (*p8>>24&0xff)*8 + (*p9>>24&0xff)*8)/100<<24
        | (Uint)((*p1>>16&0xff)*20
            + (*p2>>16&0xff)*12 + (*p3>>16&0xff)*12 + (*p4>>16&0xff)*12 + (*p5>>16&0xff)*12
            + (*p6>>16&0xff)*8 + (*p7>>16&0xff)*8 + (*p8>>16&0xff)*8 + (*p9>>16&0xff)*8)/100<<16
        | (Uint)((*p1>> 8&0xff)*20
            + (*p2>> 8&0xff)*12 + (*p3>> 8&0xff)*12 + (*p4>> 8&0xff)*12 + (*p5>> 8&0xff)*12
            + (*p6>> 8&0xff)*8 + (*p7>> 8&0xff)*8 + (*p8>> 8&0xff)*8 + (*p9>> 8&0xff)*8)/100<<8;
        p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++; p9++; rp++;
    }
}
#endif
```

```
for (top>PAD; top<HT-PAD; top++) { /* scan-lines */
    Uint *p0 = p+(top )*WD ;
    Uint *p1 = p+(top )*WD ;
    Uint *p2 = p+(top )*WD+1;
    Uint *p3 = p+(top )*WD+1;
    Uint *p4 = p+(top-1)*WD ;
    Uint *p5 = p+(top+1)*WD ;
    Uint *p6 = p+(top-1)*WD-1;
    Uint *p7 = p+(top-1)*WD+1;
    Uint *p8 = p+(top+1)*WD-1;
    Uint *p9 = p+(top+1)*WD+1;
    Uint *rp = r+(top )*WD ;
    for (cofs=0; cofs<WD; cofs++) {
        Uint s0,s1,s2,s3,s4,s5,s6,s7,s8;
        Uint t0,t1,t2;
        s0=*p1;s1=*p2;s2=*p3;s3=*p4;s4=*p5;s5=*p6;s6=*p7;s7=*p8;s8=*p9;
        /*[ 5 | 3 | 6 | 5 < 3 < ★ | 5 | 3 | 2 | 5 < 3 < ★ | 5 | 3 | 5 < < ★ | 5 ]
         |V+V-V-| | 1 < 0 < 2 | | 1 < 0 < 2 | | 0 | | 1 | 0 | | 0 | | 1 中間値確定
         | 1 | 0 | 2 | | 1 < 0 < 2 | | 1 < 0 < 2 | | 0 | | 1 | 0 | | 0 |
         |V+V-V-| | 7 | 4 | 8 | | ★ < 4 < 8 | | 1 | 4 | 8 | | ★ < 4 < 8 | | 4 | 8 | | ★ < 8 | | 8 |
        t0 = pmax3(s5,s1,s7); t1 = pmid3(s5,s1,s7); t2 = pmin3(s5,s1,s7); s5 = t0; s1 = t1; s7 = t2;
        t0 = pmax3(s3,s0,s4); t1 = pmid3(s3,s0,s4); t2 = pmin3(s3,s0,s4); s3 = t0; s0 = t1; s4 = t2;
        t0 = pmax3(s6,s2,s8); t1 = pmid3(s6,s2,s8); t2 = pmin3(s6,s2,s8); s6 = t0; s2 = t1; s8 = t2;

        t0 = pmin3(s5,s3,s6); t1 = pmid3(s5,s3,s6); s5 = t0; s3 = t1;
        t0 = pmin3(s3,s0,s2); t1 = pmid3(s3,s0,s2); t2 = pmax3(s1,s0,s2); s1 = t0; s0 = t1; s2 = t2;
        t0 = pmid3(s7,s4,s8); t1 = pmid3(s7,s4,s8); s7 = t0; s4 = t1; s8 = t2;

        t0 = pmax2(s5,s1); t1 = pmin2(s5,s1); s5 = t0; s1 = t1;
        t0 = pmax3(s3,s0,s4); t1 = pmid3(s3,s0,s4); t2 = pmin3(s3,s0,s4); s3 = t0; s0 = t1; s4 = t2;
        t0 = pmid2(s2,s8); t1 = pmid2(s2,s8); s2 = t0; s8 = t1;

        t0 = pmin3(s5,s3,s2); t1 = pmid3(s5,s3,s2); s5 = t0; s3 = t1;
        t0 = pmid3(s1,s4,s8); t1 = pmid3(s1,s4,s8); s4 = t0; s8 = t1;

        t0 = pmax2(s5,s4); t1 = pmin2(s5,s4); s5 = t0; s4 = t1;
        t0 = pmax3(s3,s0,s8); t1 = pmid3(s3,s0,s8); t2 = pmin3(s3,s0,s8); s3 = t0; s0 = t1; s8 = t2;

        s5 = pmin2(s5,s3); s8 = pmax2(s4,s8);
        *rp = pmid3(s5,s0,s8);
        p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++; p9++; rp++;
    }
}
```

```

U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
for (top=0; top<RRANGE; top+=RMGRP) { /* scan-lines */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        Uint *pp0[NCHIP], *pc0[NCHIP], *pn0[NCHIP];
        Uint *pp1[NCHIP], *pc1[NCHIP], *pn1[NCHIP];
        Uint *pp2[NCHIP], *pc2[NCHIP], *pn2[NCHIP];
        Uint *pp3[NCHIP], *pc3[NCHIP], *pn3[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = (CHIP*RRANGE+DMAP+top*rofs)*WD;
            pp0[CHIP] = p+idx+RRANGE*WD*0+WD; pc0[CHIP] = p+idx+RRANGE*WD*0+WD; rc0[CHIP] = r+idx+RRANGE*WD*0;
            pp1[CHIP] = p+idx+RRANGE*WD*1+WD; pc1[CHIP] = p+idx+RRANGE*WD*1+WD; rc1[CHIP] = r+idx+RRANGE*WD*1;
            pp2[CHIP] = p+idx+RRANGE*WD*2+WD; pc2[CHIP] = p+idx+RRANGE*WD*2+WD; rc2[CHIP] = r+idx+RRANGE*WD*2;
            pp3[CHIP] = p+idx+RRANGE*WD*3+WD; pc3[CHIP] = p+idx+RRANGE*WD*3+WD; rc3[CHIP] = r+idx+RRANGE*WD*3;
        }
    }
//EMAX5A begin blur mapdist=1
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    /*1*/for (INIT0=0,LOOP0=WD,cofs=0;4; LOOP0++; INIT0=0) { /* stage#0 */ /* mapped to FDR() on BR[63][0][0] */
    /*@0,1*/ exe(OP_ADD, &cofs, cof, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
    /*map0*/
    /*@01,0*/ exe(OP_ADD, &poofs, pc0[CHIP], EXP_H3210, cof, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@02,0*/ m0p(OP_LDWR, 1, &r7, pofs, -1276, MSK_DO, pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@02,1*/ m0p(OP_LDWR, 1, &r1, pofs, -1280, MSK_DO, pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@02,2*/ m0p(OP_LDWR, 1, &r5, pofs, -1284, MSK_DO, pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@03,0*/ exe(OP_MMINS, &r17, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@03,1*/ m0p(OP_LDWR, 1, &r4, pofs, 4, MSK_DO, pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@03,2*/ m0p(OP_LDWR, 1, &r0, pofs, 0, MSK_DO, pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@03,1*/ exe(OP_MMID3, &r11, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@03,1*/ m0p(OP_LDWR, 1, &r3, pofs, -4, MSK_DO, pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@03,2*/ exe(OP_MMMAX3, &r15, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@04,0*/ exe(OP_MMINS, &r14, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@04,0*/ m0p(OP_LDWR, 1, &r8, pofs, 1284, MSK_DO, pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@04,1*/ m0p(OP_MMID3, &r2, pofs, 1280, MSK_DO, pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@04,1*/ exe(OP_MMID3, &r10, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@04,1*/ m0p(OP_LDWR, 1, &r6, pofs, 1276, MSK_DO, pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@04,2*/ exe(OP_MMMAX3, &r13, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@05,0*/ exe(OP_MMINS, &r18, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@05,1*/ exe(OP_MMID3, &r12, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@05,2*/ exe(OP_MMMAX3, &r16, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*step-2*/
    /*@06,0*/ exe(OP_MMMAX3, &r2, r11, EXP_H3210, r10, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@06,1*/ exe(OP_MMID3, &r0, r11, EXP_H3210, r10, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@06,2*/ exe(OP_MMINS, &r1, r11, EXP_H3210, r10, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@06,3*/ exe(OP_MMMAX3, &r8, r17, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@07,0*/ exe(OP_MMID3, &r4, r17, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@07,1*/ exe(OP_MMID3, &r3, r15, EXP_H3210, r13, EXP_H3210, r16, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@07,2*/ exe(OP_MMINS, &r5, r15, EXP_H3210, r13, EXP_H3210, r16, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*step-3*/
    /*@08,0*/ exe(OP_MMINS, &r14, r3, EXP_H3210, r0, EXP_H3210, r4, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@08,1*/ exe(OP_MMID3, &r10, r3, EXP_H3210, r0, EXP_H3210, r4, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@08,2*/ exe(OP_MMMAX3, &r13, r3, EXP_H3210, r0, EXP_H3210, r4, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@08,3*/ exe(OP_MMIN, &r18, r2, EXP_H3210, r1, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@09,0*/ exe(OP_MMMAX, &r12, r2, EXP_H3210, r8, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@09,1*/ exe(OP_MMIN, &r11, r5, EXP_H3210, r1, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@09,2*/ exe(OP_MMMAX, &r15, r5, EXP_H3210, r1, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*step-4*/
    /*@10,0*/ exe(OP_MMID3, &r4, r11, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@10,1*/ exe(OP_MMINS, &r5, r15, EXP_H3210, r13, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*step-5*/
    /*@10,2*/ exe(OP_MMMAX3, &r8, r11, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@10,3*/ exe(OP_MMID3, &r3, r15, EXP_H3210, r13, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@11,0*/ exe(OP_MMIN, &r14, r5, EXP_H3210, r4, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@11,1*/ exe(OP_MMMAX, &r15, r5, EXP_H3210, r4, EXP_H3210, r11, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@11,2*/ exe(OP_MMINS, &r18, r3, EXP_H3210, r10, EXP_H3210, r18, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@11,3*/ exe(OP_MMID3, &r10, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@12,0*/ exe(OP_MMAX3, &r13, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*step-6*/
    /*@12,1*/ exe(OP_MMAX, &r8, r14, EXP_H3210, r18, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@13,0*/ exe(OP_MMIN, &r5, r15, EXP_H3210, r13, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@14,0*/ exe(OP_MMID3, &r31, r5, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@14,0*/ m0p(OP_STWR, 3, &r31, rc0[CHIP], cof, MSK_DO, rc0[CHIP], WD, 0, 0, (U11)NULL, WD);
    :
    /*map3*/
    /*@40,1*/ exe(OP_ADD, &poofs, pc3[CHIP], EXP_H3210, cof, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
    /*@41,0*/ m0p(OP_LDWR, 1, &r7, pofs, -1276, MSK_DO, pp3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@41,0*/ m0p(OP_LDWR, 1, &r1, pofs, -1280, MSK_DO, pp3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@41,1*/ m0p(OP_LDWR, 1, &r5, pofs, -1284, MSK_DO, pp3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@42,0*/ m0p(OP_MMINS, &r17, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@42,0*/ m0p(OP_LDWR, 1, &r4, pofs, 4, MSK_DO, pc3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@42,0*/ m0p(OP_LDWR, 1, &r0, pofs, 0, MSK_DO, pc3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@42,1*/ exe(OP_MMID3, &r11, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@42,1*/ m0p(OP_LDWR, 1, &r3, pofs, -4, MSK_DO, pc3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@42,2*/ exe(OP_MMMAX3, &r15, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@43,0*/ exe(OP_MMINS, &r14, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@43,0*/ m0p(OP_LDWR, 1, &r8, pofs, 1284, MSK_DO, pn3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@43,0*/ m0p(OP_LDWR, 1, &r2, pofs, 1280, MSK_DO, pn3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@43,1*/ exe(OP_MMID3, &r10, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@43,1*/ m0p(OP_LDWR, 1, &r6, pofs, 1276, MSK_DO, pn3[CHIP], WD, 0, 0, (U11)NULL, WD);
    /*@43,2*/ exe(OP_MMMAX3, &r13, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@44,0*/ exe(OP_MMINS, &r18, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@44,1*/ exe(OP_MMID3, &r12, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@44,2*/ exe(OP_MMAX3, &r16, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*step-2*/
    :
    /*step-5*/
    /*@49,2*/ exe(OP_MMAX3, &r8, r11, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@49,3*/ exe(OP_MMID3, &r3, r15, EXP_H3210, r13, EXP_H3210, r12, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@50,0*/ exe(OP_MMIN, &r14, r5, EXP_H3210, r4, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@50,1*/ exe(OP_MMMAX, &r15, r5, EXP_H3210, r4, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@50,2*/ exe(OP_MMINS, &r18, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@50,3*/ exe(OP_MMID3, &r10, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@51,0*/ exe(OP_MMAX3, &r13, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*step-6*/
    /*@51,1*/ exe(OP_MMAX, &r8, r14, EXP_H3210, r18, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@52,0*/ exe(OP_MMIN, &r5, r15, EXP_H3210, r13, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@53,0*/ exe(OP_MMID3, &r31, r5, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
    /*@53,0*/ m0p(OP_STWR, 3, &r31, rc3[CHIP], cof, MSK_DO, rc3[CHIP], WD, 0, 0, (U11)NULL, WD);
    :
}
//EMAX5A end
}
}
//EMAX5A drain_dirty_lmm

```

filter+rmm-blur-emax6.obj

BR/row: max=11 min=2 ave=6 EA/row: max=3 min=0 ave=0 ARpass/row: max=0



Figure.3.33: Blur

3.3.8 Edge with stencil

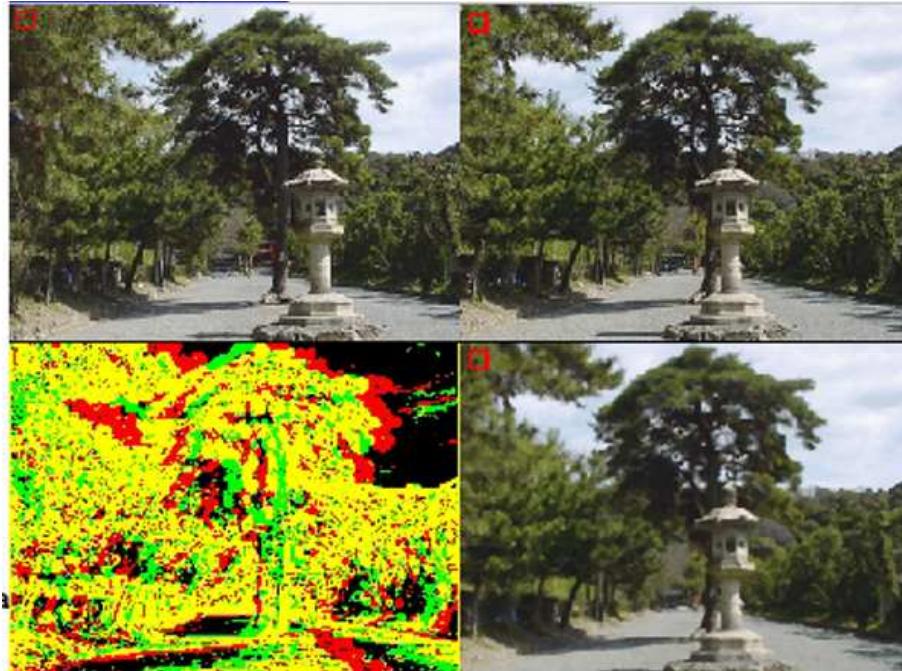


Figure 3.34: Edge

3x3 のエッジ検出フィルタである。一度のバースト演算により、6箇所 (OMAP=6) の各々について、10行分のフィルタ処理を行う (RMGRP=10)。ステンシル計算であり mapdist=1 である。

```

void edge(UInt *p, struct E *r)
#if !defined(EMAX5) && !defined(EMAX6)
for (top=PAD; top<HT-PAD; top++) { /* scan-lines */
    UInt *p0 = p+(top )*WD ;
    UInt *p1 = p+(top-1)*WD-1;
    UInt *p2 = p+(top+1)*WD+1;
    UInt *p3 = p+(top-1)*WD ;
    UInt *p4 = p+(top+1)*WD ;
    UInt *p5 = p+(top-1)*WD+1;
    UInt *p6 = p+(top+1)*WD-1;
    UInt *p7 = p+(top )*WD-1;
    UInt *p8 = p+(top )*WD+1;
    Uchar *rp = r->E[top];
    for (cofs=0; cofs<WD; cofs++) {
        int d1 = d1(*p1&MASK,*p2&MASK)+df(*p3&MASK,*p4&MASK)+df(*p5&MASK,*p6&MASK)+df(*p7&MASK,*p8&MASK);
        /* 0 < d1(42) < 256*2*4 */
        *rp = d1 < EDGEDET ? 0 : PIXMAX;
        p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++; rp++;
    }
}
#endif

```

```

U11 LOOP1, LOOPO;
U11 INIT1, INITO;
U11 AR[64][4];
U11 BR[64][4][4];
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc1, cc2, cc3, ex0, ex1;
for (top=0; top<RRANGE; top+=RMGRP) { /* scan-lines */
    for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
        Uint *pp0[NCHIP], *pc0[NCHIP], *pn0[NCHIP]; Uchar *rc0[NCHIP];
        Uint *pp1[NCHIP], *pc1[NCHIP], *pn1[NCHIP]; Uchar *rc1[NCHIP];
        Uint *pp2[NCHIP], *pc2[NCHIP], *pn2[NCHIP]; Uchar *rc2[NCHIP];
        Uint *pp3[NCHIP], *pc3[NCHIP], *pn3[NCHIP]; Uchar *rc3[NCHIP];
        Uint *pp4[NCHIP], *pc4[NCHIP], *pn4[NCHIP]; Uchar *rc4[NCHIP];
        Uint *pp5[NCHIP], *pc5[NCHIP], *pn5[NCHIP]; Uchar *rc5[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++, CHIP++) f /* will be parallelized by multi-chip (M/#chip) */
            int idx = (CHIP*RRANGE+0MAP+top+rofs)*WD;
            pp0[CHIP] = p+idx+RRANGE*WD*0*WD; pn0[CHIP] = p+idx+RRANGE*WD*0*WD; rc0[CHIP] = (Uchar*)(r->E)+idx+RRANGE*WD*0;
            pp1[CHIP] = p+idx+RRANGE*WD*1*WD; pc1[CHIP] = p+idx+RRANGE*WD*1*WD; rc1[CHIP] = (Uchar*)(r->E)+idx+RRANGE*WD*1;
            pp2[CHIP] = p+idx+RRANGE*WD*2*WD; pc2[CHIP] = p+idx+RRANGE*WD*2*WD; rc2[CHIP] = (Uchar*)(r->E)+idx+RRANGE*WD*2;
            pp3[CHIP] = p+idx+RRANGE*WD*3*WD; pc3[CHIP] = p+idx+RRANGE*WD*3*WD; rc3[CHIP] = (Uchar*)(r->E)+idx+RRANGE*WD*3;
            pp4[CHIP] = p+idx+RRANGE*WD*4*WD; pc4[CHIP] = p+idx+RRANGE*WD*4*WD; rc4[CHIP] = (Uchar*)(r->E)+idx+RRANGE*WD*4;
            pp5[CHIP] = p+idx+RRANGE*WD*5*WD; pc5[CHIP] = p+idx+RRANGE*WD*5*WD; rc5[CHIP] = (Uchar*)(r->E)+idx+RRANGE*WD*5;
    }
}

//EMAX5A begin edge mapdist=1
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    /*1*/for (INITO=1,LOOP0=WD,cofs=0; LOOP0>0; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        /*@0,1*/ exe(OP_ADD,      &cofs, cofc, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
        /*@map0*/
        /*@1,0*/ exe(OP_ADD,      &cofs, pc0[CHIP], EXP_H3210, cofc, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@2,0*/ mop(OP_LDWR, 1, &r5, pofs, -1276, MSK_D0, (U11)pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@2,0*/ mop(OP_LDWR, 1, &r3, pofs, -1280, MSK_D0, (U11)pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@2,1*/ mop(OP_LDWR, 1, &r1, pofs, -1284, MSK_D0, (U11)pp0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@3,0*/ exe(OP_NOP,      &AR[3][0], OLL, EXP_H3210, 0LL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL);
        /*@3,0*/ mop(OP_LDWR, 1, &r8, pofs, 4, MSK_D0, (U11)pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@3,0*/ mop(OP_LDWR, 1, &r7, pofs, -4, MSK_D0, (U11)pc0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@4,0*/ exe(OP_NOP,      &AR[4][0], OLL, EXP_H3210, 0LL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL);
        /*@4,0*/ mop(OP_LDWR, 1, &r2, pofs, 1284, MSK_D0, (U11)pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@4,0*/ mop(OP_LDWR, 1, &r4, pofs, 1280, MSK_D0, (U11)pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@4,1*/ mop(OP_LDWR, 1, &r6, pofs, 1276, MSK_D0, (U11)pn0[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@4,1*/ exe(OP_MSSAD,   &r7, OLL, EXP_H3210, r7, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@5,0*/ exe(OP_MSSAD,   &r1, OLL, EXP_H3210, r1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@5,1*/ exe(OP_MSSAD,   &r3, OLL, EXP_H3210, r3, EXP_H3210, r4, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@5,2*/ exe(OP_MSSAD,   &r5, OLL, EXP_H3210, r5, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@6,0*/ exe(OP_MAUAH,   &r1, r3, EXP_H3210, r1, EXP_H3210, r1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@6,1*/ exe(OP_MAUAH,   &r5, r7, EXP_H3210, r5, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@7,0*/ exe(OP_MAUAH,   &r1, r5, EXP_H3210, r1, EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP_SUMHL, OLL, OP_NOP, OLL);
        /*@8,0*/ exe(OP_MCAS,    &r31, r1, EXP_H3210, 64, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@8,0*/ mop(OP_STBR, 3, &r31, rc0[CHIP]++, 0, MSK_D0, (U11)rc0[CHIP]++, WD/4, 0, 0, (U11)NULL, WD/4);

        /*map5*/
        /*@36,1*/exe(OP_ADD,      &cofs, pc5[CHIP], EXP_H3210, cofc, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@37,0*/mop(OP_LDWR, 1, &r5, pofs, -1276, MSK_D0, (U11)pp5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@37,0*/mop(OP_LDWR, 1, &r3, pofs, -1280, MSK_D0, (U11)pp5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@37,1*/mop(OP_LDWR, 1, &r1, pofs, -1284, MSK_D0, (U11)pp5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@38,0*/exe(OP_NOP,      &AR[38][0], OLL, EXP_H3210, 0LL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL);
        /*@38,0*/mop(OP_LDWR, 1, &r8, pofs, 4, MSK_D0, (U11)pc5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@38,0*/mop(OP_LDWR, 1, &r7, pofs, -4, MSK_D0, (U11)pc5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@39,0*/exe(OP_NOP,      &AR[39][0], OLL, EXP_H3210, 0LL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL);
        /*@39,0*/mop(OP_LDWR, 1, &r2, pofs, 1284, MSK_D0, (U11)pn5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@39,0*/mop(OP_LDWR, 1, &r4, pofs, 1280, MSK_D0, (U11)pn5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@39,1*/mop(OP_LDWR, 1, &r6, pofs, 1276, MSK_D0, (U11)pn5[CHIP], WD, 0, 0, (U11)NULL, WD);
        /*@39,1*/exe(OP_MSSAD,   &r7, OLL, EXP_H3210, r7, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@40,0*/exe(OP_MSSAD,   &r1, OLL, EXP_H3210, r1, EXP_H3210, r4, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@40,1*/exe(OP_MSSAD,   &r3, OLL, EXP_H3210, r3, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@40,2*/exe(OP_MSSAD,   &r5, OLL, EXP_H3210, r5, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@41,0*/exe(OP_MAUAH,   &r1, r3, EXP_H3210, r1, EXP_H3210, r1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@41,1*/exe(OP_MAUAH,   &r5, r7, EXP_H3210, r5, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@42,0*/exe(OP_MAUAH,   &r1, r5, EXP_H3210, r1, EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP_SUMHL, OLL, OP_NOP, OLL);
        /*@43,0*/exe(OP_MCAS,    &r31, r1, EXP_H3210, 64, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        /*@43,0*/mop(OP_STBR, 3, &r31, rc5[CHIP]++, 0, MSK_D0, (U11)rc5[CHIP]++, WD/4, 0, 0, (U11)NULL, WD/4);
    }
}

//EMAX5A end
}
}

//EMAX5A drain_dirty_lmm

```

filter+rmm-edge-emax6.obj

BR/row: max=9 min=1 ave=4 EA/row: max=3 min=0 ave=1 ARpass/row: max=1



Figure.3.35: Edge

3.3.9 Stereo with stencil

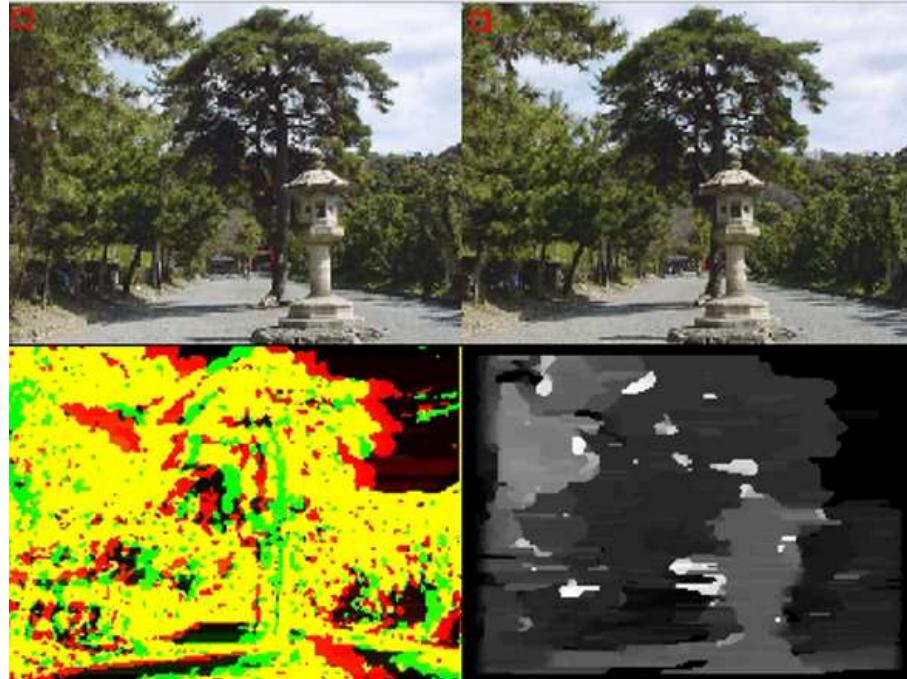


Figure 3.36: Stereo

探索サイズ 12×16 のステレオマッチングである。一度のバースト演算により 8 行分の SAD を更新する。ステンシル計算ではなく、 $\text{mapdist}=0$ である。

```

void wdifline(Uint *l, Uint *r, struct SAD2 *d, int k)
#if !defined(EMAX5) && !defined(EMAX6)
    for (top=DWIN; top<HT-DWIN; top++) { /* scan-lines */
        for (cofs=DWIN+k/2; cofs<WD-DWIN-k/2; cofs++) /* one scan-line */
            d->SAD2[top][cofs] = 0;
    }

    for (top=DWIN; top<HT-DWIN; top++) { /* scan-lines */
        Uint *lp = l + top*WD+k; /* L */
        Uint *rp = r + top*WD; /* R */
        for (pofs1=DWIN; pofs1<DWIN; pofs1++) {
            Uint *dp = &d->SAD2[top+pofs1][DWIN+k/2];
            for (cofs=0; cofs<WD-DWIN*2; cofs++) /* one scan-line */
                int x, retval = 0;
                for (x=0; x<DWIN*2; x++)
                    retval += df(((lp+cofs+x))&MASK, ((rp+cofs+x))&MASK);
                    *(dp+cofs) += retval;
            }
        }
#endif

for (top=0; top<RRANGE; top++) { /* will be parallelized by multi-chip (M/#chip) */
    for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
        int idx = CHIP*RRANGE+DWIN+top;
        for (cofs=DWIN+k/2; cofs<WD-DWIN-k/2; cofs++) /* one scan-line */
            d->SAD2[idx][cofs] = 0;
    }

    for (top=0; top<RRANGE; top++) { /* will be parallelized by multi-chip (M/#chip) */
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            int idx = CHIP*RRANGE+DWIN+top;
            Uint *lp = l + idx*WD+k; /* L */
            Uint *rp = r + idx*WD; /* R */
            for (pofs1=DWIN; pofs1<DWIN; pofs1++) {
                Uint *dp = &d->SAD2[idx+pofs1][DWIN+k/2];
                for (cofs=0; cofs<WD-DWIN*2; cofs++) /* one scan-line */
                    int x, retval = 0;
                    for (x=0; x<DWIN*2; x++)
                        retval += df(((lp+cofs+x))&MASK, ((rp+cofs+x))&MASK);
                        *(dp+cofs) += retval;
            }
        }
    }
}

```

```

U11 LOOP1, LOOPO; U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
for (top=0; top<RANGE; top++) { /* will be parallelized by multi-chip (M/#chip) */
    for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
        int idx = CHIP*RANGE+DWIN+top;
        for (cofs=DWIN+k/2; cof<DW-DWIN-k/2; cof++) /* one scan-line */
            d->SAD2[idx][cofs] = 0;
    }
}
for (top=0; top<RANGE; top++) { /* scan-lines */
    UInt *lp[NCHIP];
    UInt *dp[NCHIP], *dp1[NCHIP], *dp2[NCHIP], *dp3[NCHIP], *dp4[NCHIP], *dp5[NCHIP], *dp6[NCHIP], *dp7[NCHIP];
    UInt *dp8[NCHIP], *dp9[NCHIP], *dp10[NCHIP], *dp11[NCHIP], *dp12[NCHIP], *dp13[NCHIP];
    for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
        int idx = CHIP*RANGE+DWIN+top;
        lp[CHIP] = 1+idx*WD+k; rp[CHIP] = r+idx*WD;
        dp0[CHIP] = &d->SAD2[idx-8][DWIN+k/2];
        dp1[CHIP] = &d->SAD2[idx-7][DWIN+k/2];
        dp2[CHIP] = &d->SAD2[idx-6][DWIN+k/2];
        ;
        dpf[CHIP] = &d->SAD2[idx+7][DWIN+k/2];
    }
}
//EMAX5A begin wdifline mapdist=0
/*2*/for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*1*/for (INIT0=1,LOOP0=WD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
/*@0,1*/for (OP_ADD, &cofs, cof, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
/*map0*/
/*@1,0*/exe(OP_ADD, &rofs1, lp[CHIP], EXP_H3210, cof, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@1,1*/exe(OP_ADD, &rofs2, rofs1, 0, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,0*/mop(OP_LDWR, 1, &r2, rofs1, 4, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,0*/mop(OP_LDWR, 1, &r3, rofs1, 8, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,1*/mop(OP_LDWR, 1, &r4, rofs1, 12, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,2*/mop(OP_LDWR, 1, &r5, rofs2, 0, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,2*/mop(OP_LDWR, 1, &r7, rofs2, 4, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,3*/mop(OP_LDWR, 1, &r8, rofs2, 8, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@2,3*/mop(OP_LDWR, 1, &r9, rofs2, 12, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,0*/exe(OP_MSAD, &r22, r2, EXP_H3210, r6, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@3,0*/mop(OP_LDWR, 1, &r12, rofs1, 16, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,0*/mop(OP_LDWR, 1, &r13, rofs1, 20, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,1*/exe(OP_MSAD, &r23, r3, EXP_H3210, r7, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@3,1*/mop(OP_LDWR, 1, &r14, rofs1, 24, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,1*/mop(OP_LDWR, 1, &r15, rofs1, 28, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,2*/exe(OP_MSAD, &r24, r4, EXP_H3210, r8, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@3,2*/mop(OP_LDWR, 1, &r16, rofs1, 16, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,2*/mop(OP_LDWR, 1, &r17, rofs2, 20, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,3*/exe(OP_MSAD, &r25, r5, EXP_H3210, r9, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@3,3*/mop(OP_LDWR, 1, &r18, rofs2, 24, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@3,3*/mop(OP_LDWR, 1, &r19, rofs2, 28, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,0*/exe(OP_MSSAD, &r12, r22, EXP_H3210, r12, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,0*/mop(OP_LDWR, 1, &r2, rofs1, 32, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,0*/mop(OP_LDWR, 1, &r3, rofs1, 36, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,1*/exe(OP_MSSAD, &r13, r23, EXP_H3210, r13, EXP_H3210, r17, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,1*/mop(OP_LDWR, 1, &r4, rofs1, 40, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,1*/mop(OP_LDWR, 1, &r5, rofs1, 44, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,2*/exe(OP_MSSAD, &r14, r24, EXP_H3210, r14, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,2*/mop(OP_LDWR, 1, &r6, rofs2, 32, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,2*/mop(OP_LDWR, 1, &r7, rofs2, 36, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,3*/exe(OP_MSSAD, &r15, r25, EXP_H3210, r15, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@4,3*/mop(OP_LDWR, 1, &r8, rofs2, 40, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@4,3*/mop(OP_LDWR, 1, &r9, rofs2, 44, MSK_DO, lp[CHIP], WD, 0, 0, (U11)NULL, WD);
/*@5,0*/exe(OP_MSSAD, &r22, r12, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@5,1*/exe(OP_MSSAD, &r23, r13, EXP_H3210, r3, EXP_H3210, r7, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@5,2*/exe(OP_MSSAD, &r24, r14, EXP_H3210, r4, EXP_H3210, r8, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@5,3*/exe(OP_MSSAD, &r25, r15, EXP_H3210, r5, EXP_H3210, r9, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@6,0*/exe(OP_MAUCH, &r31, r22, EXP_H3210, r23, EXP_H3210, r24, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@7,0*/exe(OP_MAUCH, &r1, r31, EXP_H3210, r28, EXP_H3210, 0, EXP_H3210, OP_SUMHL, OLL, OP_NOP, OLL);
/*@8,0*/mop(OP_LDWR, 1, &BR[8][0][1], dp0[CHIP], cof, MSK_DO, (U11)dp0[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@8,0*/mop(OP_STWR, 3, &AR[8][0], BR[8][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*map1*/
/*@9,0*/mop(OP_LDWR, 1, &BR[9][0][1], dp1[CHIP], cof, MSK_DO, (U11)dp1[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@9,0*/exe(OP_ADD, &AR[9][0], BR[9][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@9,0*/mop(OP_STWR, 3, &AR[9][0], cof, dp1[CHIP], MSK_DO, (U11)dp1[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map2*/
/*@10,0*/mop(OP_LDWR, 1, &BR[10][0][1], dp2[CHIP], cof, MSK_DO, (U11)dp2[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@10,0*/exe(OP_ADD, &AR[10][0], BR[10][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@10,0*/mop(OP_STWR, 3, &AR[10][0], cof, dp2[CHIP], MSK_DO, (U11)dp2[CHIP], WD, 0, 1, (U11)NULL, WD);
;
/*map9*/
/*@17,0*/mop(OP_LDWR, 1, &BR[17][0][1], dp9[CHIP], cof, MSK_DO, (U11)dp9[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@17,0*/exe(OP_ADD, &AR[17][0], BR[17][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@17,0*/mop(OP_STWR, 3, &AR[17][0], cof, dp9[CHIP], MSK_DO, (U11)dp9[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map10*/
/*@18,0*/mop(OP_LDWR, 1, &BR[18][0][1], dp10[CHIP], cof, MSK_DO, (U11)dp10[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@18,0*/exe(OP_ADD, &AR[18][0], BR[18][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@18,0*/mop(OP_STWR, 3, &AR[18][0], cof, dp10[CHIP], MSK_DO, (U11)dp10[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map11*/
/*@19,0*/mop(OP_LDWR, 1, &BR[19][0][1], dp19[CHIP], cof, MSK_DO, (U11)dp19[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@19,0*/exe(OP_ADD, &AR[19][0], BR[19][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@19,0*/mop(OP_STWR, 3, &AR[19][0], cof, dp19[CHIP], MSK_DO, (U11)dp19[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map12*/
/*@20,0*/mop(OP_LDWR, 1, &BR[20][0][1], dp20[CHIP], cof, MSK_DO, (U11)dp20[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@20,0*/exe(OP_ADD, &AR[20][0], BR[20][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@20,0*/mop(OP_STWR, 3, &AR[20][0], cof, dp20[CHIP], MSK_DO, (U11)dp20[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map13*/
/*@21,0*/mop(OP_LDWR, 1, &BR[21][0][1], dp21[CHIP], cof, MSK_DO, (U11)dp21[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@21,0*/exe(OP_ADD, &AR[21][0], BR[21][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@21,0*/mop(OP_STWR, 3, &AR[21][0], cof, dp21[CHIP], MSK_DO, (U11)dp21[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map14*/
/*@22,0*/mop(OP_LDWR, 1, &BR[22][0][1], dp22[CHIP], cof, MSK_DO, (U11)dp22[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@22,0*/exe(OP_ADD, &AR[22][0], BR[22][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@22,0*/mop(OP_STWR, 3, &AR[22][0], cof, dp22[CHIP], MSK_DO, (U11)dp22[CHIP], WD, 0, 1, (U11)NULL, WD);
/*map15*/
/*@23,0*/mop(OP_LDWR, 1, &BR[23][0][1], dp23[CHIP], cof, MSK_DO, (U11)dp23[CHIP], WD, 0, 1, (U11)NULL, WD);
/*@23,0*/exe(OP_ADD, &AR[23][0], BR[23][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*@23,0*/mop(OP_STWR, 3, &AR[23][0], cof, dp23[CHIP], MSK_DO, (U11)dp23[CHIP], WD, 0, 1, (U11)NULL, WD);
}
}
//EMAX5A end
}
//EMAX5A drain_dirty_lmm

```

filter+rmm-wdifline-emax6.obj

BR/row: max=15 min=2 ave=4 EA/row: max=8 min=0 ave=2 ARpass/row: max=0

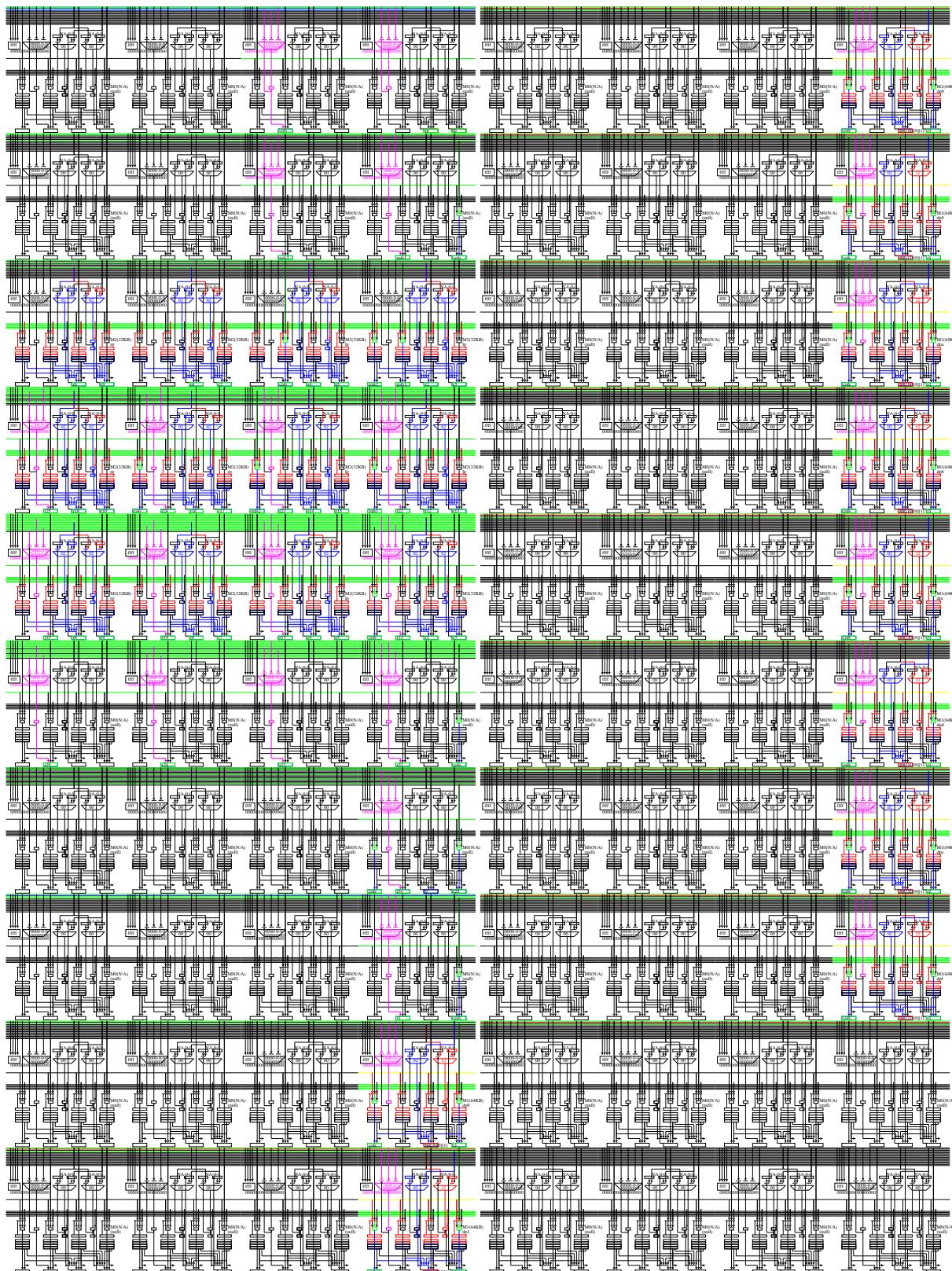


Figure 3.37: Stereo with stencil

3.4 3D-floating-point

```
cent% make -f Makefile-csim.emax6+dma all clean
cent% ..../src/csim/csim -x stencil-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean
zynq% ./stencil-zynq.emax6+dma
```

3.4.1 Grapes with stencil

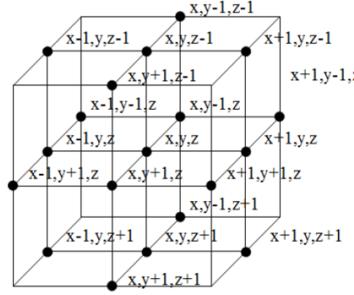


Figure.3.38: Grapes

19点浮動小数点ステンシル計算である。配列 A にステンシル性がないものの、起動回数削減のため、一度のバースト演算により 12 行分を計算する (RMGRP=12)。配列 B は再利用可能であるため mapdist=1 である。

```
grapes( float *c, float *a, float *b )
/*C3D [DP] [HT] [WD] */
/*GrA [XC] [DP] [HT] [WD] */
/*B3D [DP] [HT] [WD] */

#define NCHIP 1
#define RMGRP 12
#define OMAP 1
#define PAD 1
#define RRANGE ((HT-PAD*2)/NCHIP/OMAP)
U11 CHIP;
U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
int x, y, z;
int row, col, n;
U11 roofs, coofs, aofs, bofs, cofs;

#if !defined(EMAX5) && !defined(EMAX6)
for (z=PAD; z<WD-PAD; z++) {
    for (y=PAD; y<HT-PAD; y++) {
        for (x=PAD; x<WD-PAD; x++) {
            *(c+z*WDHT+y*WD+x) = *(b+(z-1)*WDHT+(y-1)*WD+x) * *(a+(MID-6)*WDHTDP+(z-1)*WDHT+(y-1)*WD+) /* braw00 */ /* a raw00 */
            + *(b+(z-1)*WDHT+(y )*WD+x-1) * *(a+(MID-5)*WDHTDP+(z-1)*WDHT+(y )*WD+x-1) /* braw01 */ /* a raw01 */
            + *(b+(z-1)*WDHT+(y )*WD+x) * *(a+(MID-4)*WDHTDP+(z-1)*WDHT+(y )*WD+x) /* braw01 */ /* a raw02 */
            + *(b+(z-1)*WDHT+(y )*WD+x+1) * *(a+(MID-5)*WDHTDP+(z-1)*WDHT+(y )*WD+x+1) /* braw01 */ /* a raw01 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x) * *(a+(MID-3)*WDHTDP+(z-1)*WDHT+(y+1)*WD+x) /* braw02 */ /* a raw03 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x-1) * *(a+(MID-2)*WDHTDP+(z )*WDHT+(y-1)*WD+x-1) /* braw03 */ /* a raw04 */
            + *(b+(z-1)*WDHT+(y-1)*WD+x-1) * *(a+(MID-1)*WDHTDP+(z )*WDHT+(y-1)*WD+) /* braw03 */ /* a raw05 */
            + *(b+(z-1)*WDHT+(y-1)*WD+x+1) * *(a+(MID-2)*WDHTDP+(z )*WDHT+(y-1)*WD+x+1) /* braw03 */ /* a raw04 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x-1) * *(a+(MID )*WDHTDP+(z )*WDHT+(y )*WD+x-1) /* braw04 */ /* a raw06 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x ) /* braw04 */ /* a raw06 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x+1) * *(a+(MID )*WDHTDP+(z )*WDHT+(y )*WD+x+1) /* braw04 */ /* a raw06 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x-1) * *(a+(MID+2)*WDHTDP+(z )*WDHT+(y+1)*WD+x-1) /* braw05 */ /* a raw08 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x ) * *(a+(MID+1)*WDHTDP+(z )*WDHT+(y+1)*WD+x ) /* braw05 */ /* a raw07 */
            + *(b+(z-1)*WDHT+(y+1)*WD+x+1) * *(a+(MID+2)*WDHTDP+(z )*WDHT+(y+1)*WD+x+1) /* braw05 */ /* a raw08 */
            + *(b+(z-1)*WDHT+(y-1)*WD+x ) * *(a+(MID+3)*WDHTDP+(z+1)*WDHT+(y-1)*WD+x ) /* braw06 */ /* a raw09 */
            + *(b+(z-1)*WDHT+(y-1)*WD+x-1) * *(a+(MID+5)*WDHTDP+(z+1)*WDHT+(y )*WD+x-1) /* braw07 */ /* a raw0b */
            + *(b+(z-1)*WDHT+(y )*WD+x ) * *(a+(MID+4)*WDHTDP+(z+1)*WDHT+(y )*WD+x ) /* braw07 */ /* a raw0a */
            + *(b+(z-1)*WDHT+(y )*WD+x+1) * *(a+(MID+6)*WDHTDP+(z+1)*WDHT+(y )*WD+x+1) /* braw07 */ /* a raw0b */
            + *(b+(z+1)*WDHT+(y+1)*WD+x ) * *(a+(MID+6)*WDHTDP+(z+1)*WDHT+(y+1)*WD+x );/* braw08 */ /* a raw0c */
        }
    }
}
#endif
```

```

for (z=PAD; z<DP-PAD; z++) {
    for (y=0; y<RRANGE; y+=RMGRP) {
        U11 atop[NCHIP], btop[NCHIP], ctop[NCHIP];
        U11 arow0[NCHIP], arow01[NCHIP], arow02[NCHIP], arow03[NCHIP], arow04[NCHIP], arow05[NCHIP], arow06[NCHIP], arow07[NCHIP], arow08[NCHIP],
        arow09[NCHIP], arow0a[NCHIP], arow0b[NCHIP], arow0c[NCHIP];
        U11 brow0[NCHIP], brow01[NCHIP], brow02[NCHIP], brow03[NCHIP], brow04[NCHIP], brow05[NCHIP], brow06[NCHIP], brow07[NCHIP], brow08[NCHIP];
        U11 crow0[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
            atop[CHIP] = a + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y )*WD;
            btop[CHIP] = b + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y )*WD;
            ctop[CHIP] = c + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y )*WD;
            arow00[NCHIP] = a+(MID-6)*WDHTDP+(z-1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow01[NCHIP] = a+(MID-5)*WDHTDP+(z-1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow02[NCHIP] = a+(MID-4)*WDHTDP+(z-1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow03[NCHIP] = a+(MID-3)*WDHTDP+(z-1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow04[NCHIP] = a+(MID-2)*WDHTDP+(z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow05[NCHIP] = a+(MID-1)*WDHTDP+(z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow06[NCHIP] = a+(MID-0)*WDHTDP+(z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow07[NCHIP] = a+(MID+1)*WDHTDP+(z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow08[NCHIP] = a+(MID+2)*WDHTDP+(z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow09[NCHIP] = a+(MID+3)*WDHTDP+(z+1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow0a[NCHIP] = a+(MID+4)*WDHTDP+(z+1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow0b[NCHIP] = a+(MID+5)*WDHTDP+(z+1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            arow0c[NCHIP] = a+(MID+6)*WDHTDP+(z+1)*WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow00[NCHIP] = b + (z-1) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow01[NCHIP] = b + (z-1) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow02[NCHIP] = b + (z-1) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow03[NCHIP] = b + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow04[NCHIP] = b + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow05[NCHIP] = b + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow06[NCHIP] = b + (z+1) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow07[NCHIP] = b + (z+1) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            brow08[NCHIP] = b + (z+1) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
            crow0[NCHIP] = c + (z ) *WDHT+((CHIP*RRANGE+OMAP+RRANGE+0+PAD+y-1)*WD;
        }
    }
}

//EMAX5A begin grapes mapdist=1
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*2*/for (INIT1=1,LOOP1=RMGRP,roofs=WD*4; INIT1=0; LOOP1--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
/*1*/for (INIT0=1,LOOP0=WD*2+2,coefs=(PAD-1)*4; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &coefs, INIT0?coefs:coefs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &roofs, roofs, EXP_H3210, INIT0?WD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD3, &aofs, atop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD3, &bofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD3, &cofs, ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
    /*map0*/
    mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0 -WDHT-WD )*4, MSK_D0, brow00[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#2*/
    mop(OP_LDWR, 1, &BR[2][2][1], aofs, (0+WDHTDP*(MID-6)-WDHT-WD )*4, MSK_D0, arow00[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#2 */
    exe(OP_FML, &r0, EXP_H3210, BR[2][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#3 */
    mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0 -WDHT-1)*4, MSK_D0, brow01[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#3*/
    mop(OP_LDWR, 1, &BR[3][3][0][1], bofs, (0 -WDHT+1)*4, MSK_D0, brow02[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#3*/
    mop(OP_LDWR, 1, &BR[3][3][1][1], bofs, (0 -WDHT )*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#3*/
    mop(OP_LDWR, 1, &BR[3][2][1], aofs, (0+WDHTDP*(MID-5)-WDHT-1)*4, MSK_D0, arow01[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#3 */
    mop(OP_LDWR, 1, &BR[3][2][0][1], aofs, (0+WDHTDP*(MID-5)-WDHT )*4, MSK_D0, arow01[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#3 */
    mop(OP_LDWR, 1, &BR[3][3][0][1], aofs, (0+WDHTDP*(MID-4)-WDHT )*4, MSK_D0, arow02[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#3 */
    exe(OP_FMA, &r1, r0, EXP_H3210, BR[3][0][1], EXP_H3210, BR[3][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#4 */
    exe(OP_FML, &r2, EXP_H3210, BR[3][0][0], EXP_H3210, BR[3][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#4 */
    exe(OP_FML, &r3, EXP_H3210, BR[3][1][0], EXP_H3210, BR[3][3][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#4 */
    mop(OP_LDWR, 1, &BR[4][0][1], bofs, (0 -WDHT+WD )*4, MSK_D0, brow00[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#4*/
    mop(OP_LDWR, 1, &BR[4][2][1], aofs, (0+WDHTDP*(MID-3)-WDHT+WD )*4, MSK_D0, arow03[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#4 */
    exe(OP_FMA, &r4, r1, EXP_H3210, BR[4][0][1], EXP_H3210, BR[4][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#5 */
    exe(OP_PAD, &r5, r2, EXP_H3210, r3, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#5 */
    /*stage#5*/
    mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0 -WD-1)*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#6*/
    mop(OP_LDWR, 1, &BR[6][0][0], bofs, (0 -WD+1)*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#6*/
    mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0 -WD )*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#6*/
    mop(OP_LDWR, 1, &BR[6][2][1], aofs, (0+WDHTDP*(MID-2)-WD-1)*4, MSK_D0, arow04[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#6 */
    mop(OP_LDWR, 1, &BR[6][2][0][1], aofs, (0+WDHTDP*(MID-2)-WD+1)*4, MSK_D0, arow04[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#6 */
    mop(OP_LDWR, 1, &BR[6][3][1], aofs, (0+WDHTDP*(MID-1)-WD )*4, MSK_D0, arow05[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#6 */
    exe(OP_FMA, &r6, r4, EXP_H3210, BR[6][0][1], EXP_H3210, BR[6][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#7 */
    exe(OP_FMA, &r7, r5, EXP_H3210, BR[6][0][0], EXP_H3210, BR[6][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#7 */
    exe(OP_FML, &r8, EXP_H3210, BR[6][1][1], EXP_H3210, BR[6][3][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#7 */
    mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0 -1)*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#7*/
    mop(OP_LDWR, 1, &BR[7][0][0], bofs, (0 +1)*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#7*/
    mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0 )*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#7*/
    mop(OP_LDWR, 1, &BR[7][2][1], aofs, (0+WDHTDP*(MID )-1)*4, MSK_D0, arow06[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#7 */
    mop(OP_LDWR, 1, &BR[7][2][0][1], aofs, (0+WDHTDP*(MID )+1)*4, MSK_D0, arow06[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#7 */
    exe(OP_FMA, &r9, r7, EXP_H3210, BR[7][0][1], EXP_H3210, BR[7][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#8 */
    exe(OP_FMA, &r10, r8, EXP_H3210, BR[7][0][0], EXP_H3210, BR[7][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#8 */
    exe(OP_FML, &r11, EXP_H3210, BR[7][1][1], EXP_H3210, BR[7][3][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#8 */
    mop(OP_LDWR, 1, &BR[8][0][1], bofs, (0 +1)*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#7*/
    mop(OP_LDWR, 1, &BR[8][0][0], bofs, (0 +1)*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#7*/
    mop(OP_LDWR, 1, &BR[8][1][1], bofs, (0 +WD )*4, MSK_D0, brow03[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#7*/
    mop(OP_LDWR, 1, &BR[8][2][1], aofs, (0+WDHTDP*(MID+2)+WD-1)*4, MSK_D0, arow07[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#8 */
    mop(OP_LDWR, 1, &BR[8][2][0][1], aofs, (0+WDHTDP*(MID+2)+WD+1)*4, MSK_D0, arow07[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#8 */
    mop(OP_LDWR, 1, &BR[8][3][1], aofs, (0+WDHTDP*(MID+1)+WD )*4, MSK_D0, arow08[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#8 */
    exe(OP_FMA, &r12, r9, EXP_H3210, BR[8][0][1], EXP_H3210, BR[8][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#9 */
    exe(OP_FMA, &r13, r10, EXP_H3210, BR[8][0][0], EXP_H3210, BR[8][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#9 */
    exe(OP_FML, &r14, EXP_H3210, BR[8][1][1], EXP_H3210, BR[8][3][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#9 */
    mop(OP_LDWR, 1, &BR[10][0][1], bofs, (0 +WDHT-WD )*4, MSK_D0, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#10*/
    mop(OP_LDWR, 1, &BR[10][2][1], aofs, (0+WDHTDP*(MID+3)+WDHT-WD )*4, MSK_D0, arow09[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#10*/
    exe(OP_FMA, &r15, r12, EXP_H3210, BR[10][0][1], EXP_H3210, BR[10][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#11 */
    exe(OP_FML, &r16, EXP_H3210, BR[11][0][0], EXP_H3210, BR[11][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#11 */
    exe(OP_FAD, &r17, r14, EXP_H3210, r15, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#11 */
    mop(OP_LDWR, 1, &BR[11][0][1], bofs, (0 +WDHT )*4, MSK_D0, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#11*/
    mop(OP_LDWR, 1, &BR[11][0][0], bofs, (0 +WDHT+1)*4, MSK_D0, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#11*/
    mop(OP_LDWR, 1, &BR[11][1][1], bofs, (0 +WDHT )*4, MSK_D0, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#11*/
    mop(OP_LDWR, 1, &BR[11][2][1], aofs, (0+WDHTDP*(MID+5)+WDHT-1)*4, MSK_D0, arow08[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#11 */
    mop(OP_LDWR, 1, &BR[11][2][0][1], aofs, (0+WDHTDP*(MID+5)+WDHT+1)*4, MSK_D0, arow08[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#11 */
    mop(OP_LDWR, 1, &BR[11][3][1], aofs, (0+WDHTDP*(MID+4)+WDHT )*4, MSK_D0, arow09[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#11 */
    exe(OP_FML, &r18, EXP_H3210, BR[11][0][1], EXP_H3210, BR[11][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#12 */
    exe(OP_FML, &r19, EXP_H3210, BR[11][0][0], EXP_H3210, BR[11][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#12 */
    mop(OP_LDWR, 1, &BR[12][0][1], bofs, (0 +WDHT+WD )*4, MSK_D0, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#12*/
    mop(OP_LDWR, 1, &BR[12][0][0], bofs, (0 +WDHT+1)*4, MSK_D0, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, (U11)NULL, WD*(RMGRP+PAD*2));/*st#12*/
    exe(OP_FMA, &r20, EXP_H3210, BR[12][0][1], EXP_H3210, BR[12][2][1], EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#13 */
    exe(OP_FAD, &r21, EXP_H3210, r18, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#13 */
    exe(OP_FML, &r22, EXP_H3210, r19, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#14 */
    mop(OP_STWR, 3, &r7, cofs, (0 )*4, MSK_D0, crow0[CHIP], WD*RMGRP, 0, 0, (U11)NULL, WD*RMGRP);/* stage#14*/
}
}
}

//EMAX5A end
}
//EMAX5A drain_dirty_lmm
}

```

stencil+rmm-grapes-emax6.obj

BR/row: max=12 min=3 ave=6 EA/row: max=6 min=0 ave=2 ARpass/row: max=0

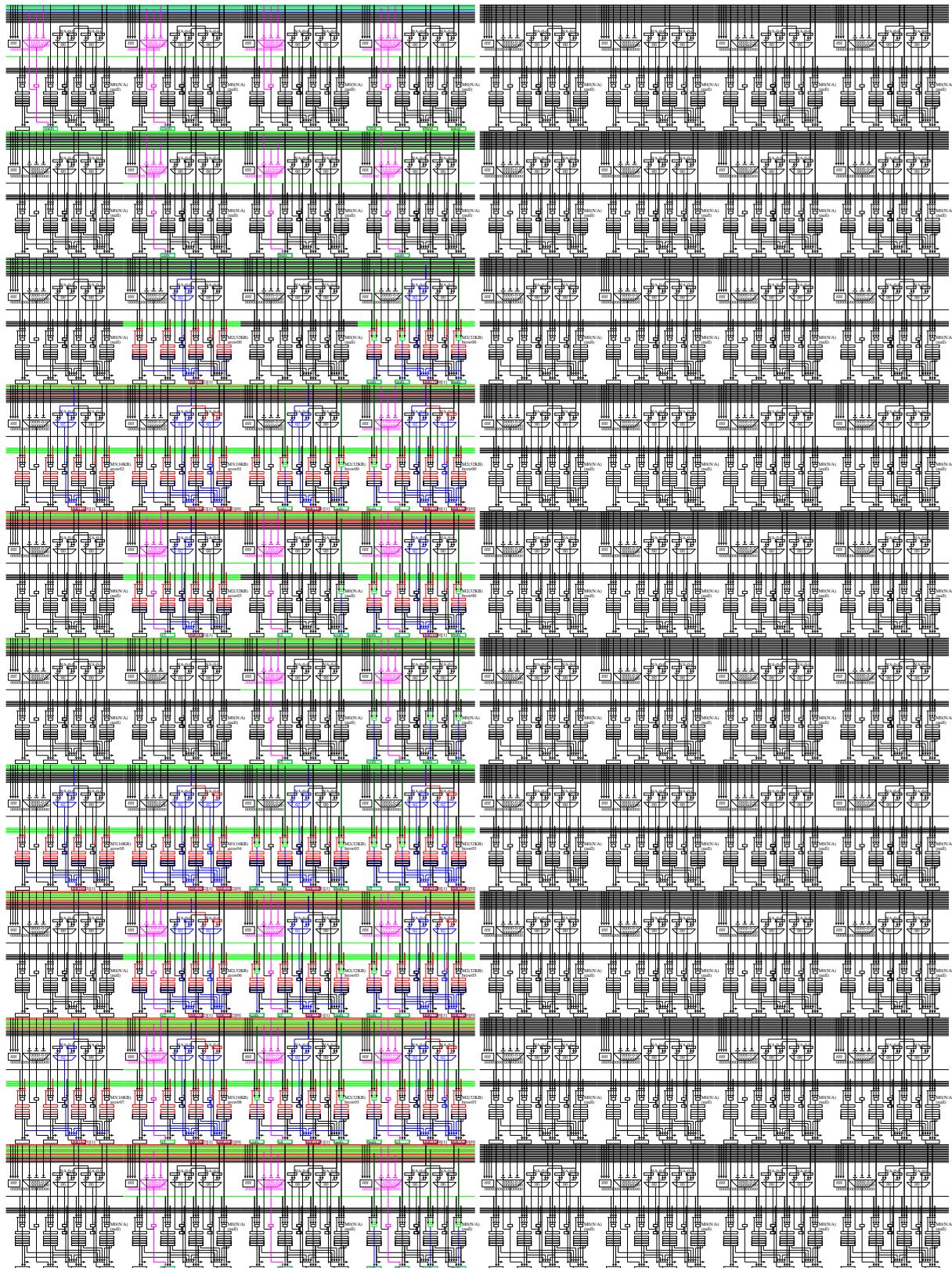


Figure.3.39: Grapes

3.4.2 Jacobi with stencil

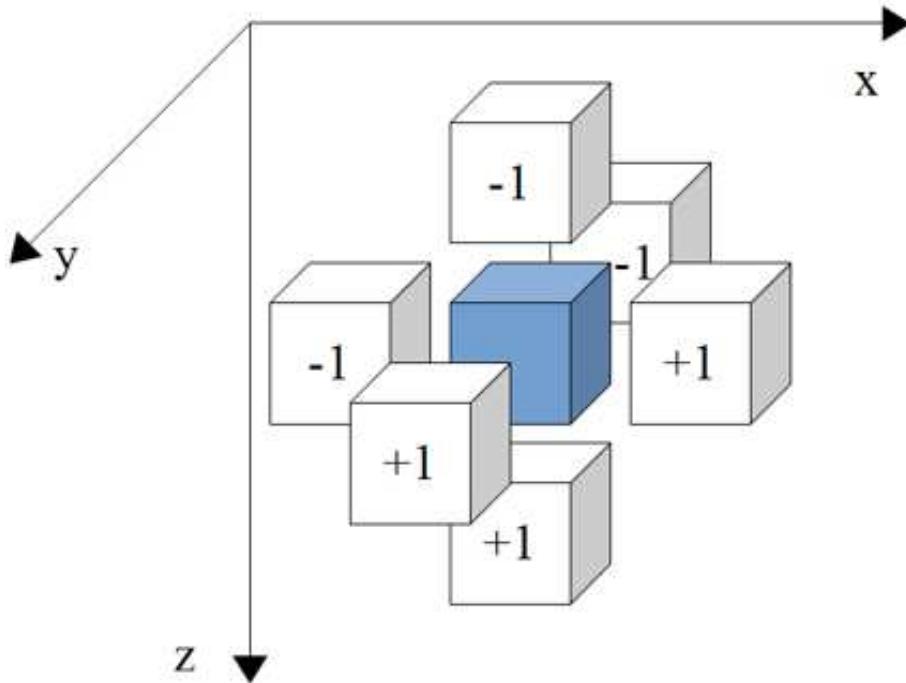


Figure.3.40: Jacobi

7 点浮動小数点ステンシル計算である。一度のバースト演算により 24 行分を計算する (RMGRP=24)。ステンシル計算であるものの、一度に 24 行分を計算するため、mapdist=0 である。

```

jacobi( float *c, float *b )
/*C3D[D][HT][WD]*/
/*B3D[D][HT][WD]*/
#ifndef NCHIP
#define RMGRP 24
#define OMAP 1
#define PAD 1
#define RRANGE ((HT-PAD*2)/NCHIP/OMAP)
#define CHIP;
#define LOOP1, LOOP0;
#define INIT1, INIT0;
#define AR[64][4]; /* output of EX in each unit */
#define BR[64][4][4]; /* output registers in each unit */
#define r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
#define r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define cc0, ccl, cc2, cc3, ex0, ex1;
#define x, y, z;
#define row, col, n;
#define roofs, coofs, aofs, bofs, cofs;
union {float f; int i;} C1, C2;
C1.f = 0.2;
C2.f = 0.3;
#define I1 C1.i;
#define I2 C2.i;
#endif

#if !defined(EMAX5) && !defined(EMAX6)
for (z=PAD; z<DP-PAD; z++) {
    for (y=PAD; y<HT-PAD; y++) {
        for (x=PAD; x<WD-PAD; x++) {
            for (c=z*WDHT+y*WD+x) = C2.f * (*((b+(z-1)*WDHT+(y )*WD+x )
                *((c+z)*WDHT+(y-1)*WD+x )
                +*((b+(z )*WDHT+(y-1)*WD+x )
                +*((b+(z )*WDHT+(y )*WD+x-1)
                +*((b+(z )*WDHT+(y )*WD+x+1)
                +*((b+(z )*WDHT+(y+1)*WD+x )
                +*((b+(z+1)*WDHT+(y )*WD+x ))
                + C1.f * *((b+(z )*WDHT+(y )*WD+x );
        }
    }
}
#endif

```

```

for (z=PAD; z<DP-PAD; z++) {
    for (y=0; y<RRANGE; y+=RMGRP) {
        U11 btop[NCHIP], ctop[NCHIP];
        U11 brow00[NCHIP], brow01[NCHIP], brow02[NCHIP], brow03[NCHIP], brow04[NCHIP];
        U11 crow0[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
            btop[CHIP] = b + (z )*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y )*WD;
            ctop[CHIP] = c + (z )*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y )*WD;
            brow00[CHIP] = b + (z-1)*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y )*WD;
            brow01[CHIP] = b + (z+1)*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y )*WD;
            brow02[CHIP] = b + (z )*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y-1)*WD;
            brow03[CHIP] = b + (z )*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y )*WD/* not used for RMGRP>1 */;
            brow04[CHIP] = b + (z )*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y+1)*WD/* not used for RMGRP>1 */;
            crow0[CHIP] = c + (z )*WDHT+(CHIP*RRANGE*OMAP+RRANGE*O+PAD+y )*WD;
        }
    }
}

//EMAX5A begin jacobi mapdist=0 /* 7 PAD>0 の場合, PLOAD と LOAD 領域が一部重複. load 中の LMM にも PLOAD を取り込むために渋滞が発生する */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/* 2*for (INIT1=1, LOOP1=RMGRP, roofs=0-WD*4; LOOP1--; INIT1=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
/* 1*for (INIT0=1, LOOP0=WD-PAD*2, coofs=(PAD-1)*4; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &coofs, INIT0?coofs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &roofs, roofs, EXP_H3210, INIT0?WD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADDS, &bofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADDS, &bofs, ctop[CHIP], EXP_H3210, roofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
/* /map0*/
    mop(OP_LDWR, 1, &BR[2][0][1], bofs, (O -WDHT )*4, MSK_DO, brow00[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][2][1], bofs, (O +WDHT )*4, MSK_DO, brow01[CHIP], WD*RMGRP, 0, 0, CHIP/*NULL, WD*RMGRP); /* stage#2 */
    exe(OP_FAD, kr0, BR[2][0][1], EXP_H3210, BR[2][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    mop(OP_LDWR, 1, &BR[3][0][1], bofs, (O -WD )*4, MSK_DO, brow02[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#3 */
    exe(OP_FAD, kr1, r0, EXP_H3210, BR[3][0][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    mop(OP_LDWR, 1, &BR[4][0][1], bofs, (O -1)*4, MSK_DO, brow02[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#4 */
    mop(OP_LDWR, 1, &BR[4][1][1], bofs, (O +4), MSK_DO, brow02[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#4 */
    mop(OP_LDWR, 1, &BR[4][2][1], bofs, (O +1)*4, MSK_DO, brow02[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#4 */
    exe(OP_FAD, kr2, r1, EXP_H3210, BR[4][0][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
    exe(OP_FML, kr3, II, EXP_H3210, BR[4][1][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
    mop(OP_LDWR, 1, &BR[5][0][1], bofs, (O +WD )*4, MSK_DO, brow02[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
    exe(OP_FAD, kr4, r2, EXP_H3210, BR[5][0][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
    exe(OP_FAD, kr5, r4, EXP_H3210, BR[4][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
    exe(OP_FMA, kr6, r3, EXP_H3210, r5, EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
    mop(OP_STWR, 3, kr6, coefs, (O +4, MSK_DO, crow0[CHIP], WD*RMGRP, 0, 0, /NULL, WD*RMGRP); /* stage#8 */
}
}

//EMAX5A end
}
//EMAX5A drain_dirty_lmm
}

```

stencil+rmm-jacobi-emax6.obj

BR/row: max=7 min=2 ave=4 EA/row: max=3 min=0 ave=0 ARpass/row: max=0

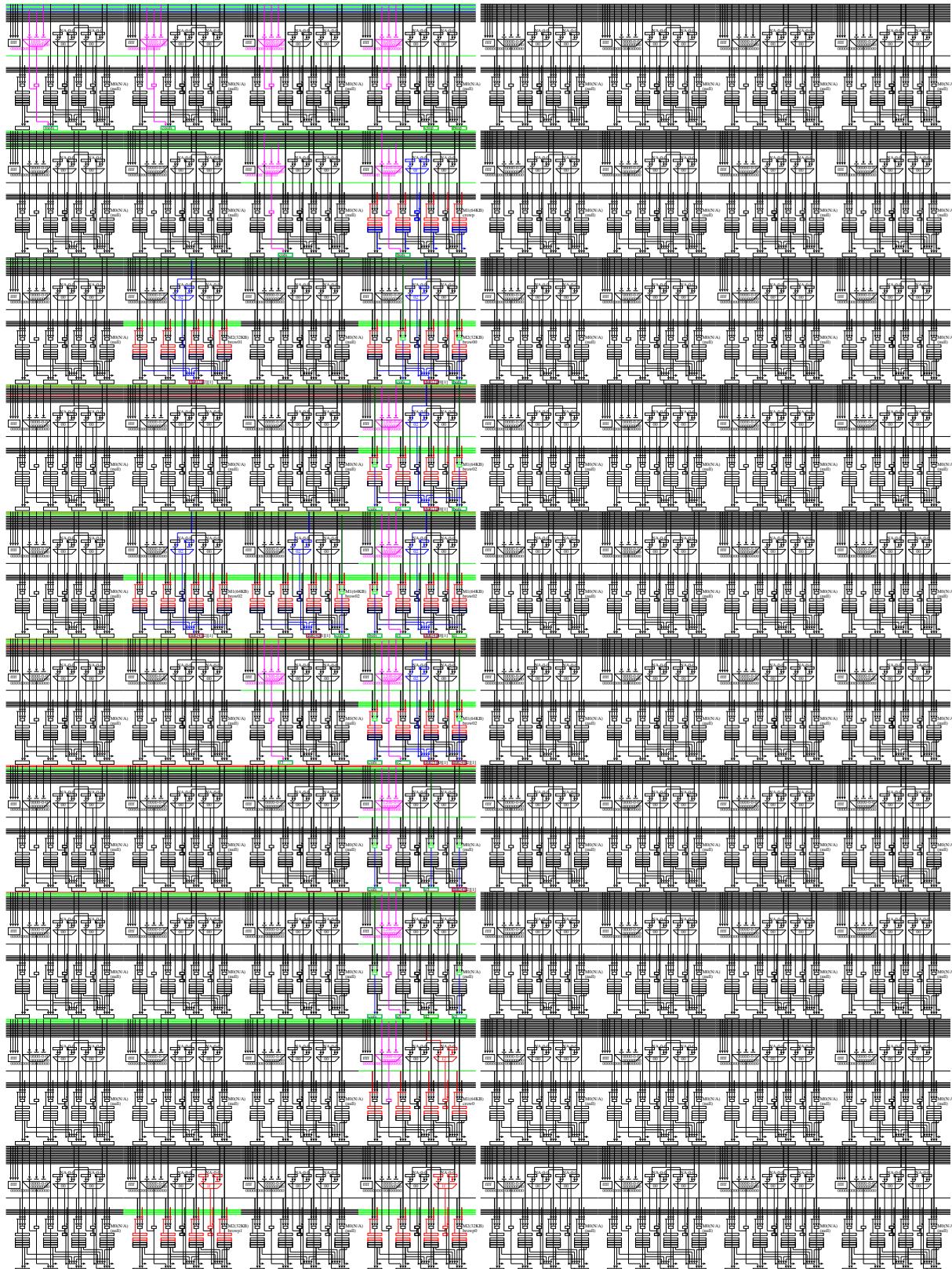


Figure.3.41: Jacobi

3.4.3 Fd6 with stencil

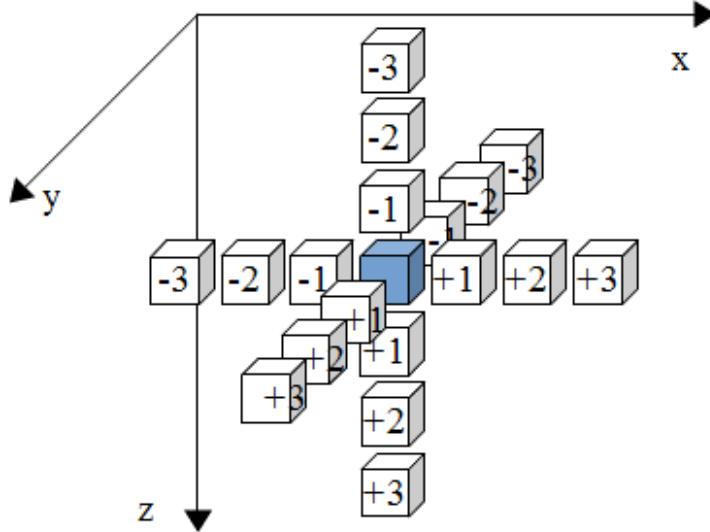


Figure.3.42: Fd6

次数 3 の 19 点浮動小数点ステンシル計算である。一度のバースト演算により 24 行分を計算する (RM-GRP=24)。ステンシル計算であるものの、一度に 24 行分を計算するため、mapdist=0 である。

```

fd6( float *c, float *b )
/*C3D[DPI][HT][WD]*/
/*B3D[DPI][HT][WD]*/
#define NCHIP 1
#define RMGRP 24
#define OMAP 1
#define PAD 3
#define RRANGE ((HT-PAD*2)/NCHIP/OMAP)
U11 CHIP;
U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
int x, y, z;
int row, col, n;
U11 roofs, coofs, aofs, bofs, cofs;
union {float f; int i;} C1, C2, C3, C4;
C1.f = 0.1;
C2.f = 0.2;
C3.f = 0.4;
C4.f = 0.8;
U11 I1 = C1.i;
U11 I2 = C2.i;
U11 I3 = C3.i;
U11 I4 = C4.i;

#if !defined(EMAX5) && !defined(EMAX6)
for (z=PAD; z<DP-PAD; z++) {
    for (y=PAD; y<HT-PAD; y++) {
        for (x=PAD; x<WD-PAD; x++) {
            *(c+z*WDHT+y*WD+x) =
                C4.f *(*b+((z-3)*WDHT)+(y )*WD+x )
                + *b+((z )*WDHT)+(y-3)*WD+x )
                + *b+((z )*WDHT)+(y )*WD+x-3)
                + *b+((z )*WDHT)+(y )*WD+x+3)
                + *b+((z )*WDHT)+(y+3)*WD+x )
                + *b+((z+3)*WDHT)+(y )*WD+x )
                + *b+((z+2)*WDHT)+(y )*WD+x )
                + C3.f *(*b+((z-2)*WDHT)+(y )*WD+x )
                + *b+((z )*WDHT)+(y-2)*WD+x )
                + *b+((z )*WDHT)+(y )*WD+x-2)
                + *b+((z )*WDHT)+(y )*WD+x+2)
                + *b+((z )*WDHT)+(y+2)*WD+x )
                + *(b+((z+2)*WDHT)+(y )*WD+x ))
                + C2.f *(*b+((z-1)*WDHT)+(y )*WD+x )
                + *b+((z )*WDHT)+(y-1)*WD+x )
                + *b+((z )*WDHT)+(y )*WD+x-1)
                + *b+((z )*WDHT)+(y )*WD+x+1)
                + *b+((z )*WDHT)+(y+1)*WD+x )
                + *(b+((z+1)*WDHT)+(y )*WD+x ))
                + C1.f * *b+((z )*WDHT)+(y )*WD+x );
        }
    }
}
#endif

```

```

for (z=PAD; z<DP-PAD; z++) {
    for (y=0; y<RRANGE; y+=RMGRP) {
        U11 btop[NCHIP], ctop[NCHIP];
        U11 brow00[NCHIP], brow01[NCHIP], brow02[NCHIP], brow03[NCHIP], brow04[NCHIP], brow05[NCHIP], brow06[NCHIP];
        U11 brow07[NCHIP], brow08[NCHIP], brow09[NCHIP], brow0a[NCHIP], brow0b[NCHIP], brow0c[NCHIP];
        U11 crow0[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
            btop[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            ctop[CHIP] = c + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow00[CHIP] = b + (z-3) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow01[CHIP] = b + (z-2) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow02[CHIP] = b + (z-1) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow03[CHIP] = b + (z+1) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow04[CHIP] = b + (z+2) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow05[CHIP] = b + (z+3) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD;
            brow06[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y-3)*WD;
            brow07[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y-2)*WD; /* not used for RMGRP>1 */
            brow08[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y-1)*WD; /* not used for RMGRP>1 */
            brow09[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y )*WD; /* not used for RMGRP>1 */
            brow0a[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y-1)*WD; /* not used for RMGRP>1 */
            brow0b[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y-2)*WD; /* not used for RMGRP>1 */
            brow0c[CHIP] = b + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y-3)*WD; /* not used for RMGRP>1 */
            crow0[CHIP] = c + (z ) *WDHT+(CHIP*RRANGE+OMAP+RANGE+0+PAD+y );
        }
    }
}

//EMAX5A begin fd6 mapdist=0 /* 11 */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    /*2*/for (INIT1=1,LOOP1=RMGRP,roofs=0-WD*4; LOOP1--; INIT1=0) /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
    /*1*/for (INIT0=1,LOOP0=WD-PAD*2,coefs=(PAD-1)*4; LOOP0--; INIT0=0) /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &coefs, INIT0?coefs:coefs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD, &roofs, roofs, EXP_H3210, INIT0?WD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD3, &bofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
        exe(OP_ADD3, &cofs, ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
        /*map0*/
        mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0 -WDHT*3 )*4, MSK_DO, brow00[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#2 */
        mop(OP_LDWR, 1, &BR[2][2][1], bofs, (0 +WDHT*3 )*4, MSK_DO, brow05[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#2 */
        exe(OP_FAD, kr3, BR[2][0][1], EXP_H3210, BR[2][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
        mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0 -WDHT*2 )*4, MSK_DO, brow01[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#3 */
        mop(OP_LDWR, 1, &BR[3][2][1], bofs, (0 +WDHT*2 )*4, MSK_DO, brow04[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#3 */
        exe(OP_FAD, kr2, BR[3][0][1], EXP_H3210, BR[3][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
        mop(OP_LDWR, 1, &BR[4][0][1], bofs, (0 -WDHT*1 )*4, MSK_DO, brow02[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#4 */
        mop(OP_LDWR, 1, &BR[4][2][1], bofs, (0 +WDHT*1 )*4, MSK_DO, brow03[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#4 */
        exe(OP_FAD, kr1, BR[4][0][1], EXP_H3210, BR[4][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
        mop(OP_LDWR, 1, &BR[5][0][1], bofs, (0 -WD*3 )*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
        mop(OP_LDWR, 1, &BR[5][0][0], bofs, (0 +WD*3 )*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
        mop(OP_LDWR, 1, &BR[5][1][1], bofs, (0 -3)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
        mop(OP_LDWR, 1, &BR[5][1][0], bofs, (0 +3)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
        exe(OP_FAD, kr13, BR[5][0][1], EXP_H3210, BR[5][0][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
        exe(OP_FAD, kr13, BR[5][1][1], EXP_H3210, BR[5][1][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0 +WD*2)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][0][0], bofs, (0 -WD*2)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0 -2)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][1][0], bofs, (0 +2)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
        exe(OP_FAD, kr12, BR[6][0][1], EXP_H3210, BR[6][0][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        exe(OP_FAD, kr22, BR[6][1][1], EXP_H3210, BR[6][1][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        exe(OP_FAD, kr23, r13, EXP_H3210, r23, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0 +WD*1)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
        mop(OP_LDWR, 1, &BR[7][0][0], bofs, (0 -WD*1)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
        mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0 -1)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
        mop(OP_LDWR, 1, &BR[7][1][0], bofs, (0 +1)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
        exe(OP_FAD, kr11, BR[7][0][1], EXP_H3210, BR[7][0][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        exe(OP_FAD, kr21, BR[7][1][1], EXP_H3210, BR[7][1][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        exe(OP_FAD, kr22, r12, EXP_H3210, r22, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        exe(OP_FAD, kr3, r23, EXP_H3210, r3, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        mop(OP_LDWR, 1, &BR[8][0][1], bofs, (0 +4)*4, MSK_DO, brow06[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#8 */
        exe(OP_FML, kr10, BR[8][0][1], EXP_H3210, I1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_FAD, kr21, r11, EXP_H3210, r21, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_FAD, kr2, r22, EXP_H3210, r2, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_FMA, kr13, r10, EXP_H3210, r3, EXP_H3210, I4, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
        exe(OP_FAD, kr1, r21, EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
        exe(OP_FMA, kr12, r13, EXP_H3210, r2, EXP_H3210, I3, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
        exe(OP_FMA, kr11, r12, EXP_H3210, r1, EXP_H3210, I2, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12 */
        mop(OP_STWR, 3, kr11, coefs, (0 +4)*4, MSK_DO, crow0[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#12 */
    }
}
}

//EMAX5A end
}

//EMAX5A drain_dirty_lmm
}

```

stencil+rmm-fd6-emax6.obj

BR/row: max=12 min=2 ave=6 EA/row: max=4 min=0 ave=1 ARpass/row: max=0

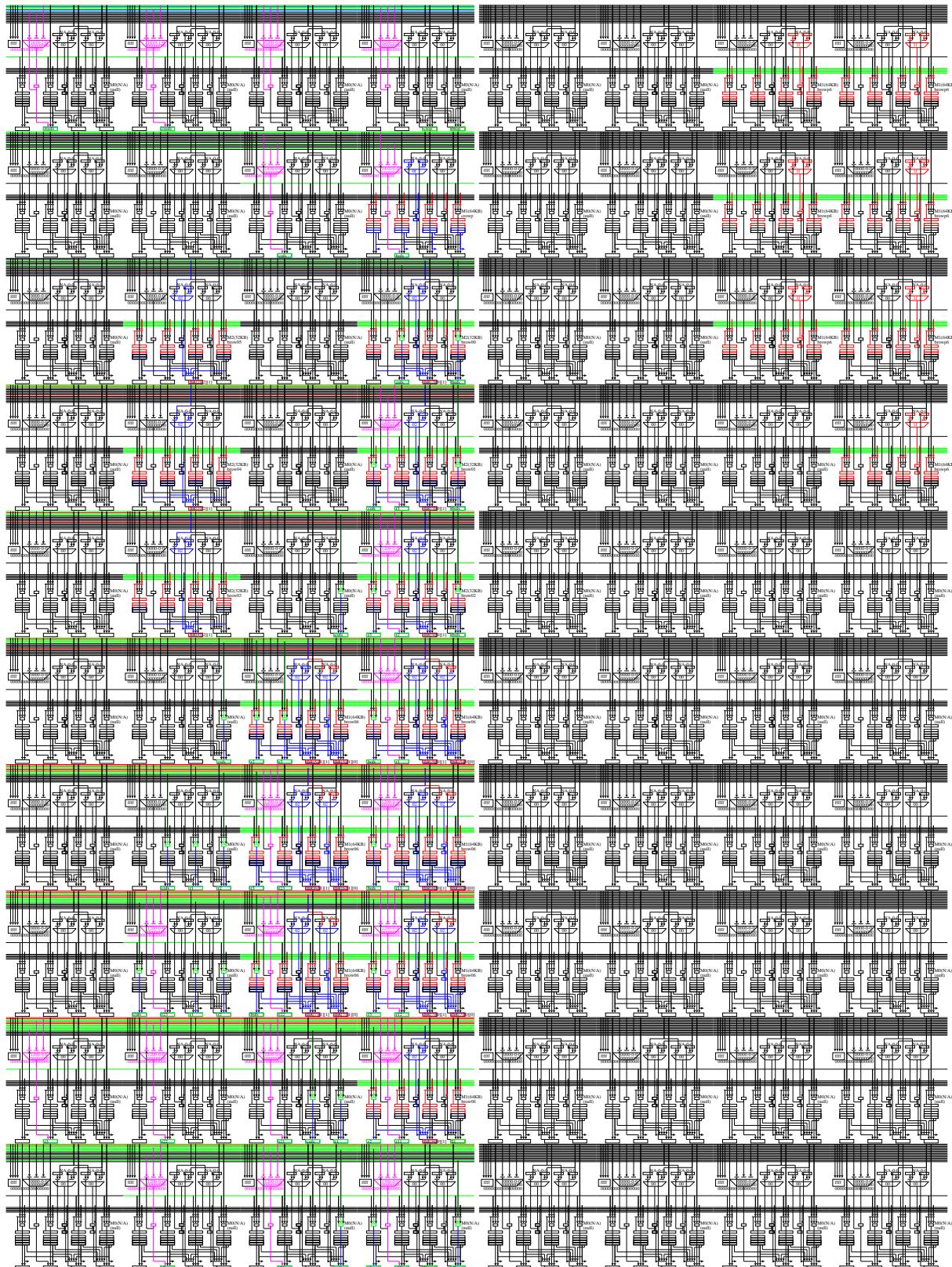


Figure.3.43: Fd6

3.4.4 Resid with stencil

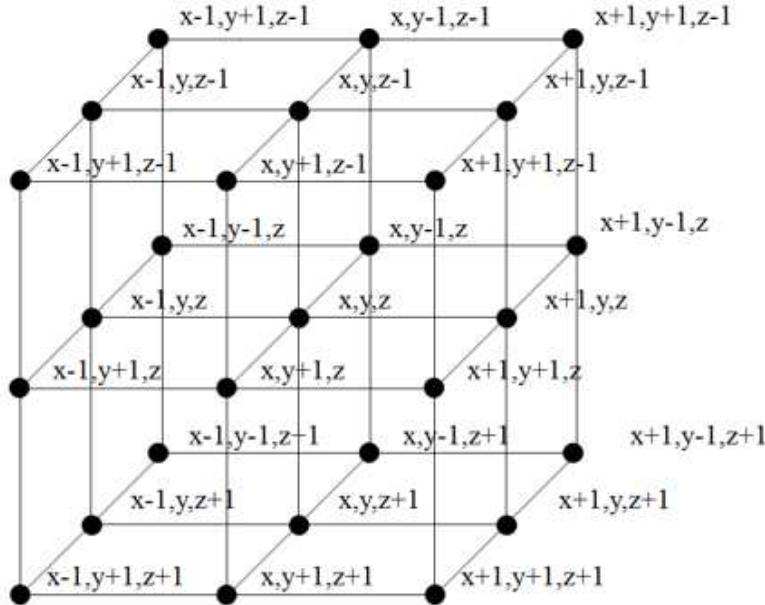


Figure.3.44: Resid

27 点浮動小数点ステンシル計算である。一度のバースト演算により 24 行分を計算する (RMGRP=24)。ステンシル計算であるものの、一度に 24 行分を計算するため、mapdist=0 である。

```

resid( float *d, float *b, float *c )
/*3D[DP][HT][WD]*/
/*B3D[DP][HT][WD]*/
/*C3D[DP][HT][WD]*/
#define NCHIP 1
#define RMGRP 24
#define OMAP 1
#define PAD 1
#define BRANGE ((HT-PAD*2)/NCHIP/OMAP)
U11 CHIP;
U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, cc1, cc2, cc3, ex0, ex1;
int x, y, z;
int row, col, n;
U11 roofs, coofs, bofs, cofs, dofs;
union {float f; int i;} A0, A1, A2, A3;
A0.f = -0.1; A1.f = -0.2; A2.f = -0.3; A3.f = -0.4;
U11 IO = A0.i; U11 I1 = A1.i; U11 I2 = A2.i; U11 I3 = A3.i;

#if !defined(EMAX5) && !defined(EMAX6)
for (z=PAD; <DP-PAD; z++) {
    for (y=PAD; y<HT-PAD; y++) {
        for (x=PAD; x<WD-PAD; x++) {
            *(d+z*WDHT+y*WD+x) =
                + A0.f * *(b+(z )*WDHT+(y )*WD+x )
                + A1.f * *(b+(z-1)*WDHT+(y )*WD+x )
                + *(b+(z )*WDHT+(y-1)*WD+x )
                + *(b+(z )*WDHT+(y )*WD+x-1)
                + *(b+(z )*WDHT+(y+1)*WD+x )
                + *(b+(z+1)*WDHT+(y )*WD+x )
                + *(b+(z+1)*WDHT+(y+1)*WD+x )
                + A2.f * *(b+(z-1)*WDHT+(y-1)*WD+x )
                + *(b+(z-1)*WDHT+(y )*WD+x-1)
                + *(b+(z-1)*WDHT+(y )*WD+x+1)
                + *(b+(z-1)*WDHT+(y+1)*WD+x )
                + *(b+(z-1)*WDHT+(y+1)*WD+x-1)
                + *(b+(z-1)*WDHT+(y+1)*WD+x+1)
                + *(b+(z+1)*WDHT+(y-1)*WD+x )
                + *(b+(z+1)*WDHT+(y )*WD+x-1)
                + *(b+(z+1)*WDHT+(y )*WD+x+1)
                + *(b+(z+1)*WDHT+(y+1)*WD+x )
                + *(b+(z+1)*WDHT+(y+1)*WD+x-1)
                + *(b+(z+1)*WDHT+(y+1)*WD+x+1);
            + A3.f * *(b+(z-1)*WDHT+(y-1)*WD+x-1)
                + *(b+(z-1)*WDHT+(y-1)*WD+x+1)
                + *(b+(z-1)*WDHT+(y+1)*WD+x-1)
                + *(b+(z-1)*WDHT+(y+1)*WD+x+1)
                + *(b+(z+1)*WDHT+(y+1)*WD+x-1)
                + *(b+(z+1)*WDHT+(y-1)*WD+x+1)
                + *(b+(z+1)*WDHT+(y+1)*WD+x-1)
                + *(b+(z+1)*WDHT+(y+1)*WD+x+1));
        }
    }
}
#endif

```

```

for (z=PAD; z<DP-PAD; z++) {
    for (y=0; y<RRANGE; y+=RMGRP) {
        U11 btop[NCHIP], ctop[NCHIP], dtop[NCHIP];
        U11 brow0[NCHIP], brow1[NCHIP], brow02[NCHIP], brow03[NCHIP], brow05[NCHIP], brow06[NCHIP], brow07[NCHIP], brow08[NCHIP];
        U11 crow0[NCHIP];
        U11 drow0[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
            btop[CHIP] = b + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD;
            ctop[CHIP] = c + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD;
            dtop[CHIP] = d + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD;
            brow00[NCHIP] = b + (z-1 ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y-1 )*WD;
            brow01[NCHIP] = b + (z-1 ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD; /* not used for RMGRP>1 */
            brow02[NCHIP] = b + (z-1 ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y-1 )*WD; /* not used for RMGRP>1 */
            brow03[NCHIP] = b + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y-1 )*WD;
            brow04[NCHIP] = b + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD; /* not used for RMGRP>1 */
            brow05[NCHIP] = b + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y-1 )*WD; /* not used for RMGRP>1 */
            brow06[NCHIP] = b + (z+1 ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y-1 )*WD;
            brow07[NCHIP] = b + (z+1 ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD; /* not used for RMGRP>1 */
            brow08[NCHIP] = b + (z+1 ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y-1 )*WD; /* not used for RMGRP>1 */
            crow0[NCHIP] = c + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD;
            drow0[NCHIP] = d + (z ) *WDHT+(CHIP*RANGE*OMAP+RRANGE*O+PAD+y )*WD;
        }
    }
}

//EMAX5A begin resid mapdist=0 /* 12 */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*2*/for (INITI=1,LOOP0=RMGRP,roofs=0-WD*4; INITI<0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
/*1*/for (INITI=0,LOOP0=WD-PAD+2,coefs=(PAD-1)*4; LOOP0--; INITI=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
    exec(OP_ADD, &coefs, INITI?coefs, EXP_H3210, OP_4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL); /* stage#0 */
    exec(OP_ADD, &roofs, roofs, EXP_H3210, INITI?WD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL); /* stage#0 */
    exec(OP_ADDS, &btop, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL); /* stage#1 */
    exec(OP_ADDS, &cofs, ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL); /* stage#1 */
    exec(OP_ADDS, &dofs, dtop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL); /* stage#1 */
/*map0*/
mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0 -WDHT-WD-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#2 */
mop(OP_LDWR, 1, &BR[2][0][0], bofs, (0 -WDHT-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#2 */
mop(OP_LDWR, 1, &BR[2][1][1], bofs, (0 -WDHT-WD-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#2 */
exec(OP_FML, &r0, BR[2][0][1], EXP_H3210, I3, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exec(OP_FML, &r1, BR[2][0][0], EXP_H3210, I2, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exec(OP_FML, &r2, BR[2][1][1], EXP_H3210, I3, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exec(OP_FML, &r3, r0, EXP_H3210, BR[3][0][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
exec(OP_FML, &r4, r1, EXP_H3210, BR[3][0][0], EXP_H3210, I1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
exec(OP_FML, &r5, r2, EXP_H3210, BR[3][1][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0 -WDHT-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][0], bofs, (0 -WDHT-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#3 */
mop(OP_LDWR, 1, &BR[3][1][1], bofs, (0 -WDHT-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#3 */
exec(OP_FMA, &r6, r3, EXP_H3210, BR[4][0][1], EXP_H3210, I3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exec(OP_FMA, &r7, r4, EXP_H3210, BR[4][0][0], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exec(OP_FMA, &r8, r5, EXP_H3210, BR[4][1][1], EXP_H3210, I3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][1], bofs, (0 -WD-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][0], bofs, (0 -WD)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][1][1], bofs, (0 -WD-1)*4, MSK_DO, brow00[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
exec(OP_FMA, &r0, r6, EXP_H3210, BR[5][0][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exec(OP_FMA, &r1, r7, EXP_H3210, BR[5][0][0], EXP_H3210, I1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exec(OP_FMA, &r2, r8, EXP_H3210, BR[5][1][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0 -1)*4, MSK_DO, brow03[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][0], bofs, (0 -WD-1)*4, MSK_DO, brow03[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0 -1)*4, MSK_DO, brow03[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#6 */
exec(OP_FMA, &r3, r0, EXP_H3210, BR[6][0][1], EXP_H3210, I1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exec(OP_FMA, &r4, r1, EXP_H3210, BR[6][0][0], EXP_H3210, I0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exec(OP_FMA, &r5, r2, EXP_H3210, BR[6][1][1], EXP_H3210, I1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0 +WD-1)*4, MSK_DO, brow03[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][0], bofs, (0 +WD-1)*4, MSK_DO, brow03[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0 +WD-1)*4, MSK_DO, brow03[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#7 */
exec(OP_FMA, &r6, r3, EXP_H3210, BR[7][0][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exec(OP_FMA, &r7, r4, EXP_H3210, BR[7][0][0], EXP_H3210, I1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exec(OP_FMA, &r8, r5, EXP_H3210, BR[7][1][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
mop(OP_LDWR, 1, &BR[8][0][1], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#9 */
mop(OP_LDWR, 1, &BR[8][0][0], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#9 */
mop(OP_LDWR, 1, &BR[8][1][1], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#9 */
exec(OP_FMA, &r9, r0, EXP_H3210, BR[8][0][1], EXP_H3210, I3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exec(OP_FMA, &r10, r1, EXP_H3210, BR[8][0][0], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exec(OP_FMA, &r11, r2, EXP_H3210, BR[8][1][1], EXP_H3210, I3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
mop(OP_LDWR, 1, &BR[9][0][1], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#9 */
mop(OP_LDWR, 1, &BR[9][0][0], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#9 */
mop(OP_LDWR, 1, &BR[9][1][1], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#9 */
exec(OP_FMA, &r12, r3, EXP_H3210, BR[9][0][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exec(OP_FMA, &r13, r4, EXP_H3210, BR[9][0][0], EXP_H3210, I1, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exec(OP_FMA, &r14, r5, EXP_H3210, BR[9][1][1], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
mop(OP_LDWR, 1, &BR[10][0][1], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#10 */
mop(OP_LDWR, 1, &BR[10][0][0], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#10 */
mop(OP_LDWR, 1, &BR[10][1][1], bofs, (0 +WDHT-1)*4, MSK_DO, brow06[NCHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#10 */
exec(OP_FMA, &r15, r6, EXP_H3210, BR[10][0][1], EXP_H3210, I3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
exec(OP_FMA, &r16, r7, EXP_H3210, BR[10][0][0], EXP_H3210, I2, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
exec(OP_FMA, &r17, r8, EXP_H3210, BR[10][1][1], EXP_H3210, I3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
mop(OP_LDWR, 1, &BR[11][0][1], bofs, (0 +WDHT-1)*4, MSK_DO, drow0[NCHIP], WD+RMGRP, 0, 0, NULL, WD+RMGRP); /* stage#11 */
exec(OP_FAD, &r1, r6, EXP_H3210, BR[11][0][1], EXP_H3210, I0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12 */
exec(OP_FAD, &r2, r7, EXP_H3210, r8, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12 */
exec(OP_FAD, &r0, r1, EXP_H3210, r2, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13 */
mop(OP_STWR, 3, &r0, dofs, (0 )*4, MSK_DO, drow0[NCHIP], WD+RMGRP, 0, 0, NULL, WD+RMGRP); /* stage#13 */
}
}

//EMAX5A end
}

//EMAX5A drain_dirty_lmm
}

```

stencil+rmm-resid-emax6.obj

BR/row: max=11 min=3 ave=7 EA/row: max=3 min=0 ave=2 ARpass/row: max=0

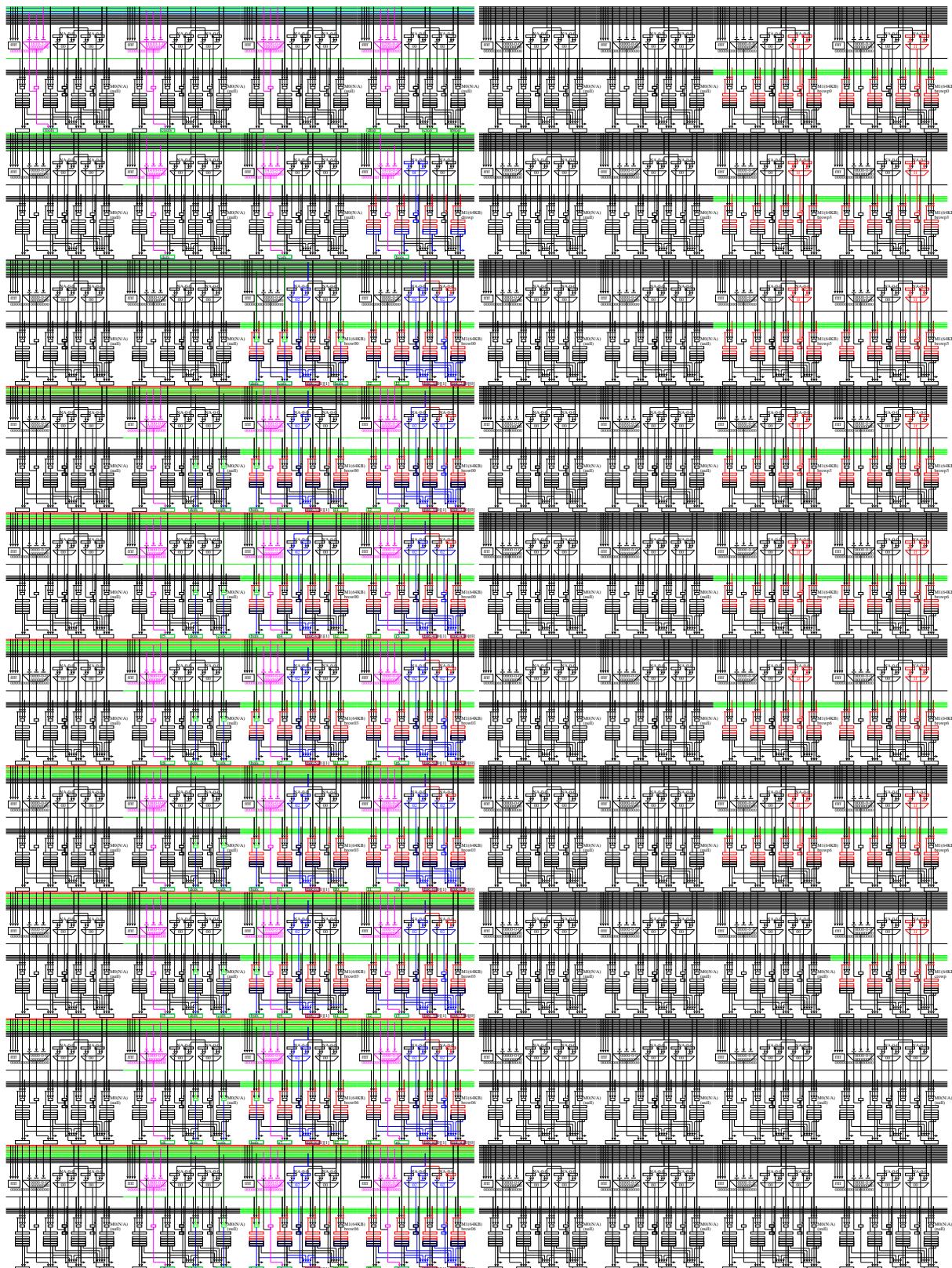


Figure.3.45: Resid

3.4.5 Wave2d with stencil

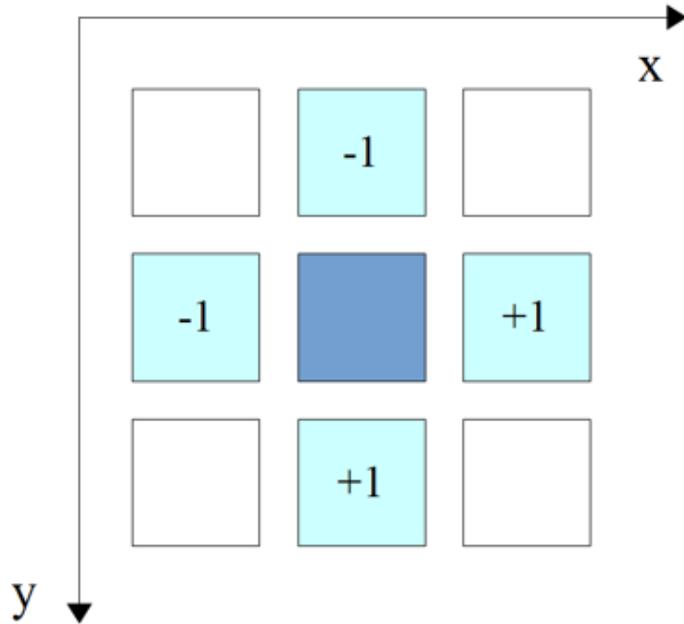


Figure.3.46: Wave2d

5点浮動小数点ステンシル計算である。一度のバースト演算により 24 行分を計算する (RMGRP=24)。ステンシル計算であるものの、一度に 24 行分を計算するため、mapdist=0 である。なお、未使用 stage を使用した PLOAD により性能向上が可能である。ただし、PAD ≥ 1 の場合、LOAD 対象 LMM と PLOAD 対象 LMM の領域が一部重複するため、PLOAD 向け AXI トランザクションが LOAD 対象 LMM にもヒットする。load サイクルにおいて AXI が待たされ、AXI が渋滞することがある。

```

wave2d( float *z2, float *z0, float *z1 )
/*WZ2[HT][WD]*/
/*WZ0[HT][WD]*/
/*WZ1[HT][WD]*/
#define NCHIP 1
#define RMGRP 24
#define OMAP 1
#define PAD 1
#define R RANGE ((HT-PAD*2)/NCHIP/OMAP)
U11 CHIP;
U11 LOOP1, LOOPO;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, cc1, cc2, cc3, ex0, ex1;
int x, y, z;
int row, col, n;
U11 roofs, coofs, z0ofs, z1ofs, z2ofs;
union {float f; int i;} C1, C2, C3, C4;
C1.f = 2.00;
C2.f = -1.00;
C3.f = 0.25;
C4.f = -4.00;
U11 I1 = C1.i;
U11 I2 = C2.i;
U11 I3 = C3.i;
U11 I4 = C4.i;

#if !defined(EMAX5) && !defined(EMAX6)
for (y=PAD; y<HT-PAD; y++) {
    for (x=PAD; x<WD-PAD; x++) {
        *(z2+y*WD+x) = C1.f * *(z1+y*WD+x)
                      + C2.f * *(z0+y*WD+x)
                      + C3.f * *(z1+(y+1)*WD+x)
                      + *(z1+(y-1)*WD+x)
                      + *(z1+(y )*WD+x-1)
                      + *(z1+(y )*WD+x+1) + C4.f * *(z1+y*WD+x);
    }
}
#endif

```

```

for (y=0; y<RRANGE; y+=RMGRP) {
    U11 z0top[NCHIP], z1top[NCHIP], z2top[NCHIP];
    U11 z1row0[NCHIP];
    U11 z1row00[NCHIP], z1row01[NCHIP], z1row02[NCHIP];
    U11 z2row0[NCHIP];
    for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
        z0top[CHIP] = z0           + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y) *WD;
        z1top[CHIP] = z1           + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y) *WD;
        z2top[CHIP] = z2           + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y) *WD;
        z1row0[CHIP] = z0           + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y) *WD;
        z1row00[CHIP] = z1          + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y-1)*WD;
        z1row01[CHIP] = z1          + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y) */* not used for RMGRP>1 */;
        z1row02[CHIP] = z1          + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y-1)*WD; /* not used for RMGRP>1 */;
        z2row0[CHIP] = z2           + (CHIP*RRANGE*OMAP+RRANGE*0+PAD+y) *WD;
    }
}

//EMAX5A begin wave2d mapdist=0 /* 8 */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    /*+2*/for (INITI=1,LOOP1=RMGRP,roofs=0-WD*4; LOOP1--; INITI=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
        /*+1*/for (INITI=0,LOOP0=WD-PAD*2,coofs=(PAD-1)*4; LOOP0--; INITI=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &coofs, INITI?coofs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000000ffffffffffL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &roofs, roofs, EXP_H3210, INITI?WD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000000ffffffffffL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD3, &z2ofs, z0top[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x0000000ffffffffffL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3, &z1ofs, z1top[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x0000000ffffffffffL, OP_NOP, OLL); /* stage#1 */
            /*map0*/
            mop(OP_LDWR, 1, &BR[2][0][1], z0ofs, (0           )*4, MSK_D0, z1row0[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#2 */
            exe(OP_FML, &r0, BR[2][0][1], EXP_H3210, I2,      EXP_H3210, 0,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
            mop(OP_LDWR, 1, &BR[3][0][1], z1ofs, (0           -WD )*4, MSK_D0, z1row00[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#3 */
            mop(OP_LDWR, 1, &BR[4][0][1], z1ofs, (0           -1)*4, MSK_D0, z1row00[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#4 */
            mop(OP_LDWR, 1, &BR[4][0][0], z1ofs, (0           )*4, MSK_D0, z1row00[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#4 */
            mop(OP_LDWR, 1, &BR[4][1][1], z1ofs, (0           +1)*4, MSK_D0, z1row00[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#4 */
            exe(OP_FAD, &r1, BR[3][0][1], EXP_H3210, BR[4][0][1], EXP_H3210, 0,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
            mop(OP_LDWR, 1, &BR[5][0][1], z1ofs, (0           +WD )*4, MSK_D0, z1row00[CHIP], WD*(RMGRP+PAD*2), 0, 0, NULL, WD*(RMGRP+PAD*2)); /* stage#5 */
            exe(OP_FMA, &r2, r1, EXP_H3210, BR[4][0][0], EXP_H3210, I4,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
            exe(OP_FAD, &r3, BR[4][1][1], EXP_H3210, BR[5][0][1], EXP_H3210, 0,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
            exe(OP_FAD, &r4, r2, EXP_H3210, r3,           EXP_H3210, 0,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
            exe(OP_FMA, &r5, r0, EXP_H3210, r4,           EXP_H3210, I3,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
            exe(OP_FMA, &r6, r5, EXP_H3210, BR[4][0][0], EXP_H3210, I1,           EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
            mop(OP_STWR, 3, &r6, z2ofs, (0           )*4, MSK_D0, z2row0[CHIP], WD*RMGRP, 0, 0, NULL, WD*RMGRP); /* stage#9 */
        }
    }
}
//EMAX5A end
}
//EMAX5A drain_dirty_lmm

```

stencil+rmm-wave2d-emax6.obj

BR/row: max=7 min=3 ave=4 EA/row: max=3 min=0 ave=0 ARpass/row: max=0

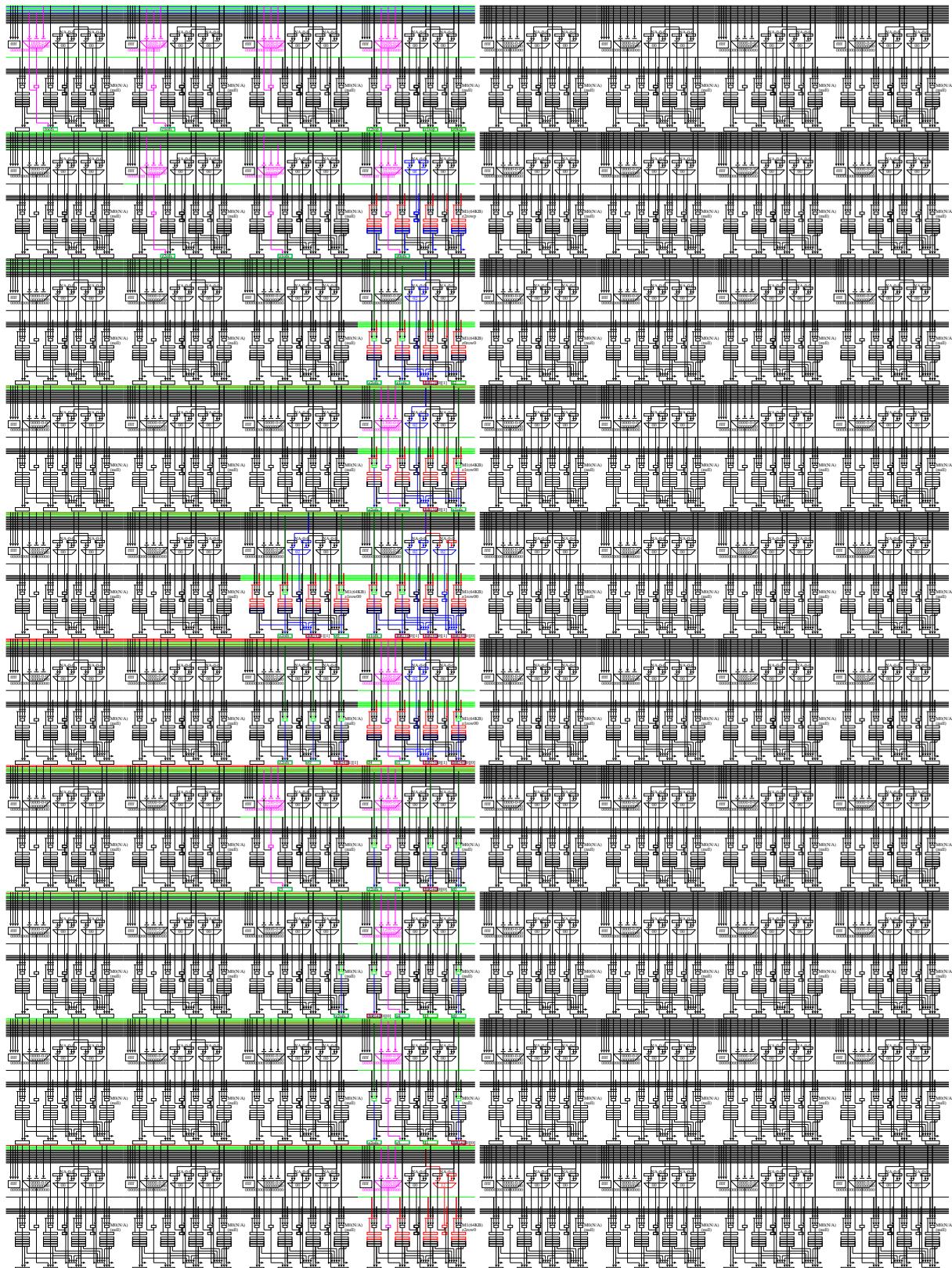


Figure.3.47: Wave2d

3.5 実用力一ネル

3.5.1 3x3畳み込み

```
cent% make -f Makefile-csim.emax6+dma cnn-csim.emax6+dma clean
cent% ../../src/csim/csim -x cnn-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma cnn-zynq.emax6+dma clean
zynq% ./cnn-zynq.emax6+dma
```

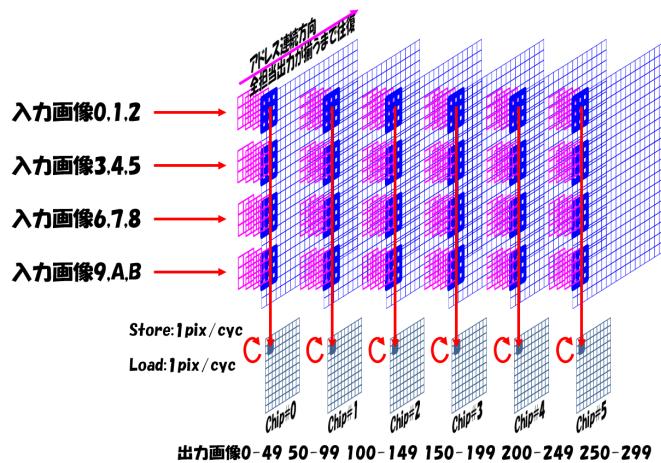


Figure.3.48: CNN の写像

図 3.48 は、応用プログラムの 1 つである Alexnet の畳み込み演算 (CNN) におけるデータの流れである。入力画像の一部とカーネル（重み）の積和演算結果を求め、複数入力画像方向に加算したものを作成する（1つの出力画像を得るために全入力画像が必要）。各入力画像の画素数は徐々に減少し枚数は増加する。出力画像は次の処理の入力画像となるため、常に入力画像枚数 ≤ 出力画像枚数である。全画像をハードウェア内に保持できない前提では、入力画像は順次ブロードキャストし、出力画像単位に演算を分割し、出力画像に次の出力値を累算できるハードウェア構成が適している。

242x242 の畳み込み演算である。一度のバースト演算により 8 行分を計算する (RMGRP=8)。ステンシル計算であるものの、一度に 8 行分を計算するため、mapdist=0 である。図 3.49(a) に、CNN のデータ構造、(b) に各 unit に積和演算器を 1 つ備える $(K^2+1) \times W$ 構成の CGRA を示す ($M \geq K$, $OC \geq W$ を仮定)。(b) は簡単のために、同一時刻のスナップショットではなく、データフローグラフを示している。実際のハードウェアでは図 3.51(c) のように各段のデータがパイプライン処理される。CNN は、入力 in の一部 ($K \times K$) と kernel ($K \times K$) の積和結果を全入力チャネル分累算し、出力 out に格納する。入力の総ワード数は $M^2 \times IC$, kernel の総ワード数は、 K^2 に入力チャネル数 (IC) と出力チャネル数 (OC) を乗じた $K^2 \times IC \times OC$, 出力の総ワード数は $M^2 \times OC$ となる。CNN に用いられる K は奇数であり、偶数が適するハードウェアパラメタ W に対応させると演算器稼働率が低下する。 $K \times K$ の積和演算を 1 列の unit[*][oc mod W] に対応させてパイプライン実行し、 W 個の出力チャネルを同時に計算すると $K^2 \times W$ の全演算器を利用でき効率がよい（ただし K 段の LMM に対するブロードキャスト機能が必要）。(c) はさらに、OC を N チップに分割して並列処理し、各チップが K^2 段の積和演算を REP 組 $\times W$ 列写像できる段数を備え、さらに各 LMM が、in の 1 行 (M 要素) \times GRP 行分と、全 kernel ($IC \times OC \times K^2 \times K$ 要素) を収容できる場合の実装である。矩形内の 7 重ループが全チップの CGRA を 1 回起動する処理に対応し、chip ループの各イタレーションが、各チップにおける REP 組の入力チャネルと kernel との畳み込み演算により W 組の出力チャネルを GRP 行分計算する処理に対応する。外側ループの oc が 0 から $OC/N-1$ まで増加する間、各チップの最終段を除く LMM は in を GRP 行分保持する。一方、最終段の LMM は oc を更新する度に入れ替える必要がある。以上の演

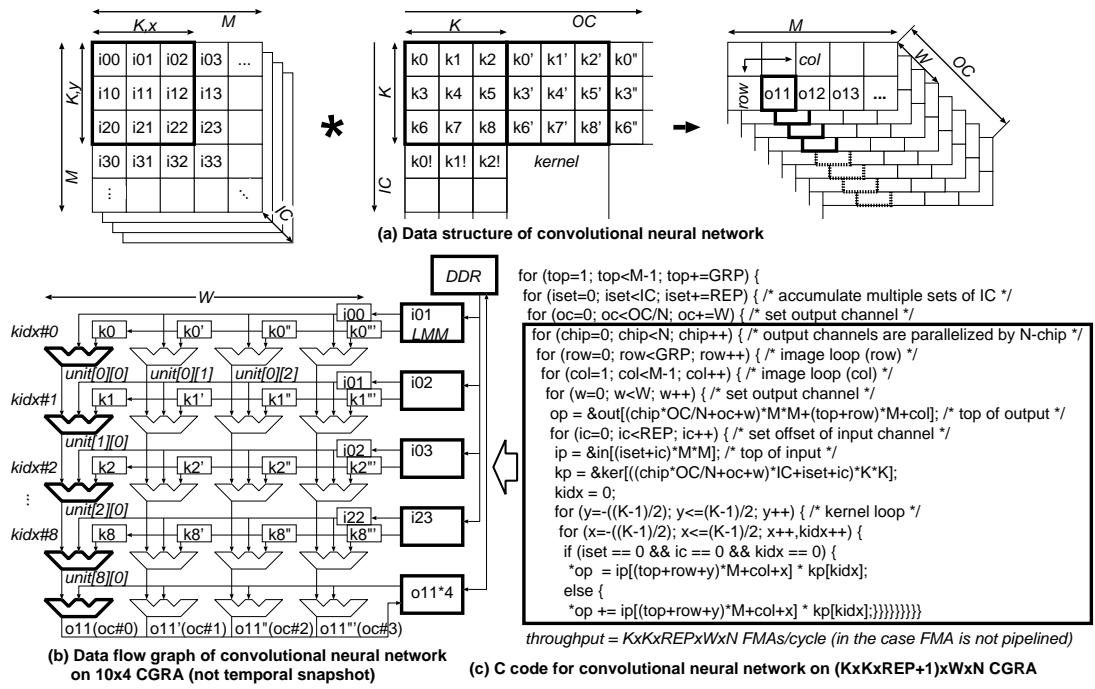


Figure 3.49: Cnn

算に要する理論的サイクル数は、CGRA 起動時の遅延 ($K^2 \times REP$ サイクル) および LMM の入れ換え時間を除くと $K^2 \times (M-2)^2 \times IC \times OC / (K^2 \times REP \times W \times N) = (M-2)^2 \times IC \times OC / (REP \times W \times N)$ となる。また、LMM が in, kernel, out 全体を収容できる場合の理論的 DDR-LMM 間転送量が $M^2 \times IC + K^2 \times IC \times OC + M^2 \times OC$ であるのに対し、各 LMM が in を $M \times GRP$, out を $M \times GRP \times W$, kernel のみ全部収容できる場合の転送量は、in と kernel が各々 $M^2 \times IC$ と $K^2 \times IC \times OC$, out は 1 回分の入れ換え量が $(M \times GRP \times W) \times 2$, 矩形外の回転数 ($OC/N/W$) $\times (IC/REP) \times (M/GRP)$ に N を乗じた回数の入れ替えが発生するため $M^2 \times 2 \times OC \times IC / REP$ ($IC=REP$ の場合 out は一度で計算でき入れ替え不要のため DDR 出力のみの $M^2 \times OC$. N に非依存) となる。

```

orig() {
    for (ic=0; ic<IC; ic++) { /* set input channel */
        ip0 = &in[ic*M*M]; /* top of input */
        for (row=1; row<M-1; row++) { /* image loop */
            for (col=1; col<M-1; col++) {
                for (oc=0; oc<OC; oc++) { /* set output channel */
                    op = &out[oc*M*M+row*M+col]; /* top of output */
                    kp = &ker[(oc*IC+ic)*K*K];
                    kidx = 0;
                    for (y=-(K-1)/2; y<=(K-1)/2; y++) { /* kernel loop */
                        for (x=-(K-1)/2; x<=(K-1)/2; x++) { /* kernel loop */
                            if (ic == 0 && kidx == 0) {
                                *(float*)&op = *(float*)&ip0[(row+y)*M+col+x] * *(float*)&kp[kidx];
                            } else {
                                *(float*)&op += *(float*)&ip0[(row+y)*M+col+x] * *(float*)&kp[kidx];
                            }
                            kidx++;
                            count0++;
                        }
                    }
                }
            }
        }
    }
}

```

```

imax() {
    U11 CHIP;
    U11 rofs;
    for (top=1; top<M-1; top+=RMGRP) {
        for (iset=0; iset<IC; iset+=IMAP) { /* accumulate multiple sets of IC */
            for (oc=0; oc<OC/NCHIP; oc+=W) { /* set output channel */
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*2*/ for (rofs=0; rofs<RMGRP; rofs++) { /* image loop (row) */
/*1*/ for (col=1; col<M-1; col++) { /* image loop (col) */
                for (w=0; w<W; w++) { /* set output channel */
                    op = &out1[(CHIP*OC/NCHIP+oc+w)*M*M+(top+rofs)*M+col]; /* top of output */
                    for (ic=0; ic<IMAP; ic++) { /* set offset of input channel */
                        ip0 = &in1[(iset+ic)*M*M]; /* top of input */
                        kp = &ker[((CHIP*OC/NCHIP+oc+w)*IC+iset+ic)*K*K];
                        kidx = 0;
                        for (y=((K-1)/2); y<=(K-1)/2; y++) { /* kernel loop */
                            for (x=((K-1)/2); x<=(K-1)/2; x++) {
                                if (iset == 0 && ic == 0 && kidx == 0) {
                                    *(float*)&op = *(float*)&ip0[(top+rofs+y)*M+col+x] * *(float*)&kp[kidx];
                                }
                                else {
                                    *(float*)&op += *(float*)&ip0[(top+rofs+y)*M+col+x] * *(float*)&kp[kidx];
                                }
                                kidx++;
                                count1++;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

imax() {
    U11 CHIP; U11 LOOP1, LOOP0; U11 INIT1, INIT0;
    U11 AR[64] [4]; /* output of EX in each unit */
    U11 BR[64] [4][4]; /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 cc0, ccl, cc2, cc3, ex0, ex1; U11 cofs, rofs, oofs;
    for (top=1; top<=M-1; top+=RMGRP) {
        for (iset=0; iset<IC; iset+=IMAP) { /* accumulate multiple sets of IC */
            Uint *ip0 = &in[(iset+0)*M*M]; /* top of input#0 */
            Uint *it00 = ip0+(top-1)*M+1-1, *ip01 = ip0+(top-1)*M+1+0, *ip02 = ip0+(top-1)*M+1+1;
            Uint *ip03 = ip0+(top+0)*M+1-1, *ip04 = ip0+(top+0)*M+1+0, *ip05 = ip0+(top+0)*M+1+1;
            Uint *ip06 = ip0+(top+1)*M+1-1, *ip07 = ip0+(top+1)*M+1+0, *ip08 = ip0+(top+1)*M+1+1;
        }
        for (oc=0; oc<OC/NCHIP; oc+=W) { /* set output channel */
            Uint *kp00[NCHIP], *kp01[NCHIP], *kp02[NCHIP], *kp03[NCHIP];
        }
        Uint *kp50[NCHIP], *kp51[NCHIP], *kp52[NCHIP], *kp53[NCHIP];
        Uint *op0[NCHIP], *op1[NCHIP], *op2[NCHIP], *op3[NCHIP];
        Uint *ot0[NCHIP], *ot1[NCHIP], *ot2[NCHIP], *ot3[NCHIP];
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
            Uint choc = CHIP*OC/NCHIP+oc;
            kp00[CHIP] = ker+((choc+0)*IC+iset+0)*K*K; kp01[CHIP] = ker+((choc+1)*IC+iset+0)*K*K; kp02[CHIP] = ker+((choc+2)*IC+iset+0)*K*K;
            kp03[CHIP] = ker+((choc+3)*IC+iset+0)*K*K;
            op0[CHIP] = out1+(choc+0)*M*M+top*M+1; op1[CHIP] = out1+(choc+1)*M*M+top*M+1; op2[CHIP] = out1+(choc+2)*M*M+top*M+1; op3[CHIP] = out1+(choc+3)*M*M+top*M+1;
            ot0[CHIP] = out1+(choc+0)*M*M+top*M+0; ot1[CHIP] = out1+(choc+1)*M*M+top*M+0; ot2[CHIP] = out1+(choc+2)*M*M+top*M+0; ot3[CHIP] = out1+(choc+3)*M*M+top*M+0;
        }
    }
    //EMAXSA begin cnn mapdist=0
    /*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
        /*2*/ for (INIT1=1,LOOP0=1;LOOP1=RMGRP,rofs=0-M*4; INIT1=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
            /*1*/ for (INIT0=1,LOOP0=M-2,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
                exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#0 */
                exe(OP_ADD, &rofs, EXP_H3210, INIT0?4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
                exe(OP_ADD, &rofs, EXP_H3210, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */
                mop(OP_LDWR, 1, &BR[2][0][1], (U11)kp00[NCHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#2 */
                mop(OP_LDWR, 1, &BR[2][0][0], (U11)kp01[NCHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#2 */
                mop(OP_LDWR, 1, &BR[2][1][1], (U11)kp02[NCHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#2 */
                mop(OP_LDWR, 1, &BR[2][1][0], (U11)kp03[NCHIP], OLL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#2 10KB */
                mop(OP_LDWR, 1, &BR[2][2][1], (U11)ip00, oofs, MSK_WO, (U11)it00, M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#2 8KB */
            }
            /****in0*****/
            exe(OP_FML, &AR[3][0], BR[2][2][1], EXP_H3210, BR[2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
            exe(OP_FML, &AR[3][1], BR[2][2][1], EXP_H3210, BR[2][0][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
            exe(OP_FML, &AR[3][2], BR[2][2][1], EXP_H3210, BR[2][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
            exe(OP_FML, &AR[3][3], BR[2][2][1], EXP_H3210, BR[2][1][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
            mop(OP_LDWR, 1, &BR[3][0][1], (U11)kp00[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#3 */
            mop(OP_LDWR, 1, &BR[3][0][0], (U11)kp01[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#3 */
            mop(OP_LDWR, 1, &BR[3][1][1], (U11)kp02[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#3 */
            mop(OP_LDWR, 1, &BR[3][1][0], (U11)kp03[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#3 */
            mop(OP_LDWR, 1, &BR[3][2][1], (U11)ip01, oofs, MSK_WO, (U11)it00, M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#3 */
            /****in5*****/
            exe(OP_FMA, &AR[48][0], AR[47][0], EXP_H3210, BR[47][2][1], EXP_H3210, BR[47][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#48 */
            exe(OP_FMA, &AR[48][1], AR[47][1], EXP_H3210, BR[47][2][1], EXP_H3210, BR[47][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#48 */
            exe(OP_FMA, &AR[48][2], AR[47][2], EXP_H3210, BR[47][2][1], EXP_H3210, BR[47][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#48 */
            exe(OP_FMA, &AR[48][3], AR[47][3], EXP_H3210, BR[47][2][1], EXP_H3210, BR[47][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#48 */
            mop(OP_LDWR, 1, &BR[48][0][1], (U11)kp00[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#48 */
            mop(OP_LDWR, 1, &BR[48][0][0], (U11)kp01[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#48 */
            mop(OP_LDWR, 1, &BR[48][1][1], (U11)kp02[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#48 */
            mop(OP_LDWR, 1, &BR[48][1][0], (U11)kp03[NCHIP], 4LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#48 */
            mop(OP_LDWR, 1, &BR[48][2][1], (U11)ip51, oofs, MSK_WO, (U11)it50, M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#48 */
            /****in55*****/
            exe(OP_FMA, &AR[53][0], AR[52][0], EXP_H3210, BR[52][2][1], EXP_H3210, BR[52][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            exe(OP_FMA, &AR[53][1], AR[52][1], EXP_H3210, BR[52][2][1], EXP_H3210, BR[52][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            exe(OP_FMA, &AR[53][2], AR[52][2], EXP_H3210, BR[52][2][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            exe(OP_FMA, &AR[53][3], AR[52][3], EXP_H3210, BR[52][2][1], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][0][1], (U11)kp50[NCHIP], 24LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][0][0], (U11)kp51[NCHIP], 24LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][1][1], (U11)kp52[NCHIP], 24LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][1][0], (U11)kp53[NCHIP], 24LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][2][1], (U11)ip56, oofs, MSK_WO, (U11)it50, M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#53 */
            /****in54*****/
            exe(OP_FMA, &AR[54][0], AR[53][0], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            exe(OP_FMA, &AR[54][1], AR[53][1], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            exe(OP_FMA, &AR[54][2], AR[53][2], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            exe(OP_FMA, &AR[54][3], AR[53][3], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][0][1], (U11)kp50[NCHIP], 28LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][0][0], (U11)kp51[NCHIP], 28LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][1][1], (U11)kp52[NCHIP], 28LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][1][0], (U11)kp53[NCHIP], 28LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][2][1], (U11)ip56, oofs, MSK_WO, (U11)it50, M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#54 */
            /****in55*****/
            exe(OP_FMA, &AR[55][0], AR[54][0], EXP_H3210, BR[54][2][1], EXP_H3210, BR[54][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            exe(OP_FMA, &AR[55][1], AR[54][1], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            exe(OP_FMA, &AR[55][2], AR[54][2], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            exe(OP_FMA, &AR[55][3], AR[54][3], EXP_H3210, BR[53][2][1], EXP_H3210, BR[53][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][0][1], (U11)kp50[NCHIP], 32LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][0][0], (U11)kp51[NCHIP], 32LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][1][1], (U11)kp52[NCHIP], 32LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][1][0], (U11)kp53[NCHIP], 32LL, MSK_DO, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*OC*K*K); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][2][1], (U11)ip58, oofs, MSK_WO, (U11)it50, M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#55 */
            /****in56*****/
            exe(OP_FMA, &AR[56][0], AR[55][0], EXP_H3210, BR[55][2][1], EXP_H3210, BR[55][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            exe(OP_FMA, &AR[56][1], AR[55][1], EXP_H3210, BR[55][2][1], EXP_H3210, BR[55][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            exe(OP_FMA, &AR[56][2], AR[55][2], EXP_H3210, BR[55][2][1], EXP_H3210, BR[55][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            exe(OP_FMA, &AR[56][3], AR[55][3], EXP_H3210, BR[55][2][1], EXP_H3210, BR[55][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            mop(OP_LDWR, 1, &BR[56][0][1], (U11)op0[CHIP], oofs, MSK_WO, (U11)ot0[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 */
            mop(OP_LDWR, 1, &BR[56][1][1], (U11)op1[CHIP], oofs, MSK_WO, (U11)ot1[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 */
            mop(OP_LDWR, 1, &BR[56][2][1], (U11)op2[CHIP], oofs, MSK_WO, (U11)ot2[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 */
            mop(OP_LDWR, 1, &BR[56][3][1], (U11)op3[CHIP], oofs, MSK_WO, (U11)ot3[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 */
            exe(OP_FAD, &AR[57][0], AR[56][0], EXP_H3210, BR[56][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
            exe(OP_FAD, &AR[57][1], AR[56][1], EXP_H3210, BR[56][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
            exe(OP_FAD, &AR[57][2], AR[56][2], EXP_H3210, BR[56][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
            exe(OP_FAD, &AR[57][3], AR[56][3], EXP_H3210, BR[56][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
            mop(OP_STWR, 1, &AR[57][0][1], (U11)op0[CHIP], oofs, MSK_WO, (U11)ot0[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 8KB */
            mop(OP_STWR, 1, &AR[57][1][1], (U11)op1[CHIP], oofs, MSK_WO, (U11)ot1[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 8KB */
            mop(OP_STWR, 1, &AR[57][2][1], (U11)op2[CHIP], oofs, MSK_WO, (U11)ot2[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 8KB */
            mop(OP_STWR, 1, &AR[57][3][1], (U11)op3[CHIP], oofs, MSK_WO, (U11)ot3[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#57 8KB */
        }
    }
    //EMAXSA end
}
//EMAXSA drain_dirty_lmm
}

```

cnn+rmm-cnn-emax6.obj

BR/row: max=11 min=1 ave=10 EA/row: max=10 min=0 ave=9 ARpass/row: max=0

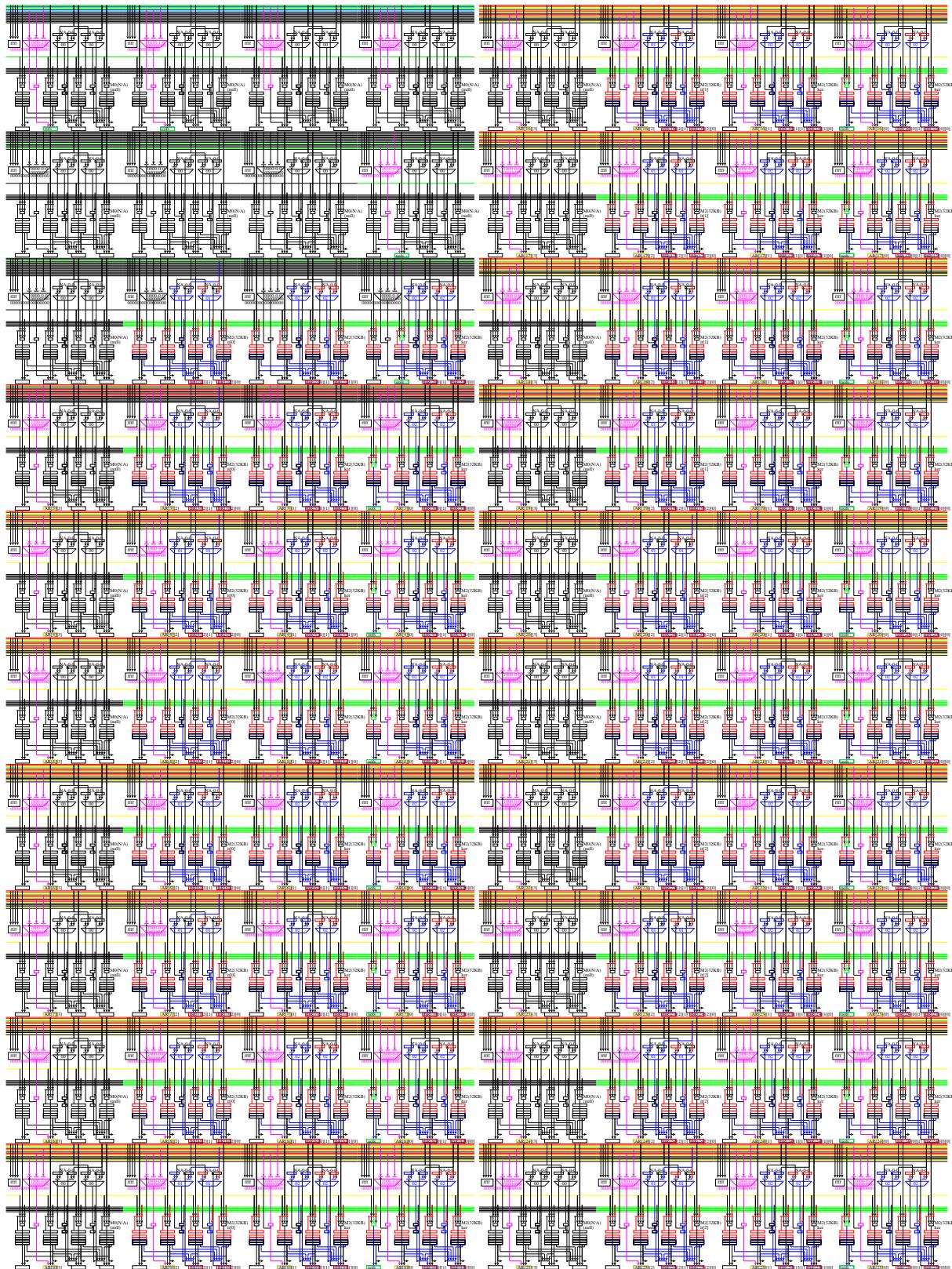


Figure.3.50: 3x3 畳み込み

Unaligned load と SIMD 演算の組み合わせ

```

imax() {
    U11 CHIP; U11 LOOP1, LOOP0; U11 INIT1, INIT0;
    U11 AR[64][4]; /* output of EX in each unit */
    U11 BR[64][4][4]; /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 cc0, ccl, cc2, cc3, ex0, ex1; U11 cofs, rofs, oofs, k;
    for (top=1; top<M-1; top+=RMGRP) {
        for (iset=0; iset<IC; iset+=IMAP) { /* accumulate multiple sets of IC */
            Uint *ip[IMAP]; *ip0[IMAP], *ip1[IMAP][K*K], *ip1[IMAP][K*K];
            for (k=0; k<IMAP; k++) {
                ip[k] = kin[(iset+k)*M+M]; /* top of input#0-5 */
                it[k] = ip[k]+(top-1)*M+1;
                ip0[k][0] = ip[k]+(top-1)*M+1; ip0[k][1] = ip[k]+(top-1)*M+1; ip0[k][2] = ip[k]+(top-1)*M+1;
                ip0[k][3] = ip[k]+(top+0)*M+1; ip0[k][4] = ip[k]+(top+0)*M+1; ip0[k][5] = ip[k]+(top+0)*M+1;
                ip0[k][6] = ip[k]+(top+1)*M+1; ip0[k][7] = ip[k]+(top+1)*M+1; ip0[k][8] = ip[k]+(top+1)*M+1;
                ip1[k][0] = ip[k]+(top-1)*M+1; ip1[k][1] = ip[k]+(top-1)*M+1; ip1[k][2] = ip[k]+(top-1)*M+1;
                ip1[k][3] = ip[k]+(top+0)*M+1; ip1[k][4] = ip[k]+(top+0)*M+1; ip1[k][5] = ip[k]+(top+0)*M+1;
                ip1[k][6] = ip[k]+(top+1)*M+1; ip1[k][7] = ip[k]+(top+1)*M+1; ip1[k][8] = ip[k]+(top+1)*M+1;
            }
            for (oc=0; oc<OC/NCHIP; oc+=W) { /* set output channel */
                Uint *xp0[IMAP][NCHIP], *kp1[IMAP][NCHIP], *kp2[IMAP][NCHIP], *kp3[IMAP][NCHIP];
                Uint *op0[NCHIP], *op1[NCHIP], *op2[NCHIP], *op3[NCHIP];
                Uint *ot0[NCHIP], *ot1[NCHIP], *ot2[NCHIP], *ot3[NCHIP];
                for (CHIP=0; CHIP<NCHIP; CHIP++) { /* parallelized by multi-chip (OC/#chip) */
                    Uint choc = CHIP*OC/NCHIP+oc;
                    for (k=0; k<IMAP; k++) {
                        kp0[CHIP] = ker+((choc+0)*IC+iset+k)*K;
                        kp1[CHIP] = ker+((choc+1)*IC+iset+k)*K;
                        kp2[CHIP] = ker+((choc+2)*IC+iset+k)*K;
                        kp3[CHIP] = ker+((choc+3)*IC+iset+k)*K;
                    }
                    op0[CHIP] = out1+(choc+0)*M*M+top*M+0; op1[CHIP] = out1+(choc+1)*M*M+top*M+0; op2[CHIP] = out1+(choc+2)*M*M+top*M+0; op3[CHIP] = out1+(choc+3)*M*M+top*M+0;
                    ot0[CHIP] = out1+(choc+0)*M*M+top*M+0; ot1[CHIP] = out1+(choc+1)*M*M+top*M+0; ot2[CHIP] = out1+(choc+2)*M*M+top*M+0; ot3[CHIP] = out1+(choc+3)*M*M+top*M+0;
                }
            }
        }
    }
}

#define cnn_core1(r, i, ofs, k, rp1) \
/*dup load*/ mop(OP_LDRW, 1, &BR[r][0][1], (U11)kp0[i][CHIP], ofs, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); \
/*dup load*/ mop(OP_LDRW, 1, &BR[r][0][0], (U11)kp1[i][CHIP], ofs, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); \
/*dup load*/ mop(OP_LDRW, 1, &BR[r][1][1], (U11)kp2[i][CHIP], ofs, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); \
/*dup load*/ mop(OP_LDRW, 1, &BR[r][1][0], (U11)kp3[i][CHIP], ofs, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); \
/*unaligned*/ mop(OP_LDR, 1, &BR[r][2][1], (U11)ip1[i][k], ofs, MSK_W0, (U11)it[i], M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); \
/*unaligned*/ mop(OP_LDR, 1, &BR[r][2][0], (U11)ip0[i][k], ofs, MSK_W0, (U11)it[i], M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); \
exe(OP_FMA, &AR[rp1][0], AR[r][0], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][0][1], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][1], AR[r][1], EXP_H3210, BR[r][0][0], EXP_H3210, BR[r][1][0], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][2], AR[r][2], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][1][1], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][3], AR[r][3], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][1][0], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL);

#define cnn_final(r, rp1) \
mop(OP_LDR, 1, &BR[rp1][0][1], (U11)op0[CHIP], ofs, MSK_W0, (U11)ot0[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_LDR, 1, &BR[rp1][1][1], (U11)op1[CHIP], ofs, MSK_W0, (U11)ot1[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_LDR, 1, &BR[rp1][2][1], (U11)op2[CHIP], ofs, MSK_W0, (U11)ot2[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_LDR, 1, &BR[rp1][3][1], (U11)op3[CHIP], ofs, MSK_W0, (U11)ot3[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
exe(OP_FAD, &AR[rp1][0], AR[r][0], EXP_H3210, BR[rp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][1], AR[r][1], EXP_H3210, BR[rp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][2], AR[r][2], EXP_H3210, BR[rp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][3], AR[r][3], EXP_H3210, BR[rp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
mop(OP_STR, 3, &AR[rp1][0], ofs, (U11)op0[CHIP], MSK_D0, (U11)ot0[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_STR, 3, &AR[rp1][1], ofs, (U11)op1[CHIP], MSK_D0, (U11)ot1[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_STR, 3, &AR[rp1][2], ofs, (U11)op2[CHIP], MSK_D0, (U11)ot2[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_STR, 3, &AR[rp1][3], ofs, (U11)op3[CHIP], MSK_D0, (U11)ot3[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP);

//EMAXSA begin cnn maplist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
/*2*/ for (INIT1=1, LOOP1=RMGRP, rofs=0-M+4; LOOP1--; INIT1=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
/*1*/ for (INIT0=1, LOOP0=M-2, rofs=0-8; LOOP0--; INIT0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &cofs, INIT0?cofs, EXP_H3210, 8, EXP_H3210, OLL, EXP_H3210, OP_AND, Ox00000000fffff1LL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?M+4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, OLL, EXP_H3210, OP_AND, Ox00000000fffff1LL, OP_NOP, OLL); /* stage#1 */

/*dup load*/ mop(OP_LDRW, 1, &BR[2][0][1], (U11)kp0[0][CHIP], OLL, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); /* stage#2 */
/*dup load*/ mop(OP_LDRW, 1, &BR[2][0][0], (U11)kp1[0][CHIP], OLL, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); /* stage#2 */
/*dup load*/ mop(OP_LDRW, 1, &BR[2][1][1], (U11)kp2[0][CHIP], OLL, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); /* stage#2 */
/*dup load*/ mop(OP_LDRW, 1, &BR[2][1][0], (U11)kp3[0][CHIP], OLL, MSK_D0, (U11)ker, IC*OC*K*K, 0, 0, (U11)NULL, IC*DC*K*K); /* stage#2 10KB */
/*unaligned*/ mop(OP_LDR, 1, &BR[2][2][1], (U11)ip0[0][0], ofs, MSK_W0, (U11)it[0], M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#2 8KB */
/*unaligned*/ mop(OP_LDR, 1, &BR[2][2][0], (U11)ip0[0][0], ofs, MSK_W0, (U11)it[0], M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); /* stage#2 8KB */
exe(OP_FML, &AR[3][0], BR[2][2][0], EXP_H3210, BR[2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exe(OP_FML, &AR[3][1], BR[2][2][0], EXP_H3210, BR[2][0][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exe(OP_FML, &AR[3][2], BR[2][2][0], EXP_H3210, BR[2][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exe(OP_FML, &AR[3][3], BR[2][2][0], EXP_H3210, BR[2][1][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */

cnn_core1( 3, 0, 4LL, 1, 4);
cnn_core1( 4, 0, 8LL, 2, 5);
cnn_core1( 5, 0, 12LL, 3, 6);
cnn_core1( 6, 0, 16LL, 4, 7);
cnn_core1( 7, 0, 20LL, 5, 8);
cnn_core1( 8, 0, 24LL, 6, 9);
cnn_core1( 9, 0, 28LL, 7, 10);
cnn_core1(10, 0, 32LL, 8, 11);

cnn_core1(11, 1, 4LL, 0, 12);
cnn_core1(12, 1, 4LL, 1, 13);
cnn_core1(13, 1, 8LL, 2, 14);
cnn_core1(14, 1, 12LL, 3, 15);
cnn_core1(15, 1, 16LL, 4, 16);
cnn_core1(16, 1, 20LL, 5, 17);
cnn_core1(17, 1, 24LL, 6, 18);
cnn_core1(18, 1, 28LL, 7, 19);
cnn_core1(19, 1, 32LL, 8, 20);

cnn_core1(55, 5, 32LL, 8, 56);
****final*****
cnn_final(55,      57);

}
}

//EMAXSA end
}
}

//EMAXSA drain_dirty_lmm
}

```

3.5.2 行列積

```
cent% make -f Makefile-csim.emax6+dma mm-csim.emax6+dma clean
cent% ../../src/csim/csim -x mm-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma mm-zynq.emax6+dma clean
zynq% ./mm-zynq.emax6+dma
```

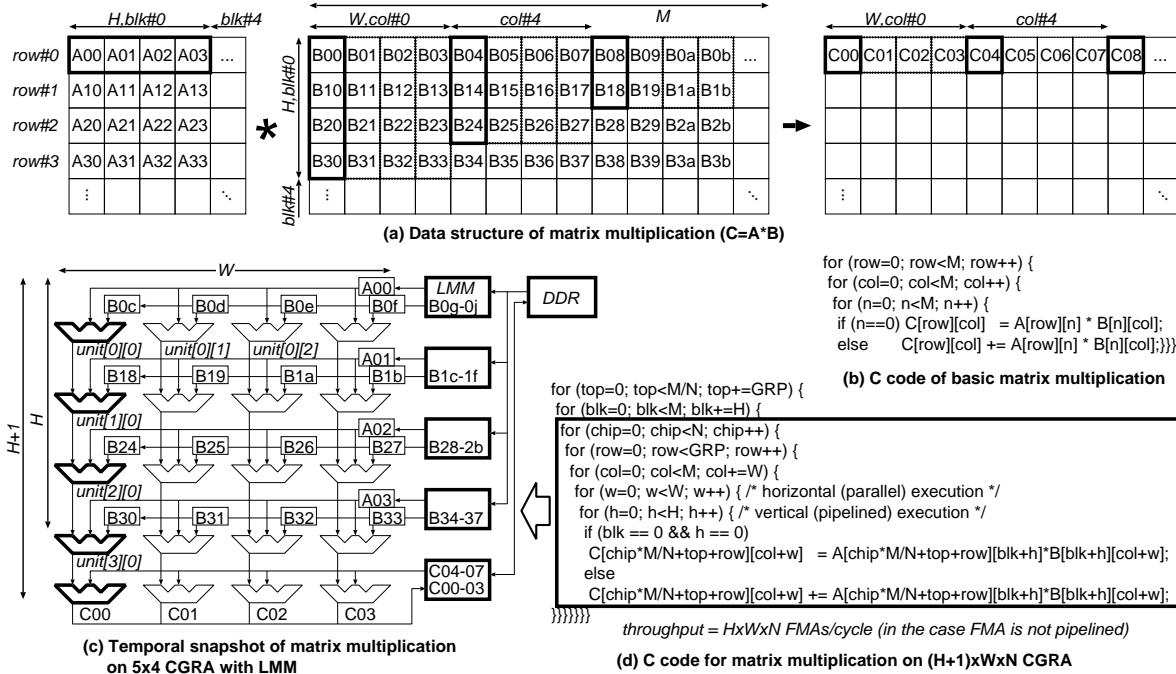


Figure 3.51: Mm

480x480 の行列積計算である。一度のバースト演算により 8 行分を計算する (RMGRP=8)。ステンシル計算であるものの、一度に 8 行分を計算するため、mapdist=0 である。図 3.51(a) に MM のデータ構造、(b) に C 言語による一般的な実装、(c) に各 unit に積和演算器を 1 つ備える $H+1$ (height) $\times W$ (width) 構成の CGRA と同一時刻におけるスナップショットを示す ($M \geq H, M \geq W$ を仮定)。出力 $C[row][col]$ に対応する $A[row][*] \times B[*][col]$ の積和演算を H 組ごとに分割し、1 列の $unit[*mod H][col mod W]$ に対応させてパイプライン実行すると $H \times W$ の全演算器を利用でき効率がよい。例えば左端列では、結果 $C0c, C08, C04, C00$ に必要な乗算 $A00 \times B0c, A01 \times B18, A02 \times B24, A03 \times B30$ を同時に実行し、最終段から $C00, C04, C08, C0c, \dots, C0(M-4)$ の部分和を毎サイクル出力し LMM に累算する。すなわち、 $H \times W$ の演算スループットにより、毎サイクル W 個の部分和を出力する。これを M/H 回繰り返すことにより、完全な $C00, C01, \dots, C0(M-1)$ が得られる。(d) はさらに、配列 A を N チップに行分割して並列処理し、各 LMM が、配列 A の 1 行 (M 要素) \times GRP 行分と、配列 B の 1 行 (M 要素) を収容できる場合の実装である。矩形内の 5 重ループが全チップの CGRA を 1 回起動する処理に対応し、chip ループの各イタレーションが、各チップにおける配列 A の GRP 行分と配列 B 全体の行列積に対応する。外側ループの blk が 0 から $M-1$ に増加する間、各チップの最終段を除く LMM は配列 A を GRP 行分保持し、最終段の LMM は $Cx0, Cx1, \dots, Cx(M-1)$ を GRP 行分保持しつつ更新する。一方、配列 B は、blk を更新する度に対応する H 行分を全チップにブロードキャストする必要がある。以上の演算に要する理論的サイクル数は、CGRA 起動時の遅延 ($H+1$ サイクル) および LMM の入れ換え時間を除くと $M^3/(H \times W \times N)$ となる。また、LMM が A, B, C 全体を収容できる場合の理論的 DDR-LMM 間転送量が $M^2 \times 3$ であるのに対し、各 LMM が A と C を $M \times \text{GRP}$ 、B を $M \times H$ のみ収容できる場合の転送量は、A と C が各々 M^2 、B が $M^2 \times M/\text{GRP}$ ($M=\text{GRP}$ の場合 M^2 に一致。 N に非依存。ブロードキャスト前提) と最適になる。

```

orig() {
    for (row=0; row<M1; row++) {
        for (col=0; col<M2; col++) {
            for (n=0; n<L; n++) {
                if (n==0) *(float*)&C0[row*M2+col] = *(float*)&A[row*L+n] * *(float*)&B[n*M2+col];
                else *(float*)&C0[row*M2+col] += *(float*)&A[row*L+n] * *(float*)&B[n*M2+col];
            count0++;
        } } }
}

imax() {
    U11 CHIP; U11 rofs;
    for (top=0; top<M1/NCHIP; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
        for (blk=0; blk<L; blk+=H) { /* 3 重ループ目 (C が確定するまでの DMA 入れ換えは R/W を伴うためオーバヘッドになる。B の broadcast 回数を増やす方が結果的に高速) */
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/*2*/ for (rofs=0; rofs<RMGRP; rofs++) { /* will be parallelized by multi-chip (M/#chip) */
/*1*/ for (col=0; col<M2; col+=W) { /* one-horizontal-line is calculated by EMAX-while(loop--) */
            for (w=0; w<W; w++) { /* horizontal (parallel) execution */
                for (h=0; h<H; h++) { /* vertical (pipelined) execution */
                    if (blk == 0 && h == 0)
                        *(float*)&C1[(CHIP*M1/NCHIP+top+rofs)*M2+col+w] = *(float*)&A[(CHIP*M1/NCHIP+top+rofs)*L+blk+h]**(float*)&B[(blk+h)*M2+col+w];
                    else
                        *(float*)&C1[(CHIP*M1/NCHIP+top+rofs)*M2+col+w] += *(float*)&A[(CHIP*M1/NCHIP+top+rofs)*L+blk+h]**(float*)&B[(blk+h)*M2+col+w];
                count1++;
            } } } } } } }
}

```

```

imax() {
    U11 CHIP; U11 LOOP1, LOOP0; U11 INIT1, INIT0;
    U11 AR[64][4]; /* output of EX in each unit */
    U11 BR[64][4]; /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 cc0, ccl, cc2, cc3, ex0, ex1; U11 cofs, rofs, oofs;
    for (top=0; top<M1/NCHIP; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
        for (blk=0; blk<L; blk+=H) { /* 3 重ループ展開の外側対象 */
            typedef struct {uint i[4];} U14;
            U14 *b00 = B+(blk*0)*M2, *b000 = b00, *b001 = (Uint*)b00+1, *b002 = (Uint*)b00+2, *b003 = (Uint*)b00+3;
            :
            Uint *a0[NCHIP]; Uint *a00[NCHIP], *a01[NCHIP], *a02[NCHIP], *a03[NCHIP], *a04[NCHIP], *a05[NCHIP], *a06[NCHIP], *a07[NCHIP];
            :
            U14 *c0[NCHIP]; U14 *c00[NCHIP], *c01[NCHIP], *c02[NCHIP], *c03[NCHIP];
            for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
                a0[CHIP] = A+(CHIP*M1/NCHIP+top)*L;
                a00[CHIP] = a0[CHIP]+blk*0; a01[CHIP] = a0[CHIP]+blk+1; a02[CHIP] = a0[CHIP]+blk+2; a03[CHIP] = a0[CHIP]+blk+3;
                :
                c0[CHIP] = C1+(CHIP*M1/NCHIP+top)*M2; c00[CHIP] = (Uint*)c0[CHIP]+0; c01[CHIP] = (Uint*)c0[CHIP]+1; c02[CHIP] = (Uint*)c0[CHIP]+2; c03[CHIP] = (Uint*)c0[CHIP]+3;
            }
        //EMAXSA begin mm mapdist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/*2*/ for (INIT1=1,LOOP1=RMGRP,rofs=0; INIT1<32; (O-M2*4)<<32){ /* stage#0 */ mapped to FOR() on BR[63][1][0] */
        INIT1=0; /* stage#0 */ mapped to FOR() on BR[63][0][0] */
/*1*/ for (INIT0=1,LOOP0=M2/W,cofs=0; O-W*4)<<32; (W*4)<<32){ /* stage#0 */ mapped to FOR() on BR[63][0][0] */
        INIT0=0; /* stage#0 */ mapped to FOR() on BR[63][0][0] */
        exec(OP_ADD, &cofs, INIT0?cofs:EXP_H3210, (W*4)<<32); EXP_H3210, OLL, EXP_H3210, OP_AND, Oxffffffffffff, OLL, OP_NOP, OLL);
        exec(OP_ADD, &cofs, EXP_H3210, INIT0?Lw4)<<32)(M2*4),0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL; /* stage#0 */
        exec(OP_ADD, &cofs, EXP_H3210, cofs, EXP_H3210, OP_AND, Oxffffffffffff, OLL, OP_NOP, OLL); /* stage#1 */
        mop(OP_LDWR, 1, &BR[1][0][1], (U11)b000, (U11)cofs, MSK_W1, (U11)b00, M2, 0, 0, (U11)NULL, M2); /* stage#1 */
        mop(OP_LDWR, 1, &BR[1][0][0], (U11)b001, (U11)cofs, MSK_W1, (U11)b00, M2, 0, 0, (U11)NULL, M2); /* stage#1 */
        mop(OP_LDWR, 1, &BR[1][1][1], (U11)b002, (U11)cofs, MSK_W1, (U11)b00, M2, 0, 0, (U11)NULL, M2); /* stage#1 */
        mop(OP_LDWR, 1, &BR[1][1][0], (U11)b003, (U11)cofs, MSK_W1, (U11)b00, M2, 0, 0, (U11)NULL, M2); /* stage#1 2KB */
        mop(OP_LDWR, 1, &BR[1][2][1], (U11)a00[CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], L*RMGRP, 0, 0, (U11)NULL, L*RMGRP); /* stage#1 16KB */
        exec(OP_FML, &AR[2][0][1], BR[1][0][1], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        exec(OP_FML, &AR[2][1][1], BR[1][0][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        exec(OP_FML, &AR[2][2][1], BR[1][1][1], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        exec(OP_FML, &AR[2][3][1], BR[1][1][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        mop(OP_LDWR, 1, &BR[2][0][1], (U11)b010, (U11)cofs, MSK_W1, (U11)b01, M2, 0, 0, (U11)NULL, M2); /* stage#2 */
        mop(OP_LDWR, 1, &BR[2][0][0], (U11)b011, (U11)cofs, MSK_W1, (U11)b01, M2, 0, 0, (U11)NULL, M2); /* stage#2 */
        mop(OP_LDWR, 1, &BR[2][1][1], (U11)b012, (U11)cofs, MSK_W1, (U11)b01, M2, 0, 0, (U11)NULL, M2); /* stage#2 */
        mop(OP_LDWR, 1, &BR[2][1][0], (U11)b013, (U11)cofs, MSK_W1, (U11)b01, M2, 0, 0, (U11)NULL, M2); /* stage#2 2KB */
        mop(OP_LDWR, 1, &BR[2][2][1], (U11)a01[CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], L*RMGRP, 0, 0, (U11)NULL, L*RMGRP); /* stage#2 16KB */
        :
        exec(OP_FMA, &AR[60][0][0], AR[59][0][1], EXP_H3210, BR[59][2][1], EXP_H3210, BR[59][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exec(OP_FMA, &AR[60][1][0], AR[59][1][1], EXP_H3210, BR[59][2][1], EXP_H3210, BR[59][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exec(OP_FMA, &AR[60][2][0], AR[59][2][1], EXP_H3210, BR[59][2][1], EXP_H3210, BR[59][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exec(OP_FMA, &AR[60][3][0], AR[59][3][1], EXP_H3210, BR[59][2][1], EXP_H3210, BR[59][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        mop(OP_LDWR, 1, &BR[60][0][1], (U11)b590, (U11)cofs, MSK_W1, (U11)b59, M2, 0, 0, (U11)NULL, M2); /* stage#60 */
        mop(OP_LDWR, 1, &BR[60][0][0], (U11)b591, (U11)cofs, MSK_W1, (U11)b59, M2, 0, 0, (U11)NULL, M2); /* stage#60 */
        mop(OP_LDWR, 1, &BR[60][1][1], (U11)b592, (U11)cofs, MSK_W1, (U11)b59, M2, 0, 0, (U11)NULL, M2); /* stage#60 */
        mop(OP_LDWR, 1, &BR[60][1][0], (U11)b593, (U11)cofs, MSK_W1, (U11)b59, M2, 0, 0, (U11)NULL, M2); /* stage#60 */
        mop(OP_LDWR, 1, &BR[60][2][1], (U11)a59[CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], L*RMGRP, 0, 0, (U11)NULL, L*RMGRP); /* stage#60 */
        exec(OP_FMA, &AR[61][0][0], AR[60][0][1], EXP_H3210, BR[60][2][1], EXP_H3210, BR[60][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#61 */
        exec(OP_FMA, &AR[61][1][0], AR[60][1][1], EXP_H3210, BR[60][2][1], EXP_H3210, BR[60][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#61 */
        exec(OP_FMA, &AR[61][2][0], AR[60][2][1], EXP_H3210, BR[60][2][1], EXP_H3210, BR[60][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#61 */
        exec(OP_FMA, &AR[61][3][0], AR[60][3][1], EXP_H3210, BR[60][2][1], EXP_H3210, BR[60][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#61 */
        mop(OP_LDWR, 1, &BR[62][0][1], (U11)c00[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        mop(OP_LDWR, 1, &BR[62][1][1], (U11)c01[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        mop(OP_LDWR, 1, &BR[62][2][1], (U11)c02[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        mop(OP_LDWR, 1, &BR[62][3][1], (U11)c03[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        exec(OP_FAD, &AR[62][0][0], AR[61][0][1], EXP_H3210, BR[62][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#62 */
        exec(OP_FAD, &AR[62][1][0], AR[61][1][1], EXP_H3210, BR[62][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#62 */
        exec(OP_FAD, &AR[62][2][0], AR[61][2][1], EXP_H3210, BR[62][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#62 */
        exec(OP_FAD, &AR[62][3][0], AR[61][3][1], EXP_H3210, BR[62][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#62 */
        mop(OP_STWR, 1, &AR[62][0][1], (U11)oofs, (U11)c00[CHIP], MSK_DO, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        mop(OP_STWR, 1, &AR[62][1][1], (U11)oofs, (U11)c01[CHIP], MSK_DO, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        mop(OP_STWR, 1, &AR[62][2][1], (U11)oofs, (U11)c02[CHIP], MSK_DO, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
        mop(OP_STWR, 1, &AR[62][3][1], (U11)oofs, (U11)c03[CHIP], MSK_DO, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); /* stage#62 */
    } } }
}

```

mm+rmm-mm-emax6.obj

BR/row: max=12 min=2 ave=11 EA/row: max=8 min=0 ave=4 ARpass/row: max=0



Figure 3.52: 行列積

Duplicated load と SIMD 演算の組み合わせ

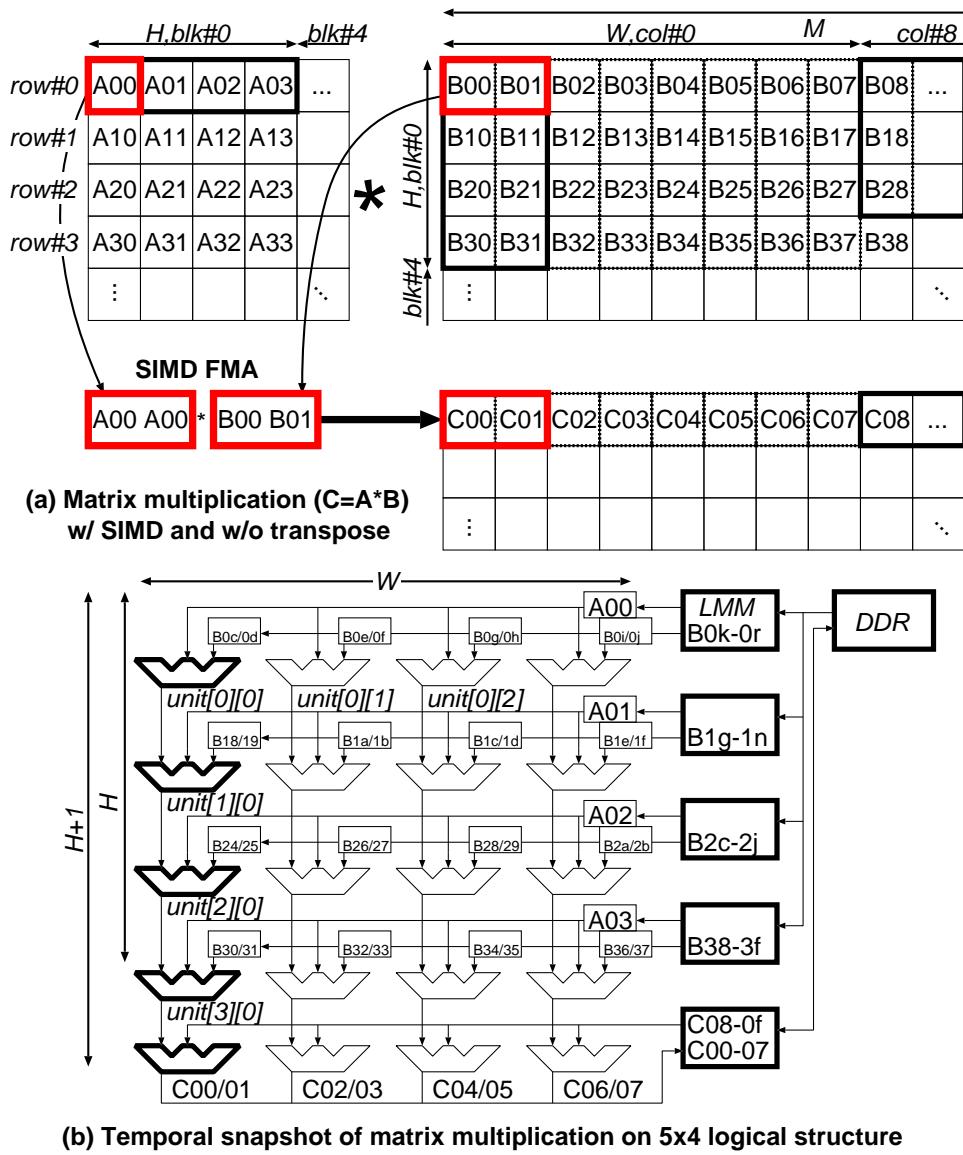


Figure.3.53: 32bit → 32bit|32bit load と SIMD 演算の組み合わせ

```

imax() {
    U11 CHIP; U11 LOOP1, LOOP0; U11 INIT1, INIT0;
    U11 AR[64][4]; /* output of EX in each unit */
    U11 BR[64][4][4];
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 cc0, ccl, cc1, cc2, cc3, ex0, ex1; U11 cof, rofs, oofs, k;
    for (top0; top0<M1/NCHIP; top0+=M1) { /* will be parallelized by multi-chip (M/#chip) */
        for (blk0; blk0<L; blk0+=H) { /* 3 重ループ展開の外側対象 */
            typedef struct {Uint i[8]} U18;
            Uint *ao[NCHIP];
            Uint *a[H][NCHIP];
            U18 *b[H], *b0[H], *b1[H], *b2[H], *b3[H];
            U18 *c0[NCHIP];
            U18 *c01[NCHIP], *c02[NCHIP], *c03[NCHIP];
            for (kx0; kx0<H; kx0++) {
                b[k] = B+(blk0+k)*M2; b0[k] = b[k]; b1[k] = (Uint*)b[k]+2; b2[k] = (Uint*)b[k]+4; b3[k] = (Uint*)b[k]+6;
            }
            for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
                a0[CHIP] = A+(CHIP*M1/NCHIP+top)*L;
                for (kx0; kx0<H; kx0++) {
                    a[k] [CHIP] = a0[CHIP]+blk0*k;
                    c0[CHIP] = C1+(CHIP*M1/NCHIP+top)*M2;
                    c00[CHIP] = (Uint*)c0[CHIP]+0; c01[CHIP] = (Uint*)c0[CHIP]+2; c02[CHIP] = (Uint*)c0[CHIP]+4; c03[CHIP] = (Uint*)c0[CHIP]+6;
                }
            }
        }
    }
#define sgemm00_core1(r, rm1, rp1) \
    mop(OP_LDR, 3, &BR[r][0][1], (U11)b0[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDR, 3, &BR[r][0][0], (U11)b1[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDR, 3, &BR[r][1][1], (U11)b2[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDR, 3, &BR[r][1][0], (U11)b3[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], M2, 0, 0, (U11)NULL, M2); \
/*dup load*/mop(OP_LDWR, 1, &BR[r][2][1], (U11)a[rm1][CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], L*RMGRP, 0, 0, (U11)NULL, L*RMGRP); \
exe(OP_FMA, &AR[rp1][0], AR[r][0], EXP_H3210, BR[r][2][1], EXP_H1010, BR[r][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][1], AR[r][1], EXP_H3210, BR[r][2][1], EXP_H1010, BR[r][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][2], AR[r][2], EXP_H3210, BR[r][2][1], EXP_H1010, BR[r][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][3], AR[r][3], EXP_H3210, BR[r][2][1], EXP_H1010, BR[r][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

#define sgemm00_final(r, rp1) \
    mop(OP_LDR, 3, &BR[rp1][0][1], (U11)c0[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
    mop(OP_LDR, 3, &BR[rp1][1][1], (U11)c01[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
    mop(OP_LDR, 3, &BR[rp1][2][1], (U11)c02[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
    mop(OP_LDR, 3, &BR[rp1][3][1], (U11)c03[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
exe(OP_FAD, &AR[rp1][0], AR[r][0], EXP_H3210, BR[rp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][1], AR[r][1], EXP_H3210, BR[rp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][2], AR[r][2], EXP_H3210, BR[rp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][3], AR[r][3], EXP_H3210, BR[rp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
mop(OP_STR, 3, &AR[rp1][0], (U11)oofs, (U11)c0[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
mop(OP_STR, 3, &AR[rp1][1], (U11)oofs, (U11)c01[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
mop(OP_STR, 3, &AR[rp1][2], (U11)oofs, (U11)c02[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
mop(OP_STR, 3, &AR[rp1][3], (U11)oofs, (U11)c03[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP);

//EMAXSA begin mm_mapdist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/*2*/ for (INIT1=1,LOOP1=RMGRP,rofs=(0-L*4)<<32|((0-M2*4)&0xffffffff); LOOP1--; INIT1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
/*1*/ for (INIT0=1,LOOP0=M2/W,rofs=(0-W*8)<<32|((0-W*8)&0xffffffff); LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, (*W*8)<<32|((W*8), EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?L(W*4)<<32|(M2*4):0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &oofs, rofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0xffffffff, OP_NOP, OLL); /* stage#1 */

    mop(OP_LDR, 3, &BR[1][0][1], (U11)b0[0], (U11)cofs, MSK_W1, (U11)b[0], M2, 0, 0, (U11)NULL, M2); /* stage#1 */
    mop(OP_LDR, 3, &BR[1][0][0], (U11)b1[0], (U11)cofs, MSK_W1, (U11)b[0], M2, 0, 0, (U11)NULL, M2); /* stage#1 */
    mop(OP_LDR, 3, &BR[1][1][1], (U11)b2[0], (U11)cofs, MSK_W1, (U11)b[0], M2, 0, 0, (U11)NULL, M2); /* stage#1 */
    mop(OP_LDR, 3, &BR[1][1][0], (U11)b3[0], (U11)cofs, MSK_W1, (U11)b[0], M2, 0, 0, (U11)NULL, M2); /* stage#1 2KB */
/*dup load*/mop(OP_LDWR, 1, &BR[1][2][1], (U11)a0[CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], L*RMGRP); /* stage#1 16KB */
exe(OP_FML, &AR[2][0], BR[1][0][1], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
exe(OP_FML, &AR[2][1], BR[1][0][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
exe(OP_FML, &AR[2][2], BR[1][1][1], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
exe(OP_FML, &AR[2][3], BR[1][1][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */

sgemm00_core1( 2, 1, 3);
sgemm00_core1( 3, 2, 4);
sgemm00_core1( 4, 3, 5);
sgemm00_core1( 5, 4, 6);
sgemm00_core1( 6, 5, 7);
sgemm00_core1( 7, 6, 8);
sgemm00_core1( 8, 7, 9);
sgemm00_core1( 9, 8, 10);
sgemm00_core1(10, 9, 11);
sgemm00_core1(11, 10, 12);
sgemm00_core1(12, 11, 13);
sgemm00_core1(13, 12, 14);
sgemm00_core1(14, 13, 15);
sgemm00_core1(15, 14, 16);
sgemm00_core1(16, 15, 17);
sgemm00_core1(17, 16, 18);
sgemm00_core1(18, 17, 19);
sgemm00_core1(19, 18, 20);
sgemm00_core1(20, 19, 21);
sgemm00_core1(21, 20, 22);
sgemm00_core1(22, 21, 23);
sgemm00_core1(23, 22, 24);
:
sgemm00_core1(59, 58, 60);
sgemm00_core1(60, 59, 61); /* H=60 */
/*final*/
sgemm00_final(61, 62);
} } }
//EMAXSA end
}
//EMAXSA drain_dirty_lmm
}

```

3.5.3 逆行列 (1/3)

```
cent% make -f Makefile-csim.emax6+dma inv-csim.emax6+dma clean
cent% ../../src/csim/csim -x inv-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma inv-zynq.emax6+dma clean
zynq% ./inv-zynq.emax6+dma
```

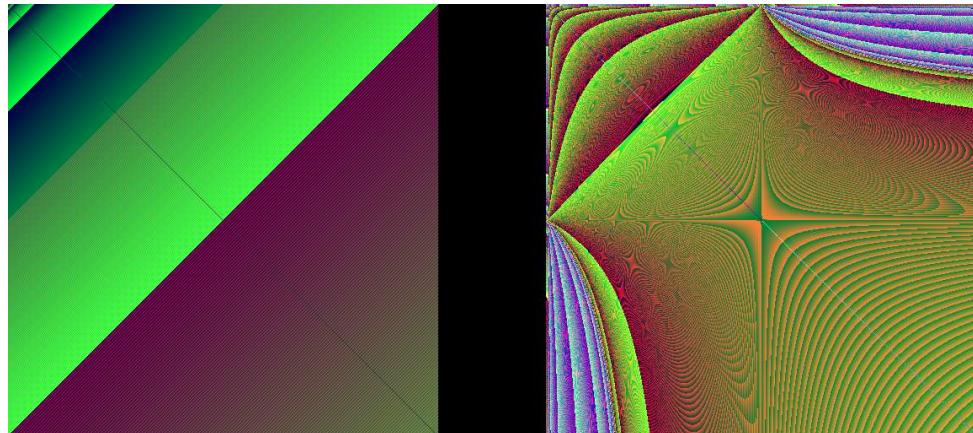


Figure 3.54: Inverse matrix

512x512 の逆行列計算（LU 分解）である。一度のバースト演算により 8 行分を計算する。ステンシル計算ではなく mapdist=0 である。

```
/* LU 分解 */
for (i=0; i<M+1; i++) {
    p[i] = i;
}
for (i=0; i<M; i++) {
    pmax = 0.0;
    k = -1;
    for (j=i; j<M; j++) {
        if (pmax < fabsf(A[p[j]*M+i])) {
            pmax = fabsf(A[p[j]*M+i]);
            k = j;
        }
    }
    if (k == -1) {
        fprintf(stderr, "can't solve\n");
        exit(1);
    }
    j = p[k]; p[k] = p[i]; p[i] = j;
    A[p[i]*M+i] = 1.0/A[p[i]*M+i];
    for (j=i+1; j<M; j++) {
        A[p[j]*M+i] *= A[p[i]*M+i];
        for (k=i+1; k<M; k++)
            A[p[j]*M+k] -= A[p[j]*M+i]*A[p[i]*M+k];
    }
}
```

```

/* LU 分解 */
for (i=0; i<M+1; i++)
    p[i] = i;
for (i=0; i<M; i++) { /* 列方向 */
    pmax = 0.0;
    k = -1;
    for (j=i; j<M; j++) { /* 行方向に探索 */
        if (pmax < fabsf(A[p[i]*M+i])) {
            pmax = fabsf(A[p[j]*M+i]);
            k = j;
        }
    }
    if (k == -1) {
        fprintf(stderr, "can't solve\n");
        exit(1);
    }
    j = p[k]; p[k] = p[i]; p[i] = j;
    A[p[i]*M+i] = 1.0/A[p[i]*M+i];
    for (j=i+1; j<M; j++) /* 行方向 */
        A[p[j]*M+i] *= A[p[i]*M+i];
    for (j=i+1; j<M; j=NCHIP*h) { /* 行方向 */
        /***** */
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            for (h=0; h<M-(i+1); h++) { /* 最内列方向 */
                for (ho=0; ho<h; ho++) { /* vertical (parallel) execution */
                    if (j+h*NCHIP+CHIP*M+i+k) -= A[p[j+h*NCHIP+CHIP]*M+i+k]*A[p[i]*M+i+1+k];
                    /* 後続の逆行列と異なり,accumulate ではなく要素毎の単独減算の繰返し */
                    /* const:A[p[j]] [0] * LMM A[p[ 0]] [*] */
                    /* ↓ */
                    /* LMM A[p[j>0]] [*] accumulate (column 方向に j,j+1,...,479 のため依存無) */
                }
            }
        }
        /***** */
        /* + - - - - - - - - - - */ /* A[p[i]] 先頭行 */ /* 先頭行は i 更新まで再利用可能 */
        /* | * > > > > > > > > */ /* A[p[j]] 次行から引く */ /* 1 行を LMM に写像 */
        /* | v + - - - - - - - - - */
        /* | v | v + - - - - - - - */ /* M/60 を収容して i 更新まで j+=60 を繰り返す */
        /* | v | v + - - - - - - - */ /* 行番号比較と cst による端数制御 */
        /* | v | v | v - + - - - - - - */ /* + CHIP#1 h=0 */
        /* | v | v | v - - + - - - - */ /* + CHIP#0 h=1 */
        /* | v | v | v - - - + - - - */ /* + CHIP#1 h=1 */
        /* | v | v | v - - - - + - - */ /* + CHIP#0 h=2 */
        /* | v | v | v - - - - - + - */ /* + CHIP#1 h=2 */
        /* | v | v | v - - - - - - + - */ /* + CHIP#0 h=3 */
        /* | v | v | v - - - - - - - + - */ /* + CHIP#1 h=3 */
        /***** */ /* 最大 60 行まで写像可能 */
    }
}

```

```

/* LU 分解 */
for (i=0; i<M+1; i++)
    p[i] = i;
for (i=0; i<M; i++) { /* 列方向 */
    pmax = 0.0;
    k = -1;
    for (j=i; j<M; j++) { /* 行方向に探索 */
        if (pmax < fabsf(A[j*M+i])) {
            pmax = fabsf(A[j*M+i]);
            k = j;
        }
    }
    if (k == -1) {
        fprintf(stderr, "can't solve\n");
        exit(1);
    }
    j = p[k]; p[k] = p[i]; p[i] = j;
    for (j=0; j<M; j++) { /* real pivoting */
        tmp = A[k*M+j]; A[k*M+j] = A[i*M+j]; A[i*M+j] = tmp;
    }
    A[i*M+i] = 1.0/A[i*M+i];
    for (j=i+1; j<M; j++) /* 行方向 */
        A[j*M+i] *= A[i*M+i];

    Uint *top = &A[i*M+i];
    Uint *topw = (U11)*top;
    Uint len = M-i;
    Uint len2 = len+(RMGRP-1)*M;
    Uint grp;
    /* FPGA 実機で j-loop の最終 (len=1) が動かないでの、ついでに ARM のほうが速そうな len を ARM で実行 2019/3/1 Nakashima */
    if (len < 16) { /* len<1 でも正常なので性能最大化で決めてよい */
        for (j=i+1; j<M; j+=NCHIP*H*RMGRP) { /* 行方向 */
            for (CHIP=0; CHIP<NCHIP; CHIP++)
                for (h=0; h<H; h++) { /* vertical (parallel) execution */
                    for (grp=0; grp<RMGRP; grp++) {
                        for (k=0; k<M-(i+1); k++) { /* 内列方向 */
                            if (j+h*NCHIP*RMGRP+CHIP*RMGRP+grp<0) A[(j+h*NCHIP*RMGRP+CHIP*RMGRP+grp)*M+i+k] -= A[(j+h*NCHIP*RMGRP+CHIP*RMGRP+grp)*M+i]*A[i*M+i+k];
                        }
                    }
                }
            for (j=i+1; j<M; j+=NCHIP*H*RMGRP) { /* 行方向 */
                ****
                Uint 100[NCHIP], 101[NCHIP], 102[NCHIP], 103[NCHIP], 104[NCHIP], 105[NCHIP], 106[NCHIP], 107[NCHIP]; /* j<M-(h*NCHIP+CHIP) */
                Uint 108[NCHIP], 109[NCHIP], 110[NCHIP], 111[NCHIP], 112[NCHIP], 113[NCHIP], 114[NCHIP], 115[NCHIP]; /* j<M-(h*NCHIP+CHIP) */
                Uint *d00[NCHIP], *d01[NCHIP], *d02[NCHIP], *d03[NCHIP], *d05[NCHIP], *d06[NCHIP], *d07[NCHIP]; /* A[p[j+b*NCHIP+CHIP]*M+k] */
                Uint *d08[NCHIP], *d09[NCHIP], *d10[NCHIP], *d11[NCHIP], *d12[NCHIP], *d13[NCHIP], *d14[NCHIP], *d15[NCHIP]; /* A[p[j+b*NCHIP+CHIP]*M+k] */
                Uint *d00w[NCHIP], *d01w[NCHIP], *d02w[NCHIP], *d03w[NCHIP], *d04w[NCHIP], *d05w[NCHIP], *d07w[NCHIP]; /* A[p[j+h*NCHIP+CHIP]*M+k] */
                Uint *d08w[NCHIP], *d09w[NCHIP], *d10w[NCHIP], *d11w[NCHIP], *d12w[NCHIP], *d13w[NCHIP], *d15w[NCHIP]; /* A[p[j+h*NCHIP+CHIP]*M+k] */
                for (CHIP=0; CHIP<NCHIP; CHIP++)
                    for (h=0; h<H; h++) { /* parallelized (OC#chip) */
                        100[NCHIP]=(j+0*NCHIP*RMGRP+CHIP*RMGRP<0)?(j+0*NCHIP*RMGRP+CHIP*RMGRP+M):101[NCHIP]=(j+1*NCHIP*RMGRP+CHIP*RMGRP+M);
                        114[NCHIP]=(j+14*NCHIP*RMGRP+CHIP*RMGRP<0)?(j+14*NCHIP*RMGRP+CHIP*RMGRP+M):115[NCHIP]=(j+15*NCHIP*RMGRP+CHIP*RMGRP+M);
                        d00[CHIP] = &A[100[NCHIP]*M+i];
                        d01[CHIP] = &A[101[NCHIP]*M+i];
                        :
                        d14[CHIP] = &A[114[CHIP]*M+i];
                        d15[CHIP] = &A[115[CHIP]*M+i];
                        d00w[CHIP]=(U11)*d00[CHIP];
                        d01w[CHIP]=(U11)*d01[CHIP];
                        :
                        d14w[CHIP]=(U11)*d14[CHIP];
                        d15w[CHIP]=(U11)*d15[CHIP];
                    }
                }
                ****
                /***** EMAX5A begin inv_xl mapdist=0 ****
                /* 2*/ for (CHIP=0; CHIP<NCHIP; CHIP++) {
                    /* 2*/ for (INITI=1,LOOP1=RMGRP,rofs=0-M+4; LOOP1--; INITI=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
                        /* 2*/ for (INITO=1,LOOP0=M-(i+1),cofs=0; LOOP0--; INITO=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
                            exe(OP_ADD, &cofs, INITO?cofs:cfs, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffff, OP_NOP, OLL); /* stage#0 */
                            exe(OP_ADD, &cofs, rofs, EXP_H3210, INITO?M+4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
                            exe(OP_ADD, &cofs, oofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffff, OP_NOP, OLL); /* stage#1 */
                            exe(OP_CMP_LT, &cco, 100[NCHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */
                            mop(OP_LDWR, 1, &BR[2][2][1], top, cofs, MSK_WO, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k] stage#2 LD
                            mop(OP_LDWR, 1, &BR[2][0][1], doo[CHIP], oofs, MSK_WO, d00w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#2 -->
                            mop(OP_LDWR, 1, &BR[2][1][1], doo1w[CHIP], rofs, MSK_WO, d01w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#2 -->
                            exe(OP_FM, &AR[2][0], BR[2][0][1], EXP_H3210, BR[2][1][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, 0); /* stage#2 | ■■■ | AR[1]
                            cex(OP_CE, &exo0, 0, 0, 0, cco, Oxaaaa); /* stage#2 | + ST v
                            mop(OP_STWR, ex0, &AR[2][0], oofs, d01[CHIP], MSK_D0, d01w[CHIP], len2, 0, 1, NULL, len2); /* stage#2 | + ST v
                            :
                            /* 2*/ if (H>1)
                                exe(OP_CMP_LT, &cco, 101[NCHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
                                mop(OP_LDWR, 1, &BR[3][2][1], top, cofs, MSK_WO, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k] stage#3 LD
                                mop(OP_LDWR, 1, &BR[3][0][1], doo[CHIP], oofs, MSK_WO, d00w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#3 -->
                                mop(OP_LDWR, 1, &BR[3][1][1], doo1w[CHIP], rofs, MSK_WO, d01w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#3 -->
                                exe(OP_FM, &AR[3][0], BR[3][0][1], EXP_H3210, BR[3][1][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, 0); /* stage#3 | ■■■ | AR[2]
                                cex(OP_CE, &exo0, 0, 0, 0, cco, Oxaaaa); /* stage#3 | AR[2]
                                mop(OP_STWR, ex0, &AR[3][0], oofs, d01[CHIP], MSK_D0, d01w[CHIP], len2, 0, 1, NULL, len2); /* stage#3 | + ST v
                            :
                            /* 2*/ if (H>12)
                                exe(OP_CMP_LT, &cco, 112[NCHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13 */
                                mop(OP_LDWR, 1, &BR[14][2][1], top, cofs, MSK_WO, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k] stage#14 LD
                                mop(OP_LDWR, 1, &BR[14][0][1], d12[CHIP], oofs, MSK_WO, d12w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#14 -->
                                mop(OP_LDWR, 1, &BR[14][1][1], d12[CHIP], rofs, MSK_WO, d12w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#14 -->
                                exe(OP_FM, &AR[14][0], BR[14][0][1], EXP_H3210, BR[14][1][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, 0); /* stage#14 | ■■■ | AR[13]
                                cex(OP_CE, &exo0, 0, 0, 0, cco, Oxaaaa); /* stage#14 | AR[13]
                                mop(OP_STWR, ex0, &AR[14][0], oofs, d12[CHIP], MSK_D0, d12w[CHIP], len2, 0, 1, NULL, len2); /* stage#14 | + ST v
                            :
                            /* 2*/ if (H>13)
                                exe(OP_CMP_LT, &cco, 113[NCHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#14 */
                                mop(OP_LDWR, 1, &BR[15][2][1], top, cofs, MSK_WO, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k] stage#15 LD
                                mop(OP_LDWR, 1, &BR[15][0][1], d13[CHIP], oofs, MSK_WO, d13w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#15 -->
                                mop(OP_LDWR, 1, &BR[15][1][1], d13[CHIP], rofs, MSK_WO, d13w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#15 -->
                                exe(OP_FM, &AR[15][0], BR[15][0][1], EXP_H3210, BR[15][1][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, 0); /* stage#15 | ■■■ | AR[14]
                                cex(OP_CE, &exo0, 0, 0, 0, cco, Oxaaaa); /* stage#15 | AR[14]
                                mop(OP_STWR, ex0, &AR[15][0], oofs, d13[CHIP], MSK_D0, d13w[CHIP], len2, 0, 1, NULL, len2); /* stage#15 | + ST v
                            :
                            /* 2*/ if (H>14)
                                exe(OP_CMP_LT, &cco, 114[NCHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#15 */
                                mop(OP_LDWR, 1, &BR[16][2][1], top, cofs, MSK_WO, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k] stage#16 LD
                                mop(OP_LDWR, 1, &BR[16][0][1], d14[CHIP], oofs, MSK_WO, d14w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#16 -->
                                mop(OP_LDWR, 1, &BR[16][1][1], d14[CHIP], rofs, MSK_WO, d14w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#16 -->
                                exe(OP_FM, &AR[16][0], BR[16][0][1], EXP_H3210, BR[16][1][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, 0); /* stage#16 | ■■■ | AR[15]
                                cex(OP_CE, &exo0, 0, 0, 0, cco, Oxaaaa); /* stage#16 | AR[15]
                                mop(OP_STWR, ex0, &AR[16][0], oofs, d14[CHIP], MSK_D0, d14w[CHIP], len2, 0, 1, NULL, len2); /* stage#16 | + ST v
                            :
                            /* 2*/ if (H>15)
                                exe(OP_CMP_LT, &cco, 115[NCHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#16 */
                                mop(OP_LDWR, 1, &BR[17][2][1], top, cofs, MSK_WO, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k] stage#17 LD
                                mop(OP_LDWR, 1, &BR[17][0][1], d15[CHIP], oofs, MSK_WO, d15w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#17 -->
                                mop(OP_LDWR, 1, &BR[17][1][1], d15[CHIP], rofs, MSK_WO, d15w[CHIP], len2, 0, 1, NULL, len2); /* A[p[j+h*NCHIP+CHIP]*M+k] stage#17 -->
                                exe(OP_FM, &AR[17][0], BR[17][0][1], EXP_H3210, BR[17][1][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, 0); /* stage#17 | ■■■ | AR[16]
                                cex(OP_CE, &exo0, 0, 0, 0, cco, Oxaaaa); /* stage#17 | AR[16]
                                mop(OP_STWR, ex0, &AR[17][0], oofs, d15[CHIP], MSK_D0, d15w[CHIP], len2, 0, 1, NULL, len2); /* stage#17 | + ST v
                            :
                            /* 2*/ endif
                        }
                    }
                }
                ****
                /* j-loop */
            }
            //***** EMAX5A drain_dirty_lmm
        }
    }

```

inv+rmm-inv_x1-emax6.obj

BR/row: max=11 min=4 ave=9 EA/row: max=4 min=0 ave=3 ARpass/row: max=0

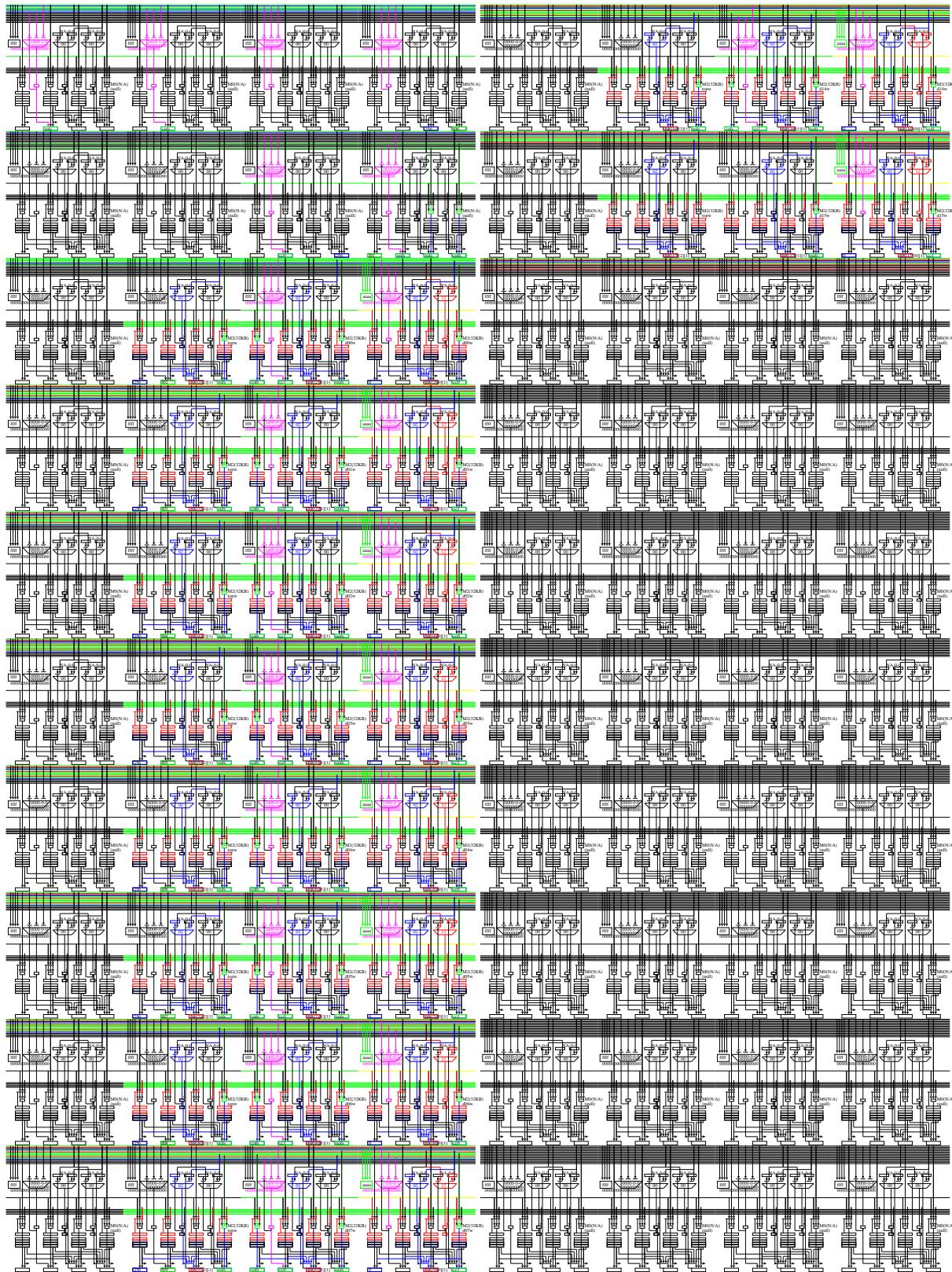


Figure.3.55: 逆行列 (1/3)

3.5.4 逆行列 (2/3)

512x512 の逆行列計算（前進消去）である。一度のバースト演算により 8 行分を計算する。ステンシル計算ではなく mapdist=0 である。

```

/* 逆行列前半 */
for (i=0; i<M; i++) {
    for (j=0; j<M; j++)
        b[p[j]] = (i==j)?1.0:0.0;
/*for (j=1; j<M; j++) { /*//通常の連立一時方程式の場合*/
    for (j=i+1; j<M; j++) { /* 逆行列 (b[]!=E) の場合, k<i では b[]==0 なので j=i+1 から開始 */
        /*for (k=0; k<j; k++) /*//通常の連立一時方程式の場合*/
            for (k=i; k<j; k++) /* 逆行列 (b[]!=E) の場合, k<i では b[]==0 なので k=i から開始 */
                b[p[j]] -= A[p[j]*M+k]*b[p[k]];
    }
}

/* 逆行列求める */
for (i=0; i<M; i++) { /* 列方向 */
    for (j=0; j<M; j++) /* 行方向 */
        b[i*M+j] = (i==j)?1.0:0.0;
}
for (i=0; i<M; i+=NCHIP*H) { /* 列方向 */
    for (j=1; j<M; j++) { /* 行方向 */
        /*****
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            for (k=0; k<j; k++) { /* 最内列方向 */
                for (h=0; h<H; h++) { /* vertical (parallel) execution */
                    b[(i+NCHIP*H+h)*M+j] -= A[p[j]*M+k]*b[(i+CHIP*H+h)*M+k];
                    /* b[*] を縦に配置する場合、も縦配置。 j が列方向に対応するが列方向の移動で k も長くなる */
                    /* 可変長 k を H 方向に展開写像するのは難しい。k は read-modify-write の回転数に写像するしかない */
                    /* b[*] と A[j][k] が同一 LMM に入る前提 最大 64KB/4/2=各 8K 要素,b[*] をいかに動かさないか */
                    /* 回転数( j を一齊適用するには,i を WxH 方向に展開するのが自然 */
                    /* ↓★ A[p[j]][*] を broadcast 可能 各 A[p[j]][*] は p[j] が不連続なので 1K 要素まで収容。つまり二重ループ展開は無理 */
                    /* +-----+ +-----+ +-----+ +-----+ */
                    /* b[ 3][j]-=A[p[j]][0:j-1] b[ 3][0:j-1] b[ 2][*] b[ 1][*] b[ 0][*] */
                    /* */
                    /* b[ 7][j]-=A[p[j]][0:j-1] b[ 7][0:j-1] b[ 6][*] b[ 5][*] b[ 4][*] */
                    /* b[59][j]-=A[p[j]][0:j-1] b[59][0:j-1] b[58][*] b[57][*] b[56][*] */
                }
            }
        }
    }
}
*****
```

```

/* 逆行行列前半 */
for (i=0; i<M; i++) { /* 列方向 */
    for (j=0; j<M; j++) { /* 行方向 */
        b[i*M+j] = (i==j)?1.0:0.0;
    }
}
for (i=0; i<M; i+=NCHIP*H) { /* 列方向 */
/*for (j=1; j<M; j++) { /* 通常の連立一時方程式の場合 */
    for (j=i+1; j<M; j++) { /* 逆行行列 (b[]-E) の場合, k<i では b[]==0 なので j=i+1 から開始 */
        Uint *top = &A[p[j]*M+i];
        Uint *topw = (U11)*top;
        /*Uint len = (j+1)/2;*/
        Uint len = j-1; /* b が単位行列の場合, k では b[]==0 なので k=i から開始 */
        /****** */
        if (len < 2) { /* len<1 でも正常なので性能最大化で決めてよい */
            for (CHIP=0; CHIP<NCHIP; CHIP++)
                /*for (k=0; k<j; k++) { /* 通常の連立一時方程式の場合 */
                    for (k=i; k<j; k++) { /* 逆行行列 (b[]-E) の場合, k では b[]==0 なので k=i から開始 */
                        for (h=0; h<H; h++) { /* vertical (parallel) execution */
                            b[(i+NCHIP*h)+M+j] -= A[p[j]*M+k]*b[(i+NCHIP*h)+M+k];
                        }
                    }
                }
            else {
                Uint jc = j-i;
                U11 Ajk; /* k=0...j-1 */
                U11 b001;
                Uint 1000[NCHIP], 1010[NCHIP], 1020[NCHIP], 1030[NCHIP], 1040[NCHIP], 1050[NCHIP], 1060[NCHIP], 1070[NCHIP]; /* (i+NCHIP*h+h+W+w) */
                Uint 1080[NCHIP], 1090[NCHIP], 1100[NCHIP], 1110[NCHIP], 1120[NCHIP], 1130[NCHIP], 1140[NCHIP], 1150[NCHIP]; /* (i+NCHIP*h+h+W+w) */
                Uint *t000[NCHIP], *t010[NCHIP], *t020[NCHIP], *t030[NCHIP], *t040[NCHIP], *t050[NCHIP], *t070[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080[NCHIP], *t090[NCHIP], *t020w[NCHIP], *t030w[NCHIP], *t040w[NCHIP], *t050w[NCHIP], *t070w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* (i+NCHIP*h+h+W+w)*M+k */
                for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
                    1000[CHIP] = (i+NCHIP*h+0)*M; 1010[CHIP] = (i+NCHIP*h+1)*M; 1020[CHIP] = (i+NCHIP*h+2)*M; 1030[CHIP] = (i+NCHIP*h+3)*M;
                    :
                    1120[CHIP] = (i+NCHIP*h+12)*M; 1130[CHIP] = (i+NCHIP*h+13)*M; 1140[CHIP] = (i+NCHIP*h+14)*M; 1150[CHIP] = (i+NCHIP*h+15)*M;
                    t000[CHIP] = &b[1000[CHIP]+i]; t010[CHIP] = &b[1010[CHIP]+i];
                    :
                    t140[CHIP] = &b[1140[CHIP]+i]; t150[CHIP] = &b[1150[CHIP]+i];
                    t000w[CHIP] = (U11)*t000[CHIP]; t010w[CHIP] = (U11)*t010[CHIP];
                    :
                    t140w[CHIP] = (U11)*t140[CHIP]; t150w[CHIP] = (U11)*t150[CHIP];
                    d000[CHIP] = &b[1000[CHIP]+j]; d010[CHIP] = &b[1010[CHIP]+j];
                    :
                    d140[CHIP] = &b[1140[CHIP]+j]; d150[CHIP] = &b[1150[CHIP]+j];
                    d000w[CHIP] = (U11)*d000[CHIP]; d010w[CHIP] = (U11)*d010[CHIP];
                    :
                    d140w[CHIP] = (U11)*d140[CHIP]; d150w[CHIP] = (U11)*d150[CHIP];
                }
            }
        }
    }
}

//EMAXSA begin inv_x2 mapdist=0
/* */ for (CHIP=0; CHIP<NCHIP; CHIP++) {
/* */ for (INITO=1,LOOP0=jc,cofs=0-4; LOOP0--; INITO=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &cofs, cof, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffLLL, OP_NOP, OLL); /* stage#0 */
    mop(OP_LDWR, 1, &Ajk, cof, MSK_W0, topw, len, 0, 0, NULL, len); /* A[p[j]*M+k] */
    mop(OP_LDWR, 1, &BR[1][3][1], t000[CHIP], cof, MSK_W0, t000w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
    mop(OP_LDWR, 1, &b000, d000[CHIP], 0, MSK_W0, d000w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
    exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2.0 +- xxxx+ST v */
    mop(OP_STWR, 1, &b000, 0, d000[CHIP], MSK_DO, d000w[CHIP], 1, 0, 1, NULL, 1); /* stage#2.0 ----- xxxx */

    #if (H>1)
        mop(OP_LDWR, 1, &BR[2][3][1], t010[CHIP], cof, MSK_W0, t010w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d010[CHIP], 0, MSK_W0, d010w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
        exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[2][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3.0 +- xxxx+ST v */
        mop(OP_STWR, 1, &b000, 0, d010[CHIP], MSK_DO, d010w[CHIP], 1, 0, 1, NULL, 1); /* stage#3.0 ----- xxxx */

        #if (H>8)
            mop(OP_LDWR, 1, &BR[9][3][1], t080[CHIP], cof, MSK_W0, t080w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
            mop(OP_LDWR, 1, &b000, d080[CHIP], 0, MSK_W0, d080w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
            exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[9][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10.0----- xxxx */
            mop(OP_STWR, 1, &b000, 0, d080[CHIP], MSK_DO, d080w[CHIP], 1, 0, 1, NULL, 1); /* stage#10.0----- xxxx */

            #if (H>9)
                mop(OP_LDWR, 1, &BR[10][3][1], t090[CHIP], cof, MSK_W0, t090w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                mop(OP_LDWR, 1, &b000, d090[CHIP], 0, MSK_W0, d090w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[10][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11.0----- xxxx */
                mop(OP_STWR, 1, &b000, 0, d090[CHIP], MSK_DO, d090w[CHIP], 1, 0, 1, NULL, 1); /* stage#11.0----- xxxx */

                #if (H>10)
                    mop(OP_LDWR, 1, &BR[11][3][1], t100[CHIP], cof, MSK_W0, t100w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                    mop(OP_LDWR, 1, &b000, d100[CHIP], 0, MSK_W0, d100w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                    exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[11][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12.0----- xxxx */
                    mop(OP_STWR, 1, &b000, 0, d100[CHIP], MSK_DO, d100w[CHIP], 1, 0, 1, NULL, 1); /* stage#12.0----- xxxx */

                    #if (H>11)
                        mop(OP_LDWR, 1, &BR[12][3][1], t110[CHIP], cof, MSK_W0, t110w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                        mop(OP_LDWR, 1, &b000, d110[CHIP], 0, MSK_W0, d110w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                        exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[12][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13.0----- xxxx */
                        mop(OP_STWR, 1, &b000, 0, d110[CHIP], MSK_DO, d110w[CHIP], 1, 0, 1, NULL, 1); /* stage#13.0----- xxxx */

                        #if (H>12)
                            mop(OP_LDWR, 1, &BR[13][3][1], t120[CHIP], cof, MSK_W0, t120w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                            mop(OP_LDWR, 1, &b000, d120[CHIP], 0, MSK_W0, d120w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                            exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[13][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#14.0----- xxxx */
                            mop(OP_STWR, 1, &b000, 0, d120[CHIP], MSK_DO, d120w[CHIP], 1, 0, 1, NULL, 1); /* stage#14.0----- xxxx */

                            #if (H>13)
                                mop(OP_LDWR, 1, &BR[14][3][1], t130[CHIP], cof, MSK_W0, t130w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                                mop(OP_LDWR, 1, &b000, d130[CHIP], 0, MSK_W0, d130w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                                exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[14][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#15.0----- xxxx */
                                mop(OP_STWR, 1, &b000, 0, d130[CHIP], MSK_DO, d130w[CHIP], 1, 0, 1, NULL, 1); /* stage#15.0----- xxxx */

                                #if (H>14)
                                    mop(OP_LDWR, 1, &BR[15][3][1], t140[CHIP], cof, MSK_W0, t140w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                                    mop(OP_LDWR, 1, &b000, d140[CHIP], 0, MSK_W0, d140w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                                    exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[15][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#16.0----- xxxx */
                                    mop(OP_STWR, 1, &b000, 0, d140[CHIP], MSK_DO, d140w[CHIP], 1, 0, 1, NULL, 1); /* stage#16.0----- xxxx */

                                    #if (H>15)
                                        mop(OP_LDWR, 1, &BR[16][3][1], t150[CHIP], cof, MSK_W0, t150w[CHIP], len, 0, 1, NULL, len); /* b[(i+NCHIP*h+h+W+w)*M+k] */
                                        mop(OP_LDWR, 1, &b000, d150[CHIP], 0, MSK_W0, d150w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+NCHIP*h+h+W+w)*M+j] */
                                        exe(OP_FMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[16][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#17.0----- xxxx */
                                        mop(OP_STWR, 1, &b000, 0, d150[CHIP], MSK_DO, d150w[CHIP], 1, 0, 1, NULL, 1); /* stage#17.0----- xxxx */
                                    #endif
                                #endif
                            #endif
                        #endif
                    #endif
                #endif
            #endif
        #endif
    #endif
}

//EMAXSA end
//EMAXSA drain_dirty_lmm
} /* else */
/****** */
} /* j-loop */
}

```

inv+rmm-inv_x2-emax6.obj

BR/row: max=5 min=1 ave=4 EA/row: max=5 min=0 ave=4 ARpass/row: max=0

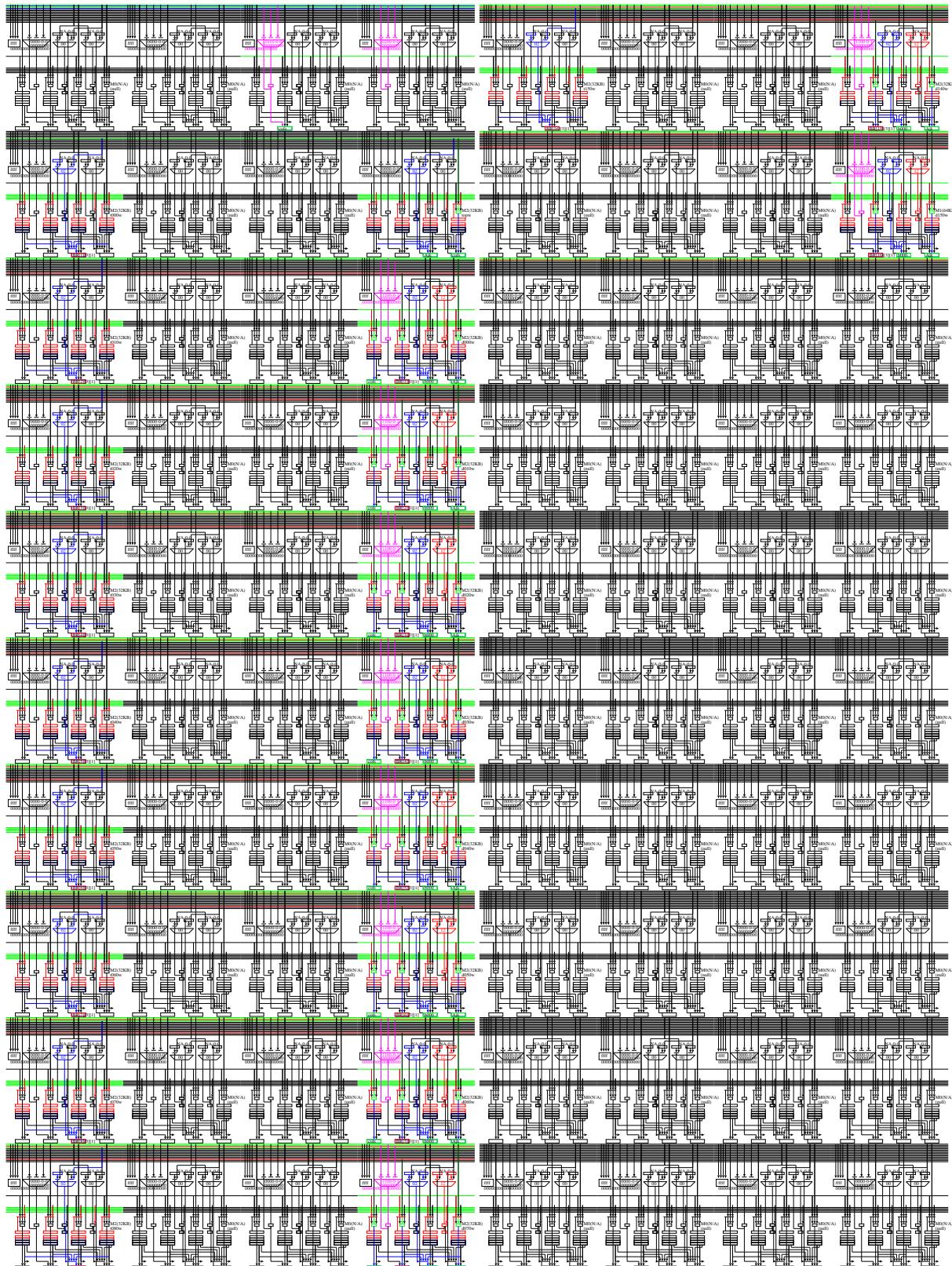


Figure.3.56: 逆行列 (2/3)

3.5.5 逆行列 (3/3)

512x512 の逆行列計算（後退代入）である。一度のバースト演算により 8 行分を計算する。ステンシル計算ではなく mapdist=0 である。

```
/* 逆行列後半 */
for (i=0; i<M; i++) {
    for (j=M-1; j>=0; j--) {
        for (k=M-1; k>j; k--) {
            b[p[j]] -= A[p[j]*M+k]*x[k];
            inv0[j*M+p[i]] = x[j] = b[p[j]]*A[p[j]*M+j];
        }
    }
}

for (i=0; i<NCHIP; i++) { /* 列方向 */
    for (j=M-1; j>=0; j--) { /* 行方向 */
        /*****
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            for (k=M-1; k>j; k--) { /* 最内列方向 */
                for (h=0; h<H; h++) { /* vertical (parallel) execution */
                    b[(i+CHIP*h+h)*M+j] -= A[p[j]*M+k]*x[(i+CHIP*h+h)*M+k];
                    /* x[*] と A[*][*] が同一 LMM に入る前提 最大 64KB/4=各 8K 要素,x[*] をいかに動かさないか */
                    /* 回転数 j を一齊適用するには,i を WxH 方向に展開するのが自然 */
                    /*
                     ↓★ A[p[j]][*] を broadcast 可能 各 A[p[j]][*] は p[j] が不連続なので 1K 要素まで収容、つまり二重ループ展開は無理 */
                    /* +-----+ +-----+ +-----+ +-----+ */
                    /* b[ 3][j]-=A[p[j]][M-1:j+1] x[ 3][M-1:j+1] b[ 2][*] b[ 1][*] b[ 0][*] */
                    /*
                    /* b[ ?][j]-=A[p[j]][M-1:j+1] x[ ?][M-1:j+1] b[ 6][*] b[ 5][*] b[ 4][*] */
                    /* b[59][j]-=A[p[j]][M-1:j+1] x[59][M-1:j+1] b[58][*] b[57][*] b[56][*] */
                }
            }
        }
        /*****
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            for (h=0; h<H; h++) { /* vertical (parallel) execution */
                inv1[j*M+p[i+CHIP*h+h+w]] = x[(i+CHIP*h+h)*M+j] = A[p[j]*M+j]*b[(i+CHIP*h+h)*M+j]; /* PIO にて LMM の x[i*M+j] を直接更新 */
                /* i はそのまま,j を切替え */
            }
        }
    }
}
```

```

/* 逆行列後半 */
for (i=0; i<M; i+=NCHIP*N) { /* 列方向 */
    for (j=M-1; j>0; j--) { /* 行方向 */
        if (j<M-1) {
            Uint *top = &A[p[j]*M+j+1];
            Uint *topw = (U11)*top;
            Uint len = M-j-1;
            /****** */
            if (len < 2) { /* len<1 でも正常なので性能最大化で決めてよい */
                for (CHIP=0; CHIP<NCHIP; CHIP++) {
                    for (k=M-1; k>j; k--) { /* 最内列方向 */
                        for (h=0; h<H; h++) { /* vertical (parallel) execution */
                            b[(i+CHIP*N+h)*M+j] = A[p[j]*M+k]*x[(i+CHIP*N+h)*M+k];
                        }
                    }
                }
            } else {
                Uint jc = M-j-1;
                U11 Ajk; /* k=j+1...M-1 */
                U11 b000, b001;
                Uint 1000[NCHIP], 1010[NCHIP], 1020[NCHIP], 1030[NCHIP], 1040[NCHIP], 1050[NCHIP], 1060[NCHIP]; /* (i+CHIP*W*H+h*W+w) */
                Uint 1080[NCHIP], 1090[NCHIP], 1100[NCHIP], 1110[NCHIP], 1120[NCHIP], 1130[NCHIP], 1140[NCHIP], 1150[NCHIP]; /* (i+CHIP*W*H+h*W+w) */
                Uint *t000[NCHIP], *t010[NCHIP], *t020[NCHIP], *t030[NCHIP], *t040[NCHIP], *t050[NCHIP], *t060[NCHIP], *t070[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *t080[NCHIP], *t090[NCHIP], *t100[NCHIP], *t110[NCHIP], *t120[NCHIP], *t130[NCHIP], *t140[NCHIP], *t150[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *t000w[NCHIP], *t010w[NCHIP], *t020w[NCHIP], *t030w[NCHIP], *t040w[NCHIP], *t050w[NCHIP], *t060w[NCHIP], *t070w[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *t080w[NCHIP], *t090w[NCHIP], *t100w[NCHIP], *t110w[NCHIP], *t120w[NCHIP], *t130w[NCHIP], *t140w[NCHIP], *t150w[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *t000o[NCHIP], *d010[NCHIP], *d020[NCHIP], *d030[NCHIP], *d040[NCHIP], *d050[NCHIP], *d060[NCHIP], *d070[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *t080o[NCHIP], *d090[NCHIP], *d100[NCHIP], *d110[NCHIP], *d120[NCHIP], *d130[NCHIP], *d140[NCHIP], *d150[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *t000w[NCHIP], *d010w[NCHIP], *d020w[NCHIP], *d030w[NCHIP], *d040w[NCHIP], *d050w[NCHIP], *d060w[NCHIP], *d070w[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                Uint *d080w[NCHIP], *d090w[NCHIP], *d100w[NCHIP], *d110w[NCHIP], *d120w[NCHIP], *d130w[NCHIP], *d140w[NCHIP], *d150w[NCHIP]; /* b[(i+CHIP*W*H+h*W+w)*M+k] */
                for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output port channels are parallelized by multi-chip (OC/#chip) */
                    1000[CHIP] = (i+CHIP*N+h)*M+j+1; 1010[CHIP] = (i+CHIP*N+h+1)*M+j+1;
                    :
                    1140[CHIP] = (i+CHIP*N+h+14)*M+j+1; 1150[CHIP] = (i+CHIP*N+h+15)*M+j+1;
                    t000[CHIP] = &x[1000[CHIP]]; t010[CHIP] = &x[1010[CHIP]];
                    :
                    t140[CHIP] = &x[1140[CHIP]]; t150[CHIP] = &x[1150[CHIP]];
                    t000w[CHIP] = (U11)t000[CHIP]; t010w[CHIP] = (U11)t010[CHIP];
                    :
                    t140w[CHIP] = (U11)t140[CHIP]; t150w[CHIP] = (U11)t150[CHIP];
                    d000[CHIP] = &b[1000[CHIP]-1]; d010[CHIP] = &b[1010[CHIP]-1];
                    :
                    d140[CHIP] = &b[1140[CHIP]-1]; d150[CHIP] = &b[1150[CHIP]-1];
                    d000w[CHIP] = (U11)d000[CHIP]; d010w[CHIP] = (U11)d010[CHIP];
                    :
                    d140w[CHIP] = (U11)d140[CHIP]; d150w[CHIP] = (U11)d150[CHIP];
                }
            }
        }
    }
}

//EMAX5A begin inv_x3 mapdist=0
/* 2 */ for (CHIP=0; CHIP<NCHIP; CHIP++) {
    /* 1 */ for (INITO=1, LOOP=jc, cofsjc=4; LOOP--; INITO=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
        exec(OP_ADD, &cofs, cofs, EXP_H3210, -4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
        mop(OP_LDWR, 1, &Ajk, top, cofs, MSK_W0, topw, len, 0, 0, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &BR[1][3][1], t000[CHIP], cofs, MSK_W0, t000w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d000[CHIP], 0, MSK_W0, d000w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2.0 */
        mop(OP_STWR, 1, &b000, 0, d000[CHIP], MSK_D0, d000w[CHIP], 1, 0, 1, NULL, 1); /* stage#2.0 */

        #if (H>1)
        mop(OP_LDWR, 1, &BR[2][3][1], t010[CHIP], cofs, MSK_W0, t010w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d010[CHIP], 0, MSK_W0, d010w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[2][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3.0 */
        mop(OP_STWR, 1, &b000, 0, d010[CHIP], MSK_D0, d010w[CHIP], 1, 0, 1, NULL, 1); /* stage#3.0 */
        :
        #endif

        #if (H>8)
        mop(OP_LDWR, 1, &BR[9][3][1], t080[CHIP], cofs, MSK_W0, t080w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d080[CHIP], 0, MSK_W0, d080w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[9][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10.0 */
        mop(OP_STWR, 1, &b000, 0, d080[CHIP], MSK_D0, d080w[CHIP], 1, 0, 1, NULL, 1); /* stage#10.0 */
        :
        #endif

        #if (H>9)
        mop(OP_LDWR, 1, &BR[10][3][1], t090[CHIP], cofs, MSK_W0, t090w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d090[CHIP], 0, MSK_W0, d090w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[10][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11.0 */
        mop(OP_STWR, 1, &b000, 0, d090[CHIP], MSK_D0, d090w[CHIP], 1, 0, 1, NULL, 1); /* stage#11.0 */
        :
        #endif

        #if (H>10)
        mop(OP_LDWR, 1, &BR[11][3][1], t100[CHIP], cofs, MSK_W0, t100w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d100[CHIP], 0, MSK_W0, d100w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[11][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12.0 */
        mop(OP_STWR, 1, &b000, 0, d100[CHIP], MSK_D0, d100w[CHIP], 1, 0, 1, NULL, 1); /* stage#12.0 */
        :
        #endif

        #if (H>11)
        mop(OP_LDWR, 1, &BR[12][3][1], t110[CHIP], cofs, MSK_W0, t110w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d110[CHIP], 0, MSK_W0, d110w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[12][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13.0 */
        mop(OP_STWR, 1, &b000, 0, d110[CHIP], MSK_D0, d110w[CHIP], 1, 0, 1, NULL, 1); /* stage#13.0 */
        :
        #endif

        #if (H>12)
        mop(OP_LDWR, 1, &BR[13][3][1], t120[CHIP], cofs, MSK_W0, t120w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d120[CHIP], 0, MSK_W0, d120w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[13][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#14.0 */
        mop(OP_STWR, 1, &b000, 0, d120[CHIP], MSK_D0, d120w[CHIP], 1, 0, 1, NULL, 1); /* stage#14.0 */
        :
        #endif

        #if (H>13)
        mop(OP_LDWR, 1, &BR[14][3][1], t130[CHIP], cofs, MSK_W0, t130w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d130[CHIP], 0, MSK_W0, d130w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[14][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#15.0 */
        mop(OP_STWR, 1, &b000, 0, d130[CHIP], MSK_D0, d130w[CHIP], 1, 0, 1, NULL, 1); /* stage#15.0 */
        :
        #endif

        #if (H>14)
        mop(OP_LDWR, 1, &BR[15][3][1], t140[CHIP], cofs, MSK_W0, t140w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d140[CHIP], 0, MSK_W0, d140w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[15][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#16.0 */
        mop(OP_STWR, 1, &b000, 0, d140[CHIP], MSK_D0, d140w[CHIP], 1, 0, 1, NULL, 1); /* stage#16.0 */
        :
        #endif

        #if (H>15)
        mop(OP_LDWR, 1, &BR[16][3][1], t150[CHIP], cofs, MSK_W0, t150w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*W*H+h*W+w)*M+k] */
        mop(OP_LDWR, 1, &b000, d150[CHIP], 0, MSK_W0, d150w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*W*H+h*W+w)*M+j] */
        exe(OP_FM3, &b000, EXP_H3210, Ajk, EXP_H3210, BR[16][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#17.0 */
        mop(OP_STWR, 1, &b000, 0, d150[CHIP], MSK_D0, d150w[CHIP], 1, 0, 1, NULL, 1); /* stage#17.0 */
        :
        #endif
    }
}

//EMAX5A end
//EMAX5A drain_dirty_lmm
} /* else */
/* if (j<M-1) */
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (h=0; h<H; h++) { /* vertical (parallel) execution */
        inv1[j*Mp+i*(CHIP*N+h)] = x[(i+CHIP*N+h)*M+j]; /* PIO にて LMM の x[i*M+j] を直接更新 */
        /* i はそのままで, j を切替え */
    }
}
} /* j-loop */
}

```

inv+rmm-inv_x3-emax6.obj

BR/row: max=5 min=1 ave=4 EA/row: max=5 min=0 ave=4 ARpass/row: max=0

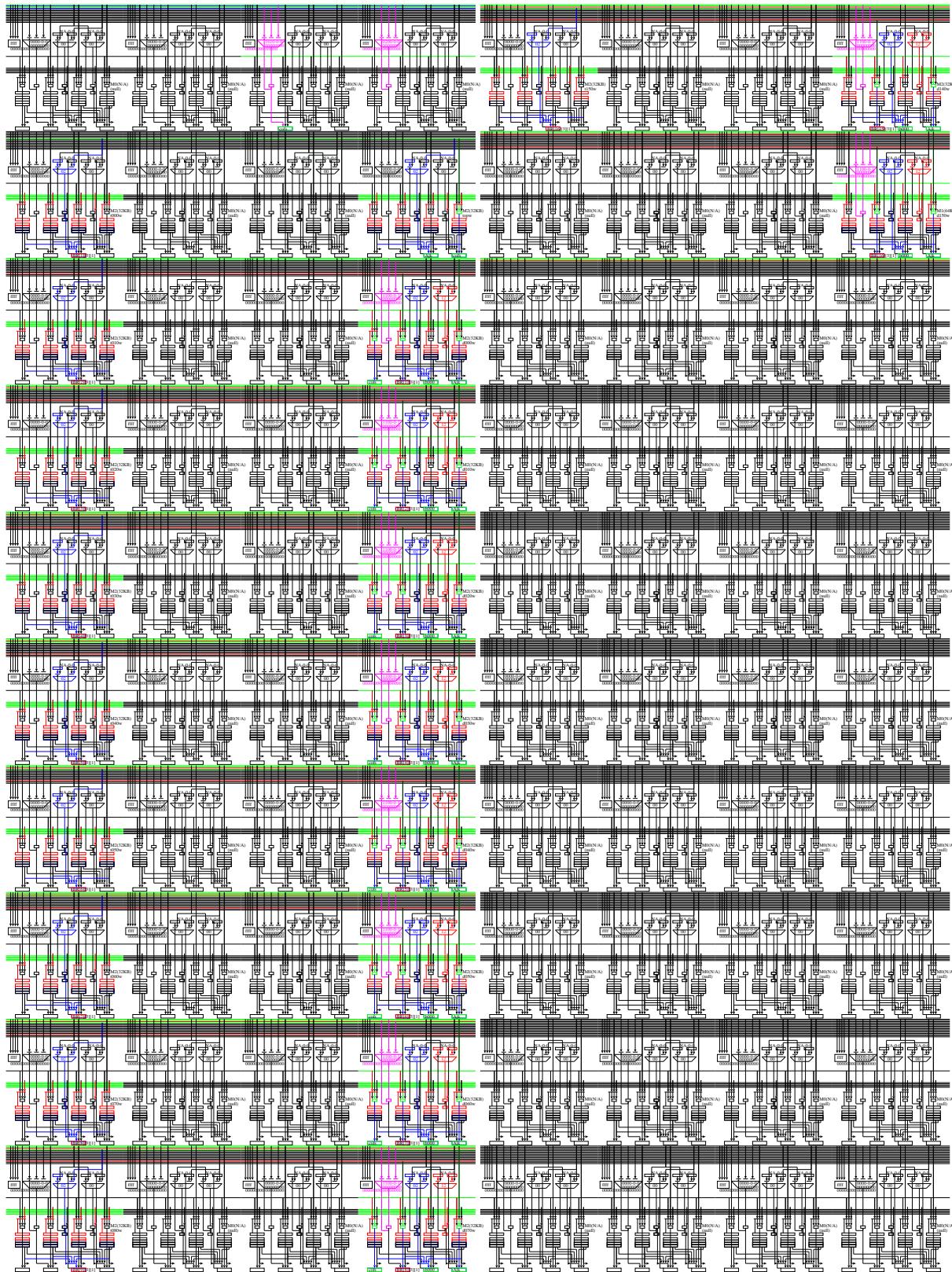


Figure.3.57: 逆行列 (3/3)

3.5.6 Lightfield レンダリング

```
cent% make -f Makefile-csim.emax6+dma gather-csim.emax6+dma clean
cent% ../../src/csim/csim -x gather-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma gather-zynq.emax6+dma clean
zynq% ./gather-zynq.emax6+dma
```



Figure 3.58: Gather

解像度 7500x7500 ライトフィールド画像処理のレンダリングである。規則的なステンシル計算ではないため mapdist=0 である。

```
orig()
{
    ry = (R+ofs)*IM;
    rx = (R+ofs);
    int w, pix, cvalR, cvalG, cvalB;

    for (row=PAD; row<OM-PAD; row++) {
        for (col=PAD; col<OM-PAD; col++) {
            c = ((row>>4)*R + (((~row&15)*ofs)>>4))*IM
                + (col>>4)*R + (((~col&15)*ofs)>>4);
            cvalR=0;
            cvalG=0;
            cvalB=0;
            for (i=-1; i<=1; i++) {
                for (j=-1; j<=1; j++) {
                    Uint pix = in[c+ry*i+rx*j];
                    w = weight[WBASE+i*MAXDELTA*2+j];
                    cvalR += ((pix>>24)&255)*w;
                    cvalG += ((pix>>16)&255)*w;
                    cvalB += ((pix>> 8)&255)*w;
                    count0++;
                }
            }
            out0[row*OM+col] = ((cvalR>>8)<<24) | ((cvalG>>8)<<16) | ((cvalB>>8)<<8);
        }
    }
}
```

```
imax()
{
    U11 CHIP;
    ry = (R+ofs)*IM;
    rx = (R+ofs);
    int w, pix, cvalR, cvalG, cvalB;

    for (row=0; row<RRANGE; row++) { /* 0..381 */
        for (CHIP=0; CHIP<CHIP; CHIP++) {
            for (col=0; col<CRANGE; col++) {
                for (oc=0; oc<OMAP; oc++) {
                    c=((CHIP*OMAP+oc)*RRANGE+row+PAD)>>4)*R + (((((CHIP*OMAP+oc)*RRANGE+row+PAD)&15)*ofs)>>4))*IM
                        + ((col+PAD)>>4)*R + ((((~(col+PAD)&15)*ofs)>>4));
                    /* 256 512 256 */
                    pix = in[c+ry*(-1)+rx*(-1)]; w = 16; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    pix = in[c+ry*(-1)+rx*( 0)]; w = 32; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    pix = in[c+ry*( 0)+rx*(-1)]; w = 16; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    /* 512 1024 512 */
                    pix = in[c+ry*( 0)+rx*(-1)]; w = 32; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    pix = in[c+ry*( 0)+rx*( 0)]; w = 64; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    pix = in[c+ry*( 1)+rx*( 0)]; w = 32; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    /* 256 512 256 */
                    pix = in[c+ry*( 1)+rx*(-1)]; w = 16; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    pix = in[c+ry*( 1)+rx*( 1)]; w = 32; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    pix = in[c+ry*( 1)+rx*( 1)]; w = 16; cvalR=((pix>>24)&255)*w; cvalG=((pix>>16)&255)*w; cvalB=((pix>> 8)&255)*w;
                    count1+=9;
                    out1[((CHIP*OMAP+oc)*RRANGE+row+PAD)*OM+(col+PAD)] = ((cvalR>>8)<<24) | ((cvalG>>8)<<16) | ((cvalB>>8)<<8);
    }
}
```

```

imax()
{
U11 CHIP; U11 LOOP1, LOOP0; U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 c0, c1, c2, c3, ex0, ex1; U11 x[NCHIP];
ry = (R+ofs)*IM; rx = (R+ofs);
UInt *ym_xm = in -ry-rx;
:

for (row=RRANGE-1; row>=0; row--) {
    int yin0[NCHIP];
    UInt *acci_y0[NCHIP];   UInt *acci_yz0[NCHIP];   UInt *acci_yp0[NCHIP];
    UInt *acco_base0[NCHIP]; UInt *acco0[NCHIP];
    :

    UInt *acco_base5[NCHIP]; UInt *acco5[NCHIP];
    for (CHIP=0; CHIP<NCHIP; CHIP++) {
        int row0 = (CHIP*OMAP+0)*RRANGE+row+PAD;
        int yout0 = row0*ON;
        yin0[CHIP] = ((row>4)*R + (((row&015)*ofs)>>4))*IM;
        acci_y0[CHIP] = in+yin0[CHIP] -ry; acci_yz0[CHIP] = in+yin0[CHIP]; acci_yp0[CHIP] = in+yin0[CHIP] +ry;
        acco_base0[CHIP] = outi+yout0+PAD; acco0[CHIP] = outi+yout0+PAD;
        :
        acco_base5[CHIP] = outi+yout5+PAD; acco5[CHIP] = outi+yout5+PAD;
    }
}

//EMAX5A begin gather mapdlist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (INIT0=1,LOOP0=CRANGE,x[CHIP]=PAD-1; LOOP0--; INIT0=0) {
        exe(OP_ADD, &x[CHIP], x[CHIP], EXP_H3210, 1LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_SUB, &x1, -1LL, EXP_H3210, x[CHIP], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 15LL, OP_NOP, OLL); /* stage#1 */
        exe(OP_MUL, &x2, 1LL, EXP_H3210, x[CHIP], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRL, 4LL); /* stage#2 */
        exe(OP_MULH, &x3, r1, EXP_H3210, (ULL)ofs, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRL, 4LL); /* stage#3 */
        exe(OP_MULH, &x4, r2, EXP_H3210, 75LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
        exe(OP_ADD3, &x5, r3, EXP_H3210, r4, EXP_H3210, (ULL)yin0[CHIP], EXP_H3210, OP_OR, OLL, OP_SLL, 2LL); /* stage#5 */
        exe(OP_ADD, &x1, r3, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL); /* stage#6 */
        mop(OP_LDWR, 1, &BR[4][0][1], r0, (ULL)ym_xm, MSK_D0, (ULL)acci_y0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#4 */
        mop(OP_LDWR, 1, &BR[4][1][1], r0, (ULL)ym_xz, MSK_D0, (ULL)acci_yz0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#4 */
        mop(OP_LDWR, 1, &BR[4][2][1], r0, (ULL)ym_xp, MSK_D0, (ULL)acci_yp0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#4 */
        exe(OP_MULH, &r10, BR[4][0][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
        exe(OP_MULH, &r11, BR[4][1][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
        exe(OP_MULH, &r12, BR[4][2][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
        exe(OP_MULH, &r13, BR[4][0][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
        exe(OP_MULH, &r14, BR[4][1][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
        exe(OP_MULH, &r15, BR[4][2][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
        exe(OP_MULHS, &r20, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][0][1], r0, (ULL)yz_xm, MSK_D0, (ULL)acci_y0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][1][1], r0, (ULL)yz_xz, MSK_D0, (ULL)acci_yz0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#6 */
        mop(OP_LDWR, 1, &BR[6][2][1], r0, (ULL)yz_xp, MSK_D0, (ULL)acci_yp0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#6 */
        exe(OP_MAUHS, &r21, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        exe(OP_MULH, &r10, BR[6][0][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        exe(OP_MULH, &r11, BR[6][1][1], EXP_B5410, 64LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        exe(OP_MULH, &r12, BR[6][2][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
        exe(OP_MULH, &r13, BR[6][0][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        exe(OP_MULH, &r14, BR[6][1][1], EXP_B7632, 64LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        exe(OP_MULH, &r15, BR[6][2][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        exe(OP_MAUHS, &r22, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
        mop(OP_LDWR, 1, &BR[8][0][1], r0, (ULL)yz_xm, MSK_D0, (ULL)acci_y0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#8 */
        mop(OP_LDWR, 1, &BR[8][1][1], r0, (ULL)yz_xz, MSK_D0, (ULL)acci_yz0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#8 */
        mop(OP_LDWR, 1, &BR[8][2][1], r0, (ULL)yz_xp, MSK_D0, (ULL)acci_yp0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#8 */
        exe(OP_MAUHS, &r23, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_MULH, &r10, BR[8][0][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_MULH, &r11, BR[8][1][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_MULH, &r12, BR[8][2][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
        exe(OP_MULH, &r13, BR[8][0][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
        exe(OP_MULH, &r14, BR[8][1][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
        exe(OP_MULH, &r15, BR[8][2][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
        exe(OP_MAUHS, &r24, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
        exe(OP_MAUHS, &r25, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
        exe(OP_MAUHS, &r30, r20, EXP_H3210, r22, EXP_H3210, r24, EXP_H3210, OP_AND, ~1LL, OP_SRLM, 8LL); /* stage#12 */
        exe(OP_MAUHS, &r31, r21, EXP_H3210, r23, EXP_H3210, r25, EXP_H3210, OP_AND, ~1LL, OP_SRLM, 8LL); /* stage#12 */
        mop(OP_MH2B8, &r29, r31, EXP_H3210, r30, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13 */
        mop(OP_STWR, 3, &r29, (ULL)(acco0[CHIP]++), OLL, MSK_D0, (ULL)acco_base0[CHIP], CRANGE, 0, 0, (ULL)NULL, CRANGE); /* stage#13 */

        ****
        exe(OP_ADD, &x0, r1, EXP_H3210, (ULL)yin0[CHIP], EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SLL, 2LL); /* stage#53 */
        ****
        mop(OP_LDWR, 1, &BR[54][0][1], r0, (ULL)ym_xm, MSK_D0, (ULL)acci_y0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#54 */
        mop(OP_LDWR, 1, &BR[54][1][1], r0, (ULL)ym_xz, MSK_D0, (ULL)acci_yz0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#54 */
        mop(OP_LDWR, 1, &BR[54][2][1], r0, (ULL)ym_xp, MSK_D0, (ULL)acci_yp0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#54 */
        exe(OP_MULH, &r10, BR[54][0][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
        exe(OP_MULH, &r11, BR[54][1][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
        exe(OP_MULH, &r12, BR[54][2][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
        exe(OP_MULH, &r13, BR[54][0][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
        exe(OP_MULH, &r14, BR[54][1][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
        exe(OP_MULH, &r15, BR[54][2][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
        exe(OP_MAUHS, &r20, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
        mop(OP_LDWR, 1, &BR[56][0][1], r0, (ULL)yz_xm, MSK_D0, (ULL)acci_y0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#56 */
        mop(OP_LDWR, 1, &BR[56][1][1], r0, (ULL)yz_xz, MSK_D0, (ULL)acci_yz0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#56 */
        mop(OP_LDWR, 1, &BR[56][2][1], r0, (ULL)yz_xp, MSK_D0, (ULL)acci_yp0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#56 */
        exe(OP_MAUHS, &r21, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
        exe(OP_MULH, &r10, BR[56][0][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
        exe(OP_MULH, &r11, BR[56][1][1], EXP_B5410, 64LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
        exe(OP_MULH, &r12, BR[56][2][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#57 */
        exe(OP_MULH, &r13, BR[56][0][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#58 */
        exe(OP_MULH, &r14, BR[56][1][1], EXP_B7632, 64LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#58 */
        exe(OP_MULH, &r15, BR[56][2][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#58 */
        exe(OP_MAUHS, &r22, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#58 */
        mop(OP_LDWR, 1, &BR[58][0][1], r0, (ULL)yp_xm, MSK_D0, (ULL)acci_y0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#58 */
        mop(OP_LDWR, 1, &BR[58][1][1], r0, (ULL)yp_xz, MSK_D0, (ULL)acci_yz0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#58 */
        mop(OP_LDWR, 1, &BR[58][2][1], r0, (ULL)yp_xp, MSK_D0, (ULL)acci_yp0[CHIP], IM, 0, 0, (ULL)NULL, IM); /* stage#58 */
        exe(OP_MAUHS, &r23, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#59 */
        exe(OP_MULH, &r10, BR[58][0][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#59 */
        exe(OP_MULH, &r11, BR[58][1][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#59 */
        exe(OP_MULH, &r12, BR[58][2][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#59 */
        exe(OP_MULH, &r13, BR[58][0][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exe(OP_MULH, &r14, BR[58][1][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exe(OP_MULH, &r15, BR[58][2][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exe(OP_MAUHS, &r24, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#60 */
        exe(OP_MAUHS, &r25, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#61 */
        exe(OP_MAUHS, &r30, r20, EXP_H3210, r22, EXP_H3210, r24, EXP_H3210, OP_AND, ~1LL, OP_SRLM, 8LL); /* stage#62 */
        exe(OP_MAUHS, &r31, r21, EXP_H3210, r23, EXP_H3210, r25, EXP_H3210, OP_AND, ~1LL, OP_SRLM, 8LL); /* stage#62 */
        exe(OP_MH2B8, &r29, r31, EXP_H3210, r30, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#63 */
        mop(OP_STWR, 3, &r29, (ULL)(acco5[CHIP]++), OLL, MSK_D0, (ULL)acco_base5[CHIP], CRANGE, 0, 0, (ULL)NULL, CRANGE); /* stage#63 */
    }

    ****
    exe(OP_ADD, &x0, r1, EXP_H3210, (ULL)yin0[CHIP], EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SLL, 2LL); /* stage#53 */
    ****
}
//EMAX5A end
//EMAX5A drain_dirty_lmm

```

gather+rmm-gather-emax6.obj

BR/row: max=13 min=1 ave=8 EA/row: max=3 min=0 ave=1 ARpass/row: max=0

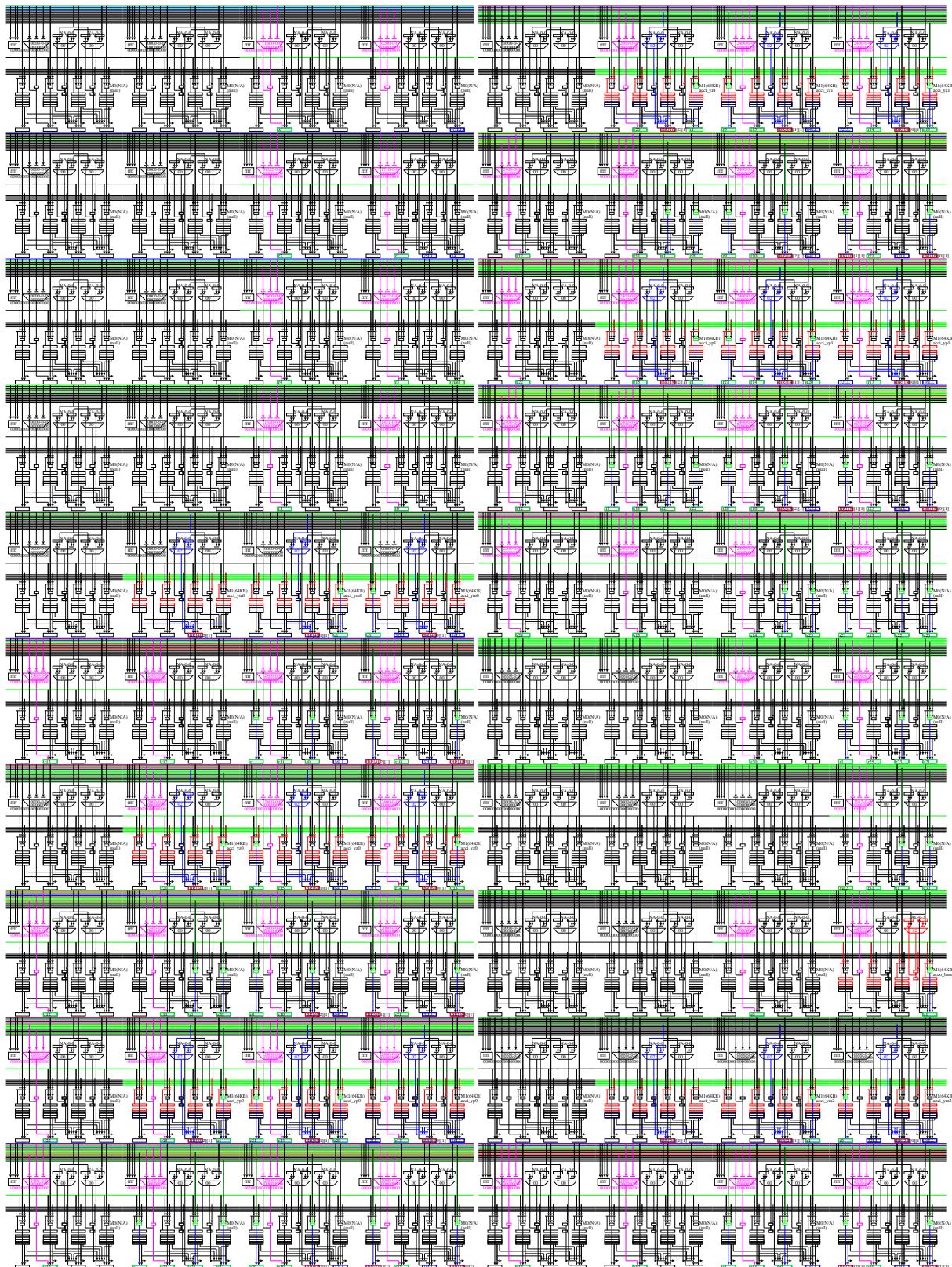


Figure.3.59: Lightfield レンダリング

3.5.7 Lightfield 距離画像生成

```
cent% make -f Makefile-csim.emax6+dma gdepth-csim.emax6+dma clean
cent% ../../src/csim/csim -x gdepth-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma gdepth-zynq.emax6+dma clean
zynq% ./gdepth-zynq.emax6+dma
```

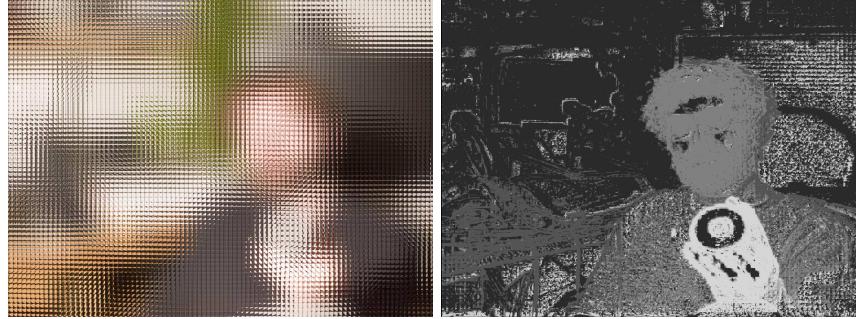
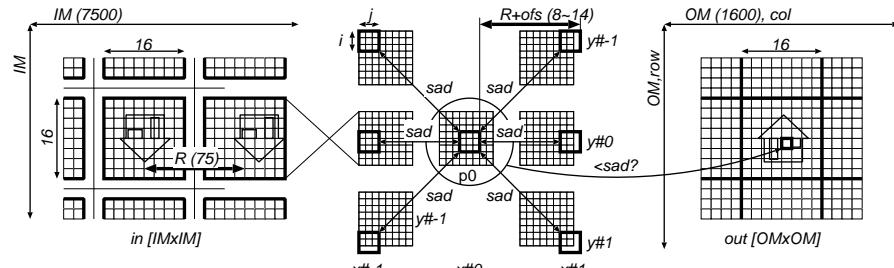
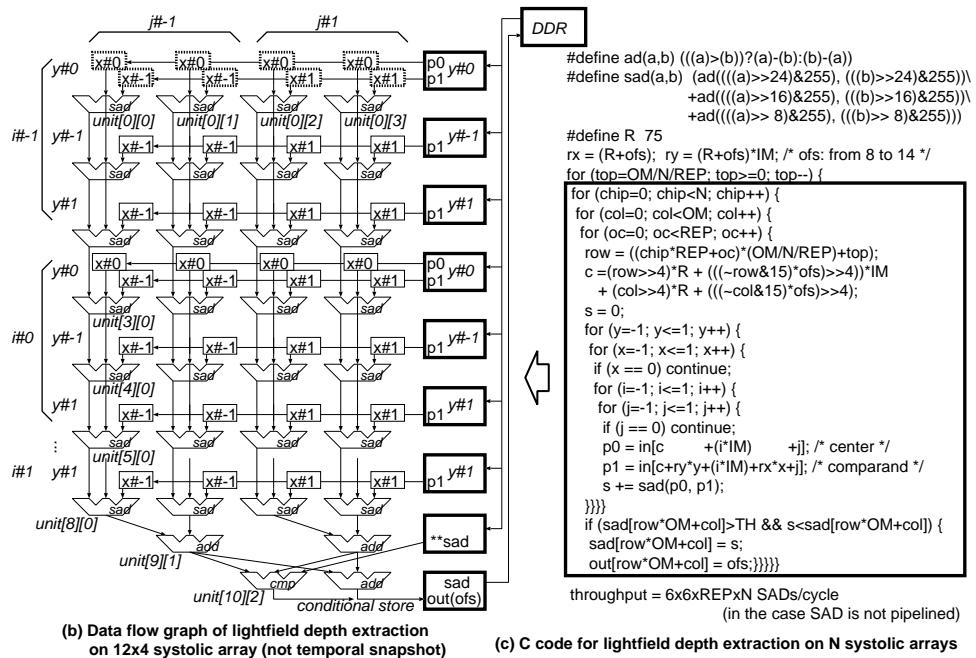


Figure 3.60: Gdepth



(a) Data structure of lightfield depth extraction



(b) Data flow graph of lightfield depth extraction on 12x4 systolic array (not temporal snapshot)

(c) C code for lightfield depth extraction on N systolic arrays

Figure 3.61: Lightfield

解像度 7500x7500 ライトフィールド画像処理の距離画像生成である。規則的なステンシル計算ではないものの適応的 mapdist を使用しており、標準値は mapdist=3 である。図 3.61(a) に、LF のデータ構造、(b)

に各 unit に Sum of absolute difference (SAD) 演算器を 1 つ備える 12x4 構成の CGRA を示す。 (b) は図 3.49 と同様、データフローグラフを示している。 75x75 画素のマイクロレンズ画像が 100x100 の格子点上に配置され、全体で 7500x7500 画素の画像 (in) が得られると仮定する。各マイクロ画像（上下左右が反転）の一部 (3x3) について、75+ofs 画素離れた複数のマイクロ画像間の SAD を求め、最小となる ofs を距離情報 (out) として生成する。具体的には、中央 (p0) の 3x3 画像のうち 6 画素と、周囲 6箇所 (p1) の 3x3 画像のうち 6 画素に対して SAD を求め、合計 36 画素の SAD を累算している。CGRA の先頭段 (unit[0][*]) に、y#0 の位置における 3x3 画素の最上行 (i#-1 に対応) を割り当て、1 行分の画像を保存する先頭 LMM から 6 箇所の離散的ロード (p0 と p1 を含む) を行う (点線矩形部分)。第 2 段 (unit[1][*]) と第 3 段 (unit[2][*]) は、各々、y#-1 および y#1 の位置 (p1) からのロードを行い、先頭段にて取得した p0 と比較する。CGRAs を 9 段通過した後に、さらに 2 段を用いて、4 列の SAD を 1 つの SAD に累算する。最外ループにおいて ofs を変更し、過去の最小 sad を保存しておき、より小さい SAD 値が得られた場合に、条件付きストアを用いて sad と out を更新する。このように、36 画素の SAD 値を求めるために 11 段を使用し、44 個中 40 個の演算器がアクティブとなる。全体の段数は、(b) に示したストア 1 段を含む 12 段に、c のアドレス計算等に必要な 6 段を前置した合計 18 段となる。(c) はさらに、画面を N チップに分割して入力画像を並列処理し、各チップが 36 画素の SAD 演算を REP 組写像できる段数を備える場合の実装である。なお ofs を更新する最外ループは省略している。以上の演算に要する理論的サイクル数は、CGRA 起動時の遅延 (REP ≥ 2 の場合 14xREP+3 サイクル) および LMM の入れ換え時間を除くと、ofsあたり $40 \times OM^2 / (REP \times N)$ となる。out を 1 画素生成するために p0 から 6 画素、p1 から 36 画素必要であるものの、p0 と p1 の水平方向参照は重複と考えると、理論的 DDR-LMM 間転送量は、in: $9 \times OM^2$ + out: OM^2 である（入力画像全体は IM^2 であるが in として参照する量は $9 \times OM^2$ ）。

```

orig()
{
    ry = (R+ofs)*IM;
    rx = (R+ofs); /* ofs: from 8 to 14 */
    for (row=PAD; row<OM-PAD; row++) {
        for (col=PAD; col<OM-PAD; col++) {
            c =((row>>4)*R + (((~row&15)*ofs)>>4))*IM
            + (col>>4)*R + (((~col&15)*ofs)>>4);
            s = 0;
            for (y=-1; y<=1; y++) {
                for (x=-1; x<=1; x++) {
                    if (x == 0) continue;
                    for (i=-1; i<=1; i++) {
                        for (j=-1; j<=1; j++) {
                            if (j == 0) continue;
                            p0 = in[c+(i*IM)      +j]; /* center */
                            p1 = in[c+ry*y+(i*IM)+rx*x+j]; /* comparand */
                            s += dif(p0, p1);
                            if (s > 0xffff) s = 0xffff;
                            count0++;
                        }
                    }
                }
            }
            if (sad0[row*OM+col]>TH && s<sad0[row*OM+col]) {
                sad0[row*OM+col] = s;
                out0[row*OM+col] = ofs;
            }
        }
    }
}

```

```

imax()
{
    U11 CHIP;
    ry = (R+ofs)*IM;
    rx = (R+ofs); /* ofs: from 8 to 14 */
    for (row=0; row<RRANGE; row++) { /* 0 .. 381 */
        for (CHIP=0; CHIP<CHIP; CHIP++) {
            for (col=0; col<CRANGE; col++) {
                for (oc=0; oc<OMAP; oc++) {
                    c =(((CHIP*OMAP+oc)*RRANGE+row+PAD)>>4)*R + (((~((CHIP*OMAP+oc)*RRANGE+row+PAD)&15)*ofs)>>4))*IM
                    + ((          col+PAD)>>4)*R + (((~(          col+PAD)&15)*ofs)>>4);
                    s = 0;
                    for (y=-1; y<=1; y++) {
                        for (x=-1; x<=1; x++) {
                            if (x == 0) continue;
                            for (i=-1; i<=1; i++) {
                                for (j=-1; j<=1; j++) {
                                    if (j == 0) continue;
                                    p0 = in[c+(i*IM)      +j]; /* center */
                                    p1 = in[c+ry*y+(i*IM)+rx*x+j]; /* comparand */
                                    s += dif(p0, p1);
                                    count1++;
                                }
                            }
                        }
                    }
                    if (sad1[((CHIP*OMAP+oc)*RRANGE+row+PAD)*OM+(col+PAD)]>TH && s<sad1[((CHIP*OMAP+oc)*RRANGE+row+PAD)*OM+(col+PAD)]) {
                        sad1[((CHIP*OMAP+oc)*RRANGE+row+PAD)*OM+(col+PAD)] = s;
                        out1[((CHIP*OMAP+oc)*RRANGE+row+PAD)*OM+(col+PAD)] = ofs;
                    }
                }
            }
        }
    }
}

```

```

imax()
{
    U11 CHIP; U11 LOOP1, LOOP0; U11 INIT1, INIT0;
    U11 AR[64][4]; /* output of EX in each unit */
    U11 BR[64][4][4];
    /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    U11 c0, c1, c2, c3, ex0, ex1; U11 x[NCHIP];
    ry = (R+ofs)*IM; rx = (R+ofs);
    Uint *yzm_xm_m4 = in-IM -rx-1; Uint *yzm_xm_p4 = in-IM -rx+1;
    :
    Uint *ypp_xp_m4 = in+IM+ry+rx-1; Uint *ypp_xp_p4 = in+IM+ry+rx+1;
    for (row=RANGE-1; row=0; row--) {
        int yin0[NCHIP];
        Uint *acci_yzm0[NCHIP]; Uint *acci_ym0[NCHIP];
        Uint *acci_yz0[NCHIP]; Uint *acci_yzp0[NCHIP];
        Uint *acci_yzp[NCHIP]; Uint *acci_ym0[NCHIP]; Uint *acci_yp0[NCHIP];
        Uint *sadx_base0[NCHIP]; Uint *sadio[NCHIP]; Uint *sado0[NCHIP];
        Uint *acco_base0[NCHIP]; Uint *acco0[NCHIP];
        :
        Uint *sadi_base0[NCHIP]; Uint *sadio[NCHIP]; Uint *sado_base0[NCHIP]; Uint *sado0[NCHIP]; Uint *acco_base0[NCHIP]; Uint *acco0[NCHIP];
        :
        for (CHIP=0; CHIP<NCHIP; CHIP++) {
            yin0[CHIP] = ((row>0)&1)*R + (((row&15)&ofs)>>4)*IM;
            acci_yzm0[CHIP] = in+yin0[CHIP]-IM; acci_ym0[CHIP] = in+yin0[CHIP]-IM+ry;
            acci_yz0[CHIP] = in+yin0[CHIP]; acci_ymz0[CHIP] = in+yin0[CHIP] -ry; acci_yzp0[CHIP] = in+yin0[CHIP] +ry;
            acci_yzp[CHIP] = in+yin0[CHIP]+IM; acci_ymp0[CHIP] = in+yin0[CHIP]+IM+ry; acci_yp0[CHIP] = in+yin0[CHIP]+IM+ry;
            sadx_base0[CHIP] = sadi+yout0+PAD; sadio[CHIP] = sadi+yout0+PAD; sado0[CHIP] = sadi+yout0+PAD;
           acco_base0[CHIP] = outi+yout0+PAD; acco0[CHIP] = outi+yout0+PAD;
            :
        }
    }
    //EMAX5A begin gdepth mapdis=3
    for (CHIP=0; CHIP<NCHIP; CHIP++) {
        for (INIT0=1,LOOP0=CRANGE,x[CHIP]=PAD-1; LOOP0--; INIT0=0) {
            exe(OP_ADD, &r1, -1LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_SUB, &r2, -1LL, EXP_H3210, x[CHIP], EXP_H3210, OLL, EXP_H3210, OP_AND, 15LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_NOP, &r2, x[CHIP], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRL, 4LL); /* stage#1 */
            exe(OP_MULH, &r3, r1, EXP_H3210, (ULL)ofs, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRL, 4LL); /* stage#2 */
            exe(OP_MULH, &r2, r2, EXP_H3210, 75LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
            exe(OP_ADDS, &r0, r3, EXP_H3210, r4, EXP_H3210, (ULL)yin0[CHIP], EXP_H3210, OP_OR, OLL, OP_SLL, 2LL);/* stage#3 */
            exe(OP_ADD, &r1, r4, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL); /* stage#3 */
            mop(OP_LDWR, 1, &BR[4][0][1], r0, (U11)*yzm_xm_m4, MSK_D0, (U11)*acci_yzm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
            mop(OP_LDWR, 1, &BR[4][0][0], r0, (U11)*yzm_xz_m4, MSK_D0, (U11)*acci_yzm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
            mop(OP_LDWR, 1, &BR[4][1][1], r0, (U11)*yzm_xz_m4, MSK_D0, (U11)*acci_yzm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
            mop(OP_LDWR, 1, &BR[4][1][0], r0, (U11)*yzm_xp_m4, MSK_D0, (U11)*acci_yzm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
            mop(OP_LDWR, 1, &BR[4][2][0], r0, (U11)*yzm_xp_m4, MSK_D0, (U11)*acci_yzm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
            exe(OP_MSSAD, &r14, OLL, EXP_H3210, BR[4][0][0], EXP_H3210, OLL, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
            exe(OP_MSSAD, &r15, OLL, EXP_H3210, BR[4][0][1], EXP_H3210, BR[4][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
            exe(OP_MSSAD, &r16, OLL, EXP_H3210, BR[4][1][0], EXP_H3210, BR[4][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
            exe(OP_MSSAD, &r17, OLL, EXP_H3210, BR[4][2][1], EXP_H3210, BR[4][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
            mop(OP_LDWR, 1, &BR[5][0][1], r0, (U11)*ymm_xm_m4, MSK_D0, (U11)*acci_ymm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#5 */
            mop(OP_LDWR, 1, &BR[5][0][0], r0, (U11)*ymm_xp_m4, MSK_D0, (U11)*acci_ymm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#5 */
            mop(OP_LDWR, 1, &BR[5][2][1], r0, (U11)*ymm_xp_m4, MSK_D0, (U11)*acci_ymm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#5 */
            mop(OP_LDWR, 1, &BR[5][2][0], r0, (U11)*ymm_xp_m4, MSK_D0, (U11)*acci_ymm0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#5 */
            :
            exe(OP_MAUH, &r24, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#14 */
            exe(OP_MAUH, &r26, r16, EXP_H3210, r17, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#14 */
            exe(OP_MAUH, &r30, r24, EXP_H3210, r26, EXP_H3210, OLL, EXP_H3210, OP_SUMHL, OLL, OP_NOP, OLL); /* stage#15 */
            mop(OP_LDWR, 1, &BR[15][1][1], (U11)*(sadx_base0[CHIP]+), OLL, MSK_D0, (U11)*sadx_base0[CHIP], CRANGE, 0, 1, (U11)NULL, CRANGE);/* stage#15 */
            exe(OP_CMP_LT, &c0, r30, EXP_H3210, BR[15][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#16 */
            exe(OP_CMP_GT, &c1, BR[15][1][1], EXP_H3210, 137LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#16 */
            exe(OP_NOP, &r31, OLL, EXP_H3210, OLL, EXP_H3210, OP_OR, (ULL)ofs, OP_NOP, OLL); /* stage#16 */
            cex(OP_CEXE, &ex1, 0, 0, c1, c0, 0x8888); /* stage#17 */
            mop(OP_STWR, ex1, r31, (U11)*(acco0[CHIP]+), OLL, MSK_D0, (U11)*acco_base0[CHIP], CRANGE, 0, 1, (U11)NULL, CRANGE); /* stage#17 */
            /*****
            exe(OP_ADD, &r0, r1, EXP_H3210, (U11)yin1[CHIP], EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SLL, 2LL); /* stage#17 */
            *****/
            :
            exe(OP_MSSAD, &r24, r14, EXP_H3210, BR[51][0][0], EXP_H3210, BR[49][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#52 */
            exe(OP_MSSAD, &r25, r15, EXP_H3210, BR[51][0][1], EXP_H3210, BR[49][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#52 */
            exe(OP_MSSAD, &r26, r16, EXP_H3210, BR[51][2][0], EXP_H3210, BR[49][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#52 */
            exe(OP_MSSAD, &r27, r17, EXP_H3210, BR[51][2][1], EXP_H3210, BR[49][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#52 */
            mop(OP_LDWR, 1, &BR[52][0][1], r0, (U11)*yzp_xm_m4, MSK_D0, (U11)*acci_yzp3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#52 */
            mop(OP_LDWR, 1, &BR[52][0][0], r0, (U11)*yzp_xm_p4, MSK_D0, (U11)*acci_yzp3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#52 */
            mop(OP_LDWR, 1, &BR[52][1][1], r0, (U11)*yzp_xz_m4, MSK_D0, (U11)*acci_yzp3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#52 */
            mop(OP_LDWR, 1, &BR[52][1][0], r0, (U11)*yzp_xz_p4, MSK_D0, (U11)*acci_yzp3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#52 */
            mop(OP_LDWR, 1, &BR[52][2][1], r0, (U11)*yzp_xp_m4, MSK_D0, (U11)*acci_yzp3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#52 */
            mop(OP_LDWR, 1, &BR[52][2][0], r0, (U11)*yzp_xp_p4, MSK_D0, (U11)*acci_yzp3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#52 */
            exe(OP_MSSAD, &r14, r24, EXP_H3210, BR[52][0][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            exe(OP_MSSAD, &r15, r25, EXP_H3210, BR[52][0][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            exe(OP_MSSAD, &r16, r26, EXP_H3210, BR[52][2][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            exe(OP_MSSAD, &r17, r27, EXP_H3210, r20, EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][0][1], r0, (U11)*ymm_xm_m4, MSK_D0, (U11)*acci_ymm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][0][0], r0, (U11)*ymm_xp_m4, MSK_D0, (U11)*acci_ymm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][2][1], r0, (U11)*ymm_xp_m4, MSK_D0, (U11)*acci_ymm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#53 */
            mop(OP_LDWR, 1, &BR[53][2][0], r0, (U11)*ymm_xp_p4, MSK_D0, (U11)*acci_ymm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#53 */
            exe(OP_MSSAD, &r24, r14, EXP_H3210, BR[53][0][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            exe(OP_MSSAD, &r25, r15, EXP_H3210, BR[53][0][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            exe(OP_MSSAD, &r26, r16, EXP_H3210, BR[53][2][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            exe(OP_MSSAD, &r27, r17, EXP_H3210, BR[53][2][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][0][1], r0, (U11)*ypm_xm_m4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][0][0], r0, (U11)*ypm_xp_m4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][2][1], r0, (U11)*ypm_xp_m4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#54 */
            mop(OP_LDWR, 1, &BR[54][2][0], r0, (U11)*ypm_xp_p4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#54 */
            exe(OP_MSSAD, &r14, r24, EXP_H3210, BR[54][0][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            exe(OP_MSSAD, &r15, r25, EXP_H3210, BR[54][0][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            exe(OP_MSSAD, &r16, r26, EXP_H3210, BR[54][2][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            exe(OP_MSSAD, &r17, r27, EXP_H3210, BR[54][2][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][0][1], r0, (U11)*ypm_xm_p4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][0][0], r0, (U11)*ypm_xp_m4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][2][1], r0, (U11)*ypm_xp_m4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#55 */
            mop(OP_LDWR, 1, &BR[55][2][0], r0, (U11)*ypm_xp_p4, MSK_D0, (U11)*acci_ypm3[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#55 */
            exe(OP_MSSAD, &r14, r24, EXP_H3210, BR[55][0][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            exe(OP_MSSAD, &r15, r25, EXP_H3210, BR[55][0][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            exe(OP_MSSAD, &r16, r26, EXP_H3210, BR[55][2][0], EXP_H3210, BR[52][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            exe(OP_MSSAD, &r17, r27, EXP_H3210, BR[55][2][1], EXP_H3210, BR[52][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#56 */
            mop(OP_LDWR, 1, &BR[56][1][1], (U11)*(sadi3[CHIP]+), OLL, MSK_D0, (U11)*sadx_base3[CHIP], CRANGE, 0, 1, (U11)NULL, CRANGE);/* stage#57 */
            exe(OP_CMP_LT, &c0, r30, EXP_H3210, BR[57][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#58 */
            exe(OP_CMP_GT, &c1, BR[57][1][1], EXP_H3210, 137LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#58 */
            cex(OP_CEXE, &ex1, 0, 0, c1, c0, 0x8888); /* stage#59 */
            mop(OP_STWR, ex1, r31, (U11)*(acco3[CHIP]+), OLL, MSK_D0, (U11)*acco_base3[CHIP], CRANGE, 0, 1, (U11)NULL, CRANGE); /* stage#59 */
            exe(OP_STWR, ex0, r30, (U11)*(sado3[CHIP]+), OLL, MSK_D0, (U11)*sadx_base3[CHIP], CRANGE, 0, 1, (U11)NULL, CRANGE); /* stage#59 */
        }
    }
    //EMAX5A end
}
//EMAX5A drain_dirty_lmm
}

```

gdepth+rmm-gdepth-emax6.obj

BR/row: max=12 min=1 ave=8 EA/row: max=6 min=0 ave=3 ARpass/row: max=0

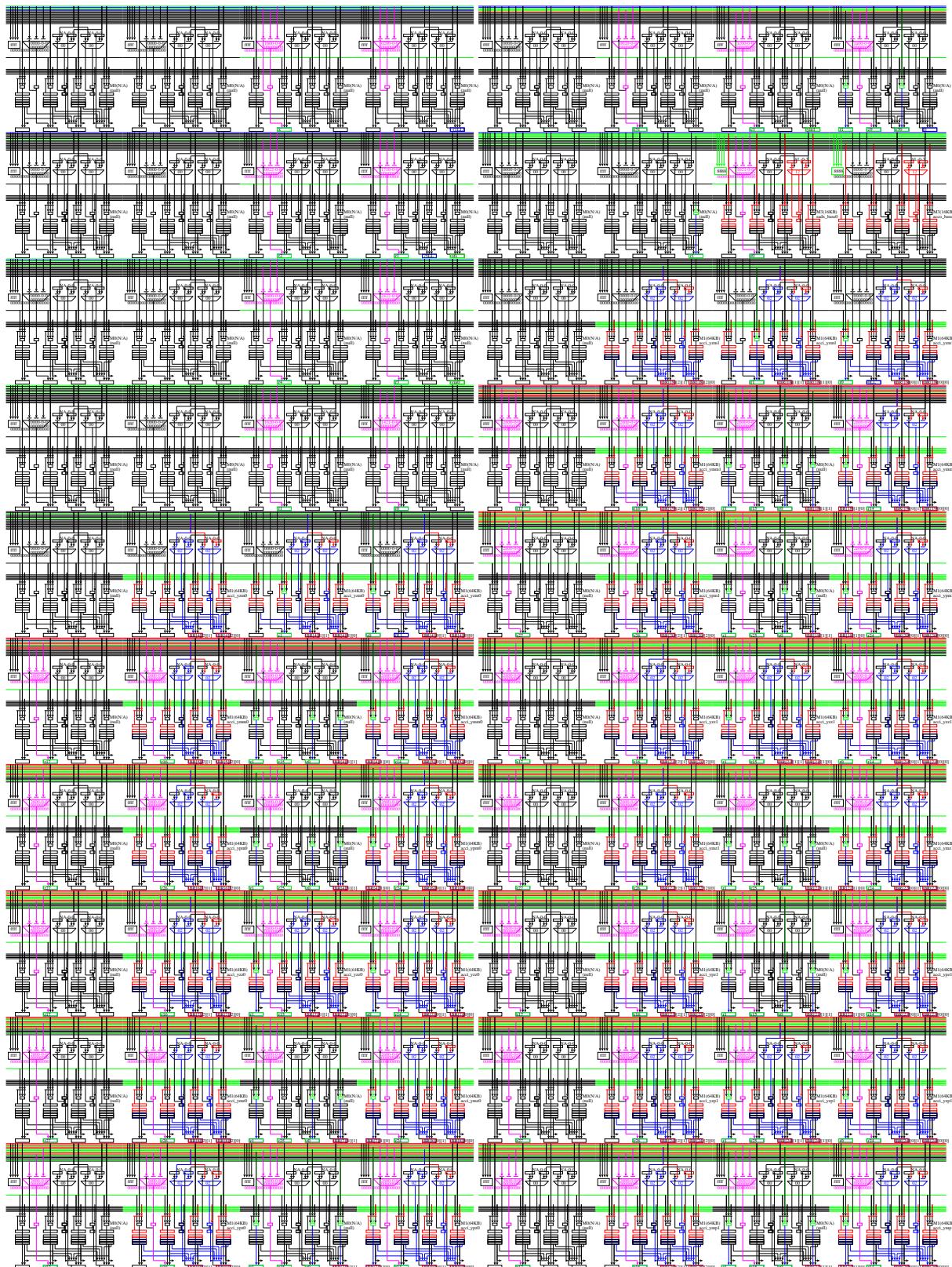


Figure 3.62: Lightfield 距離画像生成

3.6 Graph processing on EMAX5

```
cent% make -f Makefile-bsim.emax5 all clean  
cent% ../../src/bsim/bsim -x tricount-bsim.emax5 ../../graph-data/test.edges
```

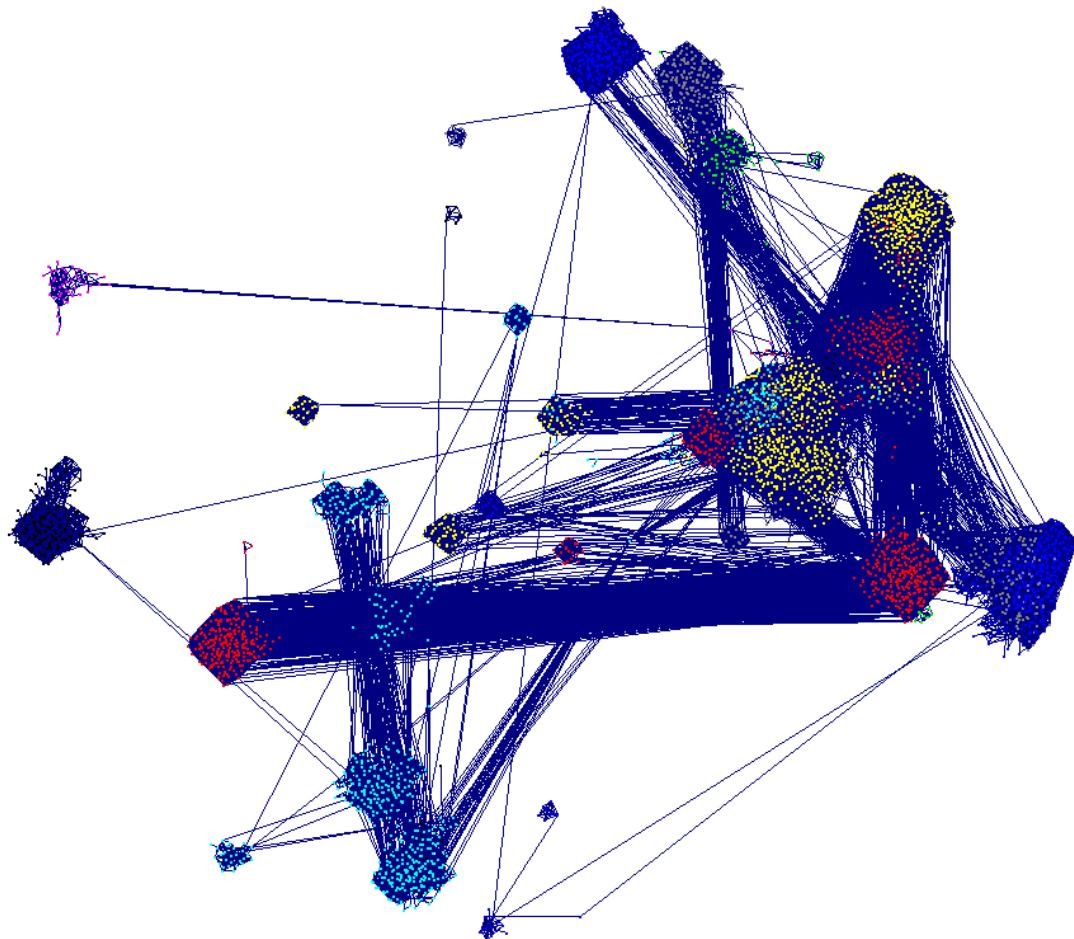


Figure.3.63: Graph

3.6.1 Triangle counting kernel0 with TCU

グラフ構造に存在する三角形を数える処理 (BFS) である。トランザクションを使用している。

```

void tri_kernel0(struct param_bfs *param)
{
    volatile int i, j, pid, qid, MVL, MEL;
    volatile struct vertex *p, *np, *q;
    volatile struct neighborvertex *n;

    i = param->i;
    p = param->p;
    np = param->nextp;
    MVL = param->maxvlist;
    MEL = param->maxelist;
    pid = p->id;

    #if !defined(EMAX5) && !defined(EMAX6)
    for (j=0; j<p->nedges; j++) {
        /* R 0段:最内ループ 256 回転程度 */
        n = p->page[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
        q = n->vp;
        qid = n->id;
        if ((q->parent) &lt;> 0) {
            /* R 0段:neighborvertex 全体を配置 pointer を使い参照 */
            /* R 0段:同上 */
            /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
            /* **** */
            while (cmplxchg(&Sem0, -1, param->th) != -1);
            /* **** */
            if (!q->parent) {
                /* R 1段:同上 */
                if (nnextfrontiers >= MVL) {
                    printf("vlist[%d] exhausted\n", MVL);
                    exit(1);
                }
                q->parent = pid;
                /* W 2段:verex 更新 */
                /* W 2段:同上 */
                q->findex = nnextfrontiers;
                /* W 2段:同上 */
                nextfrontier[nnextfrontiers] = q;
                /* W 2段:next_frontier[] 更新 */
                nnextfrontiers++;
                /* W 2段:同上 */
                nnextfrontiers->neighbors+=q->nedges;
            }
            /* **** */
            /*cmplxchg(&Sem0, param->th, -1); */
            release(&Sem0, -1);
            /* **** */
        }
    }
    else if (q->depth==depth-1 && q->findex<i) { /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
        /* **** */
        while (cmplxchg(&Sem1, -1, param->th) != -1);
        /* **** */
        if (nfrontier_edges >= MEL) {
            printf("elist[%d] exhausted\n", MEL);
            exit(1);
        }
        frontier_edge[nfrontier_edges].src = (pid<qid)?p:q; /* W 2段:frontier_edge[] 更新 */
        frontier_edge[nfrontier_edges].dst = (pid<qid)?q:p; /* W 2段:同上 */
        nfrontier_edges++;
        /* W 2段:同上 */
        nfrontier_edges->neighbors+=((pid<qid)?p:q)->nedges;
        /* **** */
        /*cmplxchg(&Sem1, param->th, -1); */
        release(&Sem1, -1);
        /* **** */
    }
}
#endif
}

#else
ULL AR[64][4]; /* output registers in each unit */
ULL BR[64][4][4]; /* output registers in each unit */
struct neighborvertex *r0 =NULL; /* n */
struct neighborvertex **r0_top =p->page;
Uint r0_len =p->nedges*4;
Uint pnedges=p->nedges;
struct neighborvertex **r0_ntop= np->page:NULL;
Uint r0_nlen=np->nedges*4:NULL;
ULL r2[4], r3[4], r4[4], r6, r7, r8;
ULL depth_1=depth-1;
ULL c0, c1, c2, c3, ex0, ex1;
int loopp->nedges;
void tri_kernel0_trans0();
void tri_kernel0_trans1();
//EMAX5A begin tri_kernel0 mapdist=1
while (loopp-) {
/*0,*/ mo4(OP_LDRQ, 1, BR[0][0], (ULL)r0++, OLL, MSK_D0, (ULL)r0_top, r0_len, 2, 0, (ULL)r0_ntop, r0_nlen);
/*1,*/ mo4(OP_CMP_LT, 1, BR[1][1], BR[0][0][3], 32LL, MSK_D0, (ULL)NULL, 0, 0, (ULL)NULL, 0);
/*1,*/ exe(OP_CMP_LT, &c0, pid, EXP_H3210, BR[0][0][2], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*2,*/ exe(OP_CMov, &r6, c0, EXP_H3210, p, EXP_H3210, BR[0][0][3], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*2,*/ exe(OP_CMov, &r7, c0, EXP_H3210, BR[0][0][3], EXP_H3210, p, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*2,*/ exe(OP_CMov, &r8, c0, EXP_H3210, pnedges, EXP_H3210, BR[1][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
/*2,*/ exe(OP_CMP_EQ, &c0, BR[1][1][1], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* q->parent==0? */
/*3,*/ exe(OP_ADD, &AR[3][0], BR[0][0][3], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* q->nedges */
/*3,*/ exe(OP_ADD, &AR[3][2], pid, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OLL); /* pid */
/*3,*/ cex(OP_CExE, &ex0, 0, 0, c0, 0xaaaa);
/*3,*/ mo4(OP_TR, ex0, AR[3], (ULL)NULL, OLL, OLL, (ULL)tri_kernel0_trans0, 0, 0, 0, (ULL)NULL, 0);
/*4,*/ exe(OP_CMP_NE, &c0, BR[1][1][1], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, /* q->parent!=0 */
/*4,*/ exe(OP_CMP_EQ, &c1, BR[1][1][2], EXP_H3210, depth_1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, /* q->depth==depth-1 */
/*4,*/ exe(OP_CMP_LT, &c2, BR[1][1][3], EXP_H3210, i, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* q->findex<i */
/*5,*/ exe(OP_ADD, &AR[5][0], r6, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* src */
/*5,*/ exe(OP_ADD, &AR[5][1], r7, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* dst */
/*5,*/ exe(OP_ADD, &AR[5][2], r8, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* nen */
/*5,*/ exe(OP_ADD, &AR[5][3], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* dummy */
/*5,*/ mo4(OP_TR, ex1, AR[5], (ULL)NULL, OLL, OLL, (ULL)tri_kernel0_trans1, 0, 0, 0, (ULL)NULL, 0);
}
//EMAX5A end
#endif
}

```

tricount-tri_kernel0-emax6.obj

BR/row: max=13 min=2 ave=6 EA/row: max=4 min=0 ave=2 ARpass/row: max=0

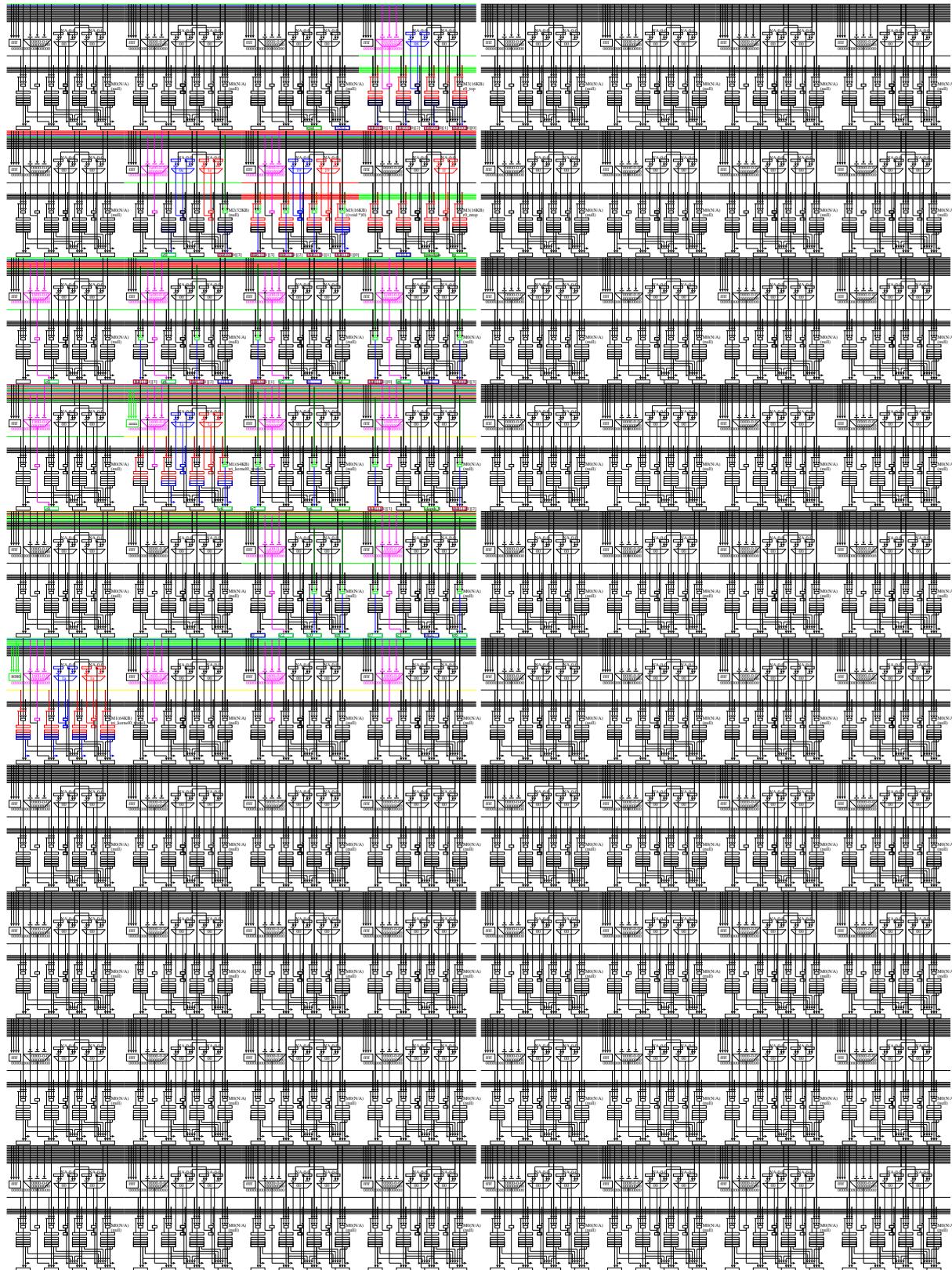


Figure.3.64: Triangle counting kernel0 with TCU

3.6.2 Triangle counting kernel1 with TCU

グラフ構造に存在する三角形を数える処理（三角形カウント）である。トランザクションを使用している。

```

void tri_kernel1(struct param_tricount *param)
{
    /* search triangle in {frontier,next} */
    /* case 1: e ∈ frontier, v ∈ prev */
    /* case 2: e ∈ frontier, v ∈ frontier */
    /* case 3: e ∈ frontier, v ∈ next */
    int i, j, pid, qid, sdepth, tdepth;
    struct vertex *p, *np, *q, *t;
    struct neighborvertex *n;

    p = param->p;
    np = param->nextp;
    t = param->t;
    pid = p->id;
    sdepth = p->depth;

    #if !defined(EMAX5) && !defined(EMAX6)
    int tricount = 0;
    for (j=0; j<p->nedges; j++) {           /* R 0段:最内ループ 256 回転程度 */
        n = p->page[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
        q = n->vp;
        qid = n->id;
        tdepth = q->depth;
        if ((tdepth==sdepth-1||(tdepth==sdepth+1))||(tdepth==sdepth && qid<pid)) { /* R 2段:比較 */
            if (search_nvertex(t->nhashtable, qid))          /* R 3段:HASH-SEARCH/CAM-SEARCH */
                tricount++;
        }
    }
    param->tricount += tricount;
    #else
    U11 BR[16][4][4]; /* output registers in each unit */
    struct neighborvertex *r0      =NULL; /* n */
    struct neighborvertex **r0_top =p->npage;
    Uint          r0_len =p->nedges*4;
    Uint          pnedges=p->nedges;
    struct neighborvertex **r0_ntop=np?p->page:NULL;
    Uint          r0_nlen=np?p->page:NULL;
    U11          r2[4], r3[4], r6, r7, r8;
    U11          sdepth_m1=sdepth-1;
    U11          tnhashtbl=t->nhashtable;
    U11          sdepth_p1=sdepth+1;
    U11          c0, c1, c2, c3, ex0, ex1;
    int loop=p->nedges;
    void tri_kernel1_trans0();
    //EMAX5 begin tri_kernel1 mapdist1
    while (loop--) {
        /*0,*/ mod(OP_LDRQ, 1, BR[0][0], (U11)(r0++), OLL, MSK_D0, (U11)r0_top, r0_len, 2, 0, (U11)r0_ntop, r0_nlen);
        /*1,*/ mo4(OP_LDDMQ, 1, BR[1][1], BR[0][0][3], 32LL, MSK_D0, (U11)NULL, 0, 0, 0, (U11)NULL, 0);
        /*2,*/ exe(OP_CMP_EQ, &c0, BR[1][1][2], EXP_H3210, sdepth_m1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* q->depth==p->depth-1 */
        /*2,*/ exe(OP_CMP_EQ, &c1, BR[1][1][2], EXP_H3210, sdepth_p1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* q->depth==p->depth+1 */
        /*2,*/ exe(OP_CMP_EQ, &c2, BR[1][1][2], EXP_H3210, sdepth, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* q->depth==p->depth */
        /*2,3*/ exe(OP_CMP_LT, &c3, BR[0][0][2], EXP_H3210, pid, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* qid<pid */
        /*3,*/ exe(OP_ADD,  &AR[3][0], tnhashtbl, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* t->nhashtable */
        /*3,1*/ exe(OP_ADD,  &AR[3][1], BR[0][0][2], EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* qid */
        /*3,2*/ exe(OP_ADD,  &AR[3][2], OLL,           EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* dummy */
        /*3,3*/ exe(OP_ADD,  &AR[3][3], OLL,           EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* dummy */
        /*3,3*/ mo4(OP_TR,   ex0,      c3, c2, c1, c0, 0xfee); OLL, (U11)tri_kernel1_trans0, 0, 0, 0, (U11)NULL, 0); /* r3[1]<-(nhashtable) r3[0]<-(qid) */
    }
    //EMAX5 end
    #endif
}

```

tricount-tri_kernel1-emax6.obj

BR/row: max=9 min=2 ave=4 EA/row: max=4 min=0 ave=2 ARpass/row: max=0

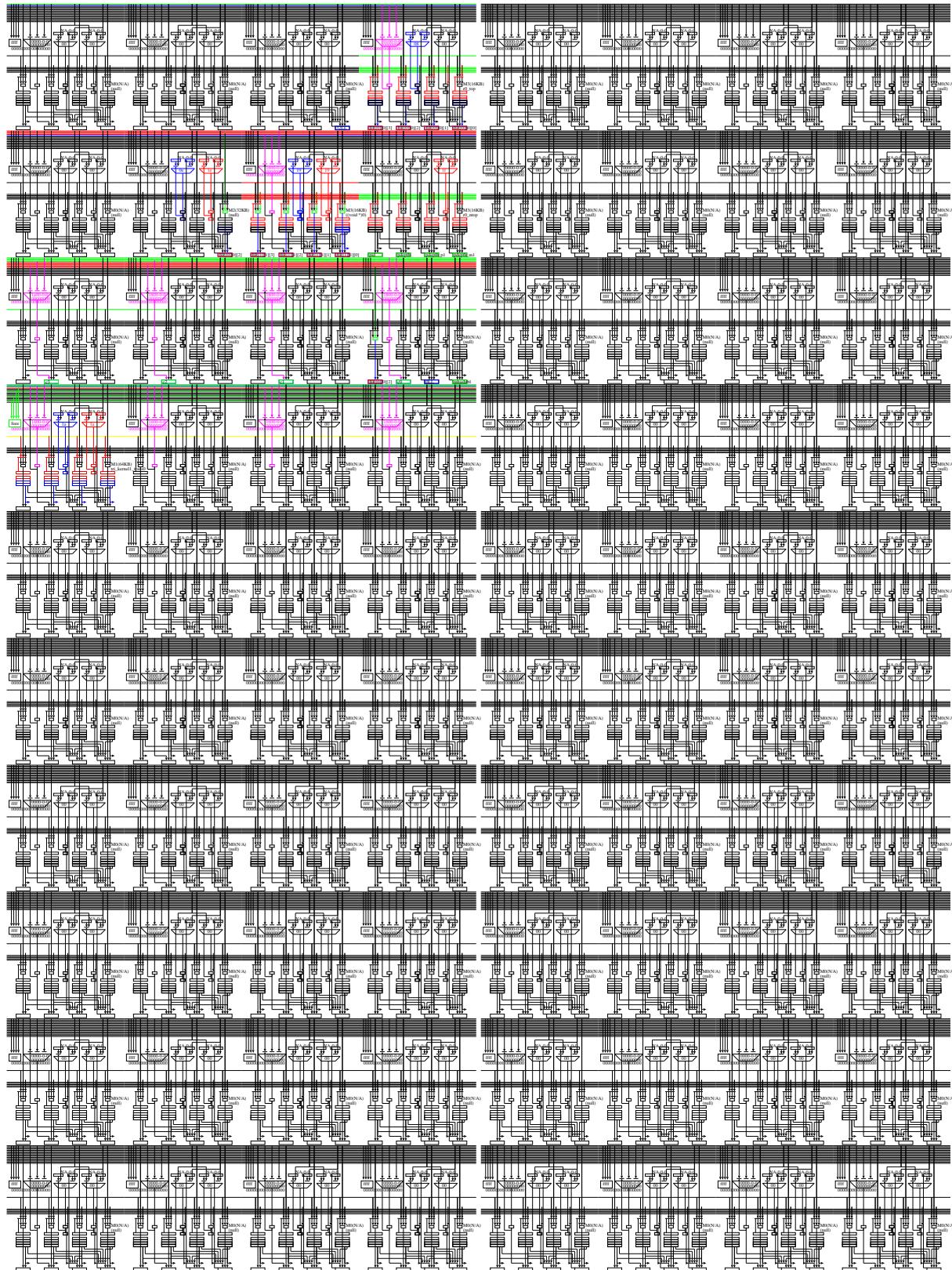


Figure.3.65: Triangle counting kernel1 with TCU

3.7 Graph processing on IMAX2

```
cent% make -f Makefile.emax6nc all clean
cent% tricount.emax6nc ../graph-data/facebook.edges
```

3.7.1 Triangle counting kernel0

```
#if !defined(EMAX6)
int i, j, pid, qid;
for (i=0; i<curfront_v->n_v; i+=4) {
    pid = curfront_v->pid[i/4]; /* sequential [LMM#0] */
    for (j=0; j<vinfo[pid].n_v; j++) { /* top+1D-sequential [LMM#1] */
        qid = vpack[pid].qid[j]; /* top+1D-sequential [LMM#2] */
        if (!vinfo[qid].parent) { /* 1D-random [LMM#1] */
            if (nxtfront_v->n_v >= MAXFRONT_V*4) {
                printf("nxtfront_v exhausted (%d)\n", MAXFRONT_V);
                exit(1);
            }
            vinfo[qid].parent = 1/pid; /* 1D-random update [LMM#1] */
            vinfo[qid].depth = depth; /* 1D-random update [LMM#1] */
            nxtfront_v->pid[nxtfront_v->n_v/4] = qid; /* sequential update [LMM#3] */
            nxtfront_v->n_v+=4; /* sequential update [LMM#3] */
        }
        else if (vinfo[qid].depth==depth-1 && pid==qid) { /* 1D-random */
            if (curfront_e->n_e >= MAXFRONT_E*4) {
                printf("curfront_e exhausted (%d)\n", MAXFRONT_E);
                exit(1);
            }
            curfront_e->e[curfront_e->n_e/4].src = pid; /* sequential update [LMM#4] */
            curfront_e->e[curfront_e->n_e/4].dst = qid; /* sequential update [LMM#4] */
            curfront_e->n_e+=4; /* sequential update [LMM#5+] */
        }
    }
}

#else
U11 CHIP;
U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
U11 cand0, cand1;
int i, j, pid, len;
U11 qofs, qid, qid_pid, qidofs, qinfo1, qinfo2, qinfo_depth, parent_depth, depth_m1;
U11 nxtfront_v_p4 = (U11)nxtfront_v + 4;
U11 curfront_e_p4 = (U11)curfront_e + 4;
U11 nxtfront_v_n, nxtfront_ofs;
U11 curfront_e_n, curfront_ofs;

parent_depth = 0x80000000 | (depth<<16);
depth_m1 = depth-1;
for (i=0; i<curfront_v->n_v; i+=4) {
    pid = curfront_v->pid[i/4]; /* sequential [LMM#0] */
    len = vinfo[pid].n_v; /* #of words */
//EMAX5A begin tri_kernel0 maplist=0
    for (INIT0=1,LOOP0=len,qofs=0-4); LOOP0--; INIT0=0) {
        exe(OP_ADD, &qofs, qofs, EXP_H3210, 4, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL);
        mop(OP_LDWR, 1, &qid, vpack[pid].qid, qofs, MSK_W0, vpack[pid].qid, len, 0, 0, NULL, 0); /* qid */
        exe(OP_ADD, &qidofs, qid, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_SLL, 2LL);
        mop(OP_LDWR, 1, &qinfo1, vinfo, qidofs, MSK_W0, vinfo, nvertices, 0, 0, NULL, 0); /* qinfo */
        mop(OP_LDWR, 1, &qinfo2, vinfo, qidofs, MSK_W0, vinfo, nvertices, 0, 0, NULL, 0); /* qinfo */
        exe(OP_CMP_LT, &cc0, qinfo1, EXP_H3210, 0x80000000OLL, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_NOP, &nxtfront_ofs, cc0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 1LL, OP_SLL, 2LL);

        exe(OP_NOP, &qinfo1, parent_depth, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, qinfo1, OP_NOP, OLL);
        cex(OP_CEXE, &ex0, 0, 0, 0, cc0, 0xaaaa);
        mop(OP_STWR, ex0, &qinfo1, vinfo, qidofs, MSK_W0, vinfo, nvertices, 0, 0, NULL, 0); /* vinfo[qid].parent=1 */

        mop(OP_LDWR, 1, &nxtfront_v_n, nxtfront_v, 0, MSK_W0, nxtfront_v, 1, 0, 1, NULL, 0); /* nxtfront_v->n_v */
        cex(OP_CEXE, &ex0, 0, 0, 0, cc0, 0xaaaa);
        mop(OP_STWR, ex0, &qid, nxtfront_v_p4, nxtfront_v_n, MSK_W0, nxtfront_v, 1, 0, 1, NULL, 0); /* nxtfront_v->n_v */
        mop(OP_LDWR, 1, &nxtfront_v_n, nxtfront_v, 0, MSK_W0, nxtfront_v, 1, 0, 1, NULL, 0); /* nxtfront_v->n_v */
        exe(OP_ADD, &nxtfront_ofs, nxtfront_v_n, EXP_H3210, nxtfront_ofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL);
        mop(OP_STWR, 1, &nxtfront_v_n, nxtfront_v, 0, MSK_W0, nxtfront_v, 1, 0, 1, NULL, 0); /* nxtfront_v->n_v */

        exe(OP_NOP, &qinfo2, qinfo2, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 0x000000007fffffLL, OP_SRL, 16LL);
        exe(OP_CMP_GE, &cc2, qinfo2, EXP_H3210, 0x80000000OLL, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_CMP_EQ, &ccl, qinfo_depth, EXP_H3210, depth_m1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_CMP_LT, &cc0, pid, EXP_H3210, qid, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
        exe(OP_NOP, &cc0, cc2, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, cc1, OP_NOP, OLL);
        exe(OP_NOP, &cand0, cand0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, cc0, OP_NOP, OLL);
        exe(OP_NOP, &curfront_ofs, cand1, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 1LL, OP_SLL, 2LL);

        exe(OP_ADD, &qid_pid, qid, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, pid, OP_OR, pid, OP_NOP, OLL);
        mop(OP_LDWR, 1, &curfront_e_n, curfront_e, 0, MSK_W0, curfront_e, 1, 0, 1, NULL, 0); /* curfront_e->n_e */
        cex(OP_CEXE, &ex1, 0, cc2, ccl, cc0, 0x8080);
        mop(OP_STWR, ex1, &qid_pid, curfront_e_p4, curfront_e_n, MSK_W0, curfront_e, 1, 0, 1, NULL, 0); /* curfront_e->n_e */
        mop(OP_LDWR, 1, &curfront_e_n, curfront_e, 0, MSK_W0, curfront_e, 1, 0, 1, NULL, 0); /* curfront_e->n_e */
        exe(OP_ADD, &curfront_e_n, curfront_e_n, EXP_H3210, curfront_ofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, OLL);
        mop(OP_STWR, 1, &curfront_e_n, curfront_e, 0, MSK_W0, curfront_e, 1, 0, 1, NULL, 0); /* curfront_e->n_e */
    }
//EMAX5A end
}
#endif
}
```

3.7.2 Triangle counting kernel1

```

#if !defined(EMAX6)
int i, j, src, dst, qid, sdepth, qdepth;

for (i=0; i<curfront_e->n_e; i+=4) {
    src = curfront_e->e[i/4].src; /* sequential [LMM#0] */
    dst = curfront_e->e[i/4].dst; /* sequential [LMM#0] */
    sdepth = vinfo[src].depth; /* vinfo [LMM#1] */
    for (j=0; j<vinfo[src].n_v; j++) { /* vinfo [LMM#1] */
        qid = vpack[src].qid[j]; /* top+ID-sequential [LMM#2] */
        qdepth = vinfo[qid].depth; /* ID-random [LMM#1] */
        if ((sdepth+1==qdepth)|| (sdepth+1==qdepth) || (sdepth==qdepth && dst<qid)) { /* src < dst < qid */
            if (search_qid_in_dst(qid, dst)) /* search */
                (*tricount)++; /* update */
        }
    }
}

#else
ULL CHIP;
ULL LOOP1, LOOPTO;
ULL INIT1, INIT0;
ULL AR[64][4]; /* output of EX in each unit */
ULL BR[64][4][4]; /* output registers in each unit */
ULL r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
ULL r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
ULL cc0, ccl, cc2, cc3, ex0, ex1;
ULL cand0, cand1, cand2;
int i, j, src, dst, len;
ULL sofs, qid, qidofs, qinfo, qinfo_depth, sdepth, sdepth_m1, sdepth_p1, vsearchqid, vsearchtop, search;
ULL tricount_ofs, tricount_r;

for (i=0; i<curfront_e->n_e; i+=4) {
    src = curfront_e->e[i/4].src; /* sequential [LMM#0] */
    dst = curfront_e->e[i/4].dst; /* sequential [LMM#0] */
    sdepth = vinfo[src].depth; /* vinfo [LMM#1] */
    sdepth_m1 = sdepth+1; /* vinfo [LMM#1] */
    sdepth_p1 = sdepth+1; /* vinfo [LMM#1] */
    len = vinfo[src].n_v; /* #of words */
//EMAX5A begin tri_kernell mapdist=0
    for (INIT0=1,LOOP0=len,sofs=0-4); LOOP0--; INIT0=0) {
        exe(OP_ADD, &zsofs, sofs, EXP_H3210, 4, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
        mop(OP_LDWR, 1, &qid, vpack[src].qid, sofs, MSK_W0, vpack[src].qid, len, 0, 0, NULL, 0);
        exe(OP_CMP_EQ, &qidofs, qid, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_SLL, 2LL);
        mop(OP_LDWR, 1, &qinfo, vinfo, qidofs, MSK_W0, vinfo, nvertices, 0, 0, NULL, 0);
        exe(OP_CMP_EQ, &cc3, qinfo_depth, qinfo, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 0xffffffffLL, OP_SRL, 16LL);
        exe(OP_CMP_EQ, &cc2, sdepth_m1, EXP_H3210, qinfo_depth, EXP_H3210, 0, EXP_H3210, OP_AND, 1LL, OP_NOP, OLL);
        exe(OP_CMP_EQ, &cc1, sdepth_p1, EXP_H3210, qinfo_depth, EXP_H3210, 0, EXP_H3210, OP_AND, 1LL, OP_NOP, OLL);
        exe(OP_CMP_LT, &cc0, dst, EXP_H3210, qid, EXP_H3210, 0, EXP_H3210, OP_AND, 1LL, OP_NOP, OLL);
        exe(OP_NOP, &cand0, cc1, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, cc0, OP_NOP, OLL);
        exe(OP_NOP, &cand1, cc3, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, cc2, OP_NOP, OLL);
        exe(OP_NOP, &cand2, cand1, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, cand0, OP_NOP, OLL);

        exe(OP_ADD, &vsearchqid, qid, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_SLL, MAXN_BIT);
        exe(OP_ADD, &vsearchtop, vsearch, EXP_H3210, vsearchqid, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
        mop(OP_LDBR, 1, &search, vsearchtop, dst, MSK_W0, vsearch, vsearchsize, 0, 0, NULL, 0); /* search */

        exe(OP_NOP, &tricount_ofs, cand2, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_AND, search, OP_NOP, OLL);

        //tricount += tricount_ofs;
        mop(OP_LDWR, 1, &tricount_r, tricount, 0, MSK_W0, tricount, 1, 0, 1, NULL, 0); /* curfront_e->n_e */
        exe(OP_ADD, &tricount_r, tricount_r, EXP_H3210, tricount_ofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000ffffffffLL, OP_NOP, OLL);
        mop(OP_STWR, 1, &tricount_r, tricount, 0, MSK_W0, tricount, 1, 0, 1, NULL, 0); /* curfront_e->n_e */
    }
//EMAX5A end
}
#endif

```

3.8 Image Recognition (ssim)

MINST

```
cent% make -f Makefile-cent.emax6nc all clean  
cent% cd ..;/ ssim/ssim-cent.emax6nc -x -t -I0 -C1 -F1
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean  
zynq% cd ..;/ ssim/ssim-zynq.emax6+dma -x -t -I0 -C1 -F1
```

CIFAR10

```
cent% make -f Makefile-cent.emax6nc all clean  
cent% cd ..;/ ssim/ssim-cent.emax6nc -x -t -I1 -C6 -F2
```

```
zynq% make -f Makefile-zynq.emax6+dma all clean  
zynq% cd ..;/ ssim/ssim-zynq.emax6+dma -x -t -I1 -C6 -F2
```

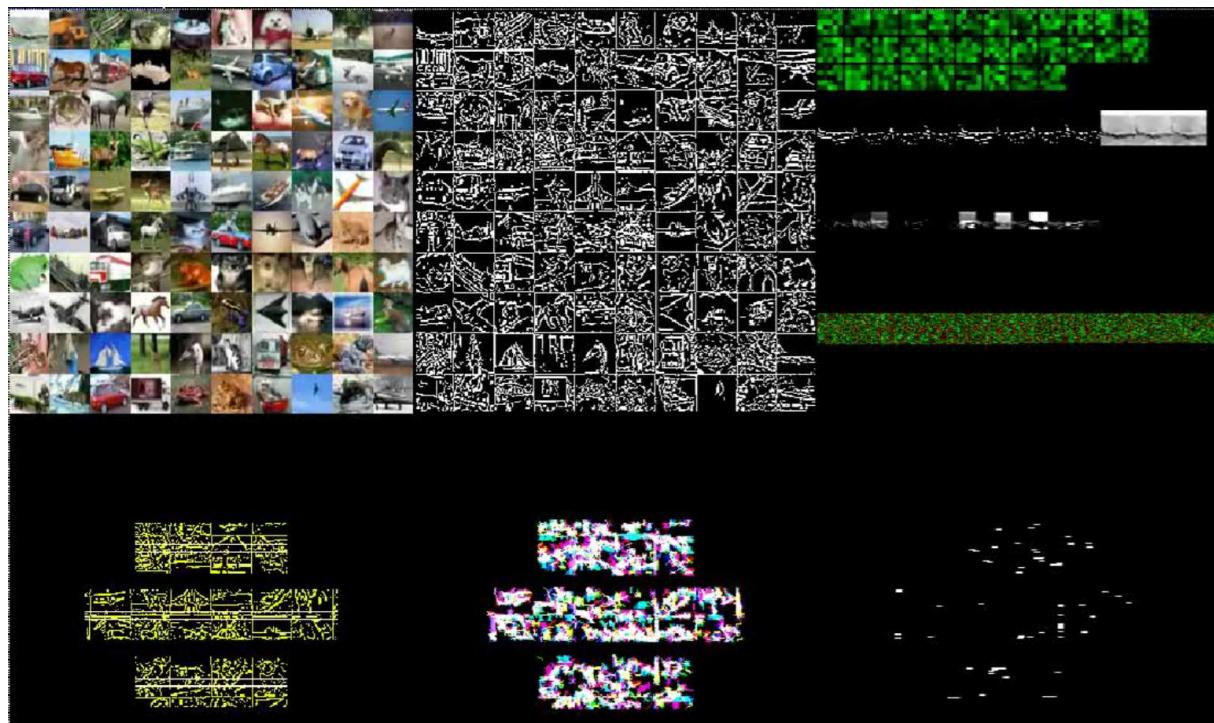


Figure 3.66: Image recognition (training + inference)

3.8.1 Cnn5x5

```

for (img=0; img<BATCH; img++) {
    for (top=0; top<M; top+=RMGRP) {
        for (iset=0; iset<IC; iset+=IMAP) { /* accumulate multiple sets of IC */
            Uint *ip0 = &i_1_ip0[(img*IC*iset+0)*IM*IM]; /* top of input#0 */
            Uint *it00 = ip0+top*IM, *ip0[25];
            ip0[ 0] = ip0+(top*0)*IM+0; ip0[ 1] = ip0+(top*0)*IM+1; ip0[ 2] = ip0+(top*0)*IM+2; ip0[ 3] = ip0+(top*0)*IM+3; ip0[ 4] = ip0+(top*0)*IM+4;
            ip0[ 5] = ip0+(top*1)*IM+0; ip0[ 6] = ip0+(top*1)*IM+1; ip0[ 7] = ip0+(top*1)*IM+2; ip0[ 8] = ip0+(top*1)*IM+3; ip0[ 9] = ip0+(top*1)*IM+4;
            ip0[10] = ip0+(top*2)*IM+0; ip0[11] = ip0+(top*2)*IM+1; ip0[12] = ip0+(top*2)*IM+2; ip0[13] = ip0+(top*2)*IM+3; ip0[14] = ip0+(top*2)*IM+4;
            ip0[15] = ip0+(top*3)*IM+0; ip0[16] = ip0+(top*3)*IM+1; ip0[17] = ip0+(top*3)*IM+2; ip0[18] = ip0+(top*3)*IM+3; ip0[19] = ip0+(top*3)*IM+4;
            ip0[20] = ip0+(top*4)*IM+0; ip0[21] = ip0+(top*4)*IM+1; ip0[22] = ip0+(top*4)*IM+2; ip0[23] = ip0+(top*4)*IM+3; ip0[24] = ip0+(top*4)*IM+4;

            for (oc=0; oc<OC4/NCHIP; oc+=W) { /* set output channel */
                Uint *kp00[NCHIP], *kp01[NCHIP], *kp02[NCHIP], *kp03[NCHIP];
                Uint *op0[NCHIP], *op1[NCHIP], *op2[NCHIP], *op3[NCHIP];
                Uint *ot0[NCHIP], *ot1[NCHIP], *ot2[NCHIP], *ot3[NCHIP];

                for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
                    Uint choc = CHIP*OC4/NCHIP+oc;
                    kp00[CHIP] = i_ker+((choc+0)*IC*iset+0)*K*K; kp01[CHIP] = i_ker+((choc+1)*IC*iset+0)*K*K;
                    kp02[CHIP] = i_ker+((choc+2)*IC*iset+0)*K*K; kp03[CHIP] = i_ker+((choc+3)*IC*iset+0)*K*K;
                    op0[CHIP] = i_out+(img*OC4+choc+0)*M*M+top*M; op1[CHIP] = i_out+(img*OC4+choc+1)*M*M+top*M;
                    op2[CHIP] = i_out+(img*OC4+choc+2)*M*M+top*M; op3[CHIP] = i_out+(img*OC4+choc+3)*M*M+top*M;
                    ot0[CHIP] = i_out+(img*OC4+choc+0)*M*M+top*M; ot1[CHIP] = i_out+(img*OC4+choc+1)*M*M+top*M;
                    ot2[CHIP] = i_out+(img*OC4+choc+2)*M*M+top*M; ot3[CHIP] = i_out+(img*OC4+choc+3)*M*M+top*M;
                }

#define cnn5x5_core1(b, o, bp1, n) \
    mop(OP_LDWR, 1, &BR[b][0][1], (U11)kp00[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][0][0], (U11)kp01[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][1], (U11)kp02[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][0], (U11)kp03[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][2][1], (U11)ip00[n], iofs, MSK_W1, (U11)it00, IMlen, 0, (U11)NULL, IMlen);\ \
    exe(OP_FMA, &AR[bp1][0], AR[b][0], EXP_H3210, BR[b][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][1], AR[b][1], EXP_H3210, BR[b][0][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][2], AR[b][2], EXP_H3210, BR[b][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][3], AR[b][3], EXP_H3210, BR[b][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);

#define cnn5x5_final(b, bp1) \
    mop(OP_LDWR, 1, &BR[bp1][0][1], (U11)op0[CHIP], oofs, MSK_W0, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][1][1], (U11)op1[CHIP], oofs, MSK_W0, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][2][1], (U11)op2[CHIP], oofs, MSK_W0, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][3][1], (U11)op3[CHIP], oofs, MSK_W0, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    exe(OP_FAD, &AR[bp1][0], AR[b][0], EXP_H3210, BR[bp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][1], AR[b][1], EXP_H3210, BR[bp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][2], AR[b][2], EXP_H3210, BR[bp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][3], AR[b][3], EXP_H3210, BR[bp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);\ \
    mop(OP_STWR, 1, &AR[bp1][0], oofs, (U11)op0[CHIP], MSK_DO, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][1], oofs, (U11)op1[CHIP], MSK_DO, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][2], oofs, (U11)op2[CHIP], MSK_DO, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][3], oofs, (U11)op3[CHIP], MSK_DO, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen)

//EMAX5A begin cnn5x5_mapdist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
/*2*/ for (INITI=0,LOOP0=M,cofs=(0-4LL)<<32|((0-M4)&0xffffffff); LOOP0--; INITI=0) {
/*1*/ for (INITI=0,LOOP1=M,cofs=(0-4LL)<<32|((0-M4)&0xffffffff); LOOP1--; INITI=0) {
/* */ /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
/* */ /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
/* */ /* mapped to FOR() on BR[63][0][1] */ /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?IM4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &iofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#1 */

    /*****in*****/
    mop(OP_LDWR, 1, &BR[2][0][1], (U11)kp00[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][0][0], (U11)kp01[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][1][1], (U11)kp02[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][1][0], (U11)kp03[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 10KB */
    mop(OP_LDWR, 1, &BR[2][2][1], (U11)ip00[0], iofs, MSK_W1, (U11)it00, IMlen, 0, 0, (U11)NULL, IMlen); /* stage#2 10KB */
    exe(OP FML, &AR[3][0], BR[2][0][1], EXP_H3210, BR[2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, EXP_H3210, OP_NOP, OLL); /* stage#3 */
    exe(OP FML, &AR[3][1], BR[2][1][1], EXP_H3210, BR[2][0][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, EXP_H3210, OP_NOP, OLL); /* stage#3 */
    exe(OP FML, &AR[3][2], BR[2][2][1], EXP_H3210, BR[2][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, EXP_H3210, OP_NOP, OLL); /* stage#3 */
    exe(OP FML, &AR[3][3], BR[2][2][1], EXP_H3210, BR[2][1][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, EXP_H3210, OP_NOP, OLL); /* stage#3 */
    cnn5x5_core1( 3, 4LL, 4, 1);
    cnn5x5_core1( 4, 8LL, 5, 2);
    cnn5x5_core1( 5, 12LL, 6, 3);
    cnn5x5_core1( 6, 16LL, 7, 4);
    cnn5x5_core1( 7, 20LL, 8, 5);
    cnn5x5_core1( 8, 24LL, 9, 6);
    cnn5x5_core1( 9, 28LL, 10, 7);
    cnn5x5_core1(10, 32LL, 11, 8);
    cnn5x5_core1(11, 36LL, 12, 9);
    cnn5x5_core1(12, 40LL, 13, 10);
    cnn5x5_core1(13, 44LL, 14, 11);
    cnn5x5_core1(14, 48LL, 15, 12);
    cnn5x5_core1(15, 52LL, 16, 13);
    cnn5x5_core1(16, 56LL, 17, 14);
    cnn5x5_core1(17, 60LL, 18, 15);
    cnn5x5_core1(18, 64LL, 19, 16);
    cnn5x5_core1(19, 68LL, 20, 17);
    cnn5x5_core1(20, 72LL, 21, 18);
    cnn5x5_core1(21, 76LL, 22, 19);
    cnn5x5_core1(22, 80LL, 23, 20);
    cnn5x5_core1(23, 84LL, 24, 21);
    cnn5x5_core1(24, 88LL, 25, 22);
    cnn5x5_core1(25, 92LL, 26, 23);
    cnn5x5_core1(26, 96LL, 27, 24);
    /*****final*****/
    cnn5x5_final(27, 28);
}

} } }

//EMAX5A end
if (Force) Force = 0;
} } }

//EMAX5A drain_dirty_lmm

```



Figure.3.67: 5x5 Convolution

3.8.2 Cnn3x3

```

for (img=0; img<BATCH; img++) {
    for (top=0; top<M; top+=RMGRP) {
        for (iset=0; iset<IC; iset+=IMAP) { /* accumulate multiple sets of IC */
            UInt *ip0 = &i_1_im[img*IC+iset+0]*IM*IM; /* top of input#0 */
            UInt *it0 = ip0+top*IM, *ip0[9];
            ip0[0] = ip0+(top+0)*IM+1; ip0[2] = ip0+(top+0)*IM+2;
            ip0[3] = ip0+(top+1)*IM+0; ip0[4] = ip0+(top+1)*IM+1; ip0[5] = ip0+(top+1)*IM+2;
            ip0[6] = ip0+(top+2)*IM+0; ip0[7] = ip0+(top+2)*IM+1; ip0[8] = ip0+(top+2)*IM+2;

            for (oc=0; oc<OC4/NCHIP; oc+=W) { /* set output channel */
                UInt *kp00[NCHIP], *kp01[NCHIP], *kp02[NCHIP], *kp03[NCHIP];
                UInt *op0[NCHIP], *op1[NCHIP], *op2[NCHIP], *op3[NCHIP];
                UInt *ot0[NCHIP], *ot1[NCHIP], *ot2[NCHIP], *ot3[NCHIP];

                for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
                    UInt choc = CHIP*OC4/NCHIP+oc;
                    kp00[CHIP] = i_ker*((choc+0)*IC+iset+0)*K*K; kp01[CHIP] = i_ker*((choc+1)*IC+iset+0)*K*K;
                    kp02[CHIP] = i_ker*((choc+2)*IC+iset+0)*K*K; kp03[CHIP] = i_ker*((choc+3)*IC+iset+0)*K*K;
                    op0[CHIP] = i_out+(img*OC4+choc+0)*M*M+top*M; op1[CHIP] = i_out+(img*OC4+choc+1)*M*M+top*M;
                    op2[CHIP] = i_out+(img*OC4+choc+2)*M*M+top*M; op3[CHIP] = i_out+(img*OC4+choc+3)*M*M+top*M;
                    ot0[CHIP] = i_out+(img*OC4+choc+0)*M*M+top*M; ot1[CHIP] = i_out+(img*OC4+choc+1)*M*M+top*M;
                    ot2[CHIP] = i_out+(img*OC4+choc+2)*M*M+top*M; ot3[CHIP] = i_out+(img*OC4+choc+3)*M*M+top*M;
                }
            }

#define cnn3x3_core1(b, o, bpi, n) \
mop(OP_LDWR, 1, &BR[b][0][1], (U11)kp00[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
mop(OP_LDWR, 1, &BR[b][0][0], (U11)kp01[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
mop(OP_LDWR, 1, &BR[b][1][1], (U11)kp02[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
mop(OP_LDWR, 1, &BR[b][1][0], (U11)kp03[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
mop(OP_LDWR, 1, &BR[b][2][1], (U11)ip00[n], iofs, MSK_W1, (U11)it00, IMlen, 0, o, (U11)NULL, IMlen);\ \
exe(OP_FMA, &AR[b][1][0], AR[b][0], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FMA, &AR[b][1][1], AR[b][1], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FMA, &AR[b][1][2], AR[b][2], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FMA, &AR[b][1][3], AR[b][3], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL)

#define cnn3x3_final(b, bp1) \
mop(OP_LDWR, 1, &BR[b][1][0][1], (U11)op0[CHIP], oofs, MSK_W0, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
mop(OP_LDWR, 1, &BR[b][1][1][1], (U11)op1[CHIP], oofs, MSK_W0, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
mop(OP_LDWR, 1, &BR[b][1][2][1], (U11)op2[CHIP], oofs, MSK_W0, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
mop(OP_LDWR, 1, &BR[b][1][3][1], (U11)op3[CHIP], oofs, MSK_W0, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
exe(OP_FAD, &AR[b][1][0], AR[b][0], EXP_H3210, BR[b][1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FAD, &AR[b][1][1], AR[b][1], EXP_H3210, BR[b][1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FAD, &AR[b][1][2], AR[b][2], EXP_H3210, BR[b][1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FAD, &AR[b][1][3], AR[b][3], EXP_H3210, BR[b][1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
mop(OP_STWR, 1, &AR[b][1][0], oofs, (U11)op0[CHIP], MSK_DO, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
mop(OP_STWR, 1, &AR[b][1][1], oofs, (U11)op1[CHIP], MSK_DO, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
mop(OP_STWR, 1, &AR[b][1][2], oofs, (U11)op2[CHIP], MSK_DO, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
mop(OP_STWR, 1, &AR[b][1][3], oofs, (U11)op3[CHIP], MSK_DO, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen)

//EMAX6 begin cnn3x3 mapdlist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
/*2*/ for (INIT1=1,LOOP1=RMGRP,rofs=(0-IM4)<<32|((0-M4)&0xffffffff); LOOP1--; INIT1=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
/*1*/ for (INIT0=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4LL)&0xffffffff); LOOP0--; INIT0=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?IM4M:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &iofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD, &oofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000f0fffffLL, OP_NOP, OLL); /* stage#1 */

    /*****in*****/
    mop(OP_LDWR, 1, &BR[2][0][1], (U11)kp00[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][0][0], (U11)kp01[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][1][1], (U11)kp02[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][1][0], (U11)kp03[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 10KB */
    mop(OP_LDWR, 1, &BR[2][2][1], (U11)ip00[0], iofs, MSK_W1, (U11)it00, IMlen, 0, o, (U11)NULL, IMlen); /* stage#2 10KB */
    exe(OP_FML, &AR[3][0], BR[2][2][1], EXP_H3210, BR[2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    exe(OP_FML, &AR[3][1], BR[2][2][1], EXP_H3210, BR[2][0][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    exe(OP_FML, &AR[3][2], BR[2][2][1], EXP_H3210, BR[2][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    exe(OP_FML, &AR[3][3], BR[2][2][1], EXP_H3210, BR[2][1][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    cnn3x3_core1( 3, 4LL, 4, 1);
    cnn3x3_core1( 4, 8LL, 5, 2);
    cnn3x3_core1( 5, 12LL, 6, 3);
    cnn3x3_core1( 6, 16LL, 7, 4);
    cnn3x3_core1( 7, 20LL, 8, 5);
    cnn3x3_core1( 8, 24LL, 9, 6);
    cnn3x3_core1( 9, 28LL, 10, 7);
    cnn3x3_core1(10, 32LL, 11, 8);

    /****final****/
    cnn3x3_final(11, 12);
}
}
//EMAX6 end
if (Force) Force = 0;
}
}
//EMAX6 drain_dirty_lmm

```

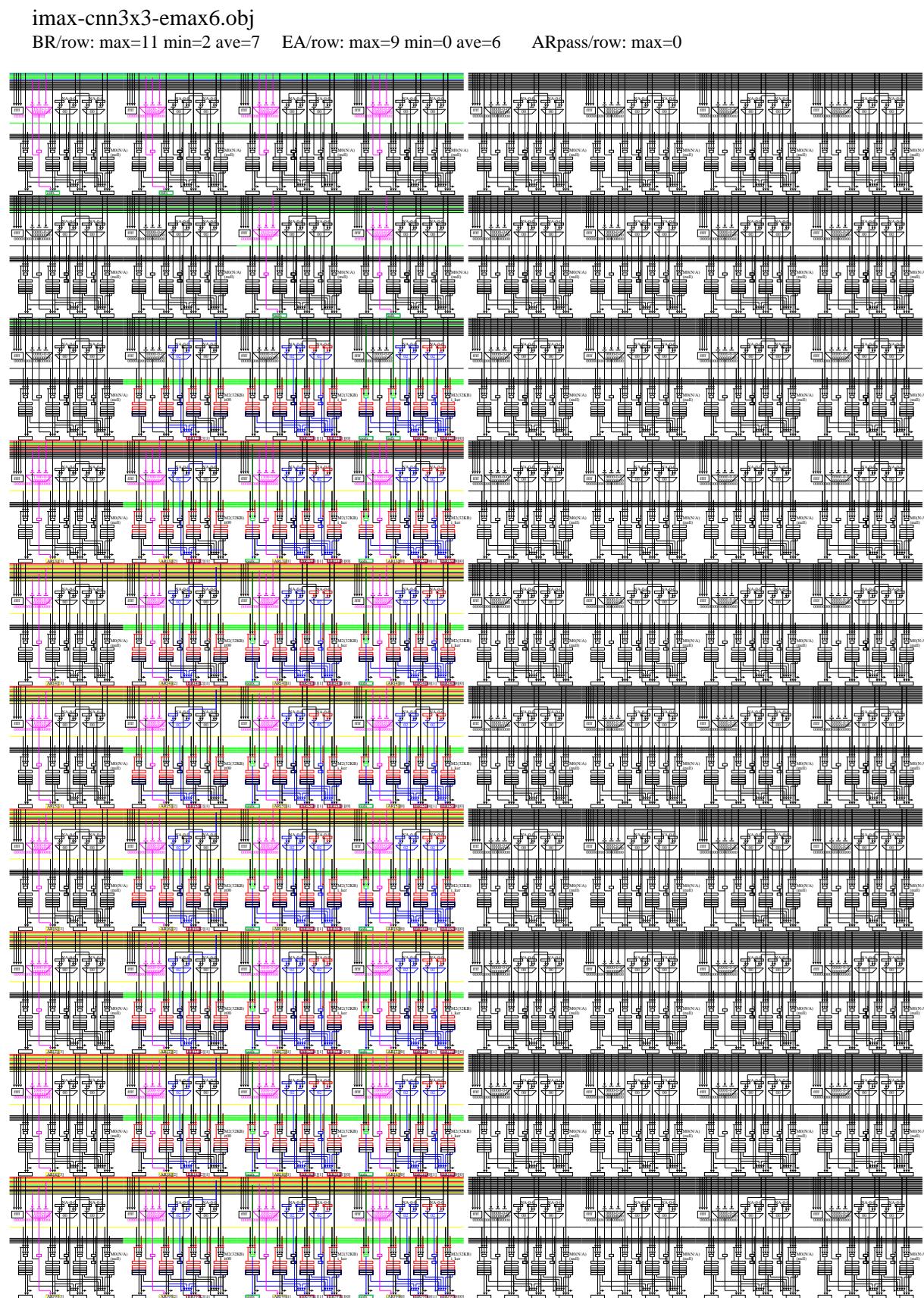


Figure.3.68: 3x3 Convolution

3.8.3 Cnn2x2

```

for (top=0; top<M; top+=RMGRP) {
    for (iset=0; iset<IC; iset+=IMAP) { /* accumulate multiple sets of IC */
        Uint *ip00 = &_l_im[iset*IM*BATCH*IM]; /* top of input#0 */
        Uint *it00 = ip0+top*IM*BATCH, *ip00[4];
        ip00[0] = ip0+(top*0)*IM*BATCH+0; ip00[1] = ip0+(top*0)*IM*BATCH+1;
        ip00[2] = ip0+(top*1)*IM*BATCH+0; ip00[3] = ip0+(top*1)*IM*BATCH+1;

        for (rofs=0; rofs<RMGRP&&(top+rofs)<M; rofs++) { /* image loop (row) */

            for (oc=0; oc<OC4*NCHIP; oc+=W) { /* set output channel */
                Uint *kp00[NCHIP], *kp01[NCHIP], *kp02[NCHIP], *kp03[NCHIP];
                Uint *op0[NCHIP], *op1[NCHIP], *op2[NCHIP], *op3[NCHIP];
                Uint *ot0[NCHIP], *ot1[NCHIP], *ot2[NCHIP], *ot3[NCHIP];

                for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
                    Uint choc = CHIP*OC4/NCHIP+oc;
                    kp00[CHIP] = i_ker*((choc*0)*IC+iset+0)*K*K;
                    kp01[CHIP] = i_ker*((choc*1)*IC+iset+0)*K*K;
                    kp02[CHIP] = i_ker*((choc*2)*IC+iset+0)*K*K;
                    kp03[CHIP] = i_ker*((choc*3)*IC+iset+0)*K*K;
                    op0[CHIP] = i_out*((choc*0)*M+top)*M*BATCH;
                    op1[CHIP] = i_out*((choc*1)*M+top)*M*BATCH;
                    op2[CHIP] = i_out*((choc*2)*M+top)*M*BATCH;
                    op3[CHIP] = i_out*((choc*3)*M+top)*M*BATCH;
                    ot0[CHIP] = i_out*((choc*0)*M+top)*M*BATCH;
                    ot1[CHIP] = i_out*((choc*1)*M+top)*M*BATCH;
                    ot2[CHIP] = i_out*((choc*2)*M+top)*M*BATCH;
                    ot3[CHIP] = i_out*((choc*3)*M+top)*M*BATCH;
                }

#define cnn2x2_core1(b, o, bpi, n) \
mop(OP_LDWR, 1, &BR[b][0][1], (U11)kp00[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\
mop(OP_LDWR, 1, &BR[b][0][0], (U11)kp01[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\
mop(OP_LDWR, 1, &BR[b][1][1], (U11)kp02[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\
mop(OP_LDWR, 1, &BR[b][1][0], (U11)kp03[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\
mop(OP_LDWR, 1, &BR[b][2][1], (U11)ip00[n], iofs, MSK_W1, (U11)it00, IMlen, 0, 0, (U11)NULL, IMlen);\
exe(OP_FMA, &AR[b][0], AR[b][0], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
exe(OP_FMA, &AR[b][1], AR[b][1], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
exe(OP_FMA, &AR[b][2], AR[b][2], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
exe(OP_FMA, &AR[b][3], AR[b][3], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL)

#define cnn2x2_final(b, bp1) \
mop(OP_LDWR, 1, &BR[bp1][0][1], (U11)op0[CHIP], oofs, MSK_W0, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
mop(OP_LDWR, 1, &BR[bp1][1][1], (U11)op1[CHIP], oofs, MSK_W0, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
mop(OP_LDWR, 1, &BR[bp1][2][1], (U11)op2[CHIP], oofs, MSK_W0, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
mop(OP_LDWR, 1, &BR[bp1][3][1], (U11)op3[CHIP], oofs, MSK_W0, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
exe(OP_FAD, &AR[bp1][0], AR[b][0], EXP_H3210, BR[bp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
exe(OP_FAD, &AR[bp1][1], AR[b][1], EXP_H3210, BR[bp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
exe(OP_FAD, &AR[bp1][2], AR[b][2], EXP_H3210, BR[bp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
exe(OP_FAD, &AR[bp1][3], AR[b][3], EXP_H3210, BR[bp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\
mop(OP_STWR, 1, &AR[bp1][0], oofs, (U11)op0[CHIP], MSK_DO, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
mop(OP_STWR, 1, &AR[bp1][1], oofs, (U11)op1[CHIP], MSK_DO, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
mop(OP_STWR, 1, &AR[bp1][2], oofs, (U11)op2[CHIP], MSK_DO, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\
mop(OP_STWR, 1, &AR[bp1][3], oofs, (U11)op3[CHIP], MSK_DO, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen)

//EMAX5A begin cnn2x2_mapdist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
/*2*/ for (INIT1=1,LOOP1=BATCH,img=(0-IM4)&<32|((0-M4)&0xfffffff); LOOP1--; INIT1=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
/*1*/ for (INIT0=1,LOOP0=M,cofs=(0-4LL)&0xfffffff; LOOP0--; INIT0=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
    exe(OP_ADD, &img, img, EXP_H3210, INIT0*IM4M+0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &cofs, INIT0*cofs, EXP_H3210, 4LL<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &iofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD, &oofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000f0000000LL, OP_NOP, OLL); /* stage#1 */

    /*****in0*****/
    mop(OP_LDWR, 1, &BR[2][0][1], (U11)kp00[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][0][0], (U11)kp01[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][1][1], (U11)kp02[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 */
    mop(OP_LDWR, 1, &BR[2][1][0], (U11)kp03[CHIP], OLL, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen); /* stage#2 10KB */
    mop(OP_LDWR, 1, &BR[2][2][1], (U11)ip00[0], iofs, MSK_W1, (U11)it00, IMlen, 0, 0, (U11)NULL, IMlen); /* stage#2 10KB */
    exe(OP FML, &AR[3][0], BR[2][2][1], EXP_H3210, BR[2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    exe(OP FML, &AR[3][1], BR[2][2][1], EXP_H3210, BR[2][0][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    exe(OP FML, &AR[3][2], BR[2][2][1], EXP_H3210, BR[2][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    exe(OP FML, &AR[3][3], BR[2][2][1], EXP_H3210, BR[2][1][0], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    cnn2x2_core1( 3, 4LL, 4, 1);
    cnn2x2_core1( 4, 8LL, 5, 2);
    cnn2x2_core1( 5, 12LL, 6, 3);
    /****final****/
    cnn2x2_final( 6, 7);

    } }
//EMAX5A end
    if (Force) Force = 0;
} } }
//EMAX5A drain_dirty_lmm

```

imax-cnn2x2-emax6.obj
 BR/row: max=11 min=2 ave=6 EA/row: max=9 min=0 ave=5 ARpass/row: max=0

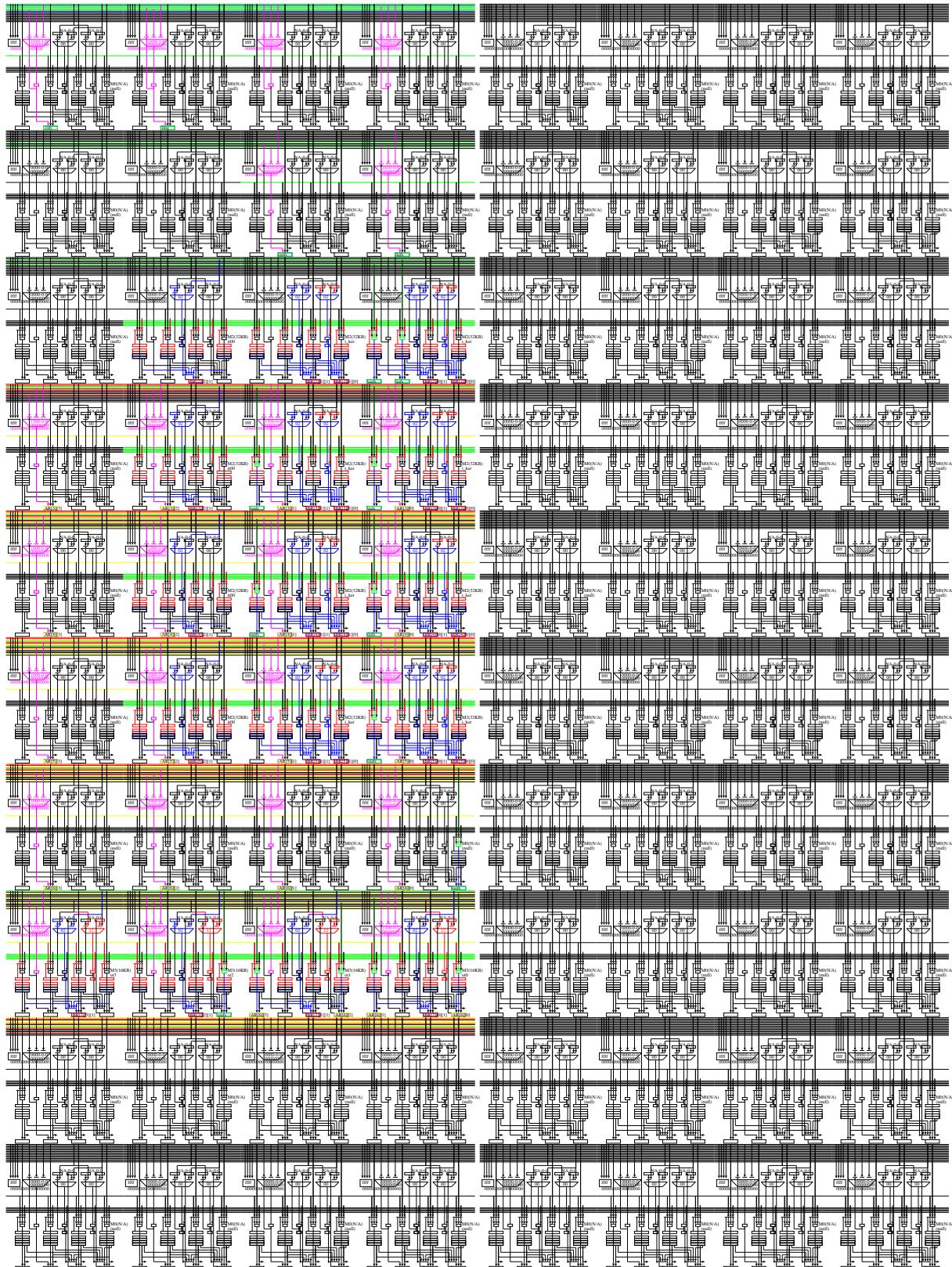


Figure.3.69: 2x2 Convolution

3.8.4 Sgemm00

```

for (top=0; top<m/NCHIP; top+=RMGRP) { /* will be parallelized by multi-chip (M/#chip) */
    for (blk=0; blk<ka; blk+=H) { /* 3 重ループ展開の外側対象 */
        typedef struct {Uint i[4]} UI4;
        Uint *a0[NCHIP];
        Uint *a[H][NCHIP];
        UI4 *b[H];
        UI4 *b0[H], *b1[H], *b2[H], *b3[H];
        UI4 *c0[NCHIP];
        UI4 *c00[NCHIP], *c01[NCHIP], *c02[NCHIP], *c03[NCHIP];
        for (k=0; k<H; k++) {
            b[k] = i_mOB(b[blk+k]*n; b0[k] = b[k]; b1[k] = (Uint*)b[k]+1; b2[k] = (Uint*)b[k]+2; b3[k] = (Uint*)b[k]+3;
        }
        for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
            a0[CHIP] = i_mOA((CHIP*m/NCHIP+top)*ka;
            for (k=0; k<H; k++)
                a[k][CHIP] = a0[CHIP]+blk*k;
            c0[CHIP] = i_mOC+(CHIP*m/NCHIP+top)*n;
            c00[CHIP]=(Uint*)c0[CHIP]+0; c01[CHIP]=(Uint*)c0[CHIP]+1; c02[CHIP]=(Uint*)c0[CHIP]+2; c03[CHIP]=(Uint*)c0[CHIP]+3;
        }
        cofslimit1 = n4- 4; /* cofslimit3 < 36 x */
        cofslimit2 = n4- 8; /* cofslimit3 < 32 x */
        cofslimit3 = n4-12; /* cofslimit3 < 28 x */
    }
#define sgemm00_core1(r, rm1, rp1) \
    map(OP_LDWR, 1, &BR[r][0][1], (U11)b0[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
    map(OP_LDWR, 1, &BR[r][0][0], (U11)b1[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
    map(OP_LDWR, 1, &BR[r][1][1], (U11)b2[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
    map(OP_LDWR, 1, &BR[r][1][0], (U11)b3[rm1], (U11)cofs, MSK_W1, (U11)a0[CHIP], Blen, 0, 0, (U11)NULL, Blen);\ \
    map(OP_FMA, &AR[rp1][0], AR[r][0], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_FMA, &AR[rp1][1], AR[r][1], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_FMA, &AR[rp1][2], AR[r][2], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_FMA, &AR[rp1][3], AR[r][3], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

#define sgemm00_final(r, rp1) \
    exe(OP_CMP_LT, &cc1, cofslimit1, EXP_H3210, cofslimit1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_CMP_LT, &cc2, cofslimit2, EXP_H3210, cofslimit2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_CMP_LT, &cc3, cofslimit3, EXP_H3210, cofslimit3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_LDWR, 1, &BR[rp1][0][1], (U11)c00[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    map(OP_LDWR, 1, &BR[rp1][1][1], (U11)c01[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    map(OP_LDWR, 1, &BR[rp1][2][1], (U11)c02[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    map(OP_LDWR, 1, &BR[rp1][3][1], (U11)c03[CHIP], (U11)cofs, MSK_W0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    map(OP_FAD, &AR[rp1][0], AR[r][0], EXP_H3210, BR[rp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_FAD, &AR[rp1][1], AR[r][1], EXP_H3210, BR[rp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_FAD, &AR[rp1][2], AR[r][2], EXP_H3210, BR[rp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_FAD, &AR[rp1][3], AR[r][3], EXP_H3210, BR[rp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    map(OP_STWR, 1, &AR[rp1][0], (U11)c00[CHIP], MSK_D0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    cex(OP_CXEX, &ex1, 0, 0, 0, cc1, Oxaaaaa);\ \
    map(OP_STWR, ex1, (U11)c01[CHIP], MSK_D0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    cex(OP_CXEX, &ex2, 0, 0, 0, cc2, Oxaaaaa);\ \
    map(OP_STWR, ex2, &AR[rp1][2], (U11)c02[CHIP], MSK_D0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen);\ \
    cex(OP_CXEX, &ex3, 0, 0, 0, cc3, Oxaaaaa);\ \
    map(OP_STWR, ex3, &AR[rp1][3], (U11)c03[CHIP], MSK_D0, (U11)c0[CHIP], Blen, 0, 1, (U11)NULL, Blen)

//EMAX5A begin sgemm00_maplist=0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
/*2*/ for (INITI1=1,LOOP1=RMGRP,rofs=(0-K4A)<<32|((W*4)&0xfffffff); LOOP1--; INITI1=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
/*1*/ for (INITI0=1,LOOP0=N/W,cofs=(W*4)<<32|((W*4)&0xfffffff); LOOP0--; INITI0=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
    exe(OP_ADD, &rofs, INITI0?cofs:cofs, EXP_H3210, (W*4)<<32|(W*4), EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, INITI0?K4A4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &rofs, rofs, EXP_H3210, cofslimit1, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffff, OP_NOP, OLL); /* stage#1 */
    map(OP_LDWR, 1, &BR[1][0][1], (U11)b0[0], (U11)cofs, MSK_W1, (U11)b[0], Blen, 0, 0, (U11)NULL, Blen); /* stage#1 */
    map(OP_LDWR, 1, &BR[1][0][0], (U11)b1[0], (U11)cofs, MSK_W1, (U11)b[0], Blen, 0, 0, (U11)NULL, Blen); /* stage#1 */
    map(OP_LDWR, 1, &BR[1][1][1], (U11)b2[0], (U11)cofs, MSK_W1, (U11)b[0], Blen, 0, 0, (U11)NULL, Blen); /* stage#1 */
    map(OP_LDWR, 1, &BR[1][1][0], (U11)b3[0], (U11)cofs, MSK_W1, (U11)a0[CHIP], Blen, 0, 0, (U11)NULL, Blen); /* stage#1 2KB */
    map(OP_LDWR, 1, &BR[1][2][1], (U11)a0[CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], Blen, 0, 0, (U11)NULL, Blen); /* stage#1 16KB */
    map(OP_FML, &AR[2][0], BR[1][0][1], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#2 */
    map(OP_FML, &AR[2][1], BR[1][0][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#2 */
    map(OP_FML, &AR[2][2], BR[1][1][1], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#2 */
    map(OP_FML, &AR[2][3], BR[1][1][0], EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL); /* stage#2 */

sgemm00_core1( 2, 1, 3);
sgemm00_core1( 3, 2, 4);
sgemm00_core1( 4, 3, 5);
sgemm00_core1( 5, 4, 6);
sgemm00_core1( 6, 5, 7);
sgemm00_core1( 7, 6, 8);
sgemm00_core1( 8, 7, 9);
sgemm00_core1( 9, 8, 10);
sgemm00_core1(10, 9, 11);
sgemm00_core1(11, 10, 12);
sgemm00_core1(12, 11, 13);
sgemm00_core1(13, 12, 14);
sgemm00_core1(14, 13, 15);
sgemm00_core1(15, 14, 16);
sgemm00_core1(16, 15, 17);
sgemm00_core1(17, 16, 18);
sgemm00_core1(18, 17, 19);
sgemm00_core1(19, 18, 20);
sgemm00_core1(20, 19, 21);
sgemm00_core1(21, 20, 22);
sgemm00_core1(22, 21, 23);
sgemm00_core1(23, 22, 24);
sgemm00_core1(24, 23, 25);
sgemm00_core1(25, 24, 26);
sgemm00_core1(26, 25, 27);
sgemm00_core1(27, 26, 28);
sgemm00_core1(28, 27, 29);
sgemm00_core1(29, 28, 30);
sgemm00_core1(30, 29, 31);
sgemm00_core1(31, 30, 32);
sgemm00_core1(32, 31, 33);
sgemm00_core1(33, 32, 34);
:
sgemm00_core1(48, 47, 49); /* 288/6 H=48 */
/*****final****/
sgemm00_final(49, 51);

} } }

//EMAX5A end
} }

//EMAX5A drain_dirty_lmm

```

imax-sgemm00-emax6.obj

BR/row: max=12 min=2 ave=11 EA/row: max=8 min=0 ave=4 ARpass/row: max=0

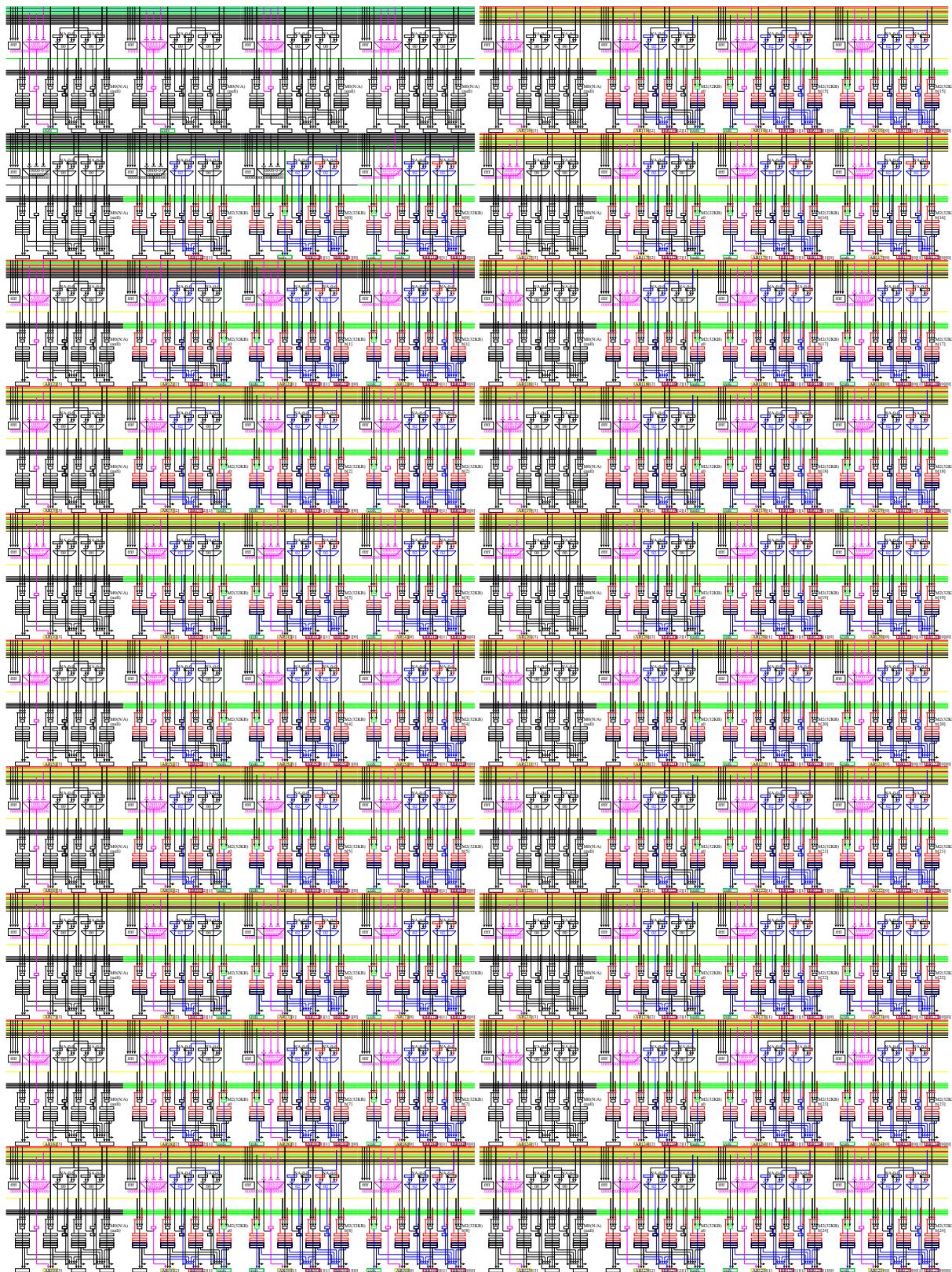


Figure.3.70: Sgemm00

3.8.5 Back_g_ker

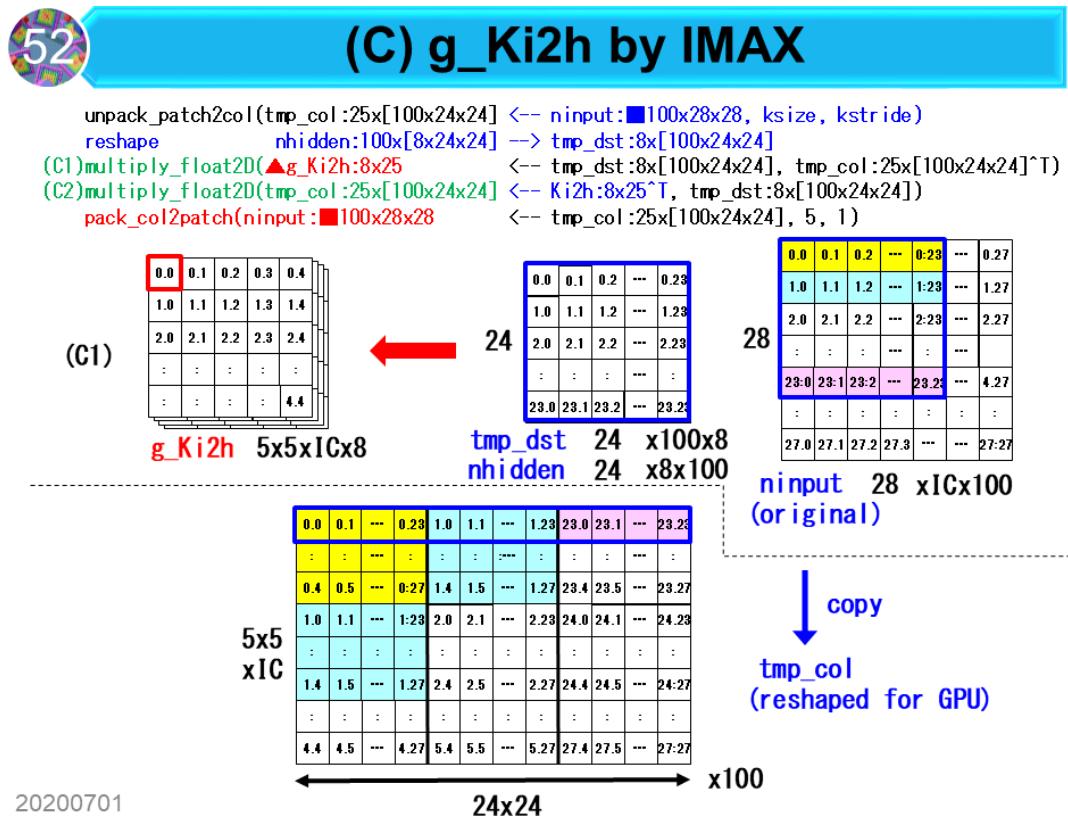


Figure.3.71: back_g_ker

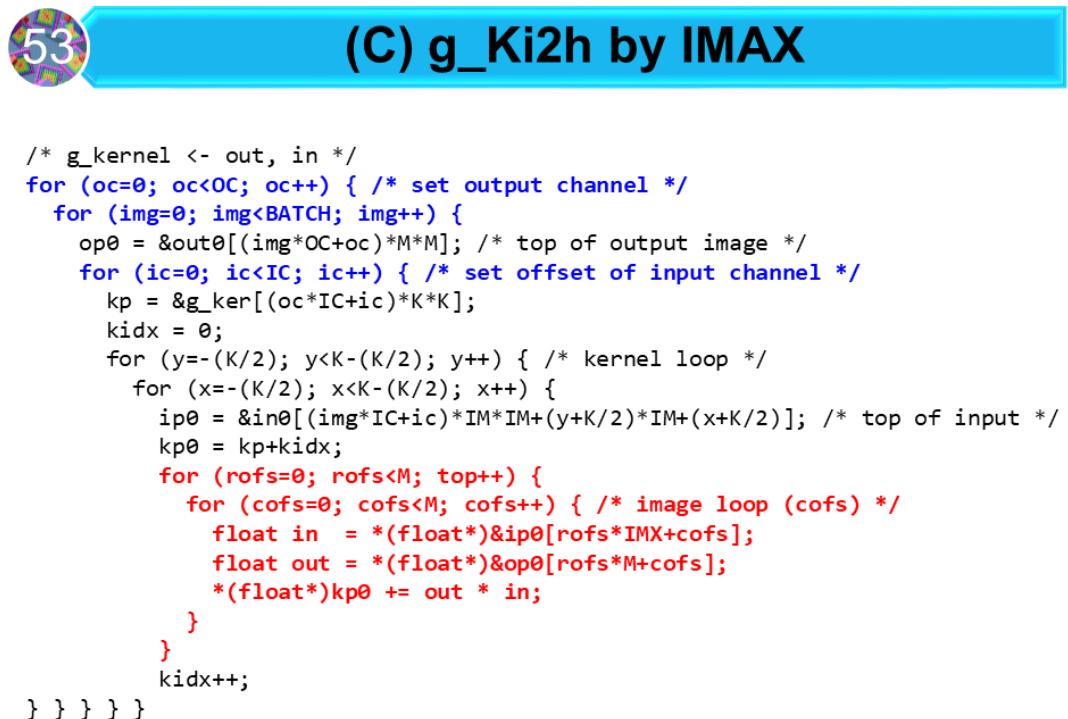


Figure.3.72: back_g_ker

```

for (oset=0; oset<((OC+OMAP-1)&~(OMAP-1)); oset+=OMAP) { /* set output channel */
    U11 cc0[IMAP], ccl[IMAP];
    Uint inum[IMAP], *ip0[IMAP], *it0[IMAP], onum[OMAP], *op0[OMAP], *ot0[OMAP], *kp0[IMAP][OMAP];
    for (rofs=0; rofs<M; rofs++) {
        for (iset=0; iset<((IC+IMAP-1)&~(IMAP-1)); iset+=IMAP) { /* set offset of input channel */
            kidx = 0;
            for (y=(K/2); y<K-(K/2); y++) { /* kernel loop */
                for (x=(K/2); x<K-(K/2); x++) {
                    for (ic=0; ic<IMAP; ic++) {
                        inum[ic] = iset+ic;
                        ip0[ic] = &i_inp[(iset+ic)*IMX*BATCH*IMX+(rofs+y*K/2)*BATCH*IMX+(x*K/2)]; /* input */
                        it0[ic] = &i_inp[(iset+ic)*IMX*BATCH*IMX+(rofs+y*K/2)*BATCH*IMX]; /* input */
                    }
                }
                for (oc=0; oc<OMAP; oc++) {
                    onum[oc] = oset+oc;
                    op0[oc] = &i_out[(oset+oc)*M*BATCH*M+rofs*BATCH*M]; /* output */
                    ot0[oc] = op0[oc];
                }
                for (ic=0; ic<IMAP; ic++) {
                    for (oc=0; oc<OMAP; oc++)
                        kp0[ic][oc] = ((iset+ic)<IC && (oset+oc)<DC) ? &i_ker[((oset+oc)*IC+iset+ic)*K*K+kidx] : 0; /* NULL skip DMA */
                }
            }
        }
    }
#define back_g_ker_core1(b, o, i) \
    exe(OP_CMP_LT, &cc0[o][i], onum[o], EXP_H3210, OC, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */ \
    exe(OP_CMP_LT, &ccl[o][i], inum[i], EXP_H3210, IC, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */ \
    mop(OP_LDWR, 1, &BR[b][1][1], (U11)op0[oc], oofs, MSK_W0, (U11)oto[o], Mlen, 0, 0, NULL, Mlen); /* stage#2 */ \
    mop(OP_LDWR, 1, &BR[b][2][1], (U11)ip0[i], iofs, MSK_W1, (U11)ito[i], IMXlen, 0, 0, NULL, IMXlen); /* stage#2 */ \
    exe(OP_NOP, &AR[b][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 (dummy) */ \
    mop(OP_LDWR, 1, &b0o, (U11)kp0[o][i], OLL, MSK_W0, (U11)kp0[o][i], ILL, 0, 1, NULL, ILL); /* stage#2 fold:unit[0] */ \
    exe(OP_FMA, &b0o, b0o, EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */ \
    cex(OP_CEZE, &ex0, 0, cc1[o][i], cc0[o][i], 0x8888); \
    mop(OP_STWR, ex0, &b0o, (U11)kp0[o][i], OLL, MSK_D0, (U11)kp0[o][i], ILL, 0, 1, NULL, ILL) /* stage#2 */

//EMAX5A begin back_g_ker mapdist0
/* */ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
/* */ for (INIT1=1,LOOP0=BATCH,img0=0-IMX4)<<32|((0-M4)&0xffffffff); LOOP0--; INIT1=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
/* */ /* */ for (INIT0=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xffffffff); LOOP0--; INIT0=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
    exe(OP_ADD, &img, img, EXP_H3210, INIT0?IMX4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffffff, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &iofs, img, EXP_H3210, cofis, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffff000000000LL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD, &ofs, img, EXP_H3210, cofis, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffffLL, OP_NOP, OLL); /* stage#1 */

    back_g_ker_core1( 2, 0, 0); /***** ic0 oc0*****/ \
    back_g_ker_core1( 3, 1, 0); /***** ic1 oc0*****/ \
    back_g_ker_core1( 4, 0, 1); /***** ic0 oc1*****/ \
    back_g_ker_core1( 5, 1, 1); /***** ic1 oc1*****/ \
    back_g_ker_core1( 6, 0, 2); /***** ic0 oc2*****/ \
    back_g_ker_core1( 7, 1, 2); /***** ic1 oc2*****/ \
    back_g_ker_core1( 8, 0, 3); /***** ic0 oc3*****/ \
    back_g_ker_core1( 9, 1, 3); /***** ic1 oc3*****/ \
    back_g_ker_core1(10, 0, 4); /***** ic0 oc4*****/ \
    back_g_ker_core1(11, 1, 4); /***** ic1 oc4*****/ \
    back_g_ker_core1(12, 0, 5); /***** ic0 oc5*****/ \
    back_g_ker_core1(13, 1, 5); /***** ic1 oc5*****/ \
    back_g_ker_core1(14, 0, 6); /***** ic0 oc6*****/ \
    back_g_ker_core1(15, 1, 6); /***** ic1 oc6*****/ \
    back_g_ker_core1(16, 0, 7); /***** ic0 oc7*****/ \
    back_g_ker_core1(17, 1, 7); /***** ic1 oc7*****/ \
} } }

//EMAX5A end
    kidx++;
} } } }

//EMAX5A drain_dirty_lmm

```

imax-back_g_ker-emax6.obj

BR/row: max=15 min=6 ave=13 EA/row: max=6 min=0 ave=5 ARpass/row: max=0

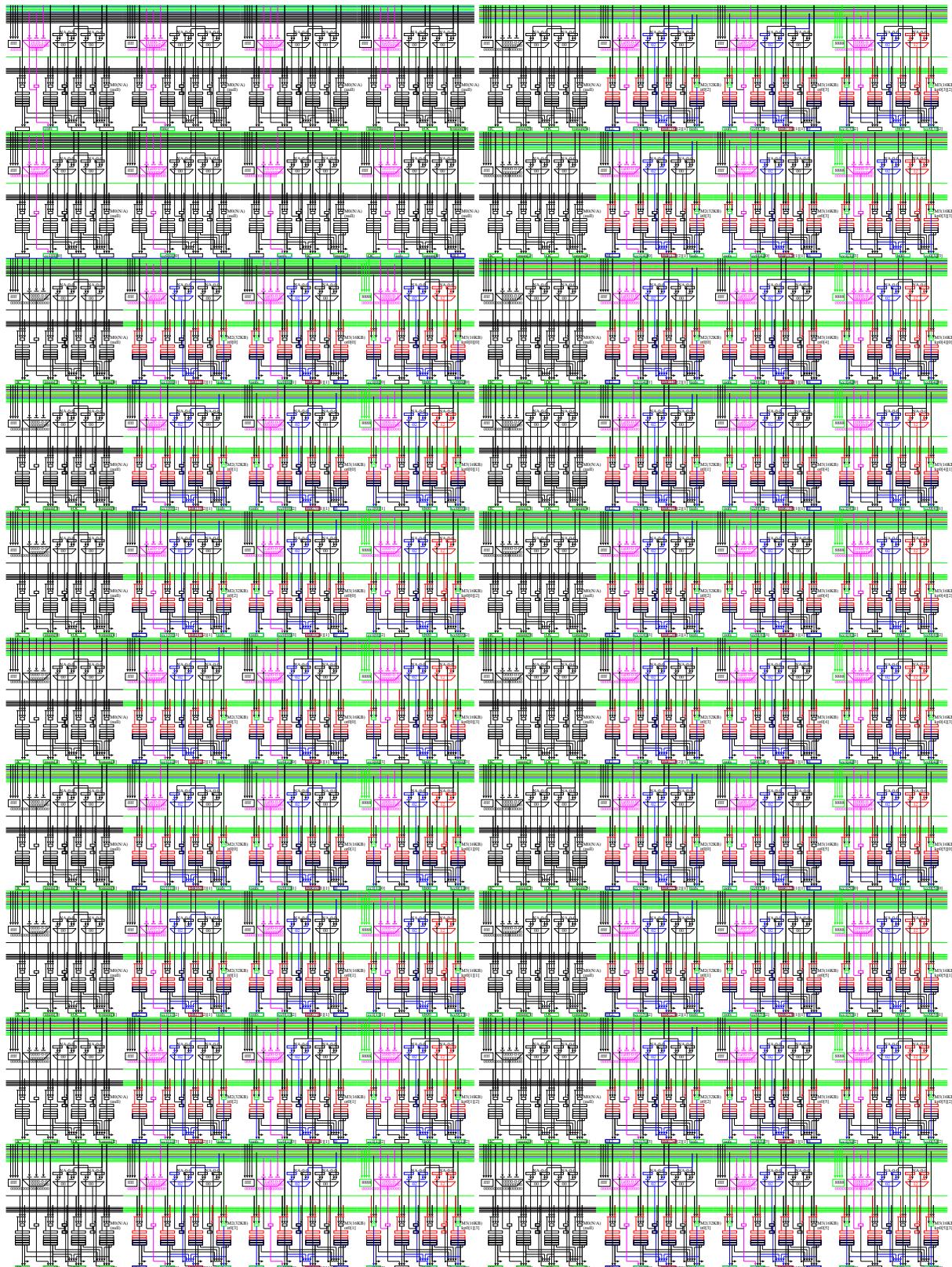


Figure.3.73: Back propagation to g_ker

3.8.6 Back_in

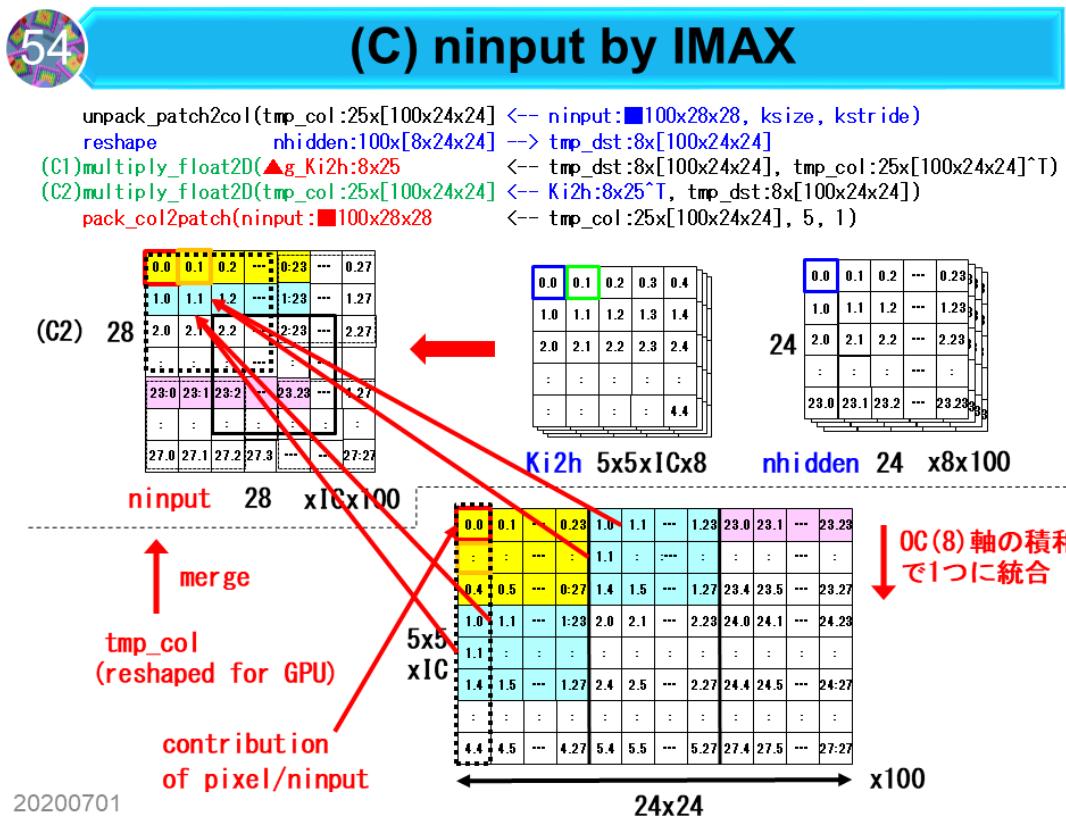


Figure.3.74: back_in



```

/* in <- kernel, out */
memset(in0, 0, sizeof(in0[0])*BATCH*IC*IM*IM);
if (K == 1 || IM-K+1 == M) { y0 = 0; x0 = 0; }
else if (IM == M) { y0 = -K/2; x0 = -K/2; }
for (ch=0;ch<IC*K*K;ch++) { /*5x5, 8x3x3*/
    ic = ch/(K*K);
    y = ch%(K*K)/K + y0;
    x = ch%(K*K)%K + x0;
    for (img=0;img<BATCH;img++) { /*100, 100*/
        for (oc=0; oc<OC; oc++) {
            op0 = &i_out[img*M*M*OC+oc*M*M];
            ip0 = &i_inp[(img*IC+ic)*IM*IM+y*IM+x];
            float cker = *(float*)&i_ker[oc*IC*K*K+ch];
            for (rofs=0;rofs<M;rofs++) { /*24, 10*/
                for (cofs=0;cofs<M;cofs++) { /*24, 10*/
                    if (0<=rofs+y && rofs+y<IM && 0<=cofs+x && cofs+x<IM)
                        *(float*)&ip0[rofs*IM+cofs]
                        += cker * *(float*)&op0[rofs*M+cofs];
                }
            }
        }
    }
}
  
```

20200701

Figure.3.75: back_in

```

for (oset=0; oset<((OC+OMAP-1)&~(OMAP-1)); oset+=OMAP) { /* set output channel */
    Uint inum[IMAP], *ip0[IMAP], *ito[IMAP], onum[OMAP], *op0[OMAP], *ot0[IMAP][OMAP];
    for (rofs=0;rofs<M;rofs++) { /*24, 10*/
        for (iset=0; iset<((IC+IMAP-1)&~(IMAP-1)); iset+=IMAP) { /* set offset of input channel */
            for (xy=0;xy<K;xy++) { /*5x5, 8x3x3*/
                y = xy/K + y0;
                x = xy%K + x0;
                U11 yIM4 = y*IM4;
                U11 x4 = x*4;
                U11 IMIM4 = IM*IM4;
                if (0<=rofs+y && rofs+y<IM) {
                    for (ic=0; ic<IMAP; ic++) {
                        inum[ic] = iset+ic;
                        ip0[ic] = &i_in[(iset+ic)*IM*BATCH*IM+(rofs+y)*BATCH*IM*x];
                        ito[ic] = &i_out[(iset+ic)*IM*BATCH*IM+(rofs+y)*BATCH*IM]; // x のマイナス成分を除去
                    }
                    for (oc=0; oc<OMAP; oc++) {
                        onum[oc] = oset+oc;
                        op0[oc] = &i_out[(oset+oc)*M*BATCH*M + rofs*BATCH*M];
                        ot0[oc] = op0[oc];
                    }
                    for (ic=0; ic<IMAP; ic++) {
                        for (oc=0; oc<OMAP; oc++) {
                            kp0[ic][oc] = (iset+ic)<IC&&(oset+oc)<OC ? (U11)i_ker[(oset+oc)*IC*K*K+(iset+ic)*K*K+xy] : 0; /* 0.0 */
                        }
                    }
                }
            }
        }
    }
}

#define back_in_core1(b, bpi, i, o) \
mop(OP_LDWR, 1, &BR[b][0][1], (U11)op0[0], oofs, MSK_W1, (U11)ot0[0], Mlen, 0, 0, NULL, Mlen); /* stage#2 */ \
exe(OP_FMA, &AR[bpi][0], AR[b][0], EXP_H3210, kp0[i][o], EXP_H3210, BR[b][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL) /* stage#3 */

#define back_in_final(b, bp2, i) \
exe(OP_ADD, &r10, cof, EXP_H3210, x4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffLL, OP_NOP, OLL); /* stage#5 */ \
exe(OP_CMP_LT, &c0, r10, EXP_H3210, IM4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */ \
mop(OP_LDWR, 1, &BR[bp2][0][1], (U11)ip0[1], iofs, MSK_W0, (U11)it0[i], IMlen, 0, 1, NULL, IMlen); /* stage#7 */ \
exe(OP_FAD, &AR[bp2][0], AR[b][0], EXP_H3210, BR[bp2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */ \
cex(OP_CEXE, &exo, 0, 0, cco, Oxaaaa); /* stage#7 */ \
mop(OP_STWR, exo, &AR[bp2][0], iofs, (U11)ip0[i], MSK_D0, (U11)it0[i], IMlen, 0, 1, NULL, IMlen) /* stage#7 */

//EMAX5A begin back_in_mapdist0
/*3*/ for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
/*2*/ for (INIT1=1,LOOP1=BATCH,img=(0-M4)<<32|((0-IM4)&0xfffffff); LOOP1--; INIT1=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
/*1*/ for (INIT0=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xfffffff); LOOP0--; INIT0=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
    exe(OP_ADD, &img, img, EXP_H3210, INIT0?M4IM4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &cofs, cof, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffLL, OP_NOP, OLL); /* stage#0 */
    exe(OP_ADD, &iofs, img, EXP_H3210, cof, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffLL, OP_NOP, OLL); /* stage#1 */
    exe(OP_ADD, &img, img, EXP_H3210, cof, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffLL, OP_NOP, OLL); /* stage#1 */
    /****ic0*****/
    mop(OP_LDWR, 1, &BR[2][0][1], (U11)op0[0], oofs, MSK_W1, (U11)ot0[0], Mlen, 0, 0, NULL, Mlen); /* stage#2 */
    exe(OP_FML, &AR[3][0], kp0[0][0], EXP_H3210, BR[2][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    back_in_core1( 3, 4, 0, 1); /***** ic0 oc1*****/
    back_in_core1( 4, 5, 0, 2); /***** ic0 oc2*****/
    back_in_core1( 5, 6, 0, 3); /***** ic0 oc3*****/
    back_in_core1( 6, 7, 0, 4); /***** ic0 oc4*****/
    back_in_core1( 7, 8, 0, 5); /***** ic0 oc5*****/
    back_in_core1( 8, 9, 0, 6); /***** ic0 oc6*****/
    back_in_core1( 9, 10, 0, 7); /***** ic0 oc7*****/
    back_in_final(10, 12, 0); /*****OMAP( 8)+2,OMAP( 8)+4****/
    /****ic1*****/
    mop(OP_LDWR, 1, &BR[13][0][1], (U11)op0[0], oofs, MSK_W1, (U11)ot0[0], Mlen, 0, 0, NULL, Mlen); /* stage#2 */
    exe(OP_FML, &AR[14][0], kp0[1][0], EXP_H3210, BR[13][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
    back_in_core1(14, 15, 1, 1); /***** ic1 oc1*****/
    back_in_core1(15, 16, 1, 2); /***** ic1 oc2*****/
    back_in_core1(16, 17, 1, 3); /***** ic1 oc3*****/
    back_in_core1(17, 18, 1, 4); /***** ic1 oc4*****/
    back_in_core1(18, 19, 1, 5); /***** ic1 oc5*****/
    back_in_core1(19, 20, 1, 6); /***** ic1 oc6*****/
    back_in_core1(20, 21, 1, 7); /***** ic1 oc7*****/
    back_in_final(21, 23, 1); /*****OMAP( 8)+2,OMAP( 8)+4****/
} } }

//EMAX5A end
} } }

//EMAX5A drain_dirty_lmm

```



Figure.3.76: Back propagation to input

3.9 Crypto

```
cent% make -f Makefile-csim.emax6+dma sha256-csim.emax6+dma clean
cent% ../../src/csim/csim sha256-csim.emax6+dma
```

```
zynq% make -f Makefile-zynq.emax6+dma sha256-zynq.emax6+dma clean
zynq% ./sha256-zynq.emax6+dma
```

3.9.1 SHA256

```
WORD i, j, th, thm;
WORD a, b, c, d, e, f, g, h, t1, t2;
printf("<<CPU>>mbuf=%08x mbuflen=%08.8x\n", (UInt)mbuf, (UInt)ctx->mbuflen);
for (i=0; i<ctx->mbuflen; i+=BLKSIZE) { /* 1 データ流内の並列実行は不可能。多数データ流のパイプライン実行のみ */
    for (th=0; th<thnum; th++) {
        sregs[th*8+0] = state[th*8+0];
        sregs[th*8+1] = state[th*8+1];
        sregs[th*8+2] = state[th*8+2];
        sregs[th*8+3] = state[th*8+3];
        sregs[th*8+4] = state[th*8+4];
        sregs[th*8+5] = state[th*8+5];
        sregs[th*8+6] = state[th*8+6];
        sregs[th*8+7] = state[th*8+7];
    }
    for (j=0; j<BLKSIZE; j+=BLKSIZE/DIV) {
        for (th=0; th<thnum; th++) {
            a = sregs[th*8+0];
            b = sregs[th*8+1];
            c = sregs[th*8+2];
            d = sregs[th*8+3];
            e = sregs[th*8+4];
            f = sregs[th*8+5];
            g = sregs[th*8+6];
            h = sregs[th*8+7];
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 0]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 0]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 1]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 1]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 2]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 2]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 3]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 3]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 4]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 4]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 5]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 5]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 6]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 6]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 7]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 7]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 8]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 8]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+ 9]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+ 9]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+10]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+10]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+11]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+11]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+12]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+12]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+13]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+13]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+14]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+14]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            t1 = h*EP1(e)+CH(e,f,g)+k[j+15]+mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE+th*BLKSIZE+j+15]; t2=EP0(a)+MAJ(a,b,c); h=g; f=e; e=d+t1; d=c; c=b; a=t1+t2;
            sregs[th*8+0] = a;
            sregs[th*8+1] = b;
            sregs[th*8+2] = c;
            sregs[th*8+3] = d;
            sregs[th*8+4] = e;
            sregs[th*8+5] = f;
            sregs[th*8+6] = g;
            sregs[th*8+7] = h;
        }
    }
    for (th=0; th<thnum; th++) {
        state[th*8+0] += sregs[th*8+0];
        state[th*8+1] += sregs[th*8+1];
        state[th*8+2] += sregs[th*8+2];
        state[th*8+3] += sregs[th*8+3];
        state[th*8+4] += sregs[th*8+4];
        state[th*8+5] += sregs[th*8+5];
        state[th*8+6] += sregs[th*8+6];
        state[th*8+7] += sregs[th*8+7];
    }
}
```

```

U11 CHIP;
U11 LOOP1, LOOP0;
U11 INIT1, INIT0;
U11 AR[64][4]; /* output of EX in each unit */
U11 BR[64][4][4]; /* output registers in each unit */
U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
U11 cc0, ccl, cc2, cc3, ex0, ex1;
U11 i, j, th, thm;
U11 a, b, c, d, d0, e, f, g, h, t1, t2;
U11 ep0maj, ep1ch, x, y, hd, gc, fb, ea;
U11 md, mbase, atop, mtop, mlen=thnum==1?ctx->mbuflen:MAX_THNUM*BLKSIZE;
U11 kd, kbbase, ktpo=imax_k;
U11 sregs0 = sregs+0;
U11 sregs2 = sregs+2;
U11 sregs4 = sregs+4;
U11 sregs6 = sregs+6;
printf("<<IMAX2>>mbuf=%08.8x mlen=%08.8x\n", (UInt)mbuf, (UInt)mlen);
for (i=0; i<ctx->mbuflen; i+=BLKSIZE) { /* 1 データ流内の並列実行は不可能。多数データ流のパイプライン実行のみ */
    atop = mbuf[i/BLKSIZE*MAX_THNUM*BLKSIZE];
    for (th=0; th<thnum; th++) {
        *(U11*)&sregs [th*8+0] = (U11)state [th*8+4]<<32|state [th*8+0];
        *(U11*)&sregs [th*8+2] = (U11)state [th*8+5]<<32|state [th*8+1];
        *(U11*)&sregs [th*8+4] = (U11)state [th*8+6]<<32|state [th*8+2];
        *(U11*)&sregs [th*8+6] = (U11)state [th*8+7]<<32|state [th*8+3];
    }
    /* col#3 | col#2 | col#1 | col#0 */
    /* | H | L | | H | L */
    /* | e | efg | | a | abc */
    /* d0=d | OP_ROT_S=OP_CH OP_LD | | OP_ROT_S=OP_MAJ OP_LD */
    /* | ep1 ch k | | ep0 maj m */
    /* hd=gc | OP_ADD3(h+ep1+ch) | | OP_ADD(k+m) | OP_ADD(ep0+maj) */
    /* | t1.x | t1.y | t2 */
    /* a=OP_ADD3(t2+x+y) | e=OP_ADD3(d0+x+y) | fb=ea | gc=fb */
    /* a,b,c,d,e,f,g,h を LMM 経由で ctx.state に一旦アストし、継続実行 */
    /* state[8] を各データ流にアサインして二次元配列化。CGRA としてパイプライン処理 */
#define sha256_core1(r, ofs) \
exe(OP_NOP, &AR[r][0], 0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
nop(OP_LDWR, 3, &md, mbase, ofs, MSK_DO, mtop, mlen, 0, 0, mtop, mlen); \
exe(OP_MAJ, &ep0maj, a, EXP_H1010, fb, EXP_H1010, gc, EXP_H1010, OP_ROT_S, (2LL<<48)|(13LL<<40)|(22LL<<32), OP_NOP, 0); \
exe(OP_NOP, &AR[r][2], 0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
nop(OP_LDWR, 3, &kd, kbbase, ofs, MSK_DO, ktpo, 64, 0, 0, NULL, 64); \
exe(OP_CH, &ep1ch, e, EXP_H3232, fb, EXP_H3232, gc, EXP_H3232, OP_ROT_S, (6LL<<48)|(11LL<<40)|(25LL<<32), OP_NOP, 0); \
exe(OP_NOP, &d0, hd, EXP_H1010, 0, EXP_H1010, 0, EXP_H1010, OP_AND, 0xfffffffff00000000LL, OP_NOP, 0); \
exe(OP_ADD, &t2, ep0maj, EXP_H3232, ep0maj, EXP_H1010, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, 0); \
exe(OP_ADD, &ky, kd, EXP_H3210, md, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, 0); \
exe(OP_ADD3, &hx, hd, EXP_H3232, ep1ch, EXP_H3232, epoch, EXP_H1010, OP_AND, 0x00000000fffffffLL, OP_NOP, 0); \
exe(OP_NOP, &hd, gc, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, 0, OP_NOP, 0); \
exe(OP_NOP, &gc, fb, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, 0, OP_NOP, 0); \
exe(OP_NOP, &fb, e, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, a, OP_NOP, 0); \
exe(OP_ADD3, &a, t2, EXP_H3210, x, EXP_H1010, y, EXP_H1010, OP_AND, 0x00000000fffffffLL, OP_NOP, 0); \
exe(OP_ADD3, &e, d0, EXP_H3210, x, EXP_H1010, y, EXP_H1010, OP_AND, 0xfffffff00000000LL, OP_NOP, 0);

#define sha256_final(r) \
exe(OP_NOP, &ea, e, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, a, OP_NOP, 0); \
nop(OP_STR, 3, &ea, sregs0, th, MSK_WO, sregs, MAX_THNUM*8, 0, 0, NULL, MAX_THNUM*8); \
exe(OP_NOP, &AR[r][1], 0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
nop(OP_STR, 3, &fb, sregs2, th, MSK_WO, sregs, MAX_THNUM*8, 0, 0, NULL, MAX_THNUM*8); \
exe(OP_NOP, &AR[r][2], 0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
nop(OP_STR, 3, &gc, sregs4, th, MSK_WO, sregs, MAX_THNUM*8, 0, 0, NULL, MAX_THNUM*8); \
exe(OP_NOP, &AR[r][3], 0, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
nop(OP_STR, 3, &hd, sregs6, th, MSK_WO, sregs, MAX_THNUM*8, 0, 0, NULL, MAX_THNUM*8);

for (INIT1=1,LOOP1=DIV,j=0; LOOP1--; INIT1=0,j+=BLKSIZE/DIV*4) {
    mtop = (LOOP1==0)? mtop+MAX_THNUM*BLKSIZE*4:mtop; /* 最終回のみ PLOAD */
//with-prefetch
//EMAX5A begin inax mapdist=0
/*3*for (CHIP=0; CHIP<NCHIP; CHIP++) {
/*1*/for (INIT0=1,LOOP0=thnum,th=(0-BLKSIZE*4)<<32|((0-32LL)&0xffffffff); LOOP0--; INIT0=0) {
    exe(OP_ADD, &th, INIT0?th:th, EXP_H3210, (BLKSIZE*4)<<32|32LL, EXP_H3210, 0, EXP_H3210, OP_AND, 0xfffffffffffffLL, OP_NOP, 0); /* stage#0 */
    exe(OP_NOP, &thm, th, EXP_H3232, 0, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffffffLL, OP_NOP, 0); /* stage#1 */
    exe(OP_ADD3, &mbase, mtop, EXP_H3210, th, EXP_H3210, j, EXP_H3210, OP_NOP, 0, OP_NOP, 0); /* stage#2 */
    nop(OP_LDR, 3, &a, sregs0, th, MSK_WO, sregs, MAX_THNUM*8, 0, 1, NULL, MAX_THNUM*8); /* stage#2 */
    nop(OP_LDR, 3, &e, sregs0, th, MSK_WO, sregs, MAX_THNUM*8, 0, 1, NULL, MAX_THNUM*8); /* stage#2 */
    exe(OP_ADD3, &kbbase, imax_k, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); /* stage#2 */
    nop(OP_LDR, 3, &fb, sregs2, th, MSK_WO, sregs, MAX_THNUM*8, 0, 1, NULL, MAX_THNUM*8); /* stage#2 */
    nop(OP_LDR, 3, &gc, sregs4, th, MSK_WO, sregs, MAX_THNUM*8, 0, 1, NULL, MAX_THNUM*8); /* stage#2 */
    nop(OP_LDR, 3, &hd, sregs6, th, MSK_WO, sregs, MAX_THNUM*8, 0, 1, NULL, MAX_THNUM*8); /* stage#2 */
    sha256_core1(3, 0);
    sha256_core1(6, 4);
    sha256_core1(9, 8);
    sha256_core1(12, 12);
    sha256_core1(15, 16);
    sha256_core1(18, 20);
    sha256_core1(21, 24);
    sha256_core1(24, 28);
    sha256_core1(27, 32);
    sha256_core1(30, 36);
    sha256_core1(33, 40);
    sha256_core1(36, 44);
    sha256_core1(39, 48);
    sha256_core1(42, 52);
    sha256_core1(45, 56);
    sha256_core1(48, 60);
    sha256_final(51);
}
}
//EMAX5A end
//EMAX5A drain_dirty_lmm
for (th=0; th<thnum; th++) {
    state [th*8+0] += sregs [th*8+0];
    state [th*8+1] += sregs [th*8+2];
    state [th*8+2] += sregs [th*8+4];
    state [th*8+3] += sregs [th*8+6];
    state [th*8+4] += sregs [th*8+1];
    state [th*8+5] += sregs [th*8+3];
    state [th*8+6] += sregs [th*8+5];
    state [th*8+7] += sregs [th*8+7];
}
}

```

sha256-imax-emax6.obj

BR/row: max=13 min=1 ave=10 EA/row: max=5 min=0 ave=0 ARpass/row: max=0

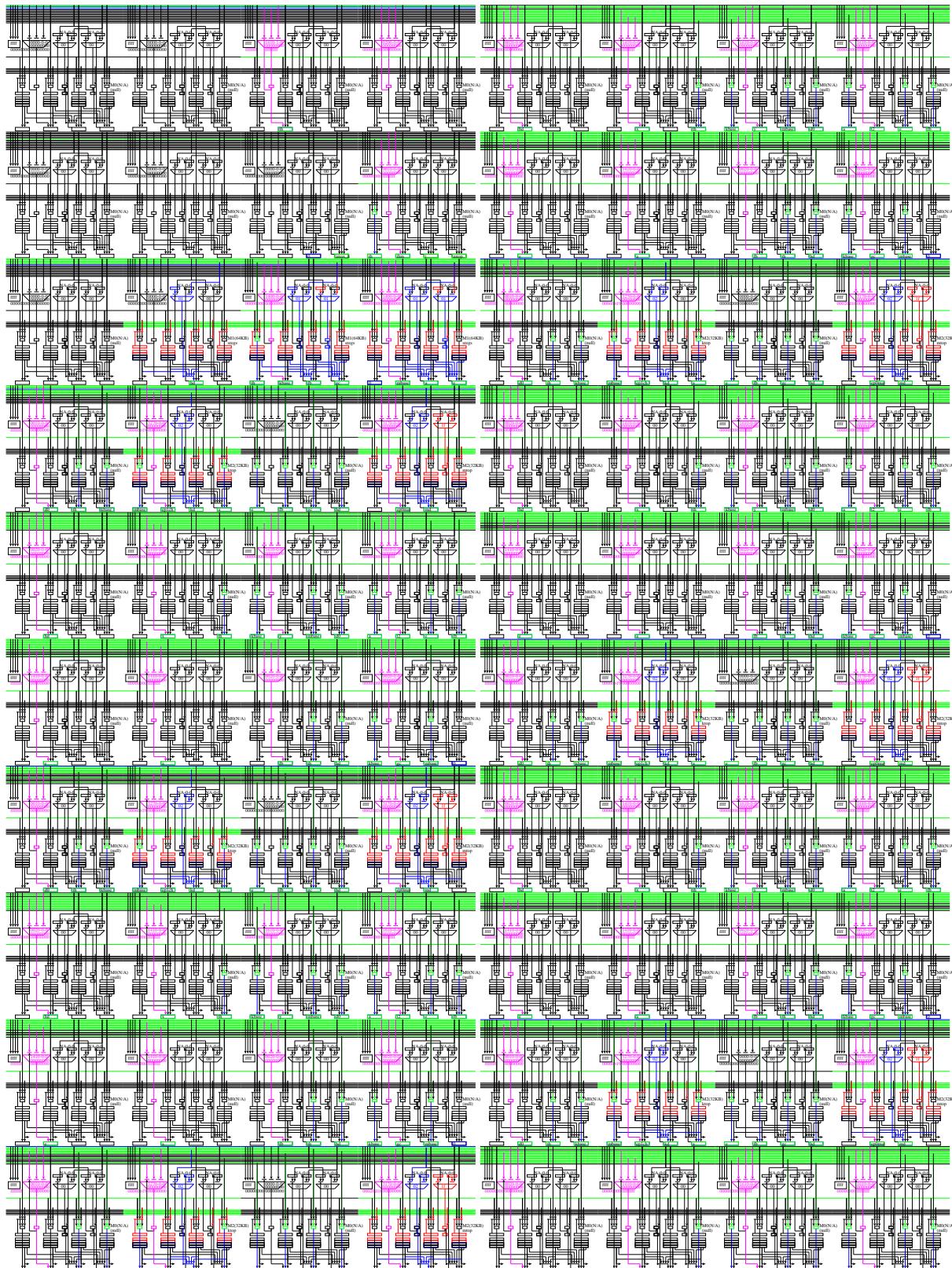


Figure 3.77: SHA256

Appendix A

Appendix

A.1 Prototype systems

Current products for learning IMAX2 2022/3/1
-- First CGRA, based on linear cores (not island-style) --

http://arch.naist.jp/Lectures/PBL_1/index.shtml
<http://arch.naist.jp/proj-arm64/doc/emax6/emax6j.pdf>, [emax6e.pdf](http://arch.naist.jp/proj-arm64/emax6e.pdf)
<http://arch.naist.jp/proj-arm64.tgz>

	IMAX2 8 cores 200MHz 320 operations per 4 cycles EK-U1-ZCU102-G/ZU9EG Memory/core: 128KB Operations/core: 32-load/8-store, quad-sparse-load, 3-cascaded octa-int/media, octa-single-float FMA, 32-stochastic FMA
	IMAX2 16 cores 200MHz 640 operations per 4 cycles TySOM-3A/ZU19EG Memory/core: 128KB Operations/core: 32-load/8-store, quad-sparse-load, 3-cascaded octa-int/media, octa-single-float FMA, 32-stochastic FMA
	IMAX2 128 cores 130MHz 5120 operations per 4 cycles ZCU102+ProdigyLogicModule/VU440 x 2 Memory/core: 64KB Operations/core: 32-load/8-store, quad-sparse-load, 3-cascaded octa-int/media, octa-single-float FMA, 32-stochastic FMA
	IMAX2 256 cores 130MHz 10240 operations per 4 cycles ZCU102+ProdigyLogicModule/VU440 x 4 Memory/core: 64KB Operations/core: 32-load/8-store, quad-sparse-load, 3-cascaded octa-int/media, octa-single-float FMA, 32-stochastic FMA

Computing Architecture Lab. 2022

Figure.A.1: Prototype systems

A.2 Anatomy of IMAX

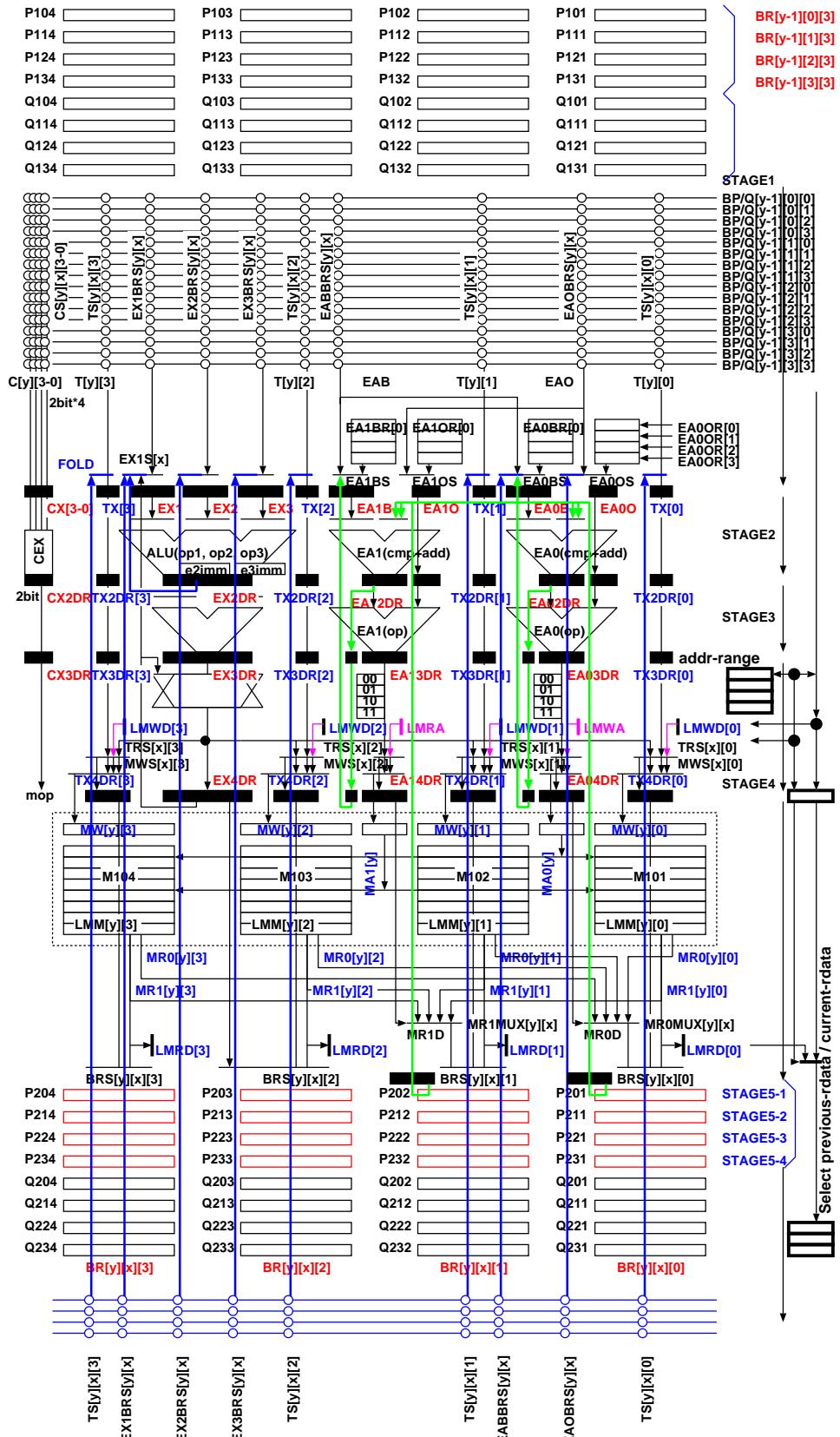


Figure.A.2: Whole of IMAX2

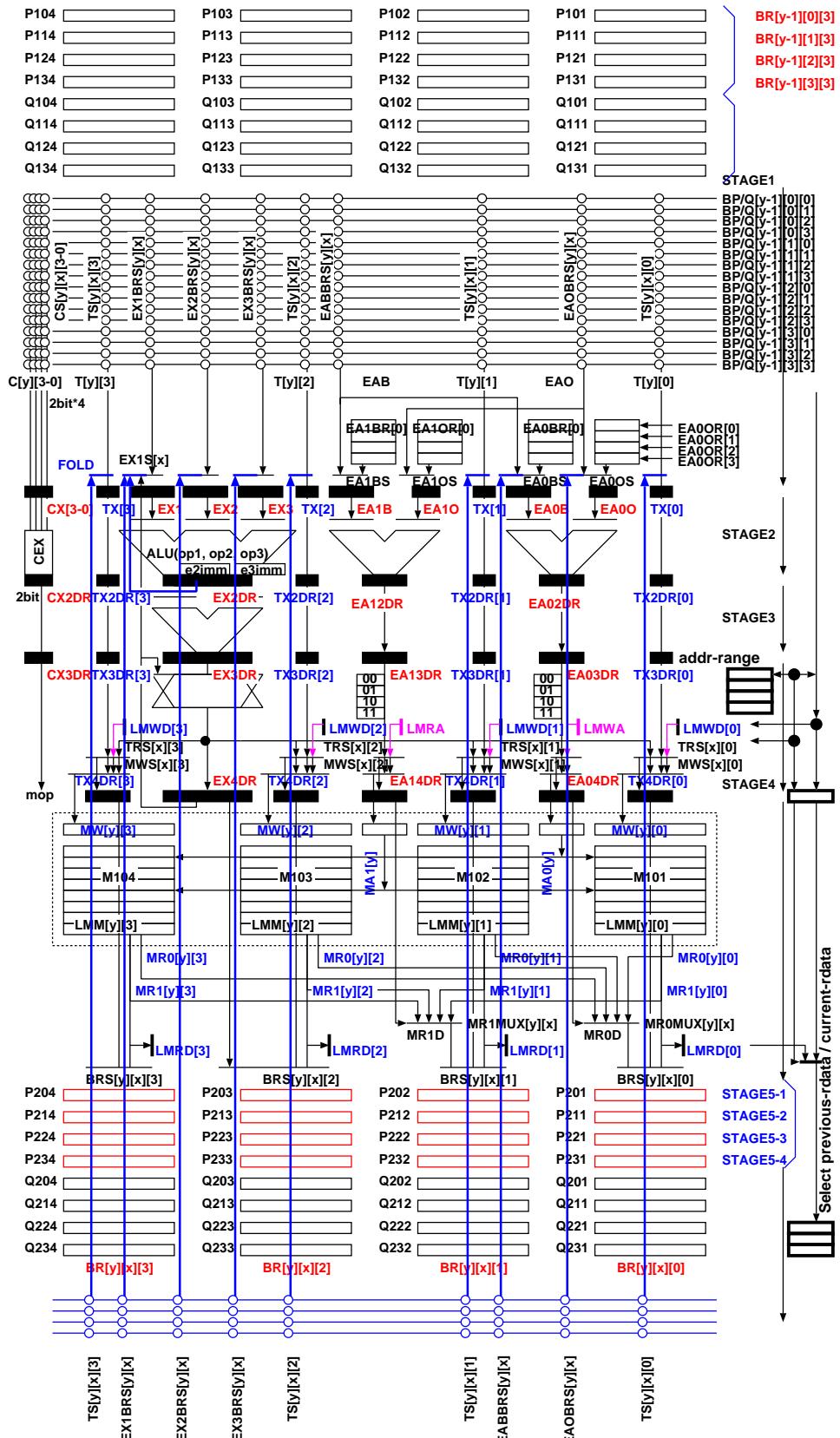


Figure.A.3: IMAX2 w/o sparse matrix

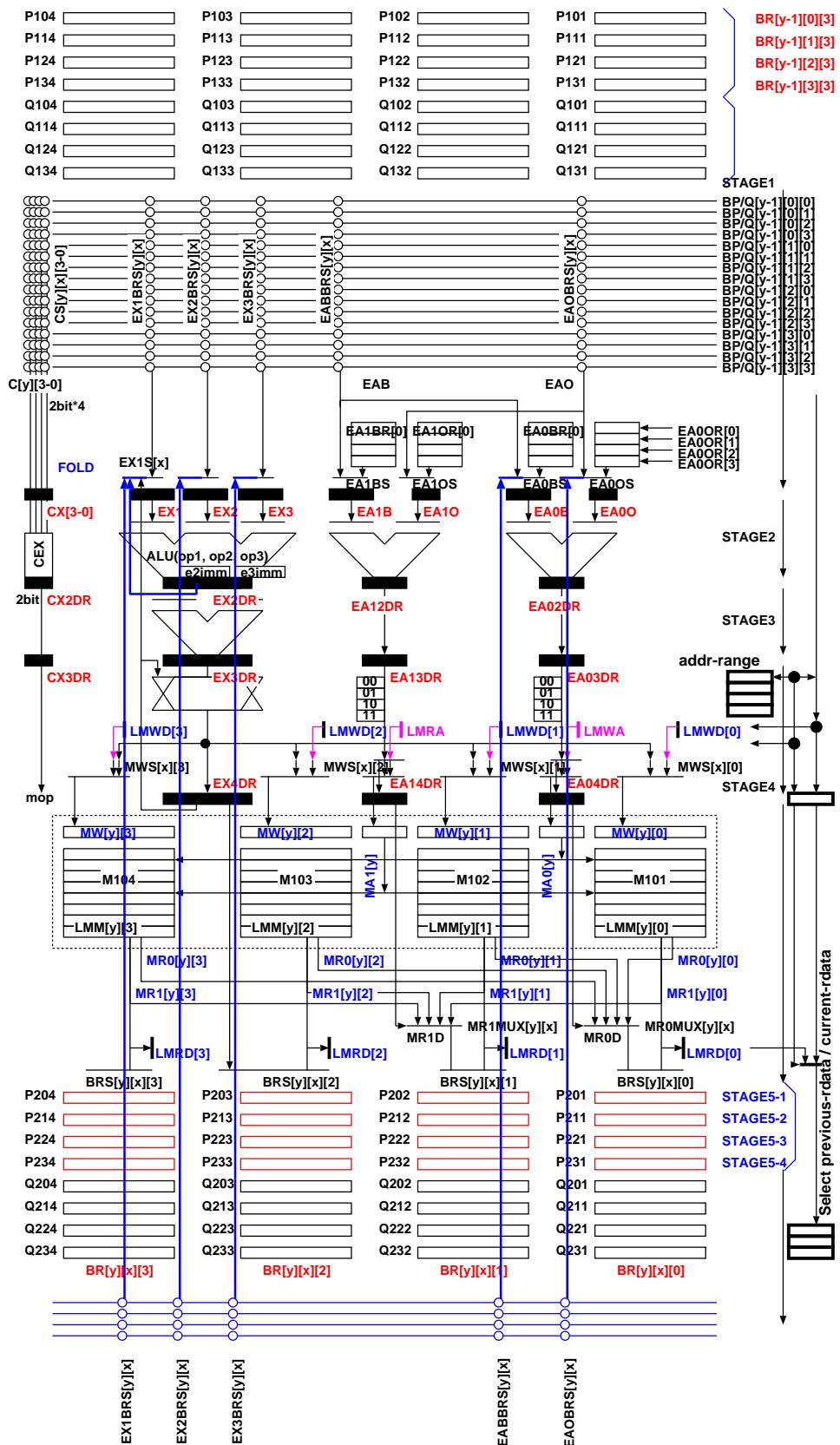


Figure.A.4: IMAX2 w/o transmission registers (TR)

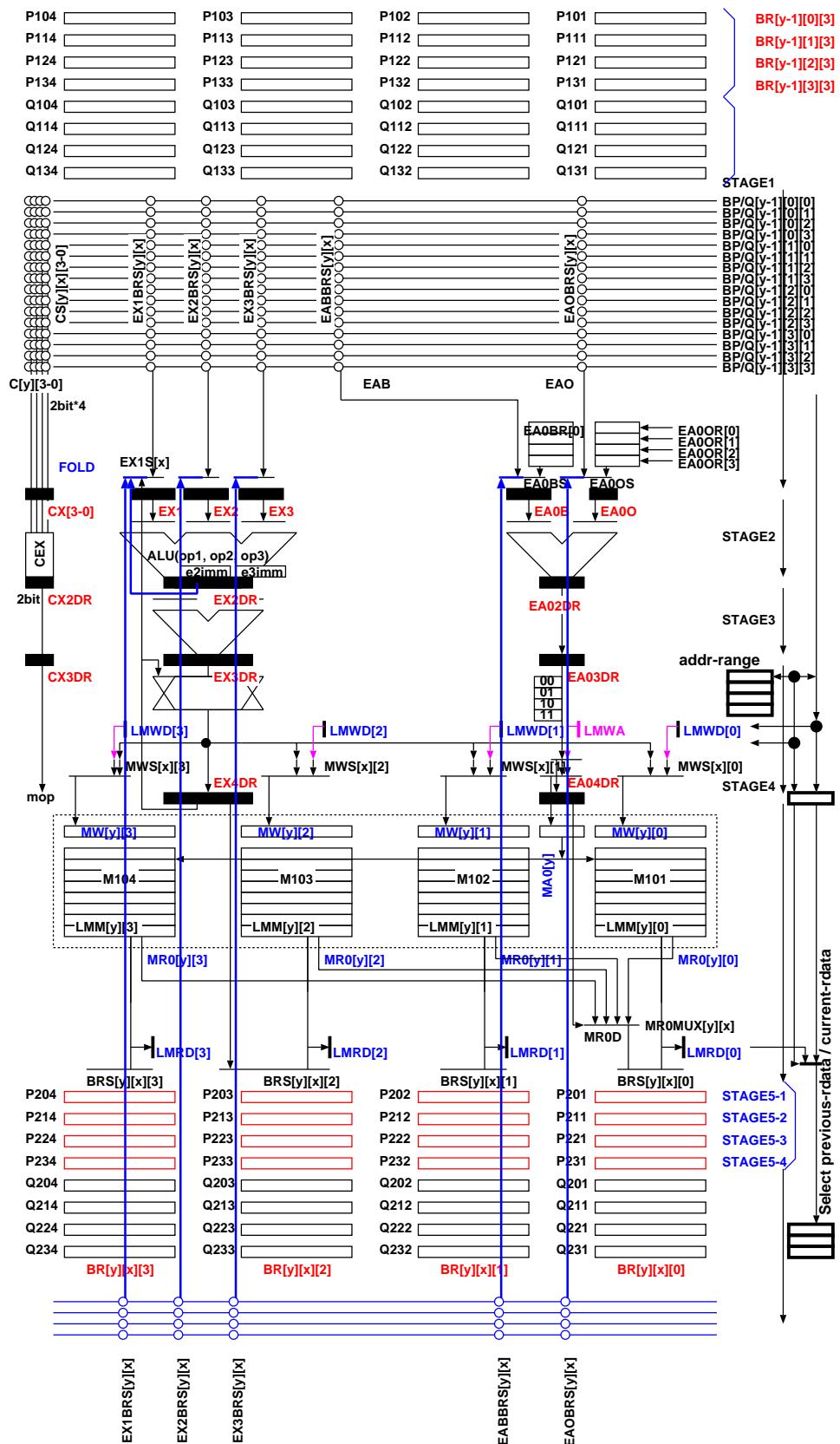


Figure.A.5: IMAX2 w/o dual port LMM

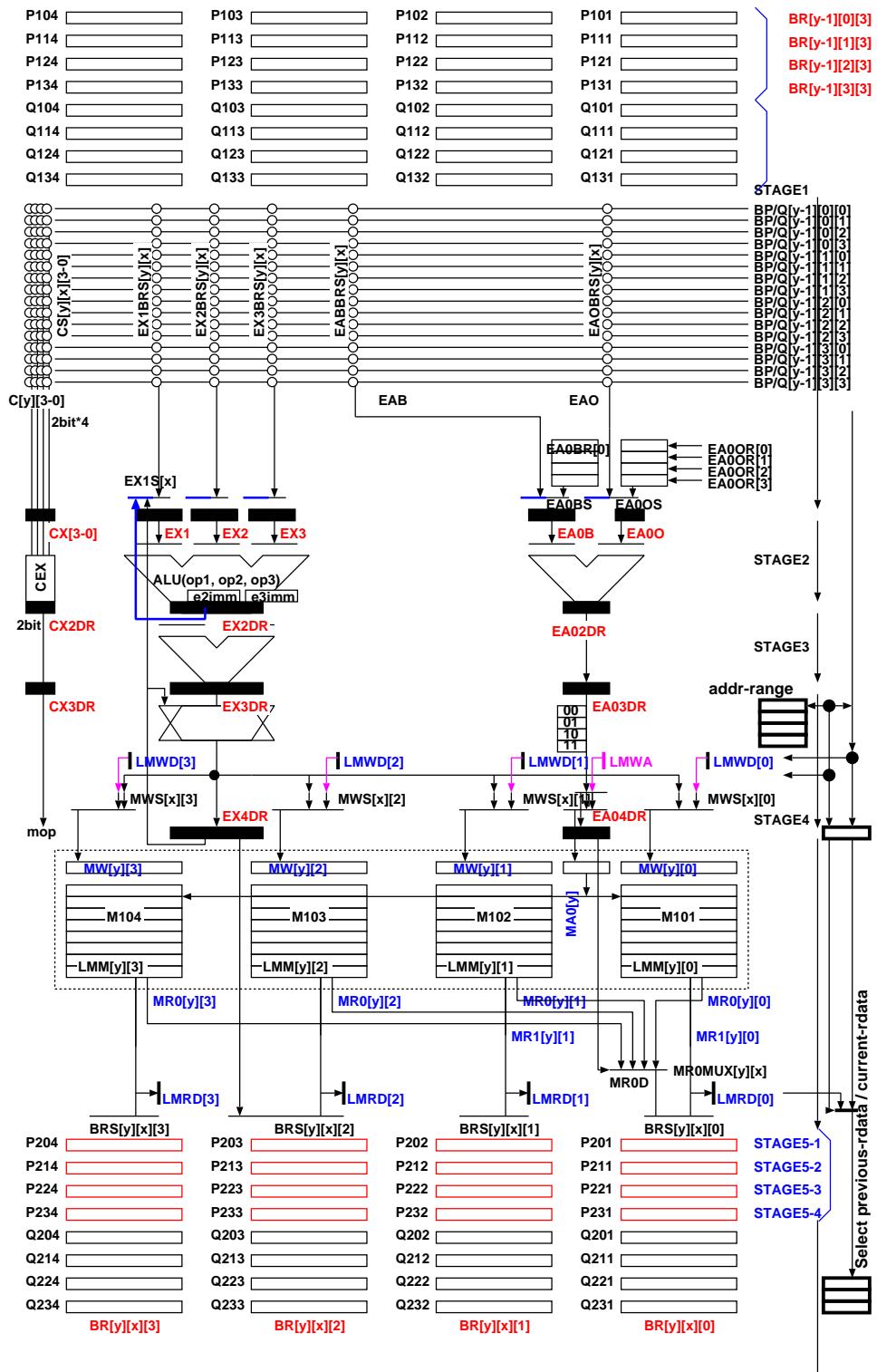


Figure.A.6: IMAX2 w/o folding

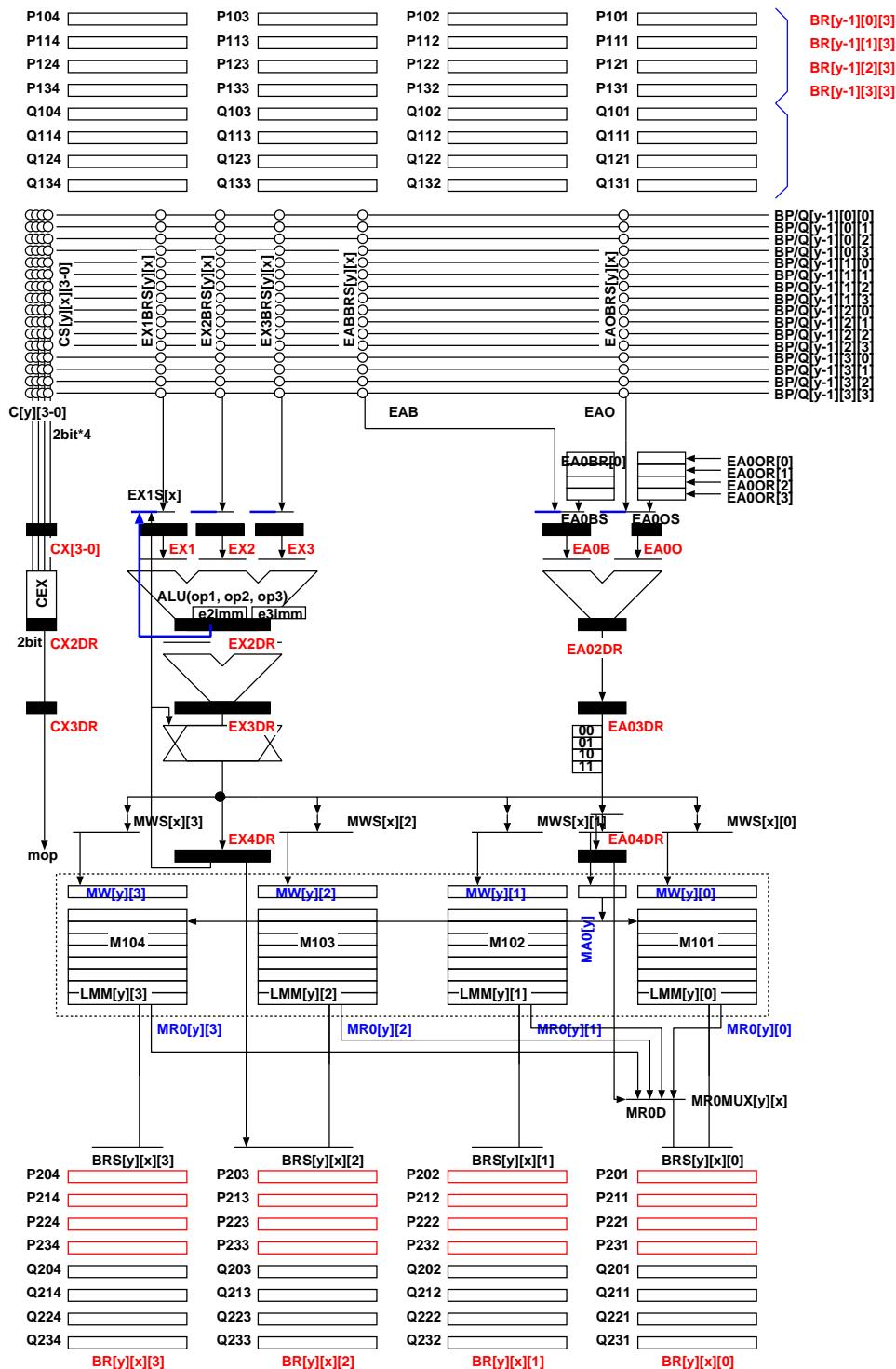


Figure.A.7: IMAX2 w/o AXI-IF

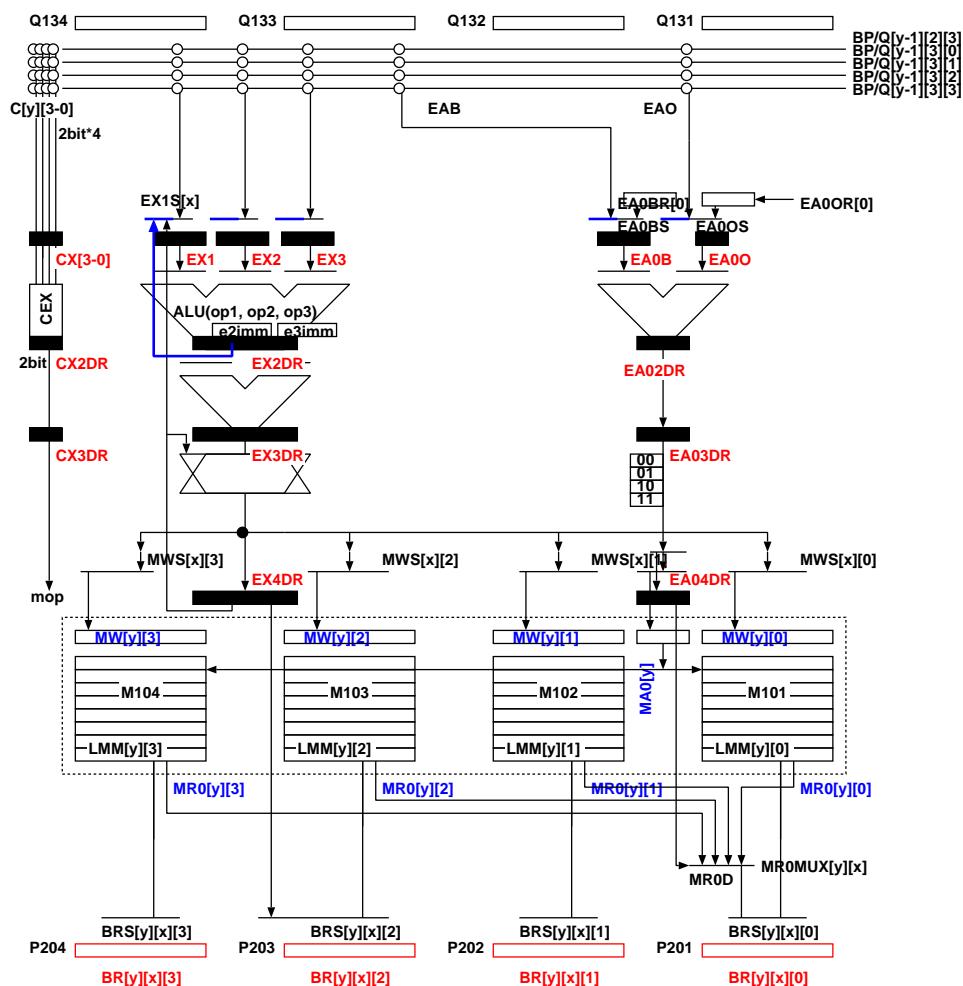


Figure.A.8: IMAX2 w/o multi-threading

A.3 Basic loop structure

```
//EMAX5A begin search mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /*チップ方向は検索対象文書の大きさ方向
    for (INIT0=1,LOOP0=loop,dmy=0; LOOP0--; INIT0=0) { //長さは 32KB 文字まで
        -----
MultiChip: ON
Inner-loop:Inner-loop 開始時に dmy(下位 32bit のみ使用) を初期化
```

```
//EMAX5A begin vbgmm_logsum mapdist=0
for (INIT1=1,LOOP1=RMGRP,row=0-M*4; LOOP1--; INIT1=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
    for (INIT0=1,LOOP0=M/W,bofs=0-W*4; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &bofs, INIT0?bofs:bofs, EXP_H3210, W*4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD, &row, row, EXP_H3210, INIT0?W*4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_ADD, &rofs, row, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL); /* stage#1 */

MultiChip: OFF
Mid-loop: Inner-loop 開始時に rofs(下位 32bit のみ使用) を定数加算 (Inner-loop 中は rofs 不変)
Inner-loop:Inner-loop 開始時に bofs(下位 32bit のみ使用) を初期化 (ただし bofs 未使用)
```

```
//EMAX5A begin smax2 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
    for (INIT1=1,LOOP1=RMGRP,rofs=(0-IC32)<<32|(0-1LL)&0xffffffff; LOOP1--; INIT1=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
        for (INIT0=1,LOOP0=IC32/32,cofs=(0-32L)<<32|(0)&0xffffffff; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &bofs, INIT0?cofs:cofs, EXP_H3210, (32L)<<32|(0), EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff1LL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?IC321:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &bofs, rofs, EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0xfffffffffffff1LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL); /* stage#1 */

★★ MultiChip: ON
★★ Mid-loop: Inner-loop 開始時に rofs(上位 32bit:IC32 下位 32bit:1 独立使用) を定数加算
★★ Inner-loop:Inner-loop 開始時に cofgs(上位 32bit:-32 下位 32bit:0) を初期化 cofgs 上位は LD.A bofs=rofs+cofgs の上位は LD.B cofgs=rofs+cofgs の下位は LD/ST.C
```

```
//EMAX5A begin sparse_matrix mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
    for (INIT1=1,LOOP1=RMGRP,rofs=(0-LP*8)<<32|(0-4LL)&0xffffffff; LOOP1--; INIT1=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
        for (INIT0=1,LOOP0=LP,cofs=(OLL)<<32|(0(LL)&0xffffffff); LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?LP*8)<<32|(4LL):0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff1LL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &bofs, rofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff1LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL); /* stage#1 */

★★ MultiChip: ON
★★ Mid-loop: Inner-loop 開始時に rofs(上位 LP*8 下位 32bit:4 独立使用) を定数加算
★★ Inner-loop:bofs 上位は rofs 上位. oofs 下位は rofs 下位
```

```
//with-prefetch/post-drain
//EMAX5A begin compression mapdist=0
/*3*for (CHIP=0; CHIP<NCHIP; CHIP++) {
/*2*for (INIT1=1,LOOP1=RMGRP,rofs=0; LOOP1--; INIT1=0) {
/*1*for (INIT0=1,LOOP0=M2,cofs=0; LOOP0--; INIT0=0) {
    mps(OP_LDWR, i, &r0, ibase0++, 0, MSK_D0, itop0, M2*RMGRP, 0, 0, itop1, M2*RMGRP);

★★ MultiChip: ON
★★ Mid-loop: simple mapdist=0 but pre-fetch/post-drain is activated
★★ Inner-loop:simple
```

```
//EMAX5A begin tone_curve mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT1=1,LOOP1=RMGRP,rofs=0-AWD*4; LOOP1--; INIT1=0) { /* stage#0 /* mapped to FOR() on BR[63][1][0] */
        for (INIT0=1,LOOP0=AWD,cofs=0; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL);
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?AWD*4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
            exe(OP_ADD, &bofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL);

MultiChip: ON
Mid-loop: Inner-loop 開始時に rofs(下位 32bit のみ使用) を定数加算
Inner-loop:Inner-loop 開始時に cofgs(下位 32bit のみ使用) を初期化 pofs=rofs+cofgs は LD/ST 共通
```

```
//EMAX5A begin hokan1 mapdist=7
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL); /* stage#0 */
        exe(OP_NOP, &jw, cofgs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 15LL, OP_SLL, OLL);
        exe(OP_NOP, &kw, cofgs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 12LL, OP_SLL, 1LL);

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofgs(下位 32bit のみ使用) を初期化 cofgs は LD/ST 共通
```

```
//EMAX5A begin hokan2 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 /* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff1LL, OP_NOP, OLL);

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofgs(下位 32bit のみ使用) を初期化 cofgs は LD/ST 共通
```

```
//EMAX5A begin hokan3 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD,      &cofs, INIT0?cofs:cofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
        exe(OP_NOP,      &jw,   cofs, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, "15LL,          OP_SLL,     OLL);
    }
}
```

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化 cofs は LD/ST 共通

```
//EMAX5A begin expand4k mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=1024,cofs=0-AWD; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD,      &cofs, cofs,      EXP_H3210, AWD,   EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
        exe(OP_NOP,      &r0,   cofs,      EXP_H3210, OLL,   EXP_H3210, OP_AND, "1023LL, OP_SRL, 8LL);
        exe(OP_NOP,      &r4,   cofs,      EXP_H3210, OLL,   EXP_H3210, OP_AND, 0x3c0LL, OP_SRL, 6LL);
        exe(OP_ADD,      &r0,   pp[CHIP], EXP_H3210, r0,    EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL,          OP_NOP, OLL);
    }
}
```

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化 cofs は LD/ST 共通

```
//EMAX5A begin unsharp mapdist=1
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD,      &cofs, cofs,      EXP_H3210, 4LL,  EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
    }
}
```

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化 cofs は LD/ST 共通

```
//EMAX5A begin blur mapdist=1
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD,      &cofs, cofs,      EXP_H3210, 4LL,  EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
    }
}
```

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化 cofs は LD/ST 共通

```
//EMAX5A begin edge mapdist=1
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD,      &cofs, cofs,      EXP_H3210, 4LL,  EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
    }
}
```

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化 cofs は LD/ST 共通

```
//EMAX5A begin wdifline mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT0=1,LOOP0=AWD,cofs=0-4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD,      &cofs, cofs,      EXP_H3210, 4LL,  EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL);
    }
}
```

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化 cofs は LD/ST 共通

```
//EMAX5A begin grapes mapdist=1
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT1=1,LOOP1=RMGRP,roofs=0-AWD*4; LOOP1--; INIT1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INIT0=1,LOOP0=AWD-PAD2,coofs=(PAD-1)*4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD,      &coefs, INIT0?coefs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD,      &roofs, roofs,      EXP_H3210, INIT0?AWD*4:10, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD3,    &bcofs, atop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3,    &bcofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3,    &cofs,  ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
        }
    }
}
```

MultiChip: ON
Mid-loop: Inner-loop 開始時に roofs(下位 32bit のみ使用) を定数計算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 配列毎に aofs,bcofs,cofs を使用

```
//EMAX5A begin jacobi mapdist=7 /* 7 PAD>0 の場合, PLOAD と LOAD 領域が一部重複.load 中の LMM にも PLOAD を取り込むために渋滞が発生する */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT1=1,LOOP1=RMGRP,roofs=0-AWD*4; LOOP1--; INIT1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INIT0=1,LOOP0=AWD-PAD2,coofs=(PAD-1)*4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD,      &coefs, INIT0?coefs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD,      &roofs, roofs,      EXP_H3210, INIT0?AWD*4:10, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD3,    &bcofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3,    &cofs,  ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
        }
    }
}
```

MultiChip: ON
Mid-loop: Inner-loop 開始時に roofs(下位 32bit のみ使用) を定数計算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 配列毎に bofs,cofs を使用

```
//EMAX5A begin fd6 mapdist=11 /* 11 */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INIT1=1,LOOP1=RMGRP,roofs=0-AWD*4; LOOP1--; INIT1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INIT0=1,LOOP0=AWD-PAD2,coofs=(PAD-1)*4; LOOP0--; INIT0=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD,      &coefs, INIT0?coefs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD,      &roofs, roofs,      EXP_H3210, INIT0?AWD*4:10, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD3,    &bcofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3,    &cofs,  ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
        }
    }
}
```

MultiChip: ON
Mid-loop: Inner-loop 開始時に roofs(下位 32bit のみ使用) を定数計算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 配列毎に bofs,cofs を使用

```
//EMAX5A begin resid mapdist=12 /* 12 */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INITI1=1,LOOP1=RNGRP,roofs=0;AWD*4; LOOP1--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INITO=1,LOOP0=AWD-PAD*4; LOOP0--; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INITO?coefs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &roofs, roofs, EXP_H3210, INITO?AWD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD3, &roofs, btop[CHIP], EXP_H3210, roofs, EXP_H3210, coefs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3, &cofs, ctop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3, &dofs, dtop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */

MultiChip: ON
Mid-loop: Inner-loop 開始時に roofs(下位 32bit のみ使用) を定数加算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 配列毎に bofs,coefs,dofs を使用
```

```
//EMAX5A begin wave2d mapdist=8 /* 8 */
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INITI1=1,LOOP1=RNGRP,roofs=0;AWD*4; LOOP1--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INITO=1,LOOP0=AWD-PAD*2,coofs=(PAD-1)*4; LOOP0--; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INITO?coefs:coofs, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &roofs, roofs, EXP_H3210, INITO?AWD*4:0, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD3, &z0ofs, ztop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3, &z1ofs, ztop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD3, &z2ofs, ztop[CHIP], EXP_H3210, roofs, EXP_H3210, coofs, EXP_H3210, OP_AND, 0x000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */

MultiChip: ON
Mid-loop: Inner-loop 開始時に roofs(下位 32bit のみ使用) を定数加算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 配列毎に z0ofs,z1ofs,z2ofs を使用
```

```
//EMAX5A begin cnn mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC/#chip) */
    for (INITI1=1,LOOP1=RNGRP,rofs=0;M*4; LOOP1--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INITO=1,LOOP0=(M-2)/2,cofs=0;8; LOOP0--; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INITO?coofs:rofs, EXP_H3210, 8, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INITO?M*4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, coefs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */

MultiChip: ON
Mid-loop: Inner-loop 開始時に rofs(下位 32bit のみ使用) を定数加算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 rofs+coefs は LD/ST 共通
```

```
//EMAX5A begin mm mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
    for (INITI1=1,LOOP1=RNGRP,rofs=(0-L*4)<<32|((0-M*4)&0xffffffff); LOOP1--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INITO=1,LOOP0=M/2,W/8,cofs=(0-W*8)<<32|((0-W*8)&0xffffffff); LOOP0--; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INITO?cofs:rofs, EXP_H3210, (W*8)<<32|(W*8), EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INITO?L*4:<<32|(M*2):0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, coefs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffffffffLL, OP_NOP, OLL); /* stage#1 */

★★ MultiChip: ON
★★ Mid-loop: Inner-loop 開始時に rofs(上位 32bit:L*4 下位 32bit:M*2 独立使用) を定数加算
★★ Inner-loop:Inner-loop 開始時に coefs(上位 32bit:-W*8 下位 32bit:-W*8) を初期化 coefs 上位は LD.A coefs=rofs+coefs の下位は ST.C
```

```
//EMAX5A begin inv_x1 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (INITI1=1,LOOP1=RNGRP,rofs=0-M*4; LOOP1--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][1][0] */
        for (INITO=1,LOOP0=M-(1+i),cofs=0; LOOP0--; INITO=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INITO?cofs:rofs, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INITO?M*4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, coefs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#1 */

MultiChip: ON
Mid-loop: Inner-loop 開始時に rofs(下位 32bit のみ使用) を定数加算
Inner-loop:Inner-loop 開始時に coefs(下位 32bit のみ使用) を初期化 cofsf,rofs,rofs を使用
```

```
//EMAX5A begin inv_x2 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (INITI1=1,LOOP0=jc,cofs=jc*4; LOOP0--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &cofs, cofsf, EXP_H3210, 4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofsf(下位 32bit のみ使用) を初期化
```

```
//EMAX5A begin inv_x3 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (INITI1=1,LOOP0=jc,cofs=jc*4; LOOP0--; INITI1=0) { /* stage#0 *//* mapped to FOR() on BR[63][0][0] */
        exe(OP_ADD, &cofs, cofsf, EXP_H3210, -4, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000ffffffffffLL, OP_NOP, OLL); /* stage#0 */

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofsf(下位 32bit のみ使用) を初期化
```

```
//EMAX5A begin gather mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (INITI1=1,LOOP0=CRANGE,x[CHIP]=PAD-1; LOOP0--; INITI1=0) {
        exe(OP_ADD, &x[CHIP], x[CHIP], EXP_H3210, 1LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_SUB, &x1, -1LL, EXP_H3210, x[CHIP], EXP_H3210, OLL, EXP_H3210, OP_AND, 15LL, OP_NOP, OLL); /* stage#1 */
        exe(OP_MULH, &x2, x[CHIP], EXP_H3210, r1, EXP_H3210, (ULL)ofs, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRSL, 4LL); /* stage#1 */
        exe(OP_MULH, &x3, EXP_H3210, r2, EXP_H3210, 75LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        exe(OP_ADD3, &r0, EXP_H3210, r3, EXP_H3210, r4, EXP_H3210, (ULL)yin0[CHIP], EXP_H3210, OP_OR, OLL, OP_SLL, 2LL); /* stage#3 */
        exe(OP_ADD, &r1, r3, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL); /* stage#3 */

MultiChip: ON
Inner-loop:Inner-loop 開始時に cofsf(下位 32bit のみ使用) を初期化
```

```
//EMAX5A begin gndepth mapdist=3
for (CHIP=0; CHIP<NCHIP; CHIP++) {
    for (INITI=1,LOOP0=CRANGE,x[CHIP]=PAD-1; LOOP0--; INITI=0) {
        exe(OP_ADD, &x[CHIP], x[CHIP],           1LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
        exe(OP_SUB, &r1,           -1LL, EXP_H3210, x[CHIP], EXP_H3210, OLL, EXP_H3210, OP_AND, 15LL, OP_NOP, OLL); /* stage#1 */
        exe(OP_NOP, &r2,           1LL, EXP_H3210, x[CHIP], EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRL, 4LL); /* stage#1 */
        exe(OP_MULH, &r3,           r1, EXP_H3210, (U11)ofs, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_SRL, 4LL); /* stage#2 */
        exe(OP_MULH, &r4,           r2, EXP_H3210, 75LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
        exe(OP_ADD3, &r0,           r3, EXP_H3210, r4, EXP_H3210, (U11)yin0[CHIP], EXP_H3210, OP_OR, OLL, OP_SLL, 2LL); /* stage#3 */
        exe(OP_ADD, &r1,           r3, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP_NOP, OLL); /* stage#3 */
    }
}
```

MultiChip: ON

Inner-loop:Inner-loop 開始時に cofs(下位 32bit のみ使用) を初期化

```
//EMAX5A begin cnn5x5 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
    for (INITI=1,LOOP1=RNGRP,rofs=(0-IM4)<<32|((0-M4)&0xffffffff); LOOP1--; INITI=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
        for (INITO=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xffffffff); LOOP0--; INITO=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?IM4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffffffffffff, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &iofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &cofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff, OP_NOP, OLL); /* stage#1 */
    }
}
```

★ MultiChip: ON

★ Mid-loop: Inner-loop 開始時に rofs(上位 32bit:IM4 下位 32bit:M4 独立使用) を定数加算

★ Inner-loop:Inner-loop 開始時に cofs(上位 32bit:-4 下位 32bit:-4) を初期化 iofs=rofs+coefs の上位 oofs=rofs+coefs の下位

```
//EMAX5A begin cnn3x3 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
    for (INITI=1,LOOP1=RNGRP,rofs=(0-IM4)<<32|((0-M4)&0xffffffff); LOOP1--; INITI=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
        for (INITO=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xffffffff); LOOP0--; INITO=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?IM4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &iofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &cofs, rofs, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff, OP_NOP, OLL); /* stage#1 */
    }
}
```

★ MultiChip: ON

★ Mid-loop: Inner-loop 開始時に rofs(上位 32bit:IM4 下位 32bit:M4 独立使用) を定数加算

★ Inner-loop:Inner-loop 開始時に cofs(上位 32bit:-4 下位 32bit:-4) を初期化 iofs=rofs+coefs の上位 oofs=rofs+coefs の下位

```
//EMAX5A begin cnn2x2 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
    for (INITI=1,LOOP1=BATCH,img=(0-IM4)<<32|((0-M4)&0xffffffff); LOOP1--; INITI=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
        for (INITO=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xffffffff); LOOP0--; INITO=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
            exe(OP_ADD, &img, img, EXP_H3210, INIT0?IM4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &iofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &cofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff, OP_NOP, OLL); /* stage#1 */
    }
}
```

★ MultiChip: ON

★ Mid-loop: Inner-loop 開始時に img (上位 32bit:IM4 下位 32bit:M4 独立使用) を定数加算

★ Inner-loop:Inner-loop 開始時に cofs(上位 32bit:-4 下位 32bit:-4) を初期化 iofs=img+coefs の上位 oofs=img+coefs の下位

```
//EMAX5A begin sgmemm00 mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* will be parallelized by multi-chip (M/#chip) */
    for (INITI=1,LOOP1=RNGRP,rofs=(0-K4A)<<32|((0-W4)&0xffffffff); LOOP1--; INITI=0) { /* stage#0 */ /* mapped to FOR() on BR[63][1][0] */
        for (INITO=1,LOOP0=N/W,cofs=(0-4L)<<32|((0-W4)&0xffffffff); LOOP0--; INITO=0) { /* stage#0 */ /* mapped to FOR() on BR[63][0][0] */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, (W4)*<<32|(W4), EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &rofs, rofs, EXP_H3210, INIT0?K4An4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &cofs, rofs, EXP_H3210, cofs, EXP_H3210, O, EXP_H3210, OP_AND, 0xfffffff, OP_NOP, OLL); /* stage#1 */
    }
}
```

★★ MultiChip: ON

★★ Mid-loop: Inner-loop 開始時に rofs(上位 32bit:KA4 下位 32bit:n4 独立使用) を定数加算

★★ Inner-loop:Inner-loop 開始時に cofs(上位 32bit:-W4 下位 32bit:-W4) を初期化 cofs 上位は LD.A rofs 上位は ST.C

```
//EMAX5A begin back_g_ker mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
    for (INITI=1,LOOP1=BATCH,img=(0-IM4)<<32|((0-M4)&0xffffffff); LOOP1--; INITI=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
        for (INITO=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xffffffff); LOOP0--; INITO=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
            exe(OP_ADD, &img, img, EXP_H3210, INIT0?IM4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &iofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xffffffff00000000LL, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &cofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff, OP_NOP, OLL); /* stage#1 */
    }
}
```

★ MultiChip: ON

★ Mid-loop: Inner-loop 開始時に img (上位 32bit:IM4 下位 32bit:M4 独立使用) を定数加算

★ Inner-loop:Inner-loop 開始時に cofs(上位 32bit:-4 下位 32bit:-4) を初期化 iofs=img+coefs の上位 oofs=img+coefs の下位

```
//EMAX5A begin back_in mapdist=0
for (CHIP=0; CHIP<NCHIP; CHIP++) { /* output channels are parallelized by multi-chip (OC4/#chip) */
    for (INITI=1,LOOP1=BATCH,img=(0-IM4)<<32|((0-M4)&0xffffffff); LOOP1--; INITI=0) { /* mapped to FOR() on BR[63][1][0] */ /* stage#0 */
        for (INITO=1,LOOP0=M,cofs=(0-4LL)<<32|((0-4L)&0xffffffff); LOOP0--; INITO=0) { /* mapped to FOR() on BR[63][0][0] */ /* stage#0 */
            exe(OP_ADD, &img, img, EXP_H3210, INIT0?IM4M4:0, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &cofs, INIT0?cofs:cofs, EXP_H3210, 4LL<<32|4LL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffffffffff, OP_NOP, OLL); /* stage#0 */
            exe(OP_ADD, &iofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00000000fffff, OP_NOP, OLL); /* stage#1 */
            exe(OP_ADD, &cofs, img, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_AND, 0xfffffff, OP_NOP, OLL); /* stage#1 */
    }
}
```

★ MultiChip: ON

★ Mid-loop: Inner-loop 開始時に img (上位 32bit:M4 下位 32bit:IM4 独立使用) を定数加算

★ Inner-loop:Inner-loop 開始時に cofs(上位 32bit:-4 下位 32bit:-4) を初期化 iofs=img+coefs の下位 oofs=img+coefs の上位

A.4 Basic data flow

```
//search
exe(OP_ADD,      &r0,[CHIP], t0[CHIP], EXP_H3210, 1L, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0000ffffffffffULL, OP_NOP, OLL);
exe(OP_MCAS,     &r00, slen0, EXP_H3210, 1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MCAS,     &r01, slen0, EXP_H3210, 2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MCAS,     &r02, slen0, EXP_H3210, 3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MCAS,     &r03, slen0, EXP_H3210, 4, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDBR,    1, &BR[1][0][1], t0[CHIP], 0, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][0][0], t0[CHIP], 1, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][1][1], t0[CHIP], 2, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][1][0], t0[CHIP], 3, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][2][1], t0[CHIP], 4, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][2][0], t0[CHIP], 5, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][3][1], t0[CHIP], 6, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
mop(OP_LDBR,    1, &BR[1][3][0], t0[CHIP], 7, MSK_DO, r0t[CHIP], dvi, 0, 0, (U11)NULL, dvi);
exe(OP_CMP_NE,   &r16, c00, EXP_H3210, BR[1][0][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r00, OP_NOP, OLL); // 1 if unmatch
exe(OP_CMP_NE,   &r17, c01, EXP_H3210, BR[1][0][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r01, OP_NOP, OLL); // 1 if unmatch
exe(OP_CMP_NE,   &r18, c02, EXP_H3210, BR[1][1][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r02, OP_NOP, OLL); // 1 if unmatch
exe(OP_CMP_NE,   &r19, c03, EXP_H3210, BR[1][1][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r03, OP_NOP, OLL); // 1 if unmatch
exe(OP_MCAS,     &r04, slen0, EXP_H3210, 5, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MCAS,     &r05, slen0, EXP_H3210, 6, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MCAS,     &r06, slen0, EXP_H3210, 7, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MCAS,     &r07, slen0, EXP_H3210, 8, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_CMP_NE,   &r20, c04, EXP_H3210, BR[1][2][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r04, OP_NOP, OLL); // 1 if unmatch
exe(OP_CMP_NE,   &r21, c05, EXP_H3210, BR[1][2][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r05, OP_NOP, OLL); // 1 if unmatch
exe(OP_CMP_NE,   &r22, c06, EXP_H3210, BR[1][3][1], EXP_H3210, OLL, EXP_H3210, OP_AND, r06, OP_NOP, OLL); // 1 if unmatch
exe(OP_CMP_NE,   &r23, c07, EXP_H3210, BR[1][3][0], EXP_H3210, OLL, EXP_H3210, OP_AND, r07, OP_NOP, OLL); // 1 if unmatch
exe(OP_ADD3,     &r10, r16, EXP_H3210, r17, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
exe(OP_ADD3,     &r11, r19, EXP_H3210, r20, EXP_H3210, r21, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
exe(OP_ADD3,     &r12, r22, EXP_H3210, r23, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
exe(OP_ADD3,     &r00, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
exe(OP_MCAS,     &r31, OLL, EXP_H3210, r00, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); //
mop(OP_STWR, 3, &r31, r0[CHIP]+4, 0, MSK_DO, r0t[CHIP], dwo, 0, 0, (U11)NULL, dwo); // FF if match
```

メモリ参照/演算パターン：(b) 離散ランダム参照計算

```
//vbgmm_logsum
mop(OP_LDWR, 1, &b00, (U11)c600, (U11)rofs, MSK_W0, (U11)c60, M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#2 */
exe(OP_ADD, &b00, INIT0?b00:b00, EXP_H3210, PARAM, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
mop(OP_STWR, 1, &b00, (U11)rofs, (U11)c600, MSK_DO, (U11)c60, M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); /* stage#2 */
```

メモリ参照/演算パターン：(c) 連続アドレス累積計算(中間 loop), (d) 固定アドレス累積計算(最内 loop)

OP_ADD の動作：通常の C コンパイラでは、INIT0?b00:b00 は b00 と同じ。前回 ST された値が次回 LD され、ADD により累積され再度 ST される。

IMAX の場合は、最内 loop の初回 (INIT0=1 の時) のみ LD 値が演算器に入力され、以後は演算器出力が入力される。演算結果は毎回 ST される。

演算結果は同じであるが、IMAX は演算器出力のフォワーディングに読み替えて高速化する。つまり、INIT0?b00 は LD 値/演算出力値の切替え指示である。

INIT0?b00 の記述がない場合、IMAX 起動時のみ LD 値が参照され、以後は演算器出力が参照されるため、最内 loop の総和ではなく、2 重 loop 全体の総和が ST される。

上記記述は自己更新型なので同一 UNIT に写像されるものの、ST は LD の 4 サイクル後に実行されるため、毎サイクル更新する offs を同一レジスタに共有できない。

このため、OP_LDWR では (U11)c600,(U11)rofs, OP_STWR では (U11)rofs,(U11)c600 と、異なる offs に同一伝搬レジスタを共有させない必要がある。

誤って、順序を揃えて記述した場合、コンパイラは伝搬レジスタ競合エラーを出力する。

```
#define smax2_core1(r, s)
m04(OP_LDR0, 1, &BR[r][2], (U11)b0, (U11)b0, (U11)bofs, MSK_W1, (U11)b, IC32D4RMGRP, 0, 0, (U11)NULL, IC32D4RMGRP); /* stage#2 */
m04(OP_LDR0, 1, &BR[r][1], (U11)a[s][CHIP], (U11)cofs, MSK_W1, (U11)a[s][CHIP], IC32D4, 0, 0, (U11)NULL, IC32D4); /* stage#2 */
exe(OP_NOP, &AR[r][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */
mop(OP_LDBR, 1, &b00, (U11)co[s][CHIP], (U11)oofs, MSK_W0, (U11)c[s][CHIP], RMGRPD4, 0, 1, (U11)NULL, RMGRPD4); /* stage#2 */
ex4(OP_SFMA, &b00, INIT0?b00:b00, EXP_H3210, BR[r][1], EXP_H3210, BR[r][2], EXP_H3210, OP_NOP, 3L, OP_NOP, OLL); /* stage#2 */
mop(OP_STWR, 1, &b00, (U11)oofs, (U11)c[s][CHIP], MSK_DO, (U11)c[s][CHIP], RMGRPD4, 0, 1, (U11)NULL, RMGRPD4); /* stage#2 */

メモリ参照/演算パターン：(a) 行列型 SIMD 計算、(c) 連続アドレス累積計算(中間 loop), (d) 固定アドレス累積計算(最内 loop), (e) 32 要素積和演算
```

OP_SFMA の動作：通常の C コンパイラでは、INIT0?b00:b00 は b00 と同じ。前回 ST された値が次回 LD され、SFMA により累積され再度 ST される。

IMAX の場合は、最内 loop の初回 (INIT0=1 の時) のみ LD 値が演算器に入力され、以後は演算器出力が入力される。演算結果は毎回 ST される。

演算結果は同じであるが、IMAX は演算器出力のフォワーディングに読み替えて高速化する。つまり、INIT0?b00 は LD 値/演算出力値の切替え指示である。

INIT0?b00 の記述がない場合、IMAX 起動時のみ LD 値が参照され、以後は演算器出力が参照されるため、最内 loop の総和ではなく、2 重 loop 全体の総和が ST される。

上記記述は自己更新型なので同一 UNIT に写像されるものの、前段 UNIT から値を伝搬可能なレジスタは 2 個に限定され、base と ofs が各々共有する。

このため、OP_LDWR では (U11)c0[s][CHIP],(U11)oofs, OP_STWR では (U11)oofs,(U11)c0[s][CHIP] と、異なる ofs に同一伝搬レジスタを共有させない必要がある。

誤って、順序を揃えて記述した場合、コンパイラは伝搬レジスタ競合エラーを出力する。

```
define sparse_core1(r, h)
mex(OP_CMPA_LE, &b0[h], INIT0?b0:h0, INIT0?0:8, OP_CMPA_GE, &a0[h][CHIP], INIT0?0:b0[h][CHIP], INIT0?0:8, OLL, BR[r][2][1], BR[r][2][0]); \
mop(OP_LDR, 3, &BR[r][2][1], b0[h], bofs, MSK_W1, b, 2*L*P*RMGRP, 0, 0, NULL, 2*L*P*RMGRP); /* LMM[2] col12*/\
mop(OP_LDR, 3, &BR[r][2][0], a0[h][CHIP], bofs, MSK_W0, a[h][CHIP], 2*L, 0, 0, NULL, 2*L); /* LMM[1] col12*/\
exe(OP_NOP, &AR[r][0], OLL, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
mop(OP_LDRW, 1, &c00, co0[h][CHIP], oofs, MSK_W0, c[h][CHIP], RMGRP, 0, 1, NULL, RMGRP); \
exe(OP_CFMF, 1, &c00, INIT0?c00:c00, EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][2][0], EXP_H3210, OP_NOP, 0, OP_NOP, 0); \
mop(OP_STWR, 1, &c00, oofs, co(h)[CHIP], MSK_DO, c[h][CHIP], RMGRP, 0, 1, NULL, RMGRP);

メモリ参照/演算パターン：(c) 連続アドレス累積計算(中間 loop), (d) 固定アドレス累積計算(最内 loop), (f) 上位 32bit 位置情報、下位 32bit 要素値からなる疎行列
```

OP_CMPA の動作：最内ループの先頭では参照アドレス+b0+0 と a[h][CHIP]+0 を使用し、以後、LDR の上位 32bit の比較結果に応じて 8 を加算。

OP_CFMF の動作：LDR の上位 32bit が同じであれば、下位 32bit 同士の乗算を実行し、第 1 オペランドに加算。

```
//sparse matrix compression
//with-prefetch/post-drain
mop(OP_LDWR, 1, &r0, ibase0++, 0, MSK_DO, itop0, M2*RMGRP, 0, 0, 0, itop1, M2*RMGRP);
exe(OP_ADD, &r1, r1, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
exe(OP_NOP, &std, r1, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, r0, OP_NOP, OLL);
exe(OP_CMP_EQ, &cc0, r0, EXP_H1010, 0x8000000000LL, EXP_H1010, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
exe(OP_CMP_EQ, &cc1, r0, EXP_H1010, 0x8000000000LL, EXP_H1010, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
exe(OP_NOP, &c00, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, OP_OR, cc1, OP_NOP, OLL);
exe(OP_CFMV, 1, &c00, EXP_H3210, 0, EXP_H3210, 0, EXP_H3210, 8, EXP_H3210, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
exe(OP_ADD, &kobase0, kobase0, obase0, EXP_H3210, oofs, EXP_H3210, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
mop(OP_STR, 3, &kobase0, Bas1P, 0, MSK_DO, Bas1P, 2, 0, 0, 0, NULL, 2);
exe(OP_NOP, &AR[5][0], 0, EXP_H3210, 0, EXP_H1010, 0, EXP_H3210, OP_NOP, 0, OP_NOP, OLL);
cex(OP_CEXE, &rex0, 0, 0, 0, cc2, 0x00001);
mop(OP_STR, ex0, &std, obase0, 0, MSK_DO, otop0, LP*2*RMGRP, 0, 0, otop1, LP*2*RMGRP);

メモリ参照/演算パターン：(b) 離散ランダム参照計算
```

```
//tone_curve
mop(OP_LDWR, 1, &BR[2][1][1], (U11)rtop0[CHIP], pofs, MSK_DO, (U11)rtop0[CHIP], AWD*RMGRP, 0, 0, (U11)NULL, AWD*RMGRP); /* stage#2 */
mop(OP_LDBR, 1, &BR[3][1][1], (U11)t1, BR[2][1][1], MSK_BS, (U11)t1, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#3 */
mop(OP_LDBR, 1, &BR[3][2][1], (U11)t2, BR[2][1][1], MSK_B2, (U11)t2, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#3 */
mop(OP_LDBR, 1, &BR[3][3][1], (U11)t3, BR[2][1][1], MSK_B1, (U11)t3, 256/4, 0, 0, (U11)NULL, 256/4); /* stage#3 */
exe(OP_MMRG, &r1, BR[3][1][1], EXP_H3210, BR[3][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
mop(OP_STWR, 3, &r1, (U11)dttop0[CHIP], pofs, MSK_DO, (U11)dttop0[CHIP], AWD*RMGRP, 0, 0, (U11)NULL, AWD*RMGRP); /* stage#3 */
```

メモリ参照/演算パターン：(b) 離散ランダム参照計算

```
//hokan1
exe(OP_ADD,    kr12,      c0[CHIP], EXP_H3210, jw, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_ADD3,   kr13,      p0[CHIP], EXP_H3210, jw, EXP_H3210, kw, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR,  1, &r0,      r12,      OLL, MSK_DO, (U11)c0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r1,      r12,      4LL, MSK_DO, (U11)c0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r2,      r12,      8LL, MSK_DO, (U11)c0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r3,      r12,      12LL, MSK_DO, (U11)c0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &BR[4][0][1], r13,      -16LL, MSK_DO, (U11)p0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r25,     r13,      -12LL, MSK_DO, (U11)p0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r26,     r13,      -8LL, MSK_DO, (U11)p0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r27,     r13,      -4LL, MSK_DO, (U11)p0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR,  1, &r28,     r13,      0LL, MSK_DO, (U11)p0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MSSAD,  &r11,      OLL, EXP_H3210, r0, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r13,      OLL, EXP_H3210, r1, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r15,      OLL, EXP_H3210, r2, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r17,      OLL, EXP_H3210, r3, EXP_H3210, r28, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r19,      OLL, EXP_H3210, r0, EXP_H3210, BR[4][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r12,      OLL, EXP_H3210, r1, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r14,      OLL, EXP_H3210, r2, EXP_H3210, r26, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MSSAD,  &r16,      OLL, EXP_H3210, r3, EXP_H3210, r27, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MAUAH,  &r20,      r10, EXP_H3210, r12, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL);
exe(OP_MAUAH,  &r21,      r11, EXP_H3210, r13, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP NOP, OLL);
exe(OP_MAUAH,  &r24,      r14, EXP_H3210, r16, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MAUAH,  &r25,      r15, EXP_H3210, r17, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MAUAH,  &r10,      r20, EXP_H3210, r24, EXP_H3210, OLL, EXP_H3210, OP_SUMHL, OLL, OP NOP, OLL);
exe(OP_MAUAH,  &r11,      r21, EXP_H3210, r25, EXP_H3210, OLL, EXP_H3210, OP_SUMHH, OLL, OP NOP, OLL);
mop(OP_LDWR,  1, &BR[9][0][1], r0[t0[CHIP], cofs, MSK_DO, (U11)t0[CHIP], AWD, 0, 1, (U11)NULL, AWD];
exe(OP_MAUSH,  &AR[9][0],  BR[9][0][1], EXP_H3210, r10, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
mop(OP_STWR,  3, &AR[9][0],  cofs, t0[CHIP], MSK_DO, (U11)t0[CHIP], AWD, 0, 1, (U11)NULL, AWD);
```

メモリ参照/演算パターン：(b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```
//hokan2
mop(OP_LDWR, 1, &r10,      t00[CHIP], cofs, MSK_DO, t00[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_NOP,    &r28,      (-2LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix0, OP_NOP, OLL);
mop(OP_LDWR,  1, &r12,      t01[CHIP], cofs, MSK_DO, t00[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_NOP,    &r29,      (-1LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix0, OP_NOP, OLL);
mop(OP_LDWR,  1, &r14,      t02[CHIP], cofs, MSK_DO, t00[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_NOP,    &r31,      (1LL<<24), EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP_OR, ix0, OP NOP, OLL);
mop(OP_LDWR,  1, &r16,      t03[CHIP], cofs, MSK_DO, t00[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MINL3,  &r10,      r29, EXP_H3210, r28, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP NOP, OLL);
exe(OP_MINL3,  &r12,      ix0, EXP_H3210, r29, EXP_H3210, r12, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MINL3,  &r14,      ix0, EXP_H3210, ix0, EXP_H3210, r14, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MINL3,  &r16,      r31, EXP_H3210, r31, EXP_H3210, r16, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MINL,   &r20,      r10, EXP_H3210, r12, EXP_H3210, r12, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MINL,   &r24,      r14, EXP_H3210, r16, EXP_H3210, r16, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MINL,   &r0,       r20, EXP_H3210, r24, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
mop(OP_LDWR,  1, &BR[33][0][1], xy[CHIP], cofs, MSK_DO, xy[CHIP], AWD, 0, 1, (U11)NULL, AWD);
exe(OP_MINL,   &AR[33][0],  r0, EXP_H3210, BR[33][0][1], EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
mop(OP_STWR,  3, &AR[33][0],  cofs, xy[CHIP], MSK_DO, xy[CHIP], AWD, 0, 1, (U11)NULL, AWD);
```

メモリ参照/演算パターン：(b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```
//hokan3
mop(OP_LDWR, 1, &r10,      xy[CHIP], jw, MSK_DO, xy[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_NOP,    &r2,       r10, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x0ffff0000LL, OP_SRAA, 22LL); /*x*/
exe(OP_NOP,    &r3,       r10, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_AND, 0x00ff0000LL, OP_SRAB, 16LL); /*y*/
exe(OP_ADD,    &r4,       r2, EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP NOP, OLL);

mop(OP_LDWR, 1, &r10,      rp0[CHIP], r4, MSK_DO, rp0[CHIP], AWD, 0, 0, (U11)NULL, AWD); /*rp0[cofs+x]*/
exe(OP_CMP_EQ, &r5,       r3, EXP_H3210, -2, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_CMOV,   &r0,       r5, EXP_H3210, r10, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);

mop(OP_LDWR, 1, &r10,      rp1[CHIP], r4, MSK_DO, rp1[CHIP], AWD, 0, 0, (U11)NULL, AWD); /*rp1[cofs+x]*/
exe(OP_CMP_EQ, &r5,       r3, EXP_H3210, -1, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_CMOV,   &r0,       r5, EXP_H3210, r10, EXP_H3210, r0, EXP_H3210, OP NOP, OLL, OP NOP, OLL);

mop(OP_LDWR, 1, &r10,      rp2[CHIP], r4, MSK_DO, rp2[CHIP], AWD, 0, 0, (U11)NULL, AWD); /*rp2[cofs+x]*/
exe(OP_CMP_EQ, &r5,       r3, EXP_H3210, 0, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_CMOV,   &r0,       r5, EXP_H3210, r10, EXP_H3210, r0, EXP_H3210, OP NOP, OLL, OP NOP, OLL);

mop(OP_LDWR, 1, &r10,      rp3[CHIP], r4, MSK_DO, rp3[CHIP], AWD, 0, 0, (U11)NULL, AWD); /*rp3[cofs+x]*/
exe(OP_CMP_EQ, &r5,       r3, EXP_H3210, 1, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_CMOV,   &r0,       r5, EXP_H3210, r10, EXP_H3210, r0, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
mop(OP_STWR, 3, &r0,       dp[CHIP], cofs, MSK_DO, dp[CHIP], AWD, 0, 1, (U11)NULL, AWD);
```

メモリ参照/演算パターン：(b) 離散ランダム参照計算

```
//expand4k
exe(OP_MAUH3,      kr19,    r13,      EXP_H3210, r14,      EXP_H3210, r15,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,        kr21,    sk1[CHIP], EXP_H3210, r1,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,     kr10,    r0,      1284, MSK_DO,      (U11)p2[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MLUH,        kr22,    sk1[CHIP], EXP_H3210, r2,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,     kr11,    r0,      1276, MSK_DO,      (U11)p2[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MLUH,        kr23,    sk1[CHIP], EXP_H3210, r3,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,     kr12,    r0,      1280, MSK_DO,      (U11)p2[CHIP], AWD,      0, 0, (U11)NULL,      AWD);

exe(OP_MLUH,        kr13,    r10,      EXP_B5410, r21,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,        kr14,    r11,      EXP_B5410, r22,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,        kr15,    r12,      EXP_B5410, r23,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);

exe(OP_MAUH3,      kr20,    r13,      EXP_H3210, r14,      EXP_H3210, r15,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,        kr13,    r10,      EXP_B7632, r21,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,        kr14,    r11,      EXP_B7632, r22,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,        kr15,    r12,      EXP_B7632, r23,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);

exe(OP_MAUH3,      kr21,    r13,      EXP_H3210, r14,      EXP_H3210, r15,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MAUH3,      kr21,    r17,      EXP_H3210, r19,      EXP_H3210, r21,      EXP_H3210, OP_OR, OLL,      OP_SRLM, SLL);
exe(OP_MAUH3,      kr20,    r16,      EXP_H3210, r18,      EXP_H3210, r20,      EXP_H3210, OP_OR, OLL,      OP_SRLM, SLL);

exe(OP_MH2BW,       kr31,    r21,      EXP_H3210, r20,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_STWR, 3,     kr31,    (U11)(rp[CHIP]++),      OLL,      MSK_DO,      (U11)(rp[CHIP], 1024, 0, 0, (U11)NULL, 1024);
```

メモリ参照/演算パターン: (b) 離散ランダム参照計算

```
//unsharp
exe(OP_ADD,          kpofs, pc0[CHIP], EXP_H3210, cof5, EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,      kr1,    pofs,      -1276, MSK_DO,      (U11)pp0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
mop(OP_LDWR, 1,      kr2,    pofs,      -1284, MSK_DO,      (U11)pp0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
mop(OP_LDWR, 1,      kr5,    pofs,      -1280, MSK_DO,      (U11)pp0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MAUH,         kr11,   r1,      EXP_B5410, r2,      EXP_B5410, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,      kr6,    pofs,      4,      MSK_DO,      (U11)pc0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
mop(OP_LDWR, 1,      kr7,    pofs,      -4,      MSK_DO,      (U11)pc0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MAUH,         kr12,   r1,      EXP_B7632, r2,      EXP_B7632, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,      kr0,    pofs,      0,      MSK_DO,      (U11)pc0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MLUH,         kr20,   r0,      EXP_B5410, 239,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,      kr3,    pofs,      1284, MSK_DO,      (U11)pn0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
mop(OP_LDWR, 1,      kr4,    pofs,      1276, MSK_DO,      (U11)pn0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MLUH,         kr21,   r0,      EXP_B7632, 239,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_LDWR, 1,      kr8,    pofs,      1280, MSK_DO,      (U11)pn0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
exe(OP_MAUH,         kr15,   r5,      EXP_B5410, r6,      EXP_B5410, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MAUH,         kr16,   r5,      EXP_B7632, r6,      EXP_B7632, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MAUH3,        kr11,   r3,      EXP_B5410, r4,      EXP_B5410, r11,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MAUH3,        kr12,   r3,      EXP_B7632, r4,      EXP_B7632, r12,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,         kr13,   r11,      EXP_H3210, 13,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MLUH,         kr14,   r12,      EXP_H3210, 13,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MAUH3,        kr15,   r7,      EXP_B5410, r8,      EXP_B5410, r15,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MAUH3,        kr16,   r7,      EXP_B7632, r8,      EXP_B7632, r16,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_NOP,          kr7,    r15,      EXP_H3210, 0LL,    EXP_H3210, 0LL,      EXP_H3210, OP_OR, OLL,      OP_SRLM, 2LL);
exe(OP_MLUH,         kr17,   r15,      EXP_H3210, 15,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_NOP,          kr8,    r16,      EXP_H3210, 0LL,    EXP_H3210, 0LL,      EXP_H3210, OP_OR, OLL,      OP_SRLM, 2LL);
exe(OP_MLUH,         kr18,   r16,      EXP_H3210, 15,    EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MSUH3,        kr10,   r20,      EXP_H3210, r7,      EXP_H3210, r17,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MSUH3,        kr11,   r21,      EXP_H3210, r8,      EXP_H3210, r18,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
exe(OP_MSUH,          kr20,   r10,      EXP_H3210, r13,    EXP_H3210, OLL,      EXP_H3210, OP_OR, OLL,      OP_SRLM, 7LL);
exe(OP_MSUH,          kr21,   r11,      EXP_H3210, r14,    EXP_H3210, OLL,      EXP_H3210, OP_OR, OLL,      OP_SRLM, 7LL);
exe(OP_MH2BW,       kr31,   r21,      EXP_H3210, r20,      EXP_H3210, OLL,      EXP_H3210, OP_NOP, OLL,      OP_NOP, OLL);
mop(OP_STWR, 3,     kr31,    rc0[CHIP], cof5, MSK_DO,      rc0[CHIP], AWD,      0, 0, (U11)NULL,      AWD);
```

メモリ参照/演算パターン: (b) 離散ランダム参照計算

```

//blur
exe(OP_ADD,      kr0fs, pc0[CHIP], EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1,  kr7,  pofs, -1276, MSK_DO, pp0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr1,  pofs, -1280, MSK_DO, pp0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr5,  pofs, -1284, MSK_DO, pp0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MMINS,   kr17, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1,  kr4,  pofs, 4, MSK_DO, pc0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr0,  pofs, 0, MSK_DO, pc0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MMID3,   kr11, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1,  kr3,  pofs, -4, MSK_DO, pc0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MMAX3,   kr15, r7, EXP_H3210, r1, EXP_H3210, r5, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMINS,   kr14, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1,  kr8,  pofs, 1284, MSK_DO, pn0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_MMID3,   kr2,  pofs, 1280, MSK_DO, pn0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MMID3,   kr10, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1,  kr6,  pofs, 1276, MSK_DO, pn0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MMAX3,   kr13, r4, EXP_H3210, r0, EXP_H3210, r3, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMINS,   kr18, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMID3,   kr12, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMAX3,   kr16, r8, EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

/*step-2*/
exe(OP_MMAX3,   kr2,  r11, EXP_H3210, r10, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMID3,   kr0,  r11, EXP_H3210, r10, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMINS,   kr1,  r11, EXP_H3210, r10, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMAX3,   kr8,  r17, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMID3,   kr4,  r17, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_MMINS,   kr3,  r15, EXP_H3210, r13, EXP_H3210, r16, EXP_H3210, OP_NOP, OLL, OP NOP, OLL);
exe(OP_MMID3,   kr5,  r15, EXP_H3210, r13, EXP_H3210, r16, EXP_H3210, OP_NOP, OLL, OP NOP, OLL);

/*step-3*/
exe(OP_MMINS,   kr14, r3, EXP_H3210, r0, EXP_H3210, r4, EXP_H3210, OP_NOP, OLL, OP NOP, OLL);
exe(OP_MMID3,   kr10, r3, EXP_H3210, r0, EXP_H3210, r4, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMAX3,   kr13, r3, EXP_H3210, r0, EXP_H3210, r4, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMIN,    kr18, r2, EXP_H3210, r8, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMMAX,   kr12, r2, EXP_H3210, r8, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMIN,    kr11, r5, EXP_H3210, r1, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMMAX,   kr15, r5, EXP_H3210, r1, EXP_H3210, r11, EXP_H3210, OP NOP, OLL, OP NOP, OLL);

/*step-4*/
exe(OP_MMID3,   kr4,  r11, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMINS,   kr5,  r15, EXP_H3210, r13, EXP_H3210, r12, EXP_H3210, OP NOP, OLL, OP NOP, OLL);

/*step-5*/
exe(OP_MMAX3,   kr8,  r11, EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMID3,   kr3,  r15, EXP_H3210, r13, EXP_H3210, r12, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMIN,    kr14, r5, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMMAX,   kr15, r5, EXP_H3210, r4, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMINS,   kr18, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMID3,   kr10, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMAX3,   kr13, r3, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);

/*step-6*/
exe(OP_MMAX,    kr8,  r14, EXP_H3210, r18, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMIN,    kr5,  r15, EXP_H3210, r13, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MMID3,   kr31, r5, EXP_H3210, r10, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
mop(OP_STWR, 3,  kr31, rc0[CHIP], cofs, MSK_DO, rc0[CHIP], AWD, 0, 0, (U11)NULL, AWD);

```

メモリ参照/演算パターン: (b) 離散ランダム参照計算

```

//edge
exe(OP_ADD,      kr0fs, pc0[CHIP], EXP_H3210, cofs, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1,  kr5,  pofs, -1276, MSK_DO, (U11)pp0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr3,  pofs, -1280, MSK_DO, (U11)pp0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr1,  pofs, -1284, MSK_DO, (U11)pp0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_NOP,      &AR[3][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP NOP, OLL);
mop(OP_LDWR, 1,  kr8,  pofs, 4, MSK_DO, (U11)pc0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr7,  pofs, -4, MSK_DO, (U11)pc0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_NOP,      &AR[4][0], OLL, EXP_H3210, OLL, EXP_H3210, OLL, EXP_H3210, OP_OR, OLL, OP NOP, OLL);
mop(OP_LDWR, 1,  kr2,  pofs, 1284, MSK_DO, (U11)pr0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr4,  pofs, 1280, MSK_DO, (U11)pr0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1,  kr6,  pofs, 1276, MSK_DO, (U11)pr0[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MSSAD,   kr7,  OLL, EXP_H3210, r7, EXP_H3210, r8, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MSSAD,   kr1,  OLL, EXP_H3210, r1, EXP_H3210, r2, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MSSAD,   kr3,  OLL, EXP_H3210, r3, EXP_H3210, r4, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MSSAD,   kr5,  OLL, EXP_H3210, r5, EXP_H3210, r6, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MAUAH,   kr1,  r3, EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MAUAH,   kr5,  r7, EXP_H3210, r5, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
exe(OP_MAUH,    kr1,  r5, EXP_H3210, r1, EXP_H3210, OLL, EXP_H3210, OP_SUMHL, OLL, OP NOP, OLL);
exe(OP_MCAS,    kr31, r1, EXP_H3210, 64, EXP_H3210, OLL, EXP_H3210, OP NOP, OLL, OP NOP, OLL);
mop(OP_STBR, 3,  kr31, rc0[CHIP]++, 0, MSK_DO, (U11)rc0[CHIP], AWD/4, 0, 0, (U11)NULL, AWD/4);

```

メモリ参照/演算パターン: (b) 離散ランダム参照計算

```

//wdifline
exe(OP_ADD,      &rofs1,      lp[CHIP],   EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
exe(OP_ADD,      &rofs2,      rp[CHIP],   EXP_H3210, cofs, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1, &r2,         rofs1, 0, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r3,         rofs1, 4, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r4,         rofs1, 8, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r5,         rofs1, 12, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r6,         rofs2, 0, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r7,         rofs2, 4, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r8,         rofs2, 8, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r9,         rofs2, 12, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MSAD,    &r22,        r2,       EXP_H3210, r6, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1, &r12,        rofs1, 16, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r13,        rofs1, 20, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r3,             EXP_H3210, r7, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
mop(OP_LDWR, 1, &r14,        rofs1, 24, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r15,        rofs1, 28, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r4,             EXP_H3210, r8, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
mop(OP_LDWR, 1, &r16,        rofs2, 16, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r17,        rofs2, 20, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r5,             EXP_H3210, r9, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
mop(OP_LDWR, 1, &r18,        rofs2, 24, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r19,        rofs2, 28, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
exe(OP_MSSAD,   &r22,        r22,        EXP_H3210, r12, EXP_H3210, r16, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1, &r2,         rofs1, 32, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r3,         rofs1, 36, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r23,            EXP_H3210, r13, EXP_H3210, r17, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
mop(OP_LDWR, 1, &r4,         rofs1, 40, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r5,         rofs1, 44, MSK_DO, lp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r24,            EXP_H3210, r14, EXP_H3210, r18, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_LDWR, 1, &r6,         rofs2, 32, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r7,         rofs2, 36, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r25,            EXP_H3210, r15, EXP_H3210, r19, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
mop(OP_LDWR, 1, &r8,         rofs2, 40, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
mop(OP_LDWR, 1, &r9,         rofs2, 44, MSK_DO, rp[CHIP], AWD, 0, 0, (U11)NULL, AWD);
r12,            EXP_H3210, r2, EXP_H3210, r6, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
r13,            EXP_H3210, r3, EXP_H3210, r7, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
r14,            EXP_H3210, r4, EXP_H3210, r8, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
r15,            EXP_H3210, r5, EXP_H3210, r9, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
r22,            EXP_H3210, r23, EXP_H3210, r24, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
r31,            EXP_H3210, r1, EXP_H3210, r25, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL;
mop(OP_LDWR, 1, &BR[8][0][1], dp0[CHIP], cofs, MSK_DO, (U11)dp0[CHIP], AWD, 0, 1, (U11)NULL, AWD);
exe(OP_ADD,      &BR[8][0],  BR[8][0][1], EXP_H3210, r1, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);
mop(OP_STWR, 3, &BR[8][0],  cofs, dp0[CHIP], MSK_DO, (U11)dp0[CHIP], AWD, 0, 1, (U11)NULL, AWD);

```

メモリ参照/演算パターン: (b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```

//grapes
mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0           -WDHT-AWD )*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#2 */
mop(OP_LDWR, 1, &BR[2][2][1], bofs, (0           -(WDHT+PAD*(MID-6)-WDHT-AWD )*4, MSK_DO, arwo00[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#2 */
exe(OP_FML, &r0, BR[2][0][1], EXP_H3210, BR[2][2][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0           -WDHT )*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][0], bofs, (0           -WDHT +1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#3 */
mop(OP_LDWR, 1, &BR[3][1][1], bofs, (0           -WDHT )*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#3 */
mop(OP_LDWR, 1, &BR[3][2][1], bofs, (0           -(WDHT+PAD*(MID-5)-WDHT -1)*4, MSK_DO, arwo01[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][2][0], bofs, (0           -(WDHT+PAD*(MID-5)-WDHT +1)*4, MSK_DO, arwo01[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][3][1], bofs, (0           -(WDHT+PAD*(MID-4)-WDHT )*4, MSK_DO, arwo02[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][3][0], bofs, (0           -(WDHT+PAD*(MID-4)-WDHT +1)*4, MSK_DO, arwo02[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#3 */
exe(OP_FML, &r1, r0, EXP_H3210, BR[3][0][1], EXP_H3210, BR[3][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#4 */
exe(OP_FML, &r2, BR[3][0][0], EXP_H3210, BR[3][2][0], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#4 */
exe(OP_FML, &r3, BR[3][1][1], EXP_H3210, BR[3][3][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#4 */
mop(OP_LDWR, 1, &BR[4][0][1], bofs, (0           -WDHT+AWD )*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#4 */
mop(OP_LDWR, 1, &BR[4][2][1], bofs, (0           -(WDHT+AWD+PAD )*4, MSK_DO, arwo03[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#4 */
exe(OP_FMA, &r4, r1, EXP_H3210, BR[4][0][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#5 */
exe(OP_FAD, &r5, r2, EXP_H3210, r3, EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#5 */
mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0           -AWD-1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][0], bofs, (0           -AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#6 */
mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0           -AWD )*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#6 */
mop(OP_LDWR, 1, &BR[6][2][1], bofs, (0           -AWD-1)*4, MSK_DO, arwo04[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#6 */
mop(OP_LDWR, 1, &BR[6][2][0], bofs, (0           -(WDHT+PAD*(MID-2)-AWD)+1)*4, MSK_DO, arwo04[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#6 */
mop(OP_LDWR, 1, &BR[6][3][1], bofs, (0           -(WDHT+PAD*(MID-2)-AWD )*4, MSK_DO, arwo05[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#6 */
mop(OP_LDWR, 1, &BR[6][3][0], bofs, (0           -(WDHT+PAD*(MID-1)-AWD )*4, MSK_DO, arwo05[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#6 */
exe(OP_FMA, &r4, r4, EXP_H3210, BR[6][0][1], EXP_H3210, BR[6][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
exe(OP_FMA, &r5, r5, EXP_H3210, BR[6][0][0], EXP_H3210, BR[6][2][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
exe(OP_FML, &r6, BR[6][1][1], EXP_H3210, BR[6][3][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0           -AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][0], bofs, (0           +1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#7 */
mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0           -(WDHT+PAD*(MID )-1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#7 */
mop(OP_LDWR, 1, &BR[7][2][1], bofs, (0           +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#7 */
exe(OP_FMA, &r3, r0, EXP_H3210, BR[7][0][1], EXP_H3210, BR[7][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FML, &r4, r1, EXP_H3210, BR[7][0][0], EXP_H3210, BR[7][2][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FAD, &r5, r2, EXP_H3210, BR[7][1][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
mop(OP_LDWR, 1, &BR[7][7][1], bofs, (0           -AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#7 */
mop(OP_LDWR, 1, &BR[7][7][0], bofs, (0           +1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#7 */
mop(OP_LDWR, 1, &BR[7][7][1], bofs, (0           -1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#7 */
mop(OP_LDWR, 1, &BR[7][7][0], bofs, (0           +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#7 */
exe(OP_FMA, &r3, r0, EXP_H3210, BR[7][7][1], EXP_H3210, BR[7][7][2], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FML, &r4, r1, EXP_H3210, BR[7][7][0], EXP_H3210, BR[7][7][2], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FAD, &r5, r2, EXP_H3210, BR[7][7][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
mop(OP_LDWR, 1, &BR[8][0][1], bofs, (0           +AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#8 */
mop(OP_LDWR, 1, &BR[8][0][0], bofs, (0           +AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#8 */
mop(OP_LDWR, 1, &BR[8][1][1], bofs, (0           +AWD )*4, MSK_DO, arwo07[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#8 */
mop(OP_LDWR, 1, &BR[8][2][1], bofs, (0           +AWD+1)*4, MSK_DO, arwo08[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#8 */
mop(OP_LDWR, 1, &BR[8][2][0], bofs, (0           +AWD+1)*4, MSK_DO, arwo08[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#8 */
mop(OP_LDWR, 1, &BR[8][3][1], bofs, (0           +AWD )*4, MSK_DO, arwo07[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#8 */
mop(OP_LDWR, 1, &BR[8][3][0], bofs, (0           +AWD+1)*4, MSK_DO, brow08[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#9 */
exe(OP_FMA, &r3, r3, EXP_H3210, BR[8][0][1], EXP_H3210, BR[8][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#9 */
exe(OP_FML, &r4, r3, EXP_H3210, BR[8][0][0], EXP_H3210, BR[8][2][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#9 */
exe(OP_FAD, &r5, r5, EXP_H3210, BR[8][1][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#9 */
mop(OP_LDWR, 1, &BR[10][0][1], bofs, (0           +WDHT-AWD )*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#10 */
mop(OP_LDWR, 1, &BR[10][0][0], bofs, (0           +(WDHT+PAD*(MID+3)-WDHT-AWD )*4, MSK_DO, arwo09[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#10 */
exe(OP_FML, &r6, r6, EXP_H3210, BR[10][0][1], EXP_H3210, BR[10][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#11 */
mop(OP_LDWR, 1, &BR[11][0][1], bofs, (0           +WDHT -1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#11 */
mop(OP_LDWR, 1, &BR[11][0][0], bofs, (0           +WDHT +1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, (U11)NULL, AWD*(RMGRP+PAD*2));/* stage#11 */
mop(OP_LDWR, 1, &BR[11][1][1], bofs, (0           +WDHT +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#11 */
mop(OP_LDWR, 1, &BR[11][1][0], bofs, (0           +WDHT +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#11 */
mop(OP_LDWR, 1, &BR[11][2][1], bofs, (0           +WDHT +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#11 */
mop(OP_LDWR, 1, &BR[11][2][0], bofs, (0           +WDHT +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#11 */
mop(OP_LDWR, 1, &BR[11][3][1], bofs, (0           +WDHT +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#11 */
mop(OP_LDWR, 1, &BR[11][3][0], bofs, (0           +WDHT +1)*4, MSK_DO, arwo06[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#11 */
exe(OP_FMA, &r7, r7, EXP_H3210, BR[11][0][1], EXP_H3210, BR[11][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#12 */
exe(OP_FML, &r8, r7, EXP_H3210, BR[11][0][0], EXP_H3210, BR[11][2][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#12 */
exe(OP_FAD, &r9, r5, EXP_H3210, BR[8][1][1], EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#12 */
mop(OP_STWR, 3, &r7, cofs, (0           +4, MSK_DO, crow0[CHIP], AWD+RMGRP, 0, 0, (U11)NULL, AWD+RMGRP);/* stage#14 */

```

メモリ参照/演算パターン: (b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```
//jacobi
mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0
    -WDHT      )*4, MSK_DO, brow00[CHIP], AWD*RMGRP, 0, 0, browp0[CHIP], AWD*RMGRP);/* stage#2 */
mop(OP_LDWR, 1, &BR[2][2][1], bofs, (0
    +WDHT      )*4, MSK_DO, brow01[CHIP], AWD*RMGRP, 0, 0, browp1[CHIP], AWD*RMGRP);/* stage#2 */
exe(OP_FAD, kr0, BR[2][0][1], EXP_H3210, BR[2][2][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0
    -AWD      )*4, MSK_DO, brow02[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp2[CHIP], AWD*(RMGRP+PAD*2)); /* stage#3 */
exe(OP_FAD, kr1, r0,
    EXP_H3210, BR[3][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#3 */
mop(OP_LDWR, 1, &BR[4][1][1], bofs, (0
    -1)*4, MSK_DO, brow02[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp2[CHIP], AWD*(RMGRP+PAD*2)); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][1][1], bofs, (0
    )*4, MSK_DO, brow02[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp2[CHIP], AWD*(RMGRP+PAD*2)); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][2][1], bofs, (0
    +1)*4, MSK_DO, brow02[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp2[CHIP], AWD*(RMGRP+PAD*2)); /* stage#4 */
exe(OP_FAD, kr2, r1,
    EXP_H3210, BR[4][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#5 */
exe(OP_FAD, kr3, II,
    EXP_H3210, BR[4][1][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][1], bofs, (0
    +AWD      )*4, MSK_DO, brow02[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp2[CHIP], AWD*(RMGRP+PAD*2)); /* stage#5 */
exe(OP_FAD, BR[5][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#6 */
exe(OP_FAD, kr4, r2,
    EXP_H3210, BR[4][2][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
exe(OP_FAD, kr5, r4,
    EXP_H3210, BR[4][2][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FMA, kr6, r3,
    EXP_H3210, r5,
    EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
mop(OP_STWR, 3, kr6,
    cofs, (0
        )*4, MSK_DO, crow0[CHIP], AWD*RMGRP, 0, 0, crowp[CHIP], AWD*RMGRP); /* stage#8 */

メモリ参照/演算パターン：(b) 離散ランダム参照計算
```

```
//fd6
mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0
    -WDHT*3 )*4, MSK_DO, brow00[CHIP], AWD*RMGRP, 0, 0, browp0[CHIP], AWD*RMGRP);/* stage#2 */
mop(OP_LDWR, 1, &BR[2][2][1], bofs, (0
    +WDHT*3 )*4, MSK_DO, brow05[CHIP], AWD*RMGRP, 0, 0, browp5[CHIP], AWD*RMGRP);/* stage#2 */
exe(OP_FAD, kr3, BR[2][0][1], EXP_H3210, BR[2][2][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0
    -WDHT*2 )*4, MSK_DO, brow01[CHIP], AWD*RMGRP, 0, 0, browp1[CHIP], AWD*RMGRP);/* stage#3 */
mop(OP_LDWR, 1, &BR[3][2][1], bofs, (0
    +WDHT*2 )*4, MSK_DO, brow04[CHIP], AWD*RMGRP, 0, 0, browp4[CHIP], AWD*RMGRP);/* stage#3 */
exe(OP_FAD, kr2, BR[3][0][1], EXP_H3210, BR[3][2][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#4 */
mop(OP_LDWR, 1, &BR[4][1][1], bofs, (0
    -WDHT*1 )*4, MSK_DO, brow02[CHIP], AWD*RMGRP, 0, 0, browp2[CHIP], AWD*RMGRP);/* stage#4 */
mop(OP_LDWR, 1, &BR[4][2][1], bofs, (0
    +WDHT*1 )*4, MSK_DO, brow03[CHIP], AWD*RMGRP, 0, 0, browp3[CHIP], AWD*RMGRP);/* stage#4 */
exe(OP_FAD, BR[4][0][1], EXP_H3210, BR[4][2][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][1], bofs, (0
    -AWD*3)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][1], bofs, (0
    +AWD*3)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][1][1], bofs, (0
    -3)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][1][1], bofs, (0
    +3)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#5 */
exe(OP_FAD, kr13, BR[5][0][1], EXP_H3210, BR[5][1][0], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#6 */
exe(OP_FAD, kr23, BR[5][1][0], EXP_H3210, BR[5][1][0], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0
    +AWD*2)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0
    -AWD*2)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0
    -2)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0
    +2)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#6 */
exe(OP_FAD, kr12, BR[6][0][1], EXP_H3210, BR[6][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
exe(OP_FAD, kr22, BR[6][0][1], EXP_H3210, BR[6][1][0], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
exe(OP_FAD, kr23, r13,
    EXP_H3210, r23,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0
    +AWD*1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0
    -AWD*1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0
    -1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0
    +1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#7 */
exe(OP_FAD, kr11, BR[7][0][1], EXP_H3210, BR[7][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FAD, kr21, BR[7][1][0], EXP_H3210, BR[7][1][0], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FAD, kr22, r12,
    EXP_H3210, r22,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
exe(OP_FAD, kr3, r23,
    EXP_H3210, r3,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#8 */
mop(OP_LDWR, 1, &BR[8][0][1], bofs, (0
    )*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, browp6[CHIP], AWD*(RMGRP+PAD*2)); /* stage#8 */
exe(OP_FAD, kr21, r11,
    EXP_H3210, r21,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#9 */
exe(OP_FML, kr21, r11,
    EXP_H3210, r21,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#9 */
exe(OP_FAD, kr22, r22,
    EXP_H3210, r2,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#9 */
exe(OP_FMA, kr13, r10,
    EXP_H3210, r3,
    EXP_H3210, I4,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#10 */
exe(OP_FAD, kr1, r21,
    EXP_H3210, r1,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#10 */
exe(OP_FMA, kr12, r13,
    EXP_H3210, r2,
    EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#11 */
exe(OP_FMA, kr11, r12,
    EXP_H3210, r1,
    EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);/* stage#12 */
mop(OP_STWR, 3, kr11,
    cofs, (0
        )*4, MSK_DO, crow0[CHIP], AWD*RMGRP, 0, 0, crowp[CHIP], AWD*RMGRP); /* stage#12 */

メモリ参照/演算パターン：(b) 離散ランダム参照計算
```

```

//resid
mop(OP_LDWR, 1, &BR[2][0][1], bofs, (0
    -WDHT-AWD-1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#2 */
mop(OP_LDWR, 1, &BR[2][0][0], bofs, (0
    -WDHT-AWD)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#2 */
mop(OP_LDWR, 1, &BR[2][1][1], bofs, (0
    -WDHT-AWD+1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#2 */
exe(OP_FML, kr0, BR[2][0][1], EXP_H3210, I3,
    EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exe(OP_FML, kr1, BR[2][1][1], EXP_H3210, I2,
    EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
exe(OP_FML, kr2, BR[2][1][1], EXP_H3210, I3,
    EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][1], bofs, (0
    -WDHT -1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][0], bofs, (0
    -WDHT)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#3 */
mop(OP_LDWR, 1, &BR[3][1][1], bofs, (0
    -WDHT +1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#3 */
exe(OP_FML, kr3, r0,
    EXP_H3210, BR[3][0][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
exe(OP_FML, kr4, r1,
    EXP_H3210, BR[3][0][0], EXP_H3210, I1,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
exe(OP_FML, kr5, r2,
    EXP_H3210, BR[3][1][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][0][1], bofs, (0
    -WDHT+AWD-1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#4 */
mop(OP_LDWR, 1, &BR[4][0][0], bofs, (0
    -WDHT+AWD)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#4 */
mop(OP_LDWR, 1, &BR[4][1][1], bofs, (0
    -WDHT+AWD+1)*4, MSK_DO, brow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow0[CHIP], AWD*(RMGRP+PAD*2));/* stage#4 */
exe(OP_FML, kr6, r3,
    EXP_H3210, BR[4][0][1], EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exe(OP_FML, kr7, r4,
    EXP_H3210, BR[4][0][0], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exe(OP_FML, kr8, r5,
    EXP_H3210, BR[4][1][1], EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */

mop(OP_LDWR, 1, &BR[5][0][1], bofs, (0
    -AWD-1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][0], bofs, (0
    -AWD)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#5 */
mop(OP_LDWR, 1, &BR[5][1][1], bofs, (0
    -AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#5 */
exe(OP_FML, kr0, r6,
    EXP_H3210, BR[5][0][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exe(OP_FML, kr1, r7,
    EXP_H3210, BR[5][0][0], EXP_H3210, I1,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exe(OP_FML, kr2, r8,
    EXP_H3210, BR[5][1][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][1], bofs, (0
    -1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][0], bofs, (0
    +4)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#6 */
mop(OP_LDWR, 1, &BR[6][1][1], bofs, (0
    +1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#6 */
exe(OP_FML, kr3, r0,
    EXP_H3210, BR[6][0][1], EXP_H3210, I1,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exe(OP_FML, kr4, r1,
    EXP_H3210, BR[6][0][0], EXP_H3210, I0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exe(OP_FML, kr5, r2,
    EXP_H3210, BR[6][1][1], EXP_H3210, I1,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][1], bofs, (0
    +AWD-1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#7 */
mop(OP_LDWR, 1, &BR[7][0][0], bofs, (0
    +AWD)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#7 */
mop(OP_LDWR, 1, &BR[7][1][1], bofs, (0
    +AWD+1)*4, MSK_DO, brow03[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow03[CHIP], AWD*(RMGRP+PAD*2));/* stage#7 */
exe(OP_FML, kr6, r3,
    EXP_H3210, BR[7][0][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exe(OP_FML, kr7, r4,
    EXP_H3210, BR[7][0][0], EXP_H3210, I1,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exe(OP_FML, kr8, r5,
    EXP_H3210, BR[7][1][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */

mop(OP_LDWR, 1, &BR[8][0][1], bofs, (0
    +WDHT-AWD-1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#8 */
mop(OP_LDWR, 1, &BR[8][0][0], bofs, (0
    +WDHT-AWD)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#8 */
mop(OP_LDWR, 1, &BR[8][1][1], bofs, (0
    +WDHT-AWD+1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#8 */
exe(OP_FML, kr0, r6,
    EXP_H3210, BR[8][0][1], EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exe(OP_FML, kr1, r7,
    EXP_H3210, BR[8][0][0], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exe(OP_FML, kr2, r8,
    EXP_H3210, BR[8][1][1], EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
mop(OP_LDWR, 1, &BR[9][0][1], bofs, (0
    +WDHT -1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#9 */
mop(OP_LDWR, 1, &BR[9][0][0], bofs, (0
    +WDHT)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#9 */
mop(OP_LDWR, 1, &BR[9][1][1], bofs, (0
    +WDHT+1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#9 */
exe(OP_FML, kr3, r0,
    EXP_H3210, BR[9][0][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exe(OP_FML, kr4, r1,
    EXP_H3210, BR[9][0][0], EXP_H3210, I1,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exe(OP_FML, kr5, r2,
    EXP_H3210, BR[9][1][1], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
mop(OP_LDWR, 1, &BR[10][0][1], bofs, (0
    +WDHT+AHD-1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#10 */
mop(OP_LDWR, 1, &BR[10][0][0], bofs, (0
    +WDHT+AHD)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#10 */
mop(OP_LDWR, 1, &BR[10][1][1], bofs, (0
    +WDHT+AHD+1)*4, MSK_DO, brow06[CHIP], AWD*(RMGRP+PAD*2), 0, 0, brow06[CHIP], AWD*(RMGRP+PAD*2));/* stage#10 */
exe(OP_FML, kr6, r3,
    EXP_H3210, BR[10][0][1], EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
exe(OP_FML, kr7, r4,
    EXP_H3210, BR[10][0][0], EXP_H3210, I2,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
exe(OP_FML, kr8, r5,
    EXP_H3210, BR[10][1][1], EXP_H3210, I3,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
mop(OP_LDWR, 1, &BR[11][0][1], bofs, (0
    +4), MSK_DO, brow0[CHIP], AWD*RMGRP, 0, 0, brow0[CHIP], AWD*RMGRP);/* stage#11 */
exe(OP_FAD, kr1, r6,
    EXP_H3210, BR[11][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12 */
exe(OP_FAD, kr2, r7,
    EXP_H3210, r8,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#12 */
exe(OP_FAD, kr0, r1,
    EXP_H3210, r2,
    EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13 */
mop(OP_STWR, 3, kr6,
    bofs, (0
        )*4, MSK_DO, brow0[CHIP], AWD*RMGRP, 0, 0, brow0[CHIP], AWD*RMGRP);/* stage#13 */

```

メモ参照/演算バターン：(b) 離散ランダム参照計算

```

//wave2d
mop(OP_LDWR, 1, &BR[2][0][1], z0ofs, (0
    )*4, MSK_DO, z0row0[CHIP], AWD*RMGRP, 0, 0, z0rowp[CHIP], AWD*RMGRP); /* stage#2 */
exe(OP_FML, kr0, BR[2][0][1], EXP_H3210, I2,
    EXP_H3210, 0, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#3 */
mop(OP_LDWR, 1, &BR[3][0][1], z1ofs, (0
    -AWD)*4, MSK_DO, zirow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, zirowp0[CHIP], AWD*(RMGRP+PAD*2)); /* stage#3 */
mop(OP_LDWR, 1, &BR[4][0][1], z1ofs, (0
    -1)*4, MSK_DO, zirow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, zirowp0[CHIP], AWD*(RMGRP+PAD*2)); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][0][0], z1ofs, (0
    )*4, MSK_DO, zirow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, zirowp0[CHIP], AWD*(RMGRP+PAD*2)); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][1][1], z1ofs, (0
    +1)*4, MSK_DO, zirow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, zirowp0[CHIP], AWD*(RMGRP+PAD*2)); /* stage#4 */
exe(OP_FAD, kr1, BR[3][0][1], EXP_H3210,
    BR[4][0][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
mop(OP_LDWR, 1, &BR[5][0][1], z1ofs, (0
    +AWD)*4, MSK_DO, zirow00[CHIP], AWD*(RMGRP+PAD*2), 0, 0, zirowp0[CHIP], AWD*(RMGRP+PAD*2)); /* stage#5 */
exe(OP_FML, kr2, r1,
    EXP_H3210, BR[5][0][1], EXP_H3210, 14,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
mop(OP_LDWR, 1, &BR[5][0][0], z1ofs, (0
    )*4, MSK_DO, EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exe(OP_FML, kr3, r2,
    EXP_H3210, BR[5][1][1], EXP_H3210, 0,
    EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
mop(OP_LDWR, 1, &BR[6][0][0], z1ofs, (0
    +WDHT+AHD-1)*4, MSK_DO, brow0[CHIP], AWD*RMGRP, 0, 0, brow0[CHIP], AWD*RMGRP);/* stage#7 */
mop(OP_LDWR, 1, &BR[6][0][1], z2ofs, (0
    +WDHT+AHD)*4, MSK_DO, brow0[CHIP], AWD*RMGRP, 0, 0, brow0[CHIP], AWD*RMGRP);/* stage#7 */
mop(OP_LDWR, 1, &BR[6][1][1], z2ofs, (0
    +WDHT+AHD+1)*4, MSK_DO, brow0[CHIP], AWD*RMGRP, 0, 0, brow0[CHIP], AWD*RMGRP);/* stage#7 */

```

メモ参照/演算バターン：(b) 離散ランダム参照計算

```

#define cnn_core1(r, i, ofs, k, rp1) \
mop(OP_LDWR, 1, &BR[r][0][1], (U11)kp0[i][CHIP], ofs, MSK_DO, (U11)ker, IC*OC*K*k, 0, 0, (U11)NULL, IC*OC*K*k); \
mop(OP_LDWR, 1, &BR[r][0][0], (U11)kp1[i][CHIP], ofs, MSK_DO, (U11)ker, IC*OC*K*k, 0, 0, (U11)NULL, IC*OC*K*k); \
mop(OP_LDWR, 1, &BR[r][1][1], (U11)kp2[i][CHIP], ofs, MSK_DO, (U11)ker, IC*OC*K*k, 0, 0, (U11)NULL, IC*OC*K*k); \
mop(OP_LDWR, 1, &BR[r][1][0], (U11)kp3[i][CHIP], ofs, MSK_DO, (U11)ker, IC*OC*K*k, 0, 0, (U11)NULL, IC*OC*K*k); \
mop(OP_LDR, 1, &BR[r][2][1], (U11)ip1[i][k], ofs, MSK_WO, (U11)it1[], M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); \
mop(OP_LDR, 1, &BR[r][2][0], (U11)ip1[i][k], ofs, MSK_WO, (U11)it1[], M*(RMGRP+2), 0, 0, (U11)NULL, M*(RMGRP+2)); \
exe(OP_FMA, AR[rp1][0], AR[r][0], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][0][1], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][1], AR[r][1], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][0][0], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][2], AR[r][2], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][1][1], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][3], AR[r][3], EXP_H3210, BR[r][2][0], EXP_H3210, BR[r][1][0], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL);

#define cnn_final(r, rp1) \
mop(OP_LDR, 1, &BR[rp1][0][1], (U11)op0[CHIP], ofs, MSK_WO, (U11)ot0[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_LDR, 1, &BR[rp1][1][1], (U11)op1[CHIP], ofs, MSK_WO, (U11)ot1[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_LDR, 1, &BR[rp1][2][1], (U11)op2[CHIP], ofs, MSK_WO, (U11)ot2[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_LDR, 1, &BR[rp1][3][1], (U11)op3[CHIP], ofs, MSK_WO, (U11)ot3[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
exe(OP_FAD, &AR[rp1][0], AR[r][0], EXP_H3210, BR[rp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][1], AR[r][1], EXP_H3210, BR[rp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][2], AR[r][2], EXP_H3210, BR[rp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][3], AR[r][3], EXP_H3210, BR[rp1][2][0], EXP_H3210, BR[r][1][1], EXP_H1010, OP_NOP, OLL, OP_NOP, OLL); \
mop(OP_STR, 3, &AR[rp1][0], ofs, (U11)op0[CHIP], MSK_DO, (U11)ot0[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_STR, 3, &AR[rp1][1], ofs, (U11)op1[CHIP], MSK_DO, (U11)ot1[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_STR, 3, &AR[rp1][2], ofs, (U11)op2[CHIP], MSK_DO, (U11)ot2[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP); \
mop(OP_STR, 3, &AR[rp1][3], ofs, (U11)op3[CHIP], MSK_DO, (U11)ot3[CHIP], M*RMGRP, 0, 1, (U11)NULL, M*RMGRP);

```

メモ参照/演算バターン：(b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```
#define mm_core1(r, rm1, rp1) \
    mop(OP_LDR, 3, &BR[r][0][1], (U11)b0[rm1], (U11)c0[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDR, 3, &BR[r][0][0], (U11)b1[rm1], (U11)c0[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDR, 3, &BR[r][1][1], (U11)b2[rm1], (U11)c0[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDR, 3, &BR[r][1][0], (U11)b3[rm1], (U11)c0[rm1], M2, 0, 0, (U11)NULL, M2); \
    mop(OP_LDWR, 1, &BR[r][2][1], (U11)a1[rm1][CHIP], (U11)a0[rm1][CHIP], L*RMGRP, 0, 0, (U11)NULL, L*RMGRP); \
exe(OP_FMA, &AR[rp1][0], AR[r][0], EXP_H3210, BR[r][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][1], AR[r][1], EXP_H3210, BR[r][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][2], AR[r][2], EXP_H3210, BR[r][2][1], EXP_H3210, EXP_H3210, OP_NOP, OLI, OP_NOP, OLL); \
exe(OP_FMA, &AR[rp1][3], AR[r][3], EXP_H3210, BR[r][2][1], EXP_H3210, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL)

#define mm_final(r, rp1) \
    mop(OP_LDR, 3, &BR[rp1][0][1], (U11)c00[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
    mop(OP_LDR, 3, &BR[rp1][1][1], (U11)c01[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
    mop(OP_LDR, 3, &BR[rp1][2][1], (U11)c02[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
    mop(OP_LDR, 3, &BR[rp1][3][1], (U11)c03[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
exe(OP_FAD, &AR[rp1][0], AR[r][0], EXP_H3210, BR[rp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][1], AR[r][1], EXP_H3210, BR[rp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][2], AR[r][2], EXP_H3210, BR[rp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
exe(OP_FAD, &AR[rp1][3], AR[r][3], EXP_H3210, AR[r][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); \
mop(OP_STR, 3, &AR[rp1][0], (U11)oofs, (U11)c00[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
mop(OP_STR, 3, &AR[rp1][1], (U11)oofs, (U11)c01[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
mop(OP_STR, 3, &AR[rp1][2], (U11)oofs, (U11)c02[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP); \
mop(OP_STR, 3, &AR[rp1][3], (U11)oofs, (U11)c03[CHIP], MSK_D0, (U11)c0[CHIP], M2*RMGRP, 0, 1, (U11)NULL, M2*RMGRP)
```

メモリ参照/演算パターン: (b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```
//inv_x1
exe(OP_CMP_LT, &cco, 100[CHIP], EXP_H3210, M, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 LD */
mop(OP_LDWR, 1, &BR[2][2][1], top, cof, MSK_W0, topw, len, 0, 0, NULL, len); /* A[p[i]*M+k】 stage#2 | */
mop(OP_LDWR, 1, &BR[2][0][1], d00[CHIP], cof, MSK_W0, d00w[CHIP], len, 0, 1, NULL, len); /* A[p[i]+h*NCIP+CHIP]*M+k】 stage#2 +-> | */
mop(OP_LDWR, 1, &BR[2][1][1], d00[CHIP], cof, MSK_W0, d00w[CHIP], len, 2, 0, 1, NULL, len); /* A[p[i]+h*NCIP+CHIP]*M+k】 stage#2 +-> | */
exe(OP_FMMS, &AR[2][0], BR[2][0][1], EXP_H3210, BR[2][1][1], EXP_H3210, BR[2][2][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 | ■■■ | 1.0 */
cex(OP_CEXE, &exo, 0, 0, 0, cco, 0xaaaa);
mop(OP_STWR, exo, &AR[2][0], oofs, d00[CHIP], MSK_D0, d00w[CHIP], len, 2, 0, 1, NULL, len); /* stage#2 | AR[1] | */
/* stage#2 | + ST v */

メモリ参照/演算パターン: (c) 連続アドレス累算計算
```

```
//inv_x2
mop(OP_LDWR, 1, &Ajk, top, cof, MSK_W0, topw, len, 0, 0, NULL, len); /* A[p[j]*M+k】 */// stage#1.0 */
mop(OP_LDWR, 1, &BR[1][3][1], t000[CHIP], cof, MSK_W0, t000w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*w*h+w+0)*M+k】 */// stage#1.3 +->xxx LD */
mop(OP_LDWR, 1, &b000, d000[CHIP], 0, MSK_W0, d000w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*w*h+w+0)*M+j】 */// stage#2.0 | ■■■ | */
exe(OP_FMMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2.0 +- xxx+ST v */
mop(OP_STWR, 1, &b000, 0, d000[CHIP], MSK_D0, d000w[CHIP], 1, 0, 1, NULL, 1); /* stage#2.0 +-----xxx */

メモリ参照/演算パターン: (d) 固定アドレス累算計算
```

```
//inv_x3
mop(OP_LDWR, 1, &Ajk, top, cof, MSK_W0, topw, len, 0, 0, NULL, len); /* A[p[j]*M+k】 */// stage#1.0
mop(OP_LDWR, 1, &BR[1][3][1], t000[CHIP], cof, MSK_W0, t000w[CHIP], len, 0, 1, NULL, len); /* b[(i+CHIP*w*h+w+0)*M+k】 */// stage#1.3 +->xxx LD */
mop(OP_LDWR, 1, &b000, d000[CHIP], 0, MSK_W0, d000w[CHIP], 1, 0, 1, NULL, 1); /* b[(i+CHIP*w*h+w+0)*M+j】 */// stage#2.0 | ■■■ | */
exe(OP_FMMS, &b000, b000, EXP_H3210, Ajk, EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2.0 +- xxx+ST v */
mop(OP_STWR, 1, &b000, 0, d000[CHIP], MSK_D0, d000w[CHIP], 1, 0, 1, NULL, 1); /* stage#2.0 +-----xxx */

メモリ参照/演算パターン: (d) 固定アドレス累算計算
```

```
//gather
mop(OP_LDWR, 1, &BR[4][0][1], r0, (U11)y_m_xm, MSK_D0, (U11)acci_ym0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][1][1], r0, (U11)y_m_zx, MSK_D0, (U11)acci_ym0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
mop(OP_LDWR, 1, &BR[4][2][1], r0, (U11)y_m_xy, MSK_D0, (U11)acci_ym0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#4 */
exe(OP_MLUH, kr10, BR[4][0][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exe(OP_MLUH, kr11, BR[4][1][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exe(OP_MLUH, kr12, BR[4][2][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#5 */
exe(OP_MLUH, kr13, BR[4][0][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exe(OP_MLUH, kr14, BR[4][1][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exe(OP_MLUH, kr15, BR[4][2][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
exe(OP_MAUH3, kr20, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][0][1], r0, (U11)y_z_xm, MSK_D0, (U11)acci_yz0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][1][1], r0, (U11)y_z_xy, MSK_D0, (U11)acci_yz0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#6 */
mop(OP_LDWR, 1, &BR[6][2][1], r0, (U11)y_z_xz, MSK_D0, (U11)acci_yz0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#6 */
exe(OP_MAUH3, kr21, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exe(OP_MLUH, kr10, BR[6][0][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exe(OP_MLUH, kr11, BR[6][1][1], EXP_B5410, 64LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exe(OP_MLUH, kr12, BR[6][2][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#7 */
exe(OP_MLUH, kr13, BR[6][0][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exe(OP_MLUH, kr14, BR[6][1][1], EXP_B7632, 64LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exe(OP_MLUH, kr15, BR[6][2][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
exe(OP_MAUH3, kr22, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#8 */
mop(OP_LDWR, 1, &BR[8][0][1], r0, (U11)y_p_xm, MSK_D0, (U11)acci_yp0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#8 */
mop(OP_LDWR, 1, &BR[8][1][1], r0, (U11)y_p_xy, MSK_D0, (U11)acci_yp0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#8 */
mop(OP_LDWR, 1, &BR[8][2][1], r0, (U11)y_p_xz, MSK_D0, (U11)acci_yp0[CHIP], IM, 0, 0, (U11)NULL, IM); /* stage#8 */
exe(OP_MAUH3, kr23, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exe(OP_MLUH, kr10, BR[8][0][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exe(OP_MLUH, kr11, BR[8][1][1], EXP_B5410, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exe(OP_MLUH, kr12, BR[8][2][1], EXP_B5410, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#9 */
exe(OP_MLUH, kr13, BR[8][0][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exe(OP_MLUH, kr14, BR[8][1][1], EXP_B7632, 32LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exe(OP_MLUH, kr15, BR[8][2][1], EXP_B7632, 16LL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exe(OP_MAUH3, kr24, r10, EXP_H3210, r11, EXP_H3210, r12, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#10 */
exe(OP_MAUH3, kr25, r13, EXP_H3210, r14, EXP_H3210, r15, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#11 */
exe(OP_MAUH3, kr30, r20, EXP_H3210, r22, EXP_H3210, r24, EXP_H3210, OP_AND, -1LL, OP_SRLM, 8LL); /* stage#12 */
exe(OP_MAUH3, kr31, r21, EXP_H3210, r23, EXP_H3210, r25, EXP_H3210, OP_AND, -1LL, OP_SRLM, 8LL); /* stage#12 */
exe(OP_MH2BW, kr29, r31, EXP_H3210, r30, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#13 */
mop(OP_STWR, 3, &r29, (U11)(acco0[CHIP]++), OLL, MSK_D0, (U11)acco_base0[CHIP], CRANGE, 0, 0, (U11)NULL, CRANGE); /* stage#13 */

メモリ参照/演算パターン: (b) 離散ランダム参照計算
```

メモリ参照/演算パターン: (b) 縮散ランダム参照計算

```
#define cnn5x5_core1(b, o, bp1, n) \
    mop(OP_LDWR, 1, &BR[b][0][1], (U11)kp0[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][0][0], (U11)kp1[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][1], (U11)kp2[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][0], (U11)kp3[CHIP], o, MSK_DO, (U11)i_ker, Klen, 0, Force, (U11)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][0], (U11)kp0[CHIP], oofs, MSK_WO, (U11)ito1[CHIP], Mlen, 0, 0, (U11)NULL, Mlen);\ \
    exe(OP_FMA, &AR[bp1][0], AR[b][0], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][1], AR[b][1], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][2], AR[b][2], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][3], AR[b][3], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL)\

#define cnn5x5_final(b, bp1) \
    mop(OP_LDWR, 1, &BR[bp1][0][1], (U11)op0[CHIP], oofs, MSK_WO, (U11)oto1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][1][1], (U11)op1[CHIP], oofs, MSK_WO, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][2][1], (U11)op2[CHIP], oofs, MSK_WO, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][3][1], (U11)op3[CHIP], oofs, MSK_WO, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    exe(OP_FAD, &AR[bp1][0], AR[b][0], EXP_H3210, BR[bp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][1], AR[b][1], EXP_H3210, BR[bp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][2], AR[b][2], EXP_H3210, BR[bp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][3], AR[b][3], EXP_H3210, BR[bp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    mop(OP_STWR, 1, &AR[bp1][0], oofs, (U11)op0[CHIP], MSK_DO, (U11)ot0[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][1], oofs, (U11)op1[CHIP], MSK_DO, (U11)ot1[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][2], oofs, (U11)op2[CHIP], MSK_DO, (U11)ot2[CHIP], Mlen, 0, 1, (U11)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][3], oofs, (U11)op3[CHIP], MSK_DO, (U11)ot3[CHIP], Mlen, 0, 1, (U11)NULL, Mlen)
```

メモリ参照/演算パターン: (b) 縮散ランダム参照計算, (c) 連続アドレス累算計算

```

#define cnn3x3_core1(b, bp1, n) \
    mop(OP_LDWR, 1, &BR[b][0][1], (ULL)kp0[CHIP], o, MSK_D0, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][0][0], (ULL)kp1[CHIP], o, MSK_D0, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][1], (ULL)kp2[CHIP], o, MSK_D0, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][0], (ULL)kp3[CHIP], o, MSK_D0, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][2][1], (ULL)kp0p[CHIP], oofs, MSK_W0, (ULL)ito1[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_FMA, &BR[bp1][0], AR[b][0], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &BR[bp1][1], AR[b][1], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &BR[bp1][2], AR[b][2], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &BR[bp1][3], AR[b][3], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);

#define cnn3x3_final(b, bp1) \
    mope(OP_LDWR, 1, &BR[bp1][0][1], (ULL)op0[CHIP], oofs, MSK_W0, (ULL)ot0[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_LDWR, 1, &BR[bp1][1][1], (ULL)op1[CHIP], oofs, MSK_W0, (ULL)ot1[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_LDWR, 1, &BR[bp1][2][1], (ULL)op2[CHIP], oofs, MSK_W0, (ULL)ot2[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_LDWR, 1, &BR[bp1][3][1], (ULL)op3[CHIP], oofs, MSK_W0, (ULL)ot3[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    exe(OP_FAD, &BR[bp1][0], AR[b][0], EXP_H3210, BR[bp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &BR[bp1][1], AR[b][1], EXP_H3210, BR[bp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &BR[bp1][2], AR[b][2], EXP_H3210, BR[bp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &BR[bp1][3], AR[b][3], EXP_H3210, BR[bp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    mope(OP_STWR, 1, &BR[bp1][0], oofs, (ULL)op1[CHIP], MSK_D0, (ULL)ot0[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_STWR, 1, &BR[bp1][1], oofs, (ULL)op1[CHIP], MSK_D0, (ULL)ot1[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_STWR, 1, &BR[bp1][2], oofs, (ULL)op2[CHIP], MSK_D0, (ULL)ot2[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mope(OP_STWR, 1, &BR[bp1][3], oofs, (ULL)op3[CHIP], MSK_D0, (ULL)ot3[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);

```

メモリ参照/演算パターン：(a) 離散ランダム参照計算、(b) 連續アドレス参照計算

```

#define cnn2x2_core1(b, o, bpl, n) \
    mop(OP_LDWR, 1, &BR[b][0][0], (ULL)kp0[CHIP], o, MSK_DO, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][0][0], (ULL)kp1[CHIP], o, MSK_DO, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][1], (ULL)kp2[CHIP], o, MSK_DO, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][1][0], (ULL)kp3[CHIP], o, MSK_DO, (ULL)i_ker, Klen, 0, Force, (ULL)NULL, Klen);\ \
    mop(OP_LDWR, 1, &BR[b][2][1], (ULL)op0[n], iofs, MSK_W1, (ULL)ito0, IMlen, 0, 0, (ULL)NULL, IMlen);\ \
    exe(OP_FMA, &AR[bp1][0], AR[b][0], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][1], AR[b][1], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][2], AR[b][2], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FMA, &AR[bp1][3], AR[b][3], EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL)

#define cnn2x2_final(b, bp1) \
    mop(OP_LDWR, 1, &BR[bp1][0][1], (ULL)op0[CHIP], oofs, MSK_W0, (ULL)ot0[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][1][1], (ULL)op1[CHIP], oofs, MSK_W0, (ULL)ot1[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][2][1], (ULL)op2[CHIP], oofs, MSK_W0, (ULL)ot2[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mop(OP_LDWR, 1, &BR[bp1][3][1], (ULL)op3[CHIP], oofs, MSK_W0, (ULL)ot3[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    exe(OP_FAD, &AR[bp1][0], AR[b][0], EXP_H3210, BR[bp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][1], AR[b][1], EXP_H3210, BR[bp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][2], AR[b][2], EXP_H3210, BR[bp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    exe(OP_FAD, &AR[bp1][3], AR[b][3], EXP_H3210, BR[bp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
    mop(OP_STWR, 1, &AR[bp1][0], oofs, (ULL)op0[CHIP], MSK_DO, (ULL)ot0[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][1], oofs, (ULL)op1[CHIP], MSK_DO, (ULL)ot1[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][2], oofs, (ULL)op2[CHIP], MSK_DO, (ULL)ot2[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);\ \
    mop(OP_STWR, 1, &AR[bp1][3], oofs, (ULL)op3[CHIP], MSK_DO, (ULL)ot3[CHIP], Mlen, 0, 1, (ULL)NULL, Mlen);

```

mop(BP_SIWR, 1, &AR[bp1][3], 001s, (011)0p3[CHIP], MSK_D0, (011)0p3[CHIP], 001s, 001s)

```
#define sgemmOO_core1(r, rm1, rp1) \
mop(OP_LDWR, 1, &BR[r][0][1], (U11)b0[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
mop(OP_LDWR, 1, &BR[r][0][0], (U11)b1[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
mop(OP_LDWR, 1, &BR[r][1][1], (U11)b2[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
mop(OP_LDWR, 1, &BR[r][1][0], (U11)b3[rm1], (U11)cofs, MSK_W1, (U11)b[rm1], Blen, 0, 0, (U11)NULL, Blen);\ \
mop(OP_LDWR, 1, &BR[r][2][1], (U11)a[rm1][CHIP], (U11)rofs, MSK_W1, (U11)a0[CHIP], Alen, 0, 0, (U11)NULL, Alen);\ \
exe(OP_FMA, &AR[rp1][0], AR[r][0], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][0][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FMA, &AR[rp1][1], AR[r][1], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][0][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FMA, &AR[rp1][2], AR[r][2], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][1][1], EXP_H3210, OP_NOP, OLI, OP_NOP, OLL);\ \
exe(OP_FMA, &AR[rp1][3], AR[r][3], EXP_H3210, BR[r][2][1], EXP_H3210, BR[r][1][0], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL)\ \

```

メモリ参照/演算パターン: (b) 離散ランダム参照計算, (c) 連続アドレス累算計算

```
#define sgemmOO_final(r, rp1) \
exe(OP_CMP_LT, &cc1, cof, EXP_H3210, cofslimit1, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_CMP_LT, &cc2, cof, EXP_H3210, cofslimit2, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_CMP_LT, &cc3, cof, EXP_H3210, cofslimit3, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
mop(OP_LDWR, 1, &BR[rp1][0][1], (U11)c00[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
mop(OP_LDWR, 1, &BR[rp1][1][1], (U11)c01[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
mop(OP_LDWR, 1, &BR[rp1][2][1], (U11)c02[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
mop(OP_LDWR, 1, &BR[rp1][3][1], (U11)c03[CHIP], (U11)oofs, MSK_W0, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
exe(OP_FAD, &AR[rp1][0], AR[r][0], EXP_H3210, BR[rp1][0][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FAD, &AR[rp1][1], AR[r][1], EXP_H3210, BR[rp1][1][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FAD, &AR[rp1][2], AR[r][2], EXP_H3210, BR[rp1][2][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
exe(OP_FAD, &AR[rp1][3], AR[r][3], EXP_H3210, BR[rp1][3][1], EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL);\ \
mop(OP_STWR, 1, &BR[rp1][0], (U11)oofs, (U11)c00[CHIP], MSK_DO, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
cex(OP_CEXE, &ex1, 0, 0, 0, cc1, 0xaaaa);\ \
mop(OP_STWR, ex1, &AR[rp1][1], (U11)oofs, (U11)c01[CHIP], MSK_DO, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
cex(OP_CEXE, &ex2, 0, 0, 0, cc2, 0xaaaa);\ \
mop(OP_STWR, ex2, &AR[rp1][2], (U11)oofs, (U11)c02[CHIP], MSK_DO, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN);\ \
cex(OP_CEXE, &ex3, 0, 0, 0, cc3, 0xaaaa);\ \
mop(OP_STWR, ex3, &AR[rp1][3], (U11)oofs, (U11)c03[CHIP], MSK_DO, (U11)c0[CHIP], CLEN, 0, 1, (U11)NULL, CLEN)
```

メモリ参照/演算パターン: (d) 固定アドレス累算計算

```
#define back_g_ker_core1(b, o, i) \
exe(OP_CMP_LT, &cc0[o][i].onum[o], EXP_H3210, OC, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */\ \
exe(OP_CMP_LT, &cc1[o][i].inum[i][CHIP], EXP_H3210, IC, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#1 */\ \
mop(OP_LDWR, 1, &BR[b][i][1], (U11)o0[i], oofs, MSK_W0, (U11)o0[i], Mlen, 0, 0, NULL, Mlen); /* stage#2 */\ \
mop(OP_LDWR, 1, &BR[b][i][1], (U11)o1[i][CHIP], (U11)o0[i], iofs, MSK_W1, (U11)o1[i][CHIP], IMlen, 0, 0, NULL, IMlen); /* stage#2 */\ \
exe(OP_NOP, &AR[b][0], OLL, EXP_H3210, OLL, EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */\ \
mop(OP_LDWR, 1, &b0o, (U11)o0[i][1][CHIP], OLL, MSK_W0, (U11)o0[i][1][CHIP], 1LL, 0, 1, NULL, 1LL); /* stage#2 */\ \
exe(OP_FMA, &b0o, b0o, EXP_H3210, BR[b][2][1], EXP_H3210, BR[b][1][1], EXP_H3210, OP_NOP, OLL, OP_NOP, OLL); /* stage#2 */\ \
cex(OP_CEXE, &ex0, 0, 0, cc0[o][i], cc0[o][i], 0x8888); /* stage#2 */\ \
mop(OP_STWR, ex0, &b0o, (U11)o0[i][1][CHIP], OLL, MSK_DO, (U11)o0[i][1][CHIP], 1LL, 0, 1, NULL, 1LL); /* stage#2 */\ \

```

メモリ参照/演算パターン: (b) 離散ランダム参照計算, (c) 連続アドレス累算計算

A.5 Compilation of IMAX

LU分解 起動条件やら空間分割のヒントを追加

20220216
1

```

for (i=0; i<M; i++) {
    if (len < 1) {
        for (j=i+1; j<M; j+=NCHIP*H) [ 長さを見てARMかIMAXか切り替える
            for (CHIP=0; CHIP<NCHIP; CHIP++) [
                for (h=0; h<H; h++) {
                    for (k=0; k<M-(i+1); k++) [
                        if (j+h*NCHIP+CHIP*M) A[p[j+h*NCHIP+CHIP]*M+i+k] -= A[p[j+h*NCHIP+CHIP]*M+i]*A[p[i]*M+i+k];
                    ]
                ]
            ]
        }
        else [
            for (j=i+1; j<M; j+=NCHIP*H) [
//EMAX5A begin inv_x1 mapdist=0
                for (CHIP=0; CHIP<NCHIP; CHIP++) [
                    for (INIT=1, LOOPU=M-(i+1), cofis=0-4; LOOPU--; INIT=0) [
                        exe(OP_ADD, &cofs, cofs, 4LL, OLL, OP_AND, 0x00000000ffffffffLL);
                        :
                        exe(OP_CMP_LT, &cof0, 100[CHIP], M, OLL, );
                        mop(OP_LDUMR, &BR1[2][1], top, cofs, top, len); ← 主記憶から
                        mop(OP_LDUMR, &BR1[0][1], d00[CHIP], cofs, d00w[CHIP], len); ← 主記憶から
                        exe(OP_FMS, &a, BR1[0][1], t00[CHIP], BR1[2][1]);
                        cex(OP_CEXE, &ex0, 0, 0, cof0, 0xaaaa);
                        mop(OP_STWR, ex0, &a, cofs, d00[CHIP], d00w[CHIP], len); ← 主記憶へ
                        :
                        exe(OP_CMP_LT, &cof0, 102[CHIP], M, OLL, );
                        mop(OP_LDUMR, &BR3[2][1], top, cofs, top, len); ← 主記憶から
                        mop(OP_LDUMR, &BR3[0][1], d02[CHIP], cofs, d02w[CHIP], len); ← 主記憶から
                        exe(OP_FMS, &a, BR3[0][1], t02[CHIP], BR3[2][1]);
                        cex(OP_CEXE, &ex0, 0, 0, cof0, 0xaaaa);
                        mop(OP_STWR, ex0, &a, cofs, d02[CHIP], d02w[CHIP], len); ← 主記憶へ
                    ]
                ]
//EMAX5A end
            ]
//EMAX5A drain_dirty_llm
        ]
    ]
}

```

Figure.A.9: Compilation step1

コンパイル過程 src/conv-mark/conv-mark

20220216
2

```

> filter+rmm.c

void tone_curve(r, d, t)
    unsigned int *r, *d;
    unsigned char *t;
{
#if !defined(EMAX5) && !defined(EMAX6)
    int j;
    for (j=0; j<WD; j++) {
        *d = ((t[j]>>24)<<24 | (t[256+((r>>16)&255)]<<16 | (t[512+((r>>8)&255)]<<8;
        r++; d++;
    }
#else
    U11 t1 = t;
    U11 t2 = t+256;
    U11 t3 = t+512;
    U11 BR[16][4][4]; /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    int loop=WD;
//EMAX5A begin tone_curve mapdist=0
    while (loop--) {
        mop(OP_LDUR, 1, &BR[0][1][1], (U11)(r++), OLL, MSK_D0, (U11)r, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
        mop(OP_LDUBR, 1, &BR[1][1][1], (U11)t1, BR[0][1][1], MSK_B3, (U11)t1, 64, 0, 0, (U11)NULL, 64); /* stage#1 */
        mop(OP_LDUBR, 1, &BR[1][2][1], (U11)t2, BR[0][1][1], MSK_B2, (U11)t2, 64, 0, 0, (U11)NULL, 64); /* stage#1 */
        mop(OP_LDUBR, 1, &BR[1][3][1], (U11)t3, BR[0][1][1], MSK_B1, (U11)t3, 64, 0, 0, (U11)NULL, 64); /* stage#1 */
        exe(OP_MMRC, &r1, BR[1][1][1], EXP_H3210, BR[1][2][1], EXP_H3210, BR[1][3][1], EXP_H3210, OLL, OP_NOP, OLL);
        mop(OP_STWR, 3, &r1, (U11)(d++), OLL, MSK_D0, (U11)d, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
    }
//EMAX5A end
#endif
}

```

Figure.A.10: Compilation step2

コンパイル過程 cpp -P

20220216
3

```
> filter+rmm.c-mark.c

void tone_curve(r, d, t)
    unsigned int *r, *d;
    unsigned char *t;
{
#if !defined(EMAX5) && !defined(EMAX6)
    int j;
    for (j=0; j<WD; j++) {
        *d = ((t)[*r>>24])<<24 | (t[256+(*r>>16)&255])<<16 | (t[512+(*r>>8)&255])<<8;
        r++; d++;
    }
#else
    U11 t1 = t;
    U11 t2 = t+256;
    U11 t3 = t+512;
    U11 BR[16][4][4]; /* output registers in each unit */
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    int loop=WD;
#define printf(format,...)
/*EMAX5AB-/ tone_curve @
/*EMAX5AS-/ while (loop--) {
/*EMAX5AS-/     mop(OP_LDWR, 1, &BR[0][1][1], (U11)(r++), 0LL, MSK_D0, (U11)r, 320, 0, 0, (U11)NULL, 320); /* stage#0 */
/*EMAX5AS-/     mop(OP_LDUBR, 1, &BR[1][1][1], (U11)t1, BR[0][1][1], MSK_B3, (U11)t1, 64, 0, 0, (U11)NULL, 64); /* stage#1 */
/*EMAX5AS-/     mop(OP_LDUBR, 1, &BR[1][2][1], (U11)t2, BR[0][1][1], MSK_B2, (U11)t2, 64, 0, 0, (U11)NULL, 64); /* stage#1 */
/*EMAX5AS-/     mop(OP_LDUBR, 1, &BR[1][3][1], (U11)t3, BR[0][1][1], MSK_B1, (U11)t3, 64, 0, 0, (U11)NULL, 64); /* stage#1 */
/*EMAX5AS-/     exe(OP_MMRG, &r1, BR[1][1][1], EXP_H3210, BR[1][2][1], EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
/*EMAX5AS-/     mop(OP_STWR, 3, &r1, (U11)(d++), 0LL, MSK_D0, (U11)d, 320, 0, 0, (U11)NULL, 320); /* stage#2 */
/*EMAX5AS-/ }
/*EMAX5AE-/ #undef printf
#endif
}
```

Figure.A.11: Compilation step3

コンパイル過程 src/conv-c2c/conv-c2c

20220216
4

```
> filter+rmm.c-cppo.c

void tone_curve(r, d, t)
    unsigned int *r, *d;
    unsigned char *t;
{
    U11 t1 = t;
    U11 t2 = t+256;
    U11 t3 = t+512;
    U11 BR[16][4][4];
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
    U11 r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    int loop=WD;
#define tone_curve @
/*EMAX5AS-/ while (loop--) {
/*EMAX5AS-/     mop(0x02, 1, &BR[0][1][1], (U11)(r++), 0LL, 14, (U11)r, 320, 0, 0, (U11)((void *)0), 320);
/*EMAX5AS-/     mop(0x07, 1, &BR[1][1][1], (U11)+1, BR[0][1][1], 3, (U11)+1, 64, 0, 0, (U11)((void *)0), 64);
/*EMAX5AS-/     mop(0x07, 1, &BR[1][2][1], (U11)+2, BR[0][1][1], 2, (U11)+2, 64, 0, 0, (U11)((void *)0), 64);
/*EMAX5AS-/     mop(0x07, 1, &BR[1][3][1], (U11)+3, BR[0][1][1], 1, (U11)+3, 64, 0, 0, (U11)((void *)0), 64);
/*EMAX5AS-/     exe(0x25, &r1, BR[1][1][1], 3, BR[1][2][1], 3, BR[1][3][1], 3, 0x00, 0LL, 0x00, 0LL);
/*EMAX5AS-/     mop(0x12, 3, &r1, (U11)(d++), 0LL, 14, (U11)d, 320, 0, 0, (U11)((void *)0), 320);
/*EMAX5AS-/ }
/*EMAX5AE-/ }
```

Figure.A.12: Compilation step4

コンパイル過程 通常のARM-Cコンパイラ 1/3

20220216
5

```
> filter+rmm-emax6.c ../../src/conv-c2c/emax6.h ../../src/conv-c2c/emax6lib.c

void tone_curve(r, d, t)
    unsigned int *r, *d;
    unsigned char *t;
{
    U11 t1 = t;
    U11 t2 = t+256;
    U11 t3 = t+512;
    U11 BR[16][4][4];
    U11 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
    int loopNDB;
    volatile emax6_conf_tone_curve();
    emax6_lmmio = emax6_lmmic;
    emax6_lmmic = 1-emax6_lmmic;
    emax6_mapdist = 0;
    *(Uint*)&emax6_lmmi[0][0][1][emax6_lmmic] = 0x013f0001|(0<<2);
    emax6_lmmi[0][0][1][emax6_lmmic].ofs = 0; emax6_lmmi[0][0][1][emax6_lmmic].top = r;
    *(Uint*)&emax6_lmmi[0][1][1][emax6_lmmic] = 0x003f0001|(0<<2);
    emax6_lmmi[0][1][1][emax6_lmmic].ofs = 0; emax6_lmmi[0][1][1][emax6_lmmic].top = t1;
    *(Uint*)&emax6_lmmi[0][1][2][emax6_lmmic] = 0x003f0001|(0<<2);
    emax6_lmmi[0][1][2][emax6_lmmic].ofs = 0; emax6_lmmi[0][1][2][emax6_lmmic].top = t2;
    *(Uint*)&emax6_lmmi[0][1][3][emax6_lmmic] = 0x003f0001|(0<<2);
    emax6_lmmi[0][1][3][emax6_lmmic].ofs = 0; emax6_lmmi[0][1][3][emax6_lmmic].top = t3;
    *(Uint*)&emax6_lmmi[0][2][0][emax6_lmmic] = 0x013f0003|(0<<2);
    emax6_lmmi[0][2][0][emax6_lmmic].ofs = 0; emax6_lmmi[0][2][0][emax6_lmmic].top = d;
    emax6_lmmi_bitmap[0] = 0x00000000000000411;
    emax6_lmmi_bitmap[1] = 0x00000000000000311;
    emax6_lmmi_bitmap[2] = 0x00000000000000211;
    emax6_lmmi_bitmap[3] = 0x00000000000000211;
    emax6_pre_with_drain_cache();
    get_nanosec(NANOS_ARM);
    if (emax6.last_conf == emax6_conf_tone_curve) {
        emax6.status = STATUS_DRAIN;
        emax6_check_lmmi_and_dma(0, 1, 0, 0, 2, 0); /*drain*/
    }
    get_nanosec(NANOS_DRAIN);
}
```

LMMのアドレス範囲情報

必要に応じて前回演算結果の回収

Figure.A.13: Compilation step5

コンパイル過程 通常のARM-Cコンパイラ 2/3

20220216
6

```
if (emax6.last_conf != emax6_conf_tone_curve) {
    DIL "dst, *src";
    int i,j;
    emax6.status = STATUS_CONF;
    emax6.last_conf = emax6_conf_tone_curve;
    emax6.lastdist = 0;
    dst = (DIL*)((struct reg_ctrl*)emax6.reg_ctrl)->i[0].conf;
    src = (DIL*)emax6_conf_tone_curve;
    for (i=0; i<sizeof(conf)/sizeof(DIL); i++)
        *dst++ = *src++;
    for (i=0; i<64; i++) {
        for (j=0; j<4; j++)
            emax6_lmmi[0][i][j][emax6_lmmic].v = 0;
    }
    while (((struct reg_ctrl*)emax6.reg_ctrl)->i[0].stat & 0xf0); //LMRING_BUSY
}
get_nanosec(NANOS_CONF);
emax6.status = STATUS_REGV;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].breg[63][0].br[0] = loop;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].breg[63][0].br[1] = -1LL;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[0][1].ea1b = (U11)t;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[0][1].ea0b = (U11)t0LL;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[1][1].ea1b = (U11)t1;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[1][2].ea1b = (U11)t2;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[1][3].ea1b = (U11)t3;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[2][0].ea0b = (U11)d;
((struct reg_ctrl*)emax6.reg_ctrl)->i[0].addr[2][0].ea0e = (U11)t0LL;
get_nanosec(NANOS_REGV);
emax6.status = STATUS_RANGE;
(struct reg_ctrl *reg_ctrl = emax6.reg_ctrl;
    Uint lmmic = emax6_lmmic;
    *(U11*)&(reg_ctrl->i[0].addr[0][1].top)+=(U11)(emax6_lmmi[0][0][1][lmmic].top+*((Ushort*)&emax6_lmmi[0][0][1][lmmic]+1)*sizeof(Uint)+(sizeof(Uint)-1)<<32)| (U11)(Uint)emax6_lmmi[0][0][1][lmmic];
    *(U11*)&(reg_ctrl->i[0].addr[1][1].top)+=(U11)(emax6_lmmi[0][1][1][lmmic]+1)*sizeof(Uint)+(sizeof(Uint)-1)<<32)| (U11)(Uint)emax6_lmmi[0][1][1][lmmic];
    *(U11*)&(reg_ctrl->i[0].addr[1][2].top)+=(U11)(emax6_lmmi[0][1][2][lmmic]+1)*sizeof(Uint)+(sizeof(Uint)-1)<<32)| (U11)(Uint)emax6_lmmi[0][1][2][lmmic];
    *(U11*)&(reg_ctrl->i[0].addr[1][3].top)+=(U11)(emax6_lmmi[0][1][3][lmmic].top+*((Ushort*)&emax6_lmmi[0][1][3][lmmic]+1)*sizeof(Uint)+(sizeof(Uint)-1)<<32)| (U11)(Uint)emax6_lmmi[0][1][3][lmmic];
    *(U11*)&(reg_ctrl->i[0].addr[2][0].top)+=(U11)(emax6_lmmi[0][2][0][lmmic].top+*((Ushort*)&emax6_lmmi[0][2][0][lmmic]+1)*sizeof(Uint)+(sizeof(Uint)-1)<<32)| (U11)(Uint)emax6_lmmi[0][2][0][lmmic];
}
```

命令写像が前回と異なる場合は
再写像

AXIIF/PIOによるレジスタ初期化

LMMにアドレス範囲情報書き込み

Figure.A.14: Compilation step6

コンパイル過程 通常のARM-Cコンパイラ 3/3

20220216

7

```

emax6.status = STATUS_LOAD;
emax6_check_lmmi_and_dma(0, 2, emax6.lastdist, 0, 0, 1);/*load*/
emax6_check_lmmi_and_dma(0, 2, emax6.lastdist, 0, 1, 1);/*load*/
emax6_check_lmmi_and_dma(0, 2, emax6.lastdist, 0, 1, 2);/*load*/
emax6_check_lmmi_and_dma(0, 2, emax6.lastdist, 0, 1, 3);/*load*/
get_nanosec(NANOS_LOAD);

((struct reg_ctrl*)emax6.reg_ctrl)->i[0].cmd = 3LL; // EXEC
{struct reg_ctrl *reg_ctrl = emax6.reg_ctrl;
 Uint lmmic      = emax6.lmmic;
}

emax6.lmmd[2][0] = 0xff>>7;
while (((struct reg_ctrl*)emax6.reg_ctrl)->i[0].stat); //LMRING_BUSY|EXRING_BUSY
get_nanosec(NANOS_EXEC);

asm volatile("b emax6_conf_end_tone_curve\n"
".align 5\n"
".global emax6_conf_tone_curve\n"
"emax6_conf_tone_curve:\n"
"    .word    0x031e0003, 0x00000000\n"
"    .word    0xffff0000, 0x00000000\n"
"    .word    0x00000000, 0x00000000\n"
"    :
"    .word    0xffff0000, 0x00000000\n"
"    .word    0x00000000, 0x00000000\n"
"    .word    0x00000000, 0x00000000\n"
".global emax6_conf_end_tone_curve\n"
"emax6_conf_end_tone_curve:\n"
);
}

```

AXIIF/DMAによる
LMMデータ書き込み

AXIIF/PIOによるIMAX起動

演算と同時に次の次データ転送があればDMA起動

UNITのconfiguration情報

Figure.A.15: Compilation step7

A.6 References

本章では、関連仕様書・規格、参考文献、関連ソースプログラム、および、ツールチェインを列挙する。

- EMAX5 基本特許 proj-arm64/doc/pat35.tgz
- IMAX 基本特許 proj-arm64/doc/pat36.tgz
- ARMv8 アーキテクチャ仕様書 proj-arm64/doc/arm/DDI0487A_f_armv8_arm.pdf
- ARM Cortex-A53 MPCore Processor Technical Reference Manual proj-arm64/doc/arm/ARM-CORTEX-A53_R0P4.pdf
- ZYNQ Ultrascale+ SoC Technical Reference Manual proj-board/zcu102/doc/ug1085-zynq-ultrascale-trm.pdf
- AMBA AXI4 and ACE Protocol Specification proj-arm64/doc/arm/AXI4_specification.pdf
- FMC ケーブル基板回路図 proj-arm64/doc/sansei/FMC ケーブル基板回路図.pdf
- FMC ケーブル接続 proj-arm64/doc/sansei/FMC ケーブル接続.xlsx
- IMAX 仕様書 proj-arm64/doc/emax6/emax6.pdf
- IMAX C 言語ディレクトイブ変換 proj-arm64/src/conv-c2c/conv-c2c
- IMAX シミュレータ proj-arm64/src/csim/csim
- プログラム例 (FFT) proj-arm64/sample/fft/fourierf.c
- プログラム例 (SORT) proj-arm64/sample/sort/sort-merge.c
- プログラム例 (文字列検索) proj-arm64/sample/stringsearch/pbmsrch+rmm.c
- プログラム例 (16x16 署み込み) proj-arm64/sample/conv16/conv16.c
- プログラム例 (VBGMM) proj-arm64/sample/test/test016.c
- プログラム例 (Stochastic 計算) proj-arm64/sample/test/test021.c
- プログラム例 (疎行列行列積) proj-arm64/sample/test/test022.c
- プログラム例 (疎行列圧縮) proj-arm64/sample/test/test024.c
- プログラム例 (画像フィルタ) proj-arm64/sample/filter/filter+rmm.c
- プログラム例 (浮動小数点ステンシル) proj-arm64/sample/stencil/stencil+rmm.c
- プログラム例 (3x3 署み込み) proj-arm64/sample/mm_cnn_if/cnn+rmm.c
- プログラム例 (行列積) proj-arm64/sample/mm_cnn_if/mm+rmm.c
- プログラム例 (逆行列) proj-arm64/sample/mm_cnn_if/inv+rmm.c
- プログラム例 (Lightfield レンダリング) proj-arm64/sample/mm_cnn_if/gather+rmm.c
- プログラム例 (Lightfield 距離画像生成) proj-arm64/sample/mm_cnn_if/gdepth+rmm.c
- プログラム例 (グラフ処理 EMAX5) proj-arm64/sample/tricount8/tricount.c
- プログラム例 (グラフ処理 IMAX2) proj-arm64/sample/tricount9/tricount.c
- プログラム例 (画像認識) proj-arm64/sample/rsim/imax.c
- プログラム例 (画像認識+Stochastic 計算) proj-arm64/sample/ssim/smax.c
- プログラム例 (SHA256) proj-arm64/sample/crypto/sha256.c

A.7 Publications

1. ◆稻益秀成, 船井遼太郎, 中島康彦: ”リニアアレイ型 CGRA の高速コンパイルを利用した JIT 実行環境の開発”, 電子情報通信学会論文誌 D, Vol.J105-D, No.12, Dec. (2022)
2. ◇中島康彦: ”CGRA の JIT コンパイル化と高機能化の魔法教えます”, 回路とシステムワークショウ招待講演, Aug. (2022)
3. ・赤部知也, 中島康彦: ”主記憶帯域使用率向上のための CGRA タンデム化”, 信学技報, vol.122, no.133, CPSY2022-16, pp.89-92, Jul. (2022)
4. ◆Tomoya Akabe, Hidenari Inamasu, Renyuan Zhang and Yasuhiko Nakashima: ”Fusion of Multiple Core and Just-in-Time Compilable CGRA”, IEEE Symposium on Low-Power and High-Speed Chips 2022 (poster), Apr. (2022)
5. ◆ Ryotaro Funai, Hidenari Inamasu, Renyuan Zhang and Yasuhiko Nakashima: ”Evaluation of IMAX2 with Sparse Matrix-matrix Multiplication Units”, IEEE Symposium on Low-Power and High-Speed Chips 2022 (poster), Apr. (2022)
6. ・船井遼太郎, 張任遠, 中島康彦: ”IMAX2 を用いた高効率な疎行列-疎行列積の実装”, 信学技報, vol.121, no.343, CPSY2021-25, pp.38-42, Jan. (2022)
7. ◇中島康彦: ”非ノイマン型の世界 -CGRA を含む最近の研究紹介-”, JEITA デバイス技術分科会招待講演, Nov. (2021)
8. ◆ Tomoya Akabe, Renyuan Zhang, and Y. Nakashima: ”Speeding Up of CGRAs by Reshaping and Stochastic FMA”, CANDAR'21, SUSCW (Sustainable Computing Systems) workshop, Nov. (2021)
9. ◆ [Best Student Paper Award] Tran Thi Diem and Yasuhiko Nakashima: ”Exploring Versatility of Primary Visual Cortex Inspired Feature Extraction Hardware Model through Various Network Architectures”, 4th International Conference on Computing, Electronics & Communications Engineering, iCCECE '21, Aug. (2021)
10. ・赤部知也, 中島康彦: ”シストリックアレイ向け確率的コンピューティングの予備評価”, 信学技報, vol.121, no.116, CPSY2021-9, pp.49-52, Jul. (2021)
11. ◇ Yasuhiko Nakashima: ”IMAX2: A CGRA with FPU+Multithreading+Chiplet”, Panel: Coarse-Grained Reconfigurable Arrays and their Opportunities as Application Accelerators, ASAP2021, invited panel, Jul. (2021)
12. ◇ Tomoya Akabe and Hidenari Inamasu: ”IMAX2: A CGRA with FPU+Multithreading+Chiplet”, ASAP2021 poster, Jul. (2021)
13. ◆ Tran Thi Diem and Yasuhiko Nakashima: ”SLIT: An Energy-Efficient Reconfigurable Hardware Architecture for Deep Convolutional Neural Networks”, IEICE Trans., Vol.E104-C, No.7, pp.319-329, Jul. (2021)
14. ◆ [Featured Poster Award] Tomoya Akabe, Mutsumi Kimura, Yasuhiko Nakashima: ”Evaluation of Narrow Bit-Width Variation for Training Neural Networks”, IEEE Symposium on Low-Power and High-Speed Chips 2021 (poster), Apr. (2021)
15. ・中島康彦: ”IMAX2: GTH の 8 レーン化を契機とする IMAX の倍速化”, 信学技報, vol.120, no.338, CPSY2020-27, pp.31-34, Jan. (2021)
16. ・稻益秀成, 中島康彦: ”シストリッククリングアレイ (IMAX2) を用いた高効率誤差逆伝搬の実装”, 信学技報, vol.120, no.338, CPSY2020-28, pp.35-39, Jan. (2021)
17. ◆ Taku Honda, Hiroki Nishimoto, Yasuhiko Nakashima: ”Speeding Up VBGMM By Using Log-sumexp With the Approximate Exp-function”, CANDAR'20, poster, Nov. (2020)
18. ◆ [Best Paper Award] Tran Thi Diem, Mutsumi Kimura and Yasuhiko Nakashima: ”Primary Visual Cortex Inspired Feature Extraction Hardware Model”, SigTelCom2020, Aug. (2020)
19. ◇中島康彦: ”好きなことを頑固に素早く”, 情報・システムソサイエティ誌 フェローからのメッセージ, Vol.25, No.2, pp.19-20, Aug. (2020)

20. ♦ Jun Iwamoto, Yuma Kikutani, Renyuan Zhang and Yasuhiko Nakashima: "Daisy-chained Systolic Array and Reconfigurable Memory Space for Narrow Memory Bandwidth", IEICE Trans., Vol.E103-D, No.03, pp.578-589, Mar. (2020)
21. 中島康彦: "動画認識フロントエンドを想定した特徴抽出専用ハードウェアの構想", 信学技報, vol.119, no.372, CPSY2019-75, pp.147-150, Jan. (2020)
22. ♦ Jun Iwamoto, Renyuan Zhang and Yasuhiko Nakashima: "Evaluation of a Chained Systolic Array with High-Speed Links", Proc. 7'th Int'l Workshop on Computer Systems and Architectures(CSA19), Nov. (2019)
23. 本田卓, 岩本淳, 中島康彦: "リニアアレイによる逆行列計算の高速化手法と評価", 情報処理学会研究報告, Vol.2019-ARC-237, No.15, Jul. (2019)
24. ♦ Takahiro ICHIKURA, Yuma KIKUTANI, and Yasuhiko NAKASHIMA: "DSA 並みの効率を達成する CNNs 拡張機能付き CGRA の提案と評価", "A Proposal and Evaluation of a CGRA with CNNs Extension for Near Efficiency to DSA", IEICE Trans., Vol.J102-D, No.07, pp.477-490, Jul. (2019)
25. 中島康彦: "CGLA における高速コンパイルとチューニングのためのアーキテクチャ支援", 信学技報, vol.119, no.76, CPSY2019-9, pp.71-76, Jun. (2019)
26. ♦ 【Outstanding Originality Award】 Jun IWAMOTO, Yuma KIKUTANI, Renyuan ZHANG, and Yasuhiko NAKASHIMA: "CGRA Cascading for Narrow Memory Bandwidth and Low Cost", xSIG 2019: The 3rd. cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming, May. (2019)
27. ♦ Yasuhiko Nakashima: "Systolic Arrays as The Last Frontiers", Invited talk in IPB Seminar and UI seminar @ Indonesia, Jan. (2019)
28. ♦ 中島康彦: "AI 専用ハードを横目に見ながらやるべきこと", 信学技報, vol.118, no.339, CPSY2018-37, pp.3-8, Dec. (2018)
29. 岩本淳, 菊谷雄真, 中島康彦: "ユニット内フィードバックによるリニアアレイの多重ループ対応手法", 信学技報, vol.118, no.339, CPSY2018-40, pp.33-38, Dec. (2018)
30. ♦ 中島康彦: "ソザイエティ人図鑑 No.22 中島康彦さん (CPSY 研究会)", 情報・システムソサイエティ誌, Vol.23, No.2, pp.4-7, Oct. (2018)
31. ♦ Yasuhiko Nakashima: "The End of Normal Computing Era -The Opportunity of Next Generation Computing-", Invited speech in YNU-NAIST Summer Workshop @ Yunnan Univ., Jul. (2018)
32. ♦ Takahiro Ichikura, Ryusuke Yamano, Yuma Kikutani, Renyuan Zhang, and Yasuhiko Nakashima: "EMAXVR: A Programmable Accelerator Employing Near ALU Utilization to DSA", IEEE Symposium on Low-Power and High-Speed Chips 2018, Apr. (2018)
33. ♦ Yasuhiko Nakashima: "The End of Normal-computing Era. The Opportunity of Next Computations", International Workshop on Frontiers in Computing Systems and Wireless Communications (FOSCOM 2018), Mar. (2018)
34. 【電子情報通信学会関西支部学生会研究発表講演会奨励賞】菊谷雄真, 山野龍佑, 一倉孝宏, 中島康彦: "エッジコンピューティング向けアクセラレータの実装と評価", 電子情報通信学会関西支部第 23 回研究発表講演会, Mar. (2018)
35. 菊谷雄真, 山野龍佑, 一倉孝宏, 中島康彦: "時分割多重実行型ストリックリングの実装と評価", 信学技報, vol.117, no.378, CPSY2017-111, pp.31-36, Jan. (2018)
36. ♦ 中島康彦: "Approximate Computing とストリックアレイ", ジスクソフト技術講演会, Dec. (2017)
37. ♦ 中島康彦: "Deep Learning に向けた Approximate Computing とストリックアレイアーキテクチャ", 革新的コンピューティングの研究開発戦略検討会, JST, Jul. (2017)
38. ♦ 中島康彦: "Google の TPU にも使われたストリックアレイアーキテクチャと Deep Learning について", 富士通研究所技術講演会, Jul. (2017)
39. 福岡久和, 山野龍佑, 中島康彦: "各種 FPGA による畳み込み演算向けストリックリングの実装と評価", CPSY 研究会, 2017-05-23, May. (2017)

40. ・山野龍佑, 中島康彦: "時分割多重実行によるシストリックリングの面積効率向上手法", 信学技報, vol.117, no.44, CPSY2017-6, pp.27-32, May. (2017)
41. ◇中島康彦: "99CAE 計算環境研究会@関西シスラボ 第8回シンポジウム, Mar. (2017)
42. ・一倉孝宏, 山野龍佑, 福岡久和, 中島康彦: "DCNN に最適な CGRA の探索と予備評価", 信学技報, vol.116, no.416, CPSY2016-114, pp.49-54, Jan. (2017)
43. ◆ Yuttakon YUTTAKONKIT, Shinya TAKAMAEDA-YAMAZAKI and Yasuhiko NAKASHIMA: "Performance Comparison of CGRA and Mobile GPU for Light-field Image Processing", CAN-DAR'16, REGULAR PAPER, pp.174-180, Nov. (2016)
44. ・中島康彦: "EMAXV における複数バースト転送と複数ベクトル演算のオーバラップ手法", 信学技報, CPSY2016-15, pp.71-76, Aug. (2016)
45. ・中島康彦: "アルゴリズム記述と CGRA 実装を統合する C 言語フレームワーク", 信学技報, vol.115, no.342, CPSY2015-65, pp.21-26, Dec. (2015)
46. ・竹内昌平, TRAN Thi Hong, 高前田伸也, 中島康彦: "低消費電力 CGRA EMAX の Zynq を用いた実機評価", 信学技報, vol.115, no.243, CPSY2015-51, pp.39-41, Oct. (2015)
47. ◆ Shohei Takeuchi, Yuttakon Yuttakonkit, Shinya Takamaeda, Yasuhiko Nakashima: "A Distributed Memory Based Embedded CGRA for Accelerating Stencil Computations", Proc. 3rd Int'l Workshop on Computer Systems and Architectures(CSA15), pp.378-384, Dec. (2015)
48. ・Yuttakon Yuttakonkit, Tran Thi Hong, Shinya Takamaeda-Yamazaki, Yasuhiko Nakashima: "Design Space Exploration of Computational Photography Accelerator", 信学技報 CPSY2015-17 SwoPP 論文集, pp.7-12, Aug. (2015)
49. ・竹内昌平, Tran Thi Hong, 高前田伸也, 中島康彦: "Zynq を用いた ARM-EMAX 密結合アクセラレータの評価", 信学技報 CPSY2015-19 SwoPP 論文集, pp.47-52, Aug. (2015)
50. ◆ Yoshikazu Inagaki, Shinya Takamaeda-Yamazaki, Jun Yao, Yasuhiko Nakashima: "Performance Evaluation of a 3D-Stencil Library for Distributed Memory Array Accelerators", IEICE Trans., Vol.E98-D, No.12, pp.2141-2149, Dec. (2015)
51. ◆ Masakazu Tanomoto, Shinya Takamaeda-Yamazaki, Jun Yao, Yasuhiko Nakashima: "A CGRA-based Approach for Accelerating Convolutional Neural Networks", 9th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-15) Turin, Italy, Sep.23-25, (2015)
52. ・竹内昌平, TRAN Thi Hong, 高前田伸也, 中島康彦: "グラフ処理向け CGRA in Cache の提案", 信学技報 CPSY2015-7, pp.37-41, Apr. (2015)
53. ◆ [IEEE Symposium on Low-Power and High-Speed Chips 2015 Featured Poster Award] Shohei Takeuchi, Thi Hong Tran, Shinya Takamaeda, Yasuhiko Nakashima: "A Parameterized Many Core Simulator for Design Space Exploration", IEEE Symposium on Low-Power and High-Speed Chips 2015 (poster), Apr. (2015)
54. ◆ Jun Yao, Yasuhiko Nakashima, Kazutoshi Kobayashi, Makoto Ikeda, Wei Xue, Tomohiro Fujiwara, Ryo Shimizu, Masakazu Tanomoto, Yangtong Xu, Xinliang Wang, Weimin Zheng: "XStenciler: a 7.1GFLOPS/W 16-Core Coprocessor with a Ring Structure for Stencil Applications", IEEE Symposium on Low-Power and High-Speed Chips 2015 (poster), Apr. (2015)
55. ・竹内昌平, 高前田(山崎)伸也, 姚駿, 中島康彦: "次世代アプリケーションのための包括的なアーキテクチャ探索環境の検討", 信学技報 CPSY2014-89, pp.25-27, Dec. (2014)
56. ・紅林修斗, 高前田伸也, 姚駿, 中島康彦: "最短経路探索の並列化と各種プラットホームによる性能比較", 信学技報 CPSY2014-74, pp.13-18, Nov. (2014)
57. ・清水怜, 田ノ元正和, 高前田(山崎)伸也, 姚駿, 中島康彦: "メモリネットワークベースアクセラレータの試作と評価", 信学技報 CPSY2014-81, pp.51-56, Nov. (2014)
58. ・田ノ元正和, 高前田(山崎)伸也, 姚駿, 中島康彦: "メモリネットワークベースアクセラレータを用いた畳み込みニューラルネットワーク処理", 信学技報 CPSY2014-82, pp.57-62, Nov. (2014)
59. ◆ Jun Yao, Mitsutoshi Saito, Shogo Okada, Kazutoshi Kobayashi, and Yasuhiko Nakashima:

- ”EReLA: a Low-Power Reliable Coarse-Grained Reconfigurable Architecture Processor and Its Irradiation Tests”, IEEE Transactions on Nuclear Science, Vol.61, No.6, pp.3250-3257, DOI=10.1109/TNS.2014.2367541 Dec. (2014)
60. ♦ Jun Yao, Yasuhiko Nakashima, Mitsutoshi Saito, Yohei Hazama, Ryosuke Yamanaka: ”A Flexible, Self-Tuning, Fault-Tolerant Functional Unit Array Processor”, IEEE Micro, pp.54-63, Issue 6, Dec. (2014)
61. ♦ Yoshikazu Inagaki, Shinya Takamaeda-Yamazaki, Jun Yao, Yasuhiko Nakashima: ”Performance Evaluation of a 3D-Stencil Library for Distributed Memory Array Accelerators”, Proc. 2nd Int'l Workshop on Computer Systems and Architectures (CSA'14), held in conjunction with CAN-DAR'14, Shizuoka, Japan, pp.388-393, Dec. (2014)
62. · 清水怜, 高前田(山崎)伸也, 姚駿, 中島康彦: ”メモリインテンシブアレイアクセラレータを用いた高性能グラフ処理”, 信学技報 CPSY2014-11, pp.7-12, Jul. (2014)
63. ♦ Jun YAO, Yasuhiko NAKASHIMA, Naveen DEVISSETTI, Kazuhiro YOSHIMURA, Takashi NAKADA: ”A Tightly Coupled General Purpose Reconfigurable Accelerator LAPP and Its Power States for HotSpot-Based Energy Reduction”, IEICE Trans., Vol.E97-D, No.12, pp.3092-3100, Dec. (2014)
64. ♦ Yukihiko SASAGAWA, Jun YAO, Yasuhiko NAKASHIMA: ”Understanding Variations for Better Adjusting Parallel Supplemental Redundant Executions to Tolerate Timing Faults”, IEICE Trans., Vol.J97-D, No.12, pp.3083-3091, Dec. (2014)
65. ♦ Tanvir Ahmed, Jun Yao, and Yasuhiko Nakashima: ”A Two-Order Increase in Robustness of Partial Redundancy Under a Radiation Stress Test by Using SDC Prediction”, IEEE Transactions on Nuclear Science, Vol.61, Issue.4, pp.1567-1574, DOI=10.1109/TNS.2014.2314691, Aug. (2014)
66. ♦ Jun Yao, Mitsutoshi Saito, Shogo Okada, Kazutoshi Kobayashi, and Yasuhiko Nakashima: ”EReLA: a Low-Power Reliable Coarse-Grained Reconfigurable Architecture Processor and Its Irradiation Tests”, IEEE Nuclear and Space Radiation Effects Conference 2014 (poster), Jul. (2014)
67. ♦ Shuto Kurebayashi, Jun Yao, Yasuhiko Nakashima: ”A Pipelined Newton-Raphson Method for Floating Point Division and Square Root on Distributed Memory CGRAs”, IEEE Symposium on Low-Power and High-Speed Chips 2014 (poster), Apr. (2014)
68. ♦ Jun Yao, Yasuhiko Nakashima, Mitsutoshi Saito, Yohei Hazama, Ryosuke Yamanaka: ”A Flexibly Fault-Tolerant FU Array Processor and its Self-Tuning Scheme to Locate Permanently Defective Unit”, IEEE Symposium on Low-Power and High-Speed Chips 2014, Apr. (2014)
69. · 林大地, 藤原知広, 姚駿, 中島康彦: ”演算器アレイ型アクセラレータへのメモリインテンシブなアプリケーションの写像と性能評価”, 情報処理学会研究報告, 計算機アーキテクチャ研究会報告, 2014-ARC-208(17), 1-5, Jan. (2014)
70. · 楠田浩平, 姚駿, 中島康彦: ”メモリ分散型アレイアクセラレータのための命令生成手法の開発と評価”, 情報処理学会研究報告, 計算機アーキテクチャ研究会報告, 2014-ARC-208(16), 1-7, Jan. (2014)
71. ♦ Tanvir AHMED, Jun YAO, and Yasuhiko NAKASHIMA: ”A Two-Order Increase in Robustness of Partial Redundancy Under Radiation Stress Test by Using SDC Prediction”, In 2013 IEEE Conference on Radiation Effects on Components and Systems (RADECS), C-7, pp.1-7, Sep. (2013)
72. · 稲垣慶和, 原祐子, 姚駿, 中島康彦: ”リング型アレイアクセラレータ向け演算ライブラリの実装と性能評価”, 研究報告計算機アーキテクチャ (ARC) , 2013-ARC-206, No.1, pp.1-6, Jul. (2013)
73. · 林大地, 関賀, 原祐子, 姚駿, 中島康彦: ”メモリ分散型アレイアクセラレータの浮動小数点演算に関する性能考察”, 研究報告計算機アーキテクチャ (ARC) , 2013-ARC-206, No.8, pp.1-6, Jul. (2013)
74. · 藤原知広, 姚駿, 原祐子, 中島康彦: ”リング型アレイアクセラレータのマクロパイプライン化による性能見積もり”, 研究報告計算機アーキテクチャ (ARC) , 2013-ARC-206, No.14, pp.1-6, Jul. (2013)
75. ♦ Tanvir AHMED, Jun YAO, Yuko HARA-AZUMI, Shigeru YAMASHITA, and Yasuhiko NAKASHIMA: ”Selective Check of Data-Path for Effective Fault Tolerance”, IEICE Trans., Vol.J96-D, No.8, pp.1592-1601, Aug. (2013)

76. ♦ Wei Wang, Jun Yao, Youhui Zhang, Wei Xue, Yasuhiko Nakashima, and Weimin Zheng: "HW/SW Approaches to Accelerate GRAPES in an FU Array", IEEE Symposium on Low-Power and High-Speed Chips 2013, Apr. (2013)
77. ♦大上俊, 姚駿, 中島康彦: "演算器アレイにおける高信頼化命令写像手法", IEICE Trans., Vol.J96-D, No.3, pp.472-483, Mar. (2013)
78. ◇中島康彦: "LSI 化に繋がるシミュレータ開発手法と設計事例", 計算機アーキテクチャ研究会, Mar. (2013)
79. · 王昊, 姚駿, 中島康彦: "GCC の vectorizer を利用した演算器アレイ向け命令変換手法", 研究報告計算機アーキテクチャ(ARC), 2013-ARC-203 No.9, Feb. (2013)
80. · 関賀, 姚駿, 中島康彦: "リング接続を利用しデータ移動を最小限にするアクセラレータの提案", 研究報告システム LSI 設計技術 (SLDM) SIG Technical Reports, 2013-SLDM-159, Vol.17, pp.1-6, Jan. (2013)
81. · 山中良祐, 姚駿, 中島康彦: "セレクタ部に着目した演算器アレイ型アクセラレータの高信頼化手法", 信学技報 CPSY2012-13 SwoPP 論文集, pp.25-30, Aug. (2012)
82. · Tanvir Ahmed, Jun Yao, Yasuhiko Nakashima: "Achieving Near-Optimal Dependability with Minimal Hardware Costs in an FU Array Processor by Soft Error Rate Monitoring", 研究報告計算機アーキテクチャ (ARC) ,2012-ARC-201(4),1-6, Aug. (2012)
83. · 大谷友哉, Tanvir Ahmed, 姚駿, 中島康彦: "演算器アレイにおける冗長化オーバヘッドの少ない高信頼化手法の提案", 研究報告計算機アーキテクチャ (ARC) ,2012-ARC-201(19),1-6, Aug. (2012)
84. ♦ Yukihiko SASAGAWA, Jun YAO, Takashi NAKADA, Yasuhiko NAKASHIMA: "RazorProtector: Maintaining Razor DVS Efficiency in Large IR-drop Zones by an Adaptive Redundant Data-Path", IEICE Trans. on VLSI Design and CAD Algorithms, Vol.E95-A, No.12, pp.2319-2329, Dec. (2012)
85. ♦ Tanvir Ahmed, Jun Yao, Yasuhiko Nakashima: "Introducing OVP Awareness to Achieve an Efficient Permanent Defect Locating", NANOARCH 2012, pp.43-49, Netherlands, Jul. (2012)
86. · YAO Jun, NAKASHIMA Yasuhiko: "Deep DVS in FU array by Covering Process Variations with Data-Path Auto-fix", 研究報告計算機アーキテクチャ (ARC) , Vol.2012-ARC-200, No.18, pp.1-9, May. (2012)
87. ♦齊藤光俊, 下岡俊介, Devisetti Venkatarama Naveen, 大上俊, 吉村和浩, 姚駿, 中田尚, 中島康彦: "線形演算器アレイ型アクセラレータを備えた高電力効率プロセッサの開発", 電子情報通信学会論文誌 D, Vol.J95-D, No.9, pp.1729-1737, Sep. (2012)
88. ♦岩上拓矢, 吉村和浩, 中田尚, 中島康彦: "時分割実行機構による演算器アレイ型アクセラレータの効率化", 情報処理学会論文誌コンピューティングシステム, ACS39, Vol.5, No.4, pp.13-23, Aug. (2012)
89. ♦吉村和浩, 中田尚, 中島康彦, 北村俊明: "異種命令セットアーキテクチャを持つ高電力効率SMT プロセッサの開発", 電子情報通信学会論文誌 D, Vol.J95-D, No.6, pp.1334-1346, Jun. (2012)
90. ♦中田尚, 吉村和浩, 下岡俊介, 大上俊, Devisetti Venkatarama Naveen, 中島康彦: "画像処理向け線形アレイアクセラレータの性能評価", 情報処理学会論文誌コンピューティングシステム, ACS38, Vol.5, No.3, pp.74-85, May. (2012)
91. · 王昊, 姚駿, 中島康彦: "多様なアクセスパターンに適応するアクセラレータ向けメモリアクセス機構", IPSJ SIG Notes 2012-ARC-199(15), pp.1-4, 2012-03-20, 長崎, Mar. (2012)
92. · Tanvir Ahmed, Jun Yao and Yasuhiko Nakashima: "Achieving Effective Fault Tolerance in FU array by Adding AVF Awareness", IPSJ SIG Notes 2012-ARC-199(5), pp.1-4, 2012-03-20, 長崎, Mar. (2012)
93. ♦ Yukihiko SASAGAWA, Jun YAO, Takashi NAKADA, Yasuhiko NAKASHIMA: "Improving DVS Efficiency by Tolerating IR-drops with an Adaptive Redundant Data-Path", WRA 2011 : 2nd Workshop on Resilient Architectures (in conjunction with MICRO-2011), Dec. (2011)
94. · 森高晃大, 下岡俊介, 吉村和浩, 姚駿, 中田尚, 中島康彦: "大規模演算器アクセラレータのための複数 FPGA 連結手法", IEICE technical report. Computer systems 111(328), 9-14, 2011-11-22, デザインガイド 2011, Nov. (2011)

95. ・齊藤光俊, 下岡俊介, 吉村和浩, 姚駿, 中田尚, 中島康彦: ”演算器アレイ型アクセラレータの実装とその分析”, IEICE technical report. Computer systems 111(328), 9-14, 2011-11-22, デザインガイア 2011, Nov. (2011)
96. ◇中島康彦: ”高性能・低電力・高信頼を全部満たす次世代コンピュータはこんな姿?”, けいはんな情報通信研究フェア 2011, Nov. (2011)
97. ◇中島康彦: ”汎用プロセッサと相性の良い演算器アレイ型アクセラレータの構想”, ICD 第3回アクセラレーション技術発表討論会, テーマ: アクセラレータ技術の展開を目指して, Sep. (2011)
98. ・笛川幸宏, 姚駿, 中田尚, 中島康彦: ”演算器の適応的冗長化による高効率DVS方式の提案”, 信学技報, vol.111, no.164, DC2011-15, pp.1-6, Jul. (2011)
99. ・下岡俊介, 吉村和浩, 中田尚, 中島康彦: ”演算器アレイ型アクセラレータにおけるローカルバッファの最適化”, 研究報告計算機アーキテクチャ (ARC) , 2011-ARC-196(18), pp.1-6, Jul. (2011)
100. ・大上俊, 吉村和浩, 姚駿, 中田尚, 中島康彦: ”演算器アレイにおける高信頼化命令写像手法”, 研究報告計算機アーキテクチャ (ARC) , 2011-ARC-196(19), pp.1-7, Jul. (2011)
101. ♦ Naveen Devisetti, Takuya Iwakami, Kazuhiro Yoshimura, Takashi Nakada, Jun Yao, Yasuhiko Nakashima: ”LAPP: A Low Power Array Accelerator with Binary Compatibility”, HPPAC2011, pp.849-857, May. (2011)
102. ♦岩上拓矢, 吉村和浩, 中田尚, 中島康彦: ”仮想化機構による演算器アレイ型アクセラレータの効率化”, 先進的計算基盤システムシンポジウム SACSIS2011 論文集, pp.136-143, May. (2011)
103. ♦森浩大, 大上俊, 下岡俊介, 吉村和浩, 中田尚, 中島康彦: ”演算器アレイ型アクセラレータのための命令変換手法”, 先進的計算基盤システムシンポジウム SACSIS2011 論文集 (ポスター), 11-608, pp.207-208, May. (2011)
104. ・YAO Jun, Yasuhiko NAKASHIMA: ”EReLA: Exploiting Efficiency of Redundant Executions on an FU array”, 情報処理学会研究報告, Vol.2011-ARC-194(9), pp.1-5, Mar. (2011)
105. ♦Kazuhiro YOSHIMURA, Takuya IWAKAMI, Takashi NAKADA, Jun YAO, Hajime SHIMADA and Yasuhiko NAKASHIMA: ”An Instruction Mapping Scheme for FU Array Accelerator”, IEICE Trans. on Information and Systems, Vol.E94-D, No.2, pp.286-297, Feb. (2011)
106. ◇中島康彦: ”プログラムモデルを維持しつつ大幅な高性能・低電力化を可能とするプロセッサアーキテクチャ”, 第18回<けいはんな>新産業創出交流センターシーズフォーラム, Jan. (2011)
107. ・【電子情報通信学会集積回路研究会優秀若手研究ポスター賞】大上俊, 岩上拓矢, 吉村和浩, 中田尚, 中島康彦: ”アレイ型アクセラレータにおける演算器間ネットワークの設計”, 集積回路研究会 (ICD), Dec. (2010)
108. ・下岡俊介, 岩上拓矢, 吉村和浩, 中田尚, 中島康彦: ”演算器アレイ型アクセラレータにおけるメモリアクセス機構の設計”, 集積回路研究会 (ICD), Dec. (2010)
109. ・岩上拓矢, 吉村和浩, 森浩大, 中田尚, 中島康彦: ”演算器アレイを拡張する細粒度時分割機構”, 集積回路研究会 (ICD), Dec. (2010)
110. ・森浩大, 岩上拓矢, 吉村和浩, 中田尚, 中島康彦: ”演算器アレイ型アクセラレータのための命令変換手法の検討”, SWoPP2010(Vol.2010-ARC-190 No.26 2010/8/4), pp.1-6, Aug. (2010)
111. ♦岩上拓矢, 吉村和浩, 上利宗久, 中田尚, 中島康彦: ”プログラマビリティを備える低電力アクセラレータの提案と評価”, 先進的計算基盤システムシンポジウム SACSIS2010 論文集 (poster), May. (2010)
112. ♦Takuya Iwakami, Munehisa Agari, Kazuhiro Yoshimura, Takashi Nakada, Yasuhiko Nakashima: ”Area Optimization of FU Array in Low-Power Accelerators”, IEEE Symposium on Low-Power and High-Speed Chips 2010 (poster), Apr. (2010)
113. ・吉村和浩, 上利宗久, 中田尚, 中島康彦: ”演算器アレイ型プロセッサのための命令スケジューラの設計と評価”, 信学技報, Vol.109, No.474, pp.511-516, Mar. (2010)
114. ♦ Kazuhiro Yoshimura, Takashi Nakada, Yasuhiko Nakashima, Toshiaki Kitamura: ”An Energy Efficient SMT Processor with Heterogeneous Instruction Set Architectures”, IASTED Int'l Conf. on Parallel and Distributed Computing and Networks (PDCN2010), pp.201-209, Feb. (2010)
115. ・中田尚, 中島康彦: ”線形アレイ VLIW プロセッサにおける適応性検討”, 情報処理学会研究報告,

- Vol.2009-ARC-186, No.10, HOKKE-17, pp.1-9, Nov. (2009)
116. ・【情報処理学会関西支部大会学生奨励賞】上利宗久, 中田尚, 中島康彦: ”線形アレイ型VLIWプロセッサの面積効率評価”, 平成21年度情報処理学会関西支部大会講演論文集, A-03, Sep. (2009)
117. ◇中島康彦: ”グリーンコンピューターへの道～計算の低消費電力化～”, 関西学研都市6大学市民講座, Oct. (2009)
118. ◆中田尚, 片岡晶人, 中島康彦: ”VLIW型命令キューを持つスーパースカラプロセッサの命令スケジューリング機構”, 情報処理学会論文誌コンピューティングシステム, ACS26, Vol.2, No.2, pp.48-62, Jul. (2009)
119. ◆中田尚, 上利宗久, 中島康彦: ”画像処理向け線形アレイVLIWプロセッサ”, 先進的計算基盤システムシンポジウムSACSSIS2009論文集, pp.293-300, May. (2009)
120. ◆Munehisa Agari, Takashi Nakada, Yasuhiko Nakashima: ”A Linear Array VLIW Processor for Image Processing”, IEEE Symposium on Low-Power and High-Speed Chips 2009 (poster), p.153, Apr. (2009)
121. ◆Kazuhiro Yoshimura, Takashi Nakada and Yasuhiko Nakashima: ”An Instruction Decomposition Method for Reconfigurable Decoders”, IEEE International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA2009 post proceeding), Mar. (2009)
122. ◇中島康彦: ”脱マルチコアの試み－ヘテロSMT型VLIWとリニアアレイ型VLIW－”, 情報処理学会ものづくり基盤コンピューティングシステム研究会招待講演, Mar. (2009)
123. ◇中島康彦: ”3-wa yから9N-wa yに至る最近のVLIW研究紹介”, 電子情報通信学会コンピュータシステム研究会招待講演, CPSY, Vol.2008 No.43-52, pp.31-36, Dec. (2008)
124. ・上利宗久, 中田尚, 中島康彦: ”N倍速を目指すVLIWプロセッサの構想”, IPSJ SIG Technical Report, 2008-ARC-180, pp21-24, Oct. (2008)
1. ・中島康彦, 舟井遼太朗: ”CGRAによる演算ユニット”, 特願2021-209979 (2021.12.23)
 2. ・中島康彦: ”データ処理装置”, PCT/JP2020/025123 特願2021-527755 (2021.11.9)
 3. ・中島康彦, 高前田伸也: ”データ処理装置(メモリ内蔵アクセラレータの構成方法)”, 中国ZL201680019602 (2020.12.11)
 4. ・中島康彦: ”データ処理装置(高効率アクセラレータ構成方法)”, PCT/JP2020/025123 (2020.6.26)
 5. ・中島康彦: ”データ処理装置(高効率アクセラレータ構成方法)”, 特願2019-517698 (2019.9.19)
 6. ・Yasuhiko Nakashima, Shinya Takamaeda: ”Data processing Device”, United States Patent 10,275,392 (2019.4.30)
 7. ・中島康彦: ”データ処理装置(NCHIP制御方法)”, 特願2019-121853 (2019.6.28)
 8. ・Yasuhiko Nakashima, Takashi Nakada: ”Data processing Device for Performing a Plurality of Calculation Processes in Parallel”, European Patent Application No.09820420.9 (H31.1.18)
 9. ・中島康彦: ”データ処理装置(高効率アクセラレータ構成方法)”, PCT/JP2018/018169 (H30.5.10)
 10. ・中島康彦: ”データ処理装置(高効率アクセラレータ構成方法)”, 特願2017-96061 (H29.5.12)
 11. ・Jun Yao, Yasuhiko Nakashima, Tao Wang, Wei Zhang, Zuqi Liu, Shuzhan Bi: ”METHOD FOR ACCESSING MEMORY OF MULTI-CORE SYSTEM, RELATED APPARATUS, SYSTEM, AND STORAGE MEDIUM”, PCT/CN2017/083523 (2017.5.8)
 12. ・中島康彦, 高前田伸也: ”データ処理装置(メモリ内蔵アクセラレータの構成方法)”, PCT/JP2016/061302 (H28.4.6)
 13. ・中島康彦, 高前田伸也: ”データ処理装置(メモリ内蔵アクセラレータの構成方法)”, 特願2015-079552 (H27.4.8)
 14. ・中島康彦, 姚駿: ”データ供給装置及びデータ処理装置”, PCT/JP2013/057503 (H25.3.15)
 15. ・中島康彦, 姚駿: ”データ供給装置及びデータ処理装置”, 特願2012-061110 (H24.3.16)
 16. ・中島康彦, 中田尚: ”データ処理装置”, PCT/JP2009/005306 (H21.10.13)
 17. ・中田尚, 中島康彦: ”データ処理装置”, 特願2009-150788 (H21.6.25)
 18. ・中島康彦, 中田尚: ”データ処理装置”, 特願2008-265312 (H20.10.14)

Appendix B

Q&A

B.1 概要

B.1.1 CGRA とは何ですか

OHM 大学テキスト コンピューターアーキテクチャ ISBN:978-4-274-21253-6 を買って読めばわかります。

B.1.2 EMAX6 と IMAX はどこか違うのですか

違いはありません。番号が増えてわかり難いので、別名 IMAX と呼んでいるだけです。

B.1.3 実際速いんですか

Jetson TX2(GPU 1.3GHz) が相手なら、IMAX(150MHz) がぶっちぎりです。

B.2 プログラマとしての使い方

B.2.1 自分で書いたプログラムが全くコンパイルできないのですが

1. exe() や mop() のディスティネーション変数に、AR[row][col] や BR[row][col][reg] を指定すると、写像位置を固定することができます。それ以外の変数を使用した場合、コンパイラが空き演算器を割り当てます。
2. 先頭ステージには、多重ループ制御のための演算が写像されるので、4列全てを exe() に使うことはできません。
3. コンパイルが正常終了すると、tgif xxx.obj で、演算器への機能写像状況が図で確認できます。
4. コンパイルできない原因が伝搬レジスタの不足にある場合、どうすれば節約できるか、正しい結果の写像図を見て考えましょう。
5. 論理ユニットには最大 2 つの mop(LD) を配置できます。最初の LD 結果は BR[1]、次の LD 結果は BR[0] に格納するように記述しましょう。
6. 論理ユニットには最大 1 つの mo4(LDRQ) を配置できます。BR[3]-[0] に 64bit x 4 のデータが格納されます。

B.2.2 自分で書いたプログラムが全く動かないのですが

具体的にどこまで動いて、どこから動かないかを言ってくれないと、聞く耳ありません。なお、以下の可能性があります。顔を洗って出直しましょう。

1. ZCU102 の電源が入っていない。
2. ZCU102 がハングしている。電源再投入。

3. VU440 の電源が入っていない.
4. VU440 に正しい bin が書き込まれていない.
5. VU440 間の aurora 物理層がリンクアップしていない. 各リセットボタンを押す.
6. VU440 間の aurora 上位層がリセットされていない. 各リセットボタンを押す.
7. コンパイラ (conv-c2c) が思っている IMAX のステージ数 (EMAX_DEPTH) と, FPGA に書き込まれている構成のステージ数 (8 だったり 16 だったり 64 だったり) が一致していない.
8. コンパイル時エラーを見逃している.
9. ループカウンタ (for (INIT0=1,LOOP0=loop,dmy=0; LOOP0-;INIT0=0) の部分) が間違っている.
10. ハードウェアがハンギングしている. 次節を参考に, ハードがわかる人にお土産をもって行って相談しましょう.

B.2.3 自分で書いたプログラムの実行結果がおかしいのですが

1. 浮動小数点演算の場合, 演算誤差が生じことがあります.
2. LMM の先頭アドレスが変化しないと IMAX は再利用可能と判断して主記憶からロードしません. カメラ等入力デバイスから LMM に値を書き込んで使う場合, mop(force) において, force=1 を設定すれば毎回主記憶からロードします.
3. IMAX 記述部分のアルゴリズムとしての正しさは, emax6nc モード (No-CGRA mode: exe() や mop() を関数コールと見なして通常の C コンパイラにより実行し実行) にて確認できます. 通常の C プログラムとして動作するので printf() や通常のデバッガを使用できます.
4. 前項のデバッグが完了していれば, IMAX コンパイラにより実行する際の不具合は, mop() における主記憶間データ転送情報の間違いに起因することが多いです.
5. exe() においてディスティネーション変数と第 1 ソース変数が同一の場合, アキュムレート等自己ループ演算と見なされます. 前行のディスティネーション変数とも同一である場合, emax6nc モードは前行の結果を第 1 ソース変数の値に使用しますが, IMAX では自身の演算結果を第 1 ソース変数の値に使用するため, 結果が一致しません.

B.3 ハード設計者としての使い方

B.3.1 ハードが動いていないようですが

以下の可能性があります. 隅なくチェックしましょう.

1. VU440 の flush メモリが壊れている. PlayerPro を使って再度書き込む.
2. インタフェースボードが外れている. 正しく接続する.
3. VU440 に設定すべき IMAX コアの周波数, C2C の周波数が間違っている. PlayerPro を使って正しく設定する.
4. 論理設計にバグがある. Modelsim や Chipscope を使ってデバッグする.
5. タイミングエラーが起きている. 周波数を下げてみる.

B.3.2 演算機能を追加したいのですが

以下の手順で追加できます.

1. 追加機能を想定してアプリを書いてみる.
2. emax6nc で動作を確認.
3. コンパイラ (conv-c2c) に機能追加する.
4. シミュレータ (csim) に機能追加する.
5. テストプログラムを作って csim 上で動作確認する.
6. Verilog 記述に機能追加する.

7. テストプログラムを csim 上で走行し、コンバータでテストベンチを生成する。
8. Modelsim とテストベンチを使って Verilog を検証する。
9. Vivado で合成配置配線する。
10. PlayerPro を使って FPGA に書き込む。
11. テストプログラムを実機上でコンパイルして動作確認する。

B.3.3 機能追加後の csim の調べ方

1. 普通に実行する

```
% ../../src/csim/csim -x test022-csim.emax6+dma
```

```
Monitor-000:00000000_00400000 00014190 IPC=0.296 R0B=0:2 0:1 0:1 0:1
000:PIO RD adr=00000000 data=0000000000000000_0000000000000000_0000000000000000_0000000000000000
000:PIO RD adr=00000000 data=0000000000000000_0000000000000000_0000000000000000_0000000000000000
ここで何も表示されなくなる
```

2. Monitor-000:00000000_00400000 が実行命令数を表示しているので、これを使って、ARM トレースを表示させる

```
% ../../src/csim/csim -x -b400000 test022-csim.emax6+dma | egrep "RT"
```

-b400000 は、400000 命令以降の ARM トレースを表示するという意味。

"RT" は、ARM スーパスカラパイプラインのリタイア情報のみ選択

```
000:RT 00000000_00403d33 00014160 * [cond=e:upd=0] fmov :V23<-00000000_00000000_00000000_3f
000:RT 00000000_00403d34 00014164 * [cond=e:upd=0] fmov :V21<-00000000_00000000_00000000_3f
000:RT 00000000_00403d35 00014168 * [cond=e:upd=1] sub :R36<-00000000_00000002
000:RT 00000000_00403d36 0001416c * [cond=d:upd=0] b
000:RT 00000000_00403d37 00014170 * [cond=e:upd=0] mov :R02<-00000000_00000000
ここで何も表示されなくなる
```

3. 正確な実行命令数が 00000000_00403d37 であることがわかったので、次は全情報を表示

```
% ../../src/csim/csim -b403d36 -x test022-csim.emax6+dma | less
```

```
c00:RB 14[t0:2] 15[t0:4] 16[t0:1] 0[t0:1] 1[t0:1] 2[t0:1] 3[t0:1] 4[t0:4] 5[t0:1] 6[t0:1] 7[t
000:PIO RD adr=00000000 data=0000000000000000_0000000000000000_0000000000000000_0000000000000000
000:AXIIF->IORQ RD opcd=10 adr=80000000 data=0000000000000000_0000000000ffff
c00:RB 14[t0:2] 15[t0:4] 16[t0:1] 0[t0:1] 1[t0:1] 2[t0:1] 3[t0:1] 4[t0:4] 5[t0:1] 6[t0:1] 7[t
c00:RB 14[t0:2] 15[t0:4] 16[t0:1] 0[t0:1] 1[t0:1] 2[t0:1] 3[t0:1] 4[t0:4] 5[t0:1] 6[t0:1] 7[t
以後、同じメッセージが続く。
```

c00: は ARM のコア番号, 000: はスレッド番号

なお、csim の構成は、起動時にわかる。

コア 01, 02, 03 はすることがないので停止している。

```
ARM+EMAX6 Simulator Version 1.67 2021/06/14 03:53:22 nakashim Exp nakashim $
MAXTHR = 16 ... 全スレッド数
```

```

MAXCORE    = 4                                … コア数
THR/CORE   = 4.000000 (should be integer)    … コアあたりのスレッド数
ROBSIZE    = 16 (actives are CORE_ROBSIZE-1)  … リオーダバッファサイズ
LINESIZE   = 64B                               … キャッシュラインサイズ
I1SIZE     = 16384B (4way delay=16)           … I1 キャッシュパラメタ
D1SIZE     = 16384B (4way delay=16)           … D1 キャッシュパラメタ
L2SIZE     = 131072B (4way dirdl=50, cc=100, mm=150)
MAXL1BK   = 8                                … L1 キャッシュバンク数
MAXL2BK   = 8                                … L2 キャッシュバンク数
MAXMMBK   = 4                                … 主記憶バンク数
memspace   = 00000000-4fffffff
arm_hdr    = 00001000-
arm_param  = 00001080-
aloclimit = -4dfffff00
stack/thr  = 00010000
stackinit  = -4dfffff00

```

4. AXIIF に、READ/PIO adr=80000000 が出力されてることがわかる。IMAX が存在しない I/O 空間なので、当然ハングする(実機と同じといえれば同じ)。

という具合に調べると、何がおこっているかがわかる。

B.4 商売としての使い方

B.4.1 設計データとツールを一式欲しいのですが

1. 大学の産学連係部門に相談すれば、有償でお渡しできることがあります。
2. 博士後期課程に入學して IMAX グループに所属すれば、全ての情報にアクセスできます。ただし、論文書いて発表しないと、お持ち帰りはできません。