Coarse-Grained Reconfigurable Array Architecture

Jeremy Lee

PostDoc

School of Computer Science and Engineering

University New South Wales

21 April 2016

CGRA for Embedded Systems

Contents

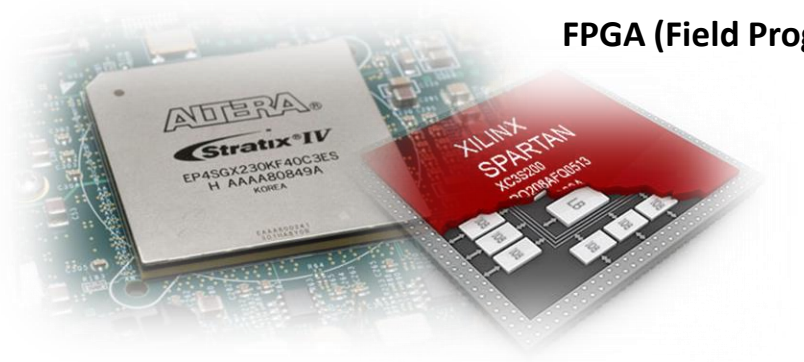
- Necessity of CGRA
- CGRA Organization
- CGRA operation example
- CGRA vs FPGA
- Trend in CGRA and design examples

Embedded Systems

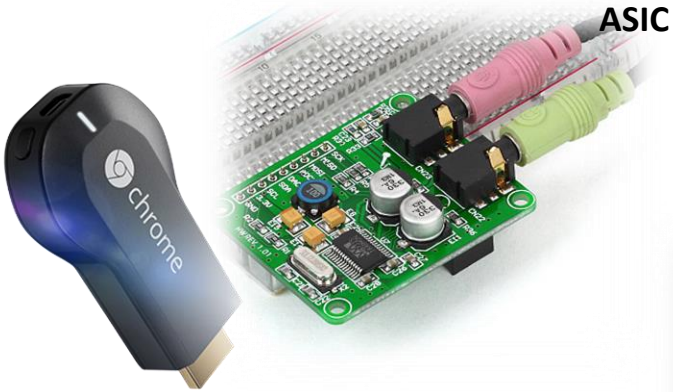
*Source – images from www.samsung.com, www.apple.com, www.bose.com

Embedded Components

FPGA (Field Programmable Gate Array)



ASIC (Application-Specific Integrated Circuit)

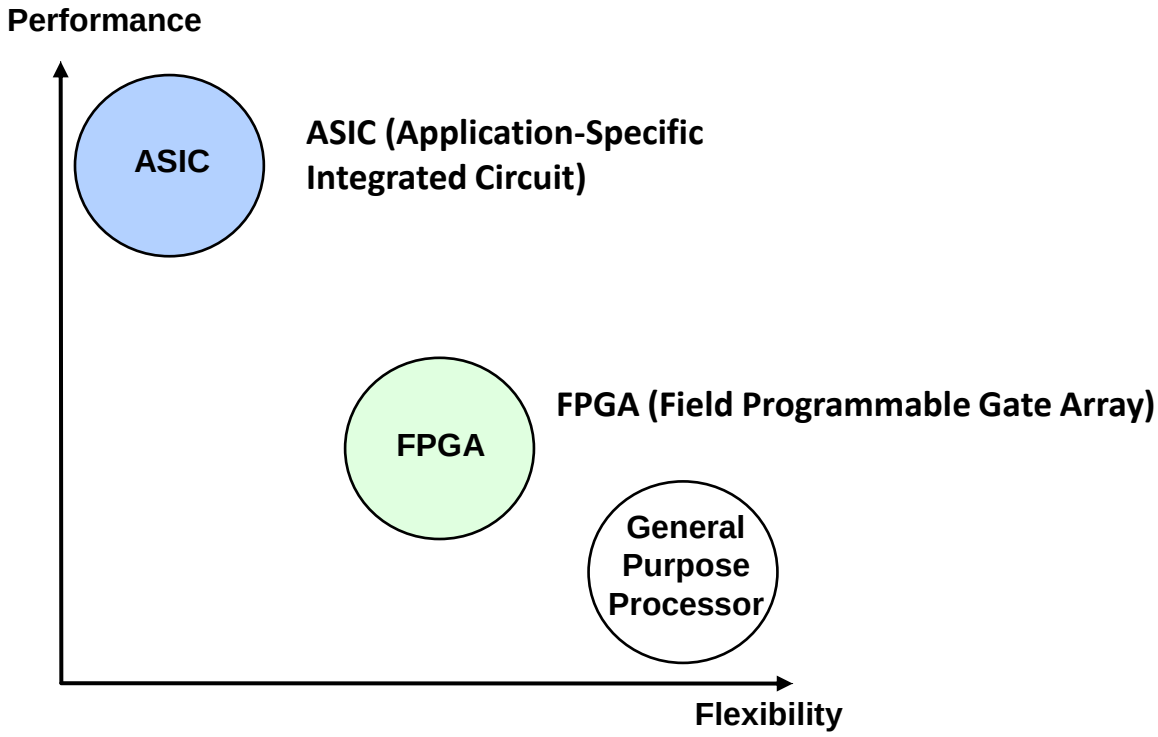


GPP (General Purpose Processor)

*Source – images from www.mikroe.com, www.ti.com, www.extreametech.com, www.altera.com, www.xilinx.com, www.google.com

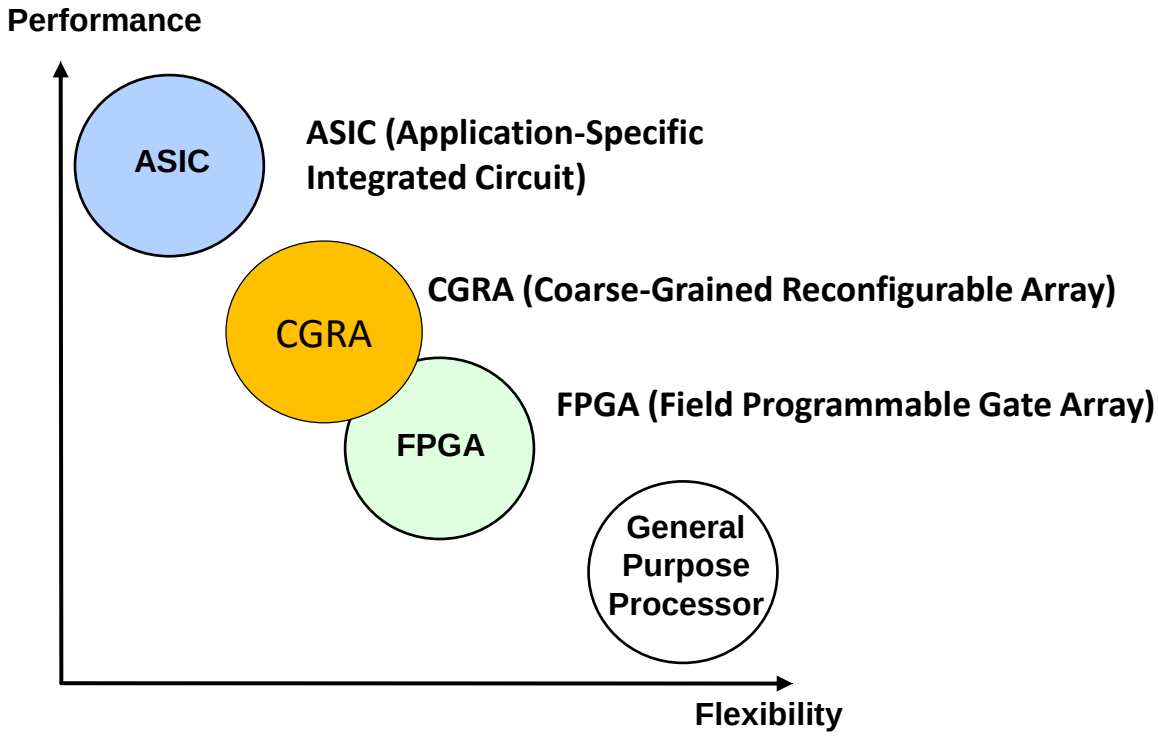
Embedded Components

- Three representative embedded components
  - Dedicated hardware (ASIC), FPGA and general purpose processor
  - Performance and flexibility tradeoffs for 3 components

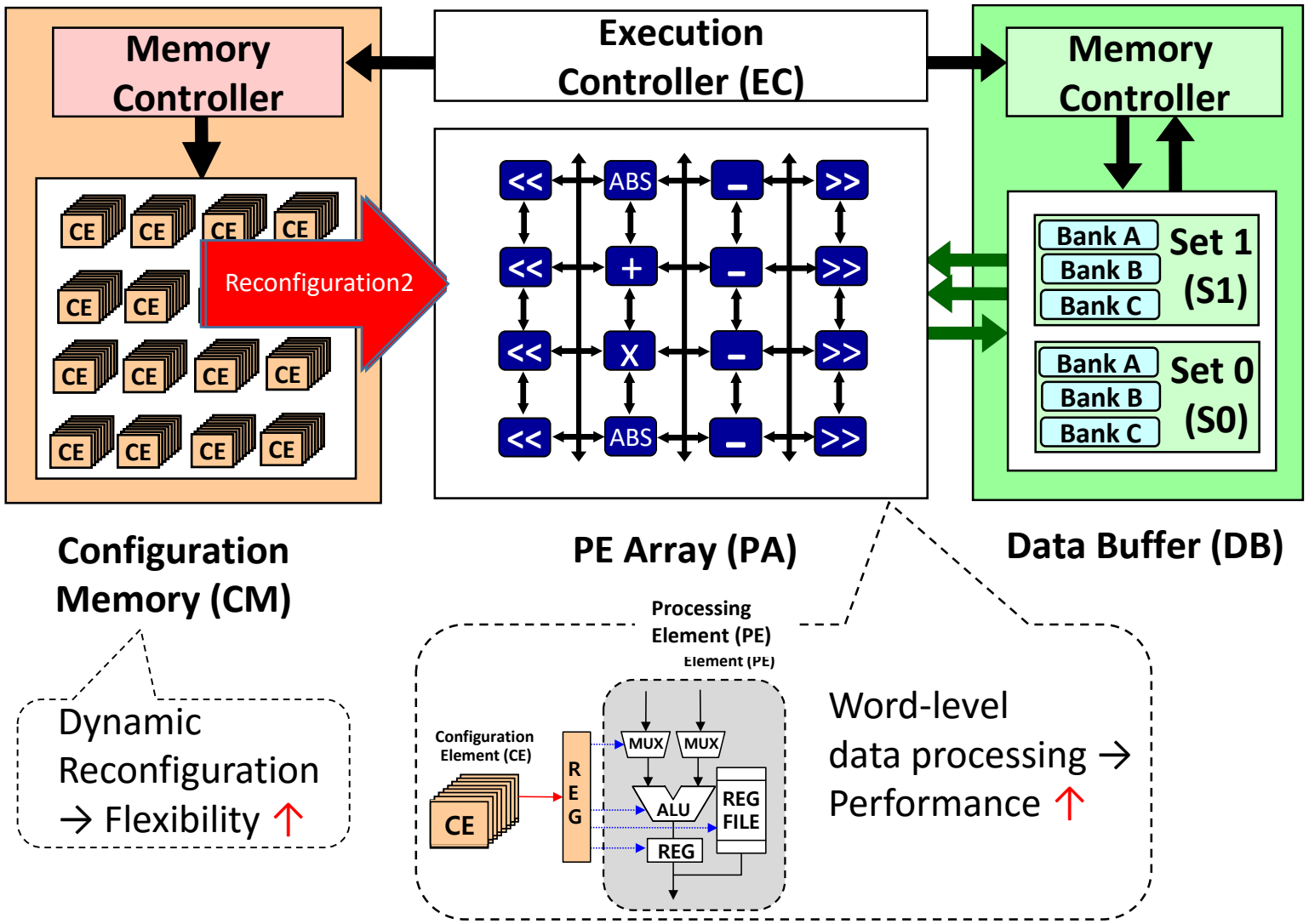


Position of CGRA

- Coarse-Grained Reconfigurable Array (CGRA)
  - ASIC-like performance
  - Higher flexibility than ASIC

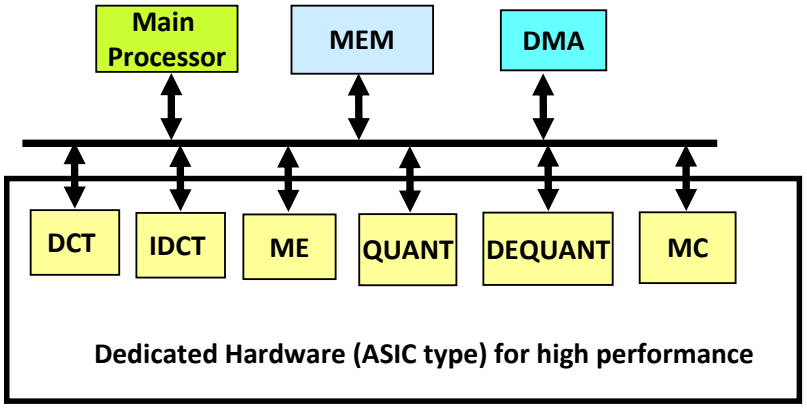


CGRA Organization

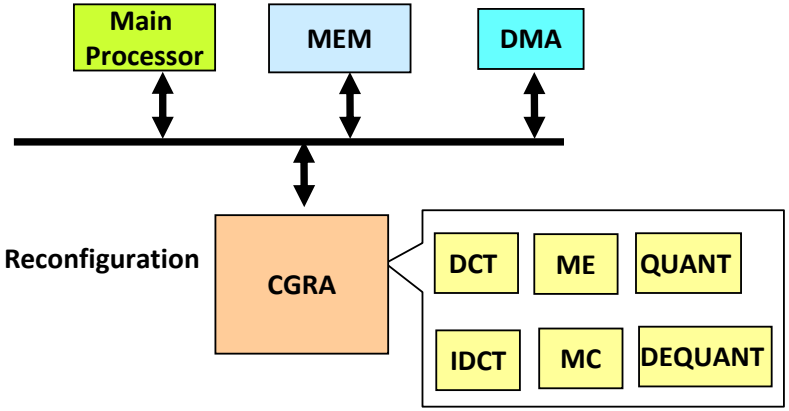


CGRA Operation Example

- CGRA example
  - System-on-Chip Platform for MPEG encoder



(a) SoC with ASICs
(high performance but not flexible)



(b) SoC with CGRA
(flexible with high performance)

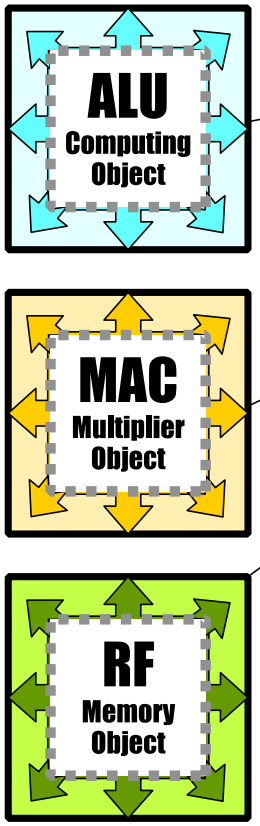
FPOA Architecture Overview

Honeywell

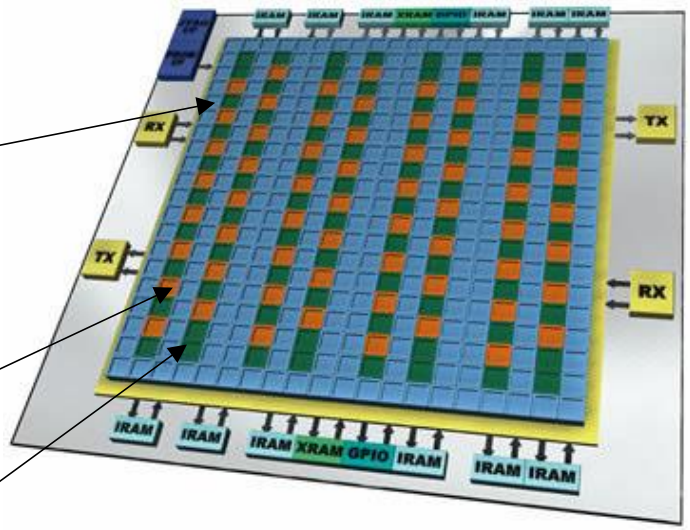
MATHSTAR.jpg

FPOA Objects

Building Blocks



1 GHz Object Array



Object Architecture

- Coarse-grained Architecture
- Same inter-object communication
- All synchronized at 1 GHz

Objects arranged by abutment

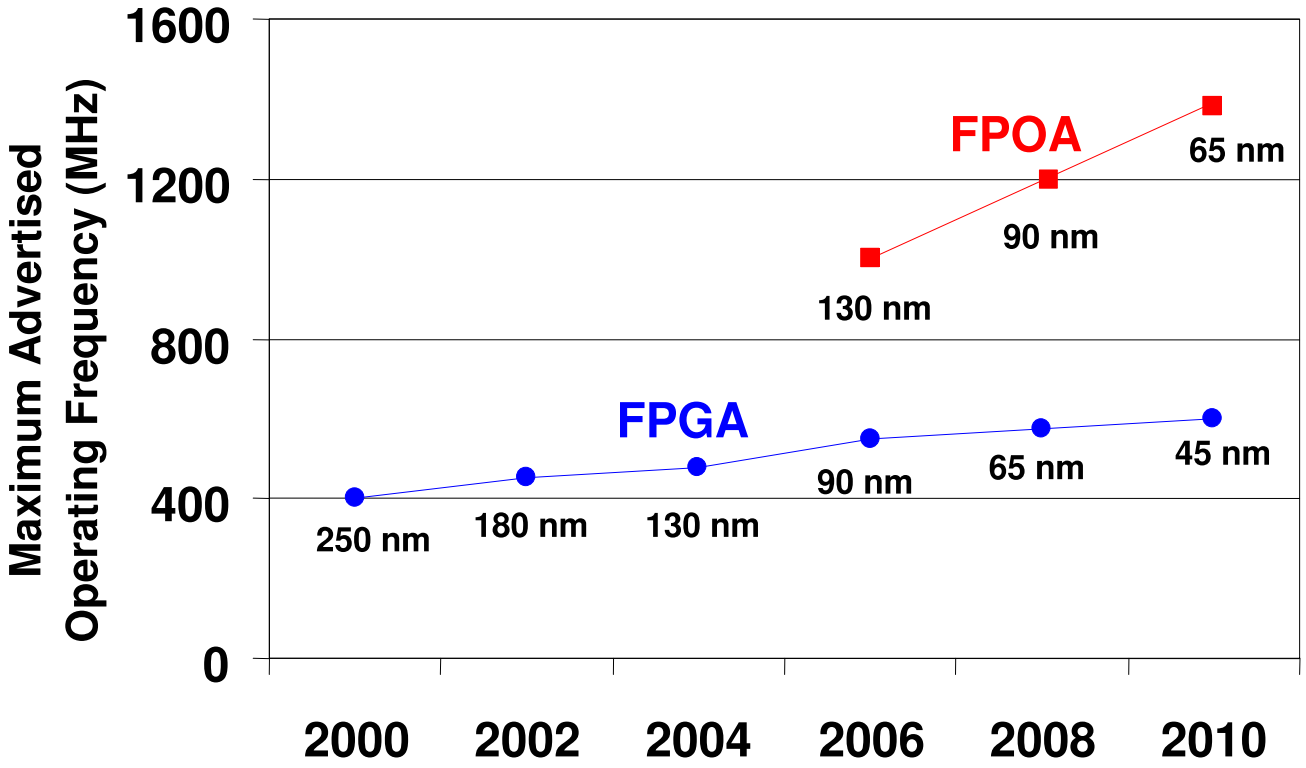
- Enables 1 GHz performance
- Manageable design with hundreds of objects -vs- sea of gates

*Source - Honeywell “Radiation Hardened Field Programmable Object Array (FPOA) for Space Processing,” MAFA 2007

FPOA Performance Scales

Honeywell

MATHSTAR.jpg

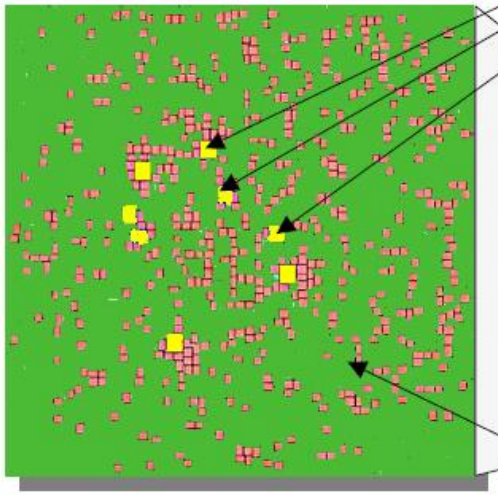


FPGA cites maximum advertised frequency, actual clock rates after timing closure are typically much lower than maximum
FPOA cites actual clock rates
All historical and forecasted numbers are estimates of MathStar, Inc.

*Source - Honeywell "Radiation Hardened Field Programmable Object Array (FPOA) for Space Processing," MAFA 2007

What is Holding FPGAs Back?

Honeywell



4X LUT4 Architecture Tile in FPGA*
35,462 µ² @ 0.18µ

- Buffer
- SRAM
- Multiplexer
- LUT
- Flip Flop
- Pass Transistor Switch
- Buffered Switch

Expensive: A small fraction of the die actually does processing. The rest is interconnect and memory

FPGA Die

Slow: Have to navigate a lot of interconnect to get to the logic.

*Source - Honeywell "Radiation Hardened Field Programmable Object Array (FPOA) for Space Processing," MAFA 2007
- "Automatic Transistor and Physical Design of FPGA Tiles from an Architectural Specification," K.Padalia, Jonathan Rose, et al.

Advantage of CGRA

Technology	Example suppliers	Developm-ent costs	Time to market	Die size	Performan-ce	Reconfigu-ration
ASIC	Qualcomm, Samsung	High	Slow	Low	High	Low
FPGA	Xilinx, Altera	Low	Fast	High	Low	High
CGRA	PACT, MathStar	Low	Fast	Medium	High	Medium

Trend in CGRA and Design Examples

- Originally, Research started in Europe, Germany.

- USA
  - Defense ministry
  - Supporting CGRA research of academy for the purpose of military mobile equipments.
  - NASA – development for aerospace equipment
    - RDPP

- Asia
  - Japan, Korea, China – Companies developing CGRA for mobile equipment

Trend in CGRA and Design Examples

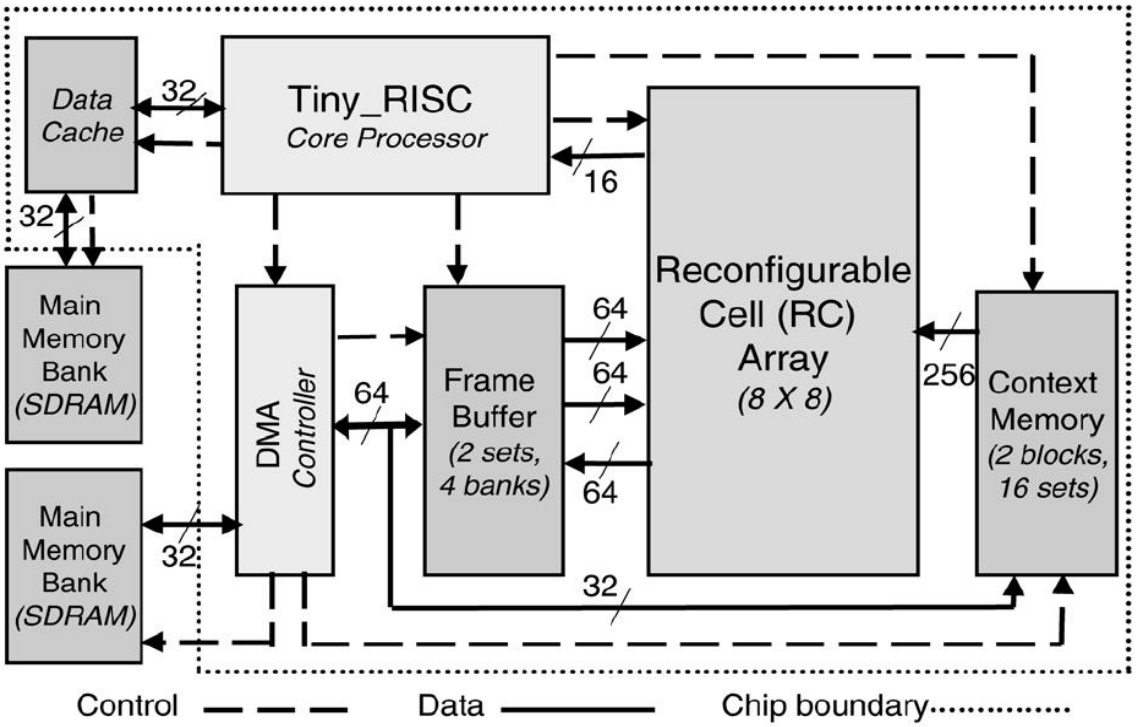
- **Summary of representative CGRA examples**

Nation	Institution	Name	Year	Design Space Exploration	Compiler	Chip
Germany	Kaiserslautern Univ’	KRESS Array	1995	O	X	X
	Darmstadt Univ' of Tech'	DreAM	2000	X	X	O
	PACT XPP Technology	**XPP64**	2002	O	X	O
Belgium	IMEG	**ADRES**	2002	O	O	O
USA	MIT	MATRIX	1996	X	X	O
	Univ' of washington	RaPiDs	*1996*	X	X	O
	UC Berkeley	Garp	1997	O	O	X
	Stanford Univ'	REMARC	1998	X	X	O
	UC Irvine	**MorphoSys**	2000	X	O	O
	Carnegie Mellon Univ'	Piperench	2000	X	O	O
Japan	Hiroshima Univ'	I-PARS	2003	X	X	O
U.K.	Univ' of Edinburgh	RICA	2009	O	O	O
Korea	Samsung Electronics	**SRP**	2008	O	O	O

Trend in CGRA and Design Examples

- **MorPhoSys**

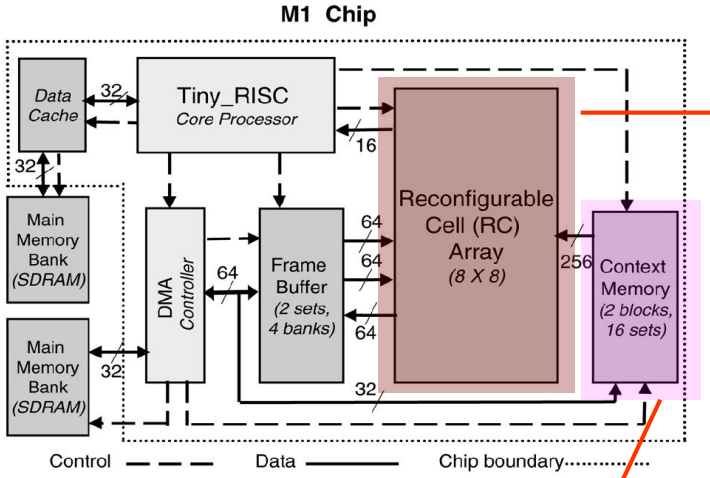
Nation	Institution	Name	Year	DSE	Compiler	Chip
USA	UC Irvine	MorphoSys	2000	X	O	O

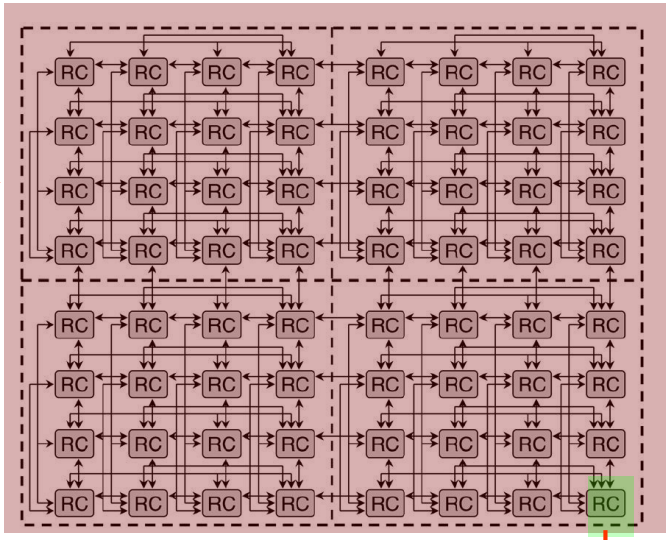


Overall Structure

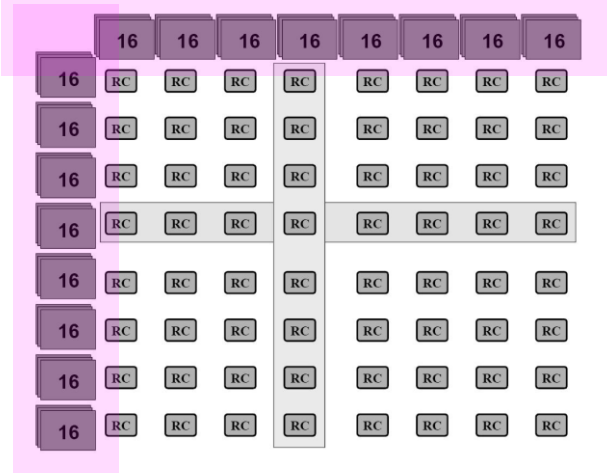
Trend in CGRA and Design Examples

- **MorPhoSys**

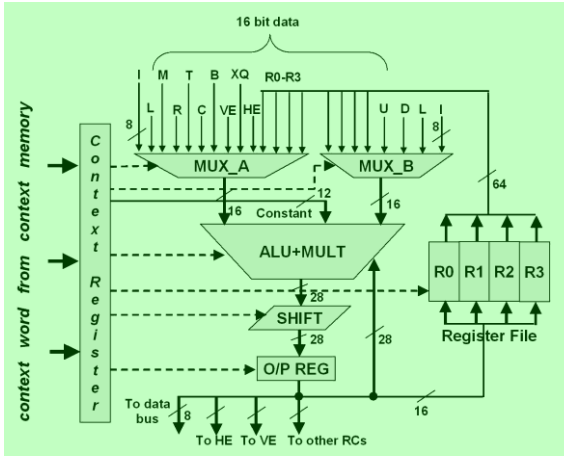




RC Array Interconnection



Organization of Context Memory

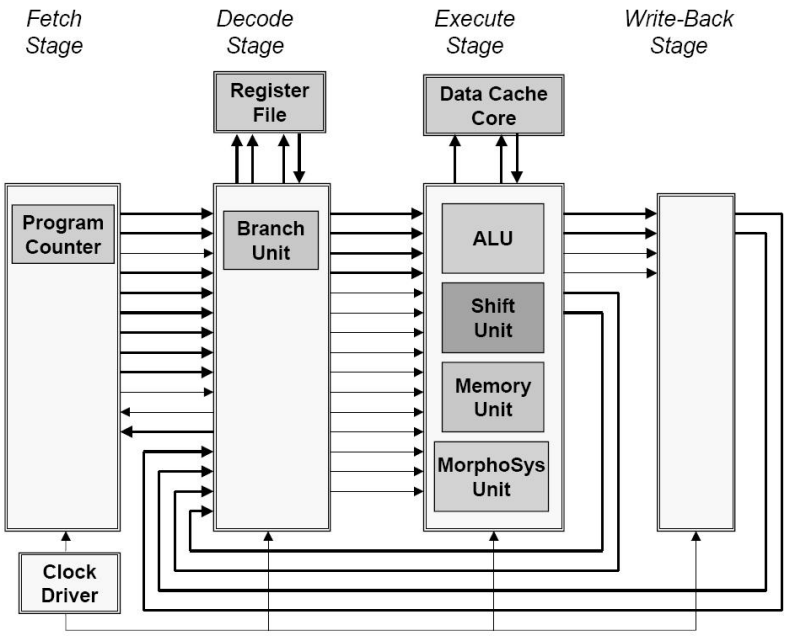


RC Inner Structure

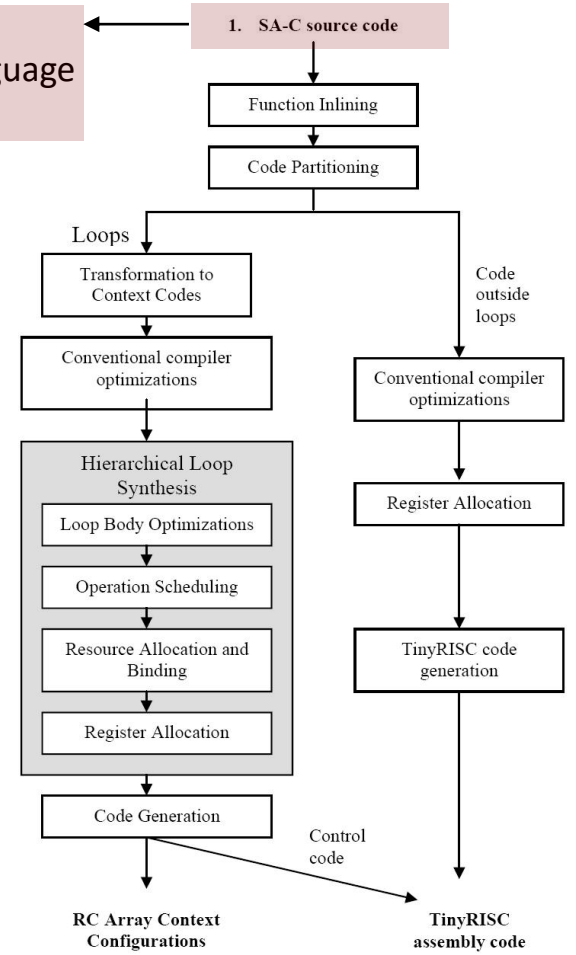
Trend in CGRA and Design Examples

- **MorPhoSys**

▪ **Single Assignment C (SA-C)**
- Functional programming language for efficient array processing



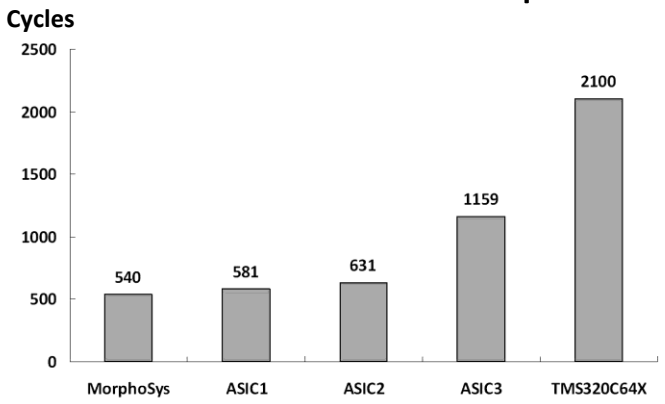
TinyRISC pipeline stage



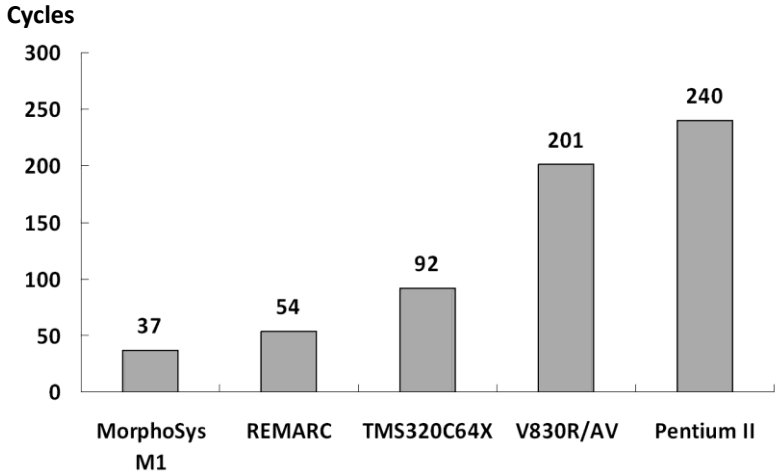
Entire Flow of Compilation

Trend in CGRA and Design Examples

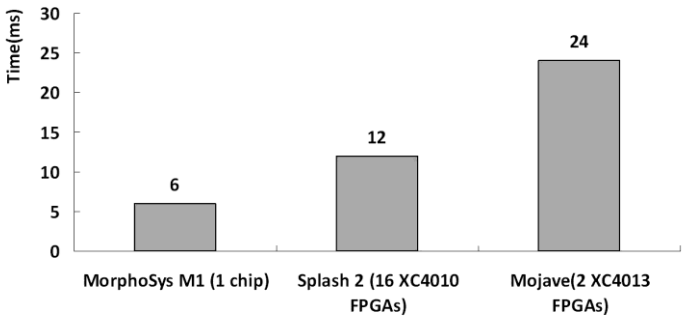
- **MorPhoSys**
  - Performance comparison



(a) Motion Estimation

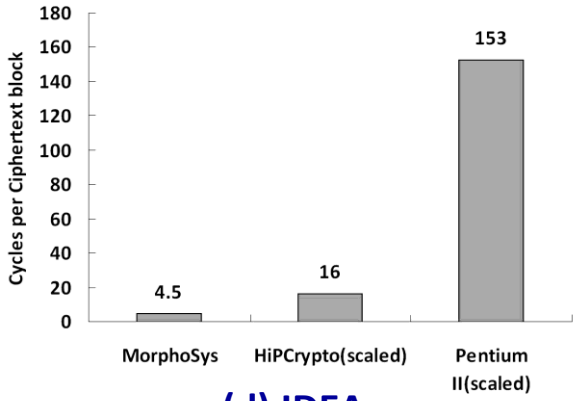


(b) DCT/IDCT



(c) SLD

Second Level of Dectection Algorithm in Automatic Target Recognition (ATR)

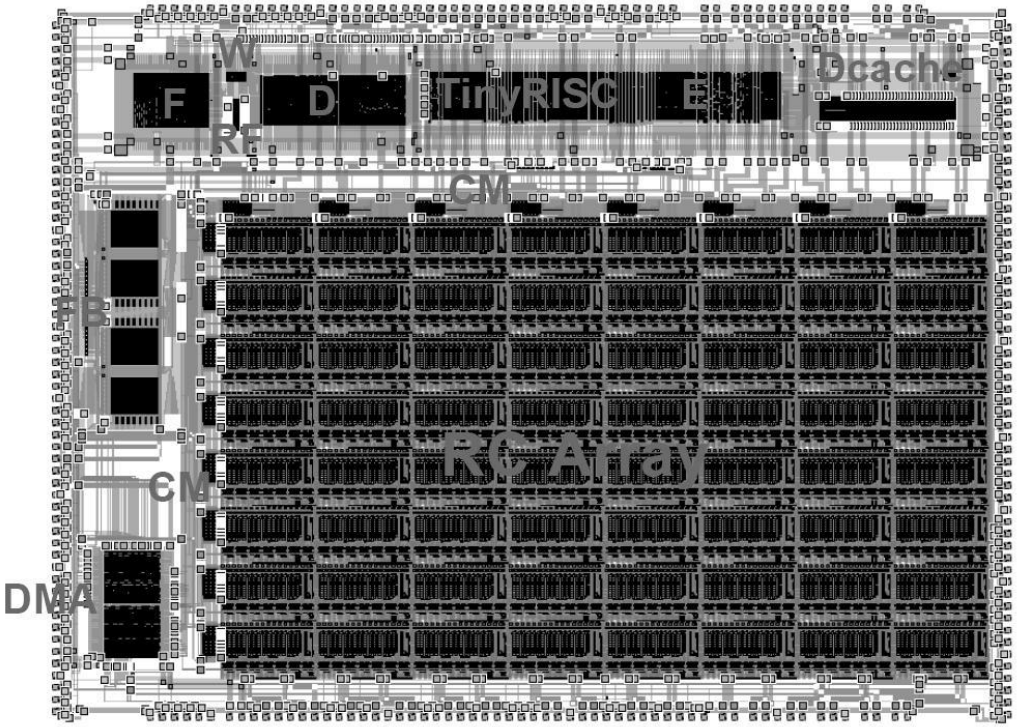


(d) IDEA

International Data Encryption Algorithm

Trend in CGRA and Design Examples

- **MorPhoSys**
  - Physical implementation (0.35/0.18/0.13 ㎛, 100/200/250 MHz)

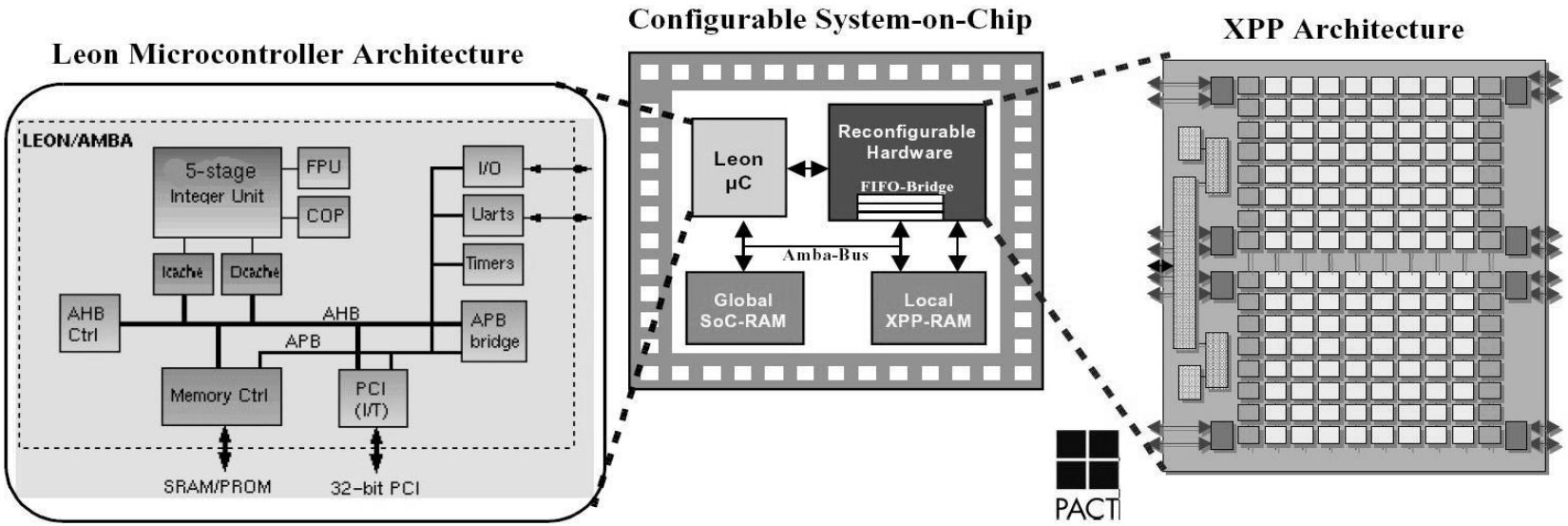


Layout of MorphoSys M1 Chip

Trend in CGRA and Design Examples

- **PACT-XPP**

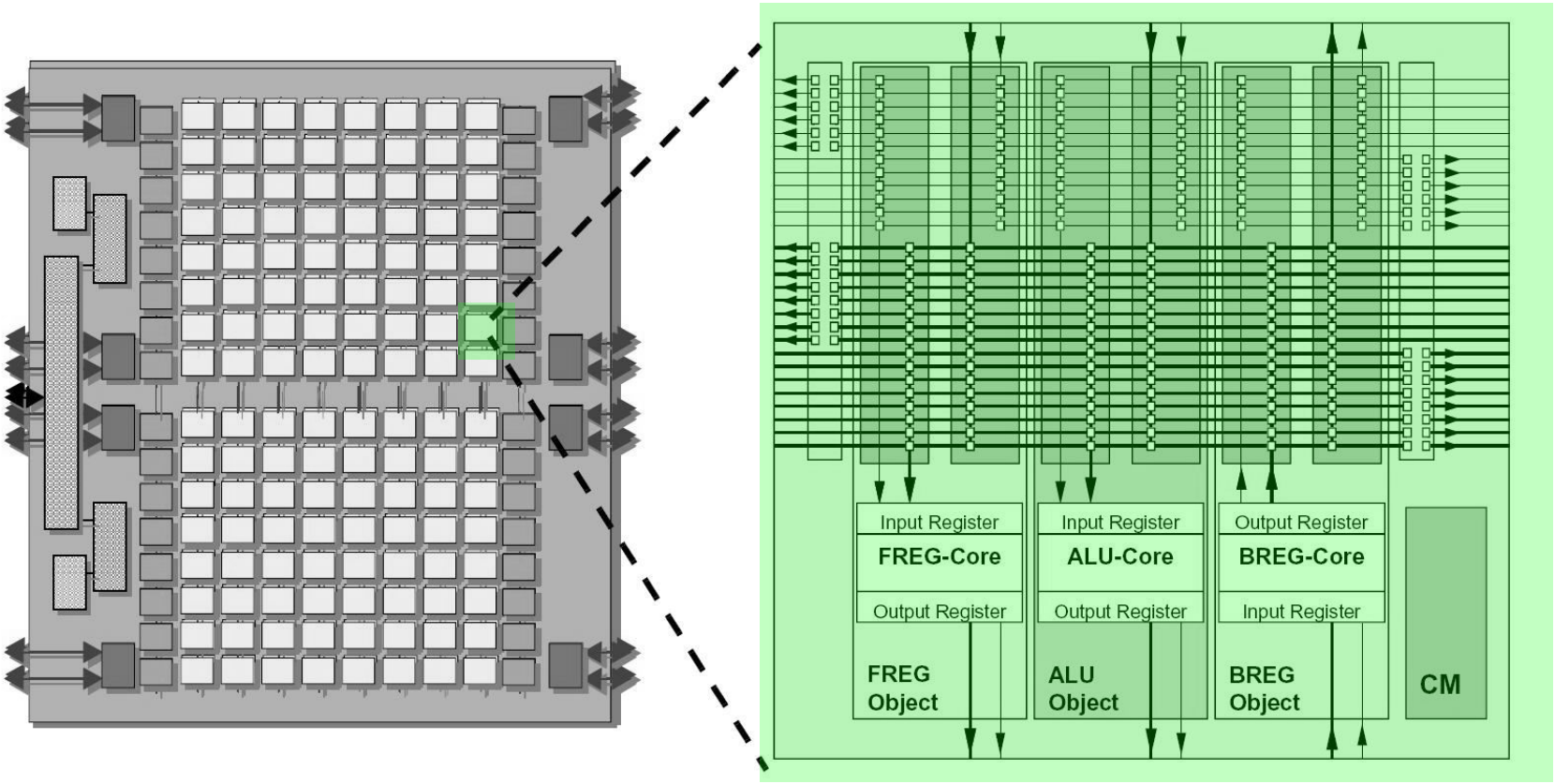
Nation	Institution	Name	Year	DSE	Compiler	Chip
Germany	PACT XPP Technology	XPP64	2002	O	X	O



Overall Structure

Trend in CGRA and Design Examples

- **PACT-XPP**



ALU-PAE (Processing Array Element) Inner Structure

Trend in CGRA and Design Examples

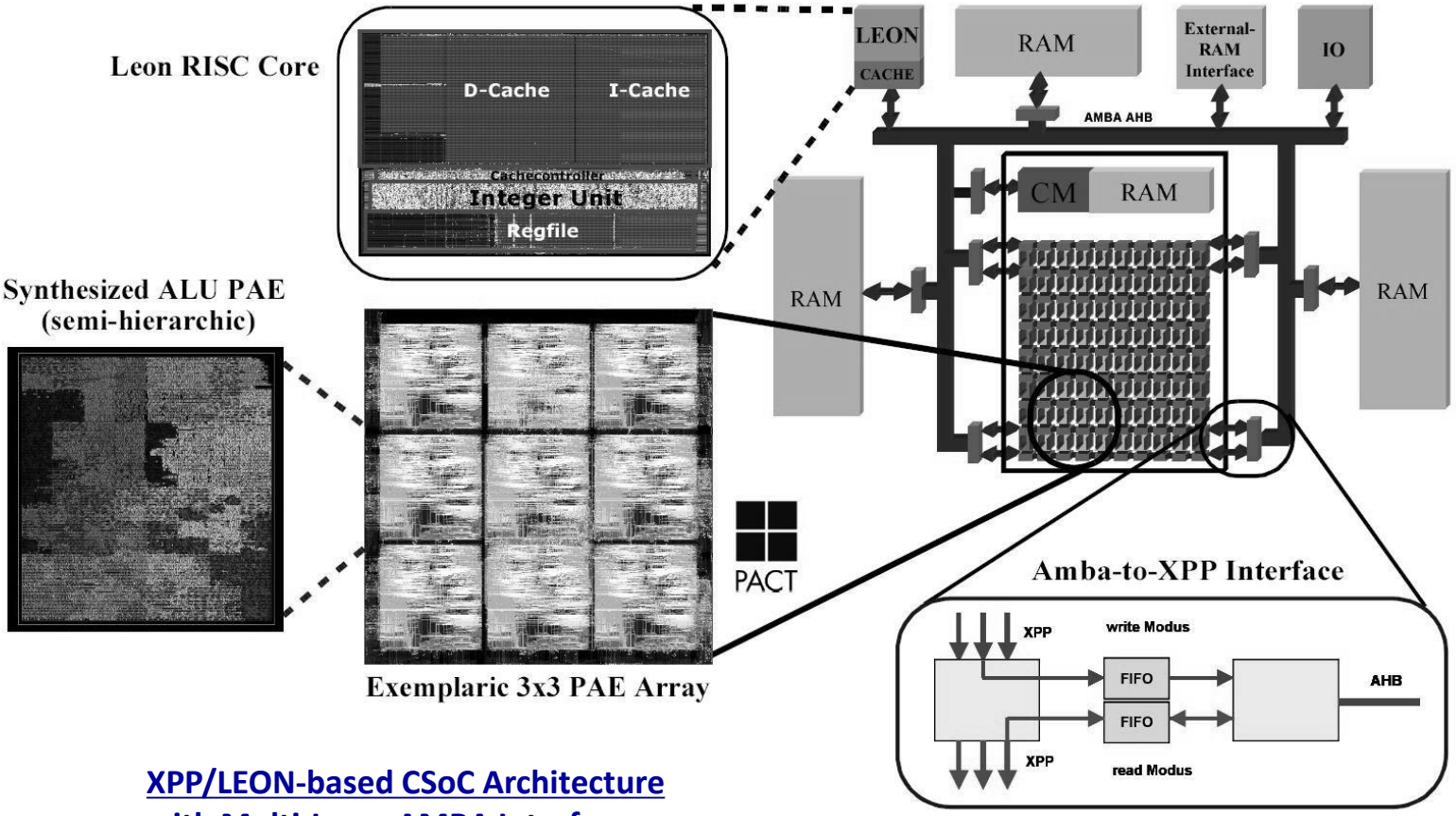
- **PACT-XPP**
  - Performance comparison

H.264 Decoder Performance Comparison

H.264(48X32) Upscaled 10 frames	ARM	PACT	Speedup
Upscaling	98M	815.8K	120X
Luma Reconstruction	6M	342.2K	17.5X
Itrans + IQ	3M	64K	47X
Color Conversion	1M	77.5K	13X
Entire Decoder	123.5M	16.7M	7.4X

Trend in CGRA and Design Examples

- **PACT-XPP**
  - Physical implementation (0.18/0.13μm, 200/300 MHz)

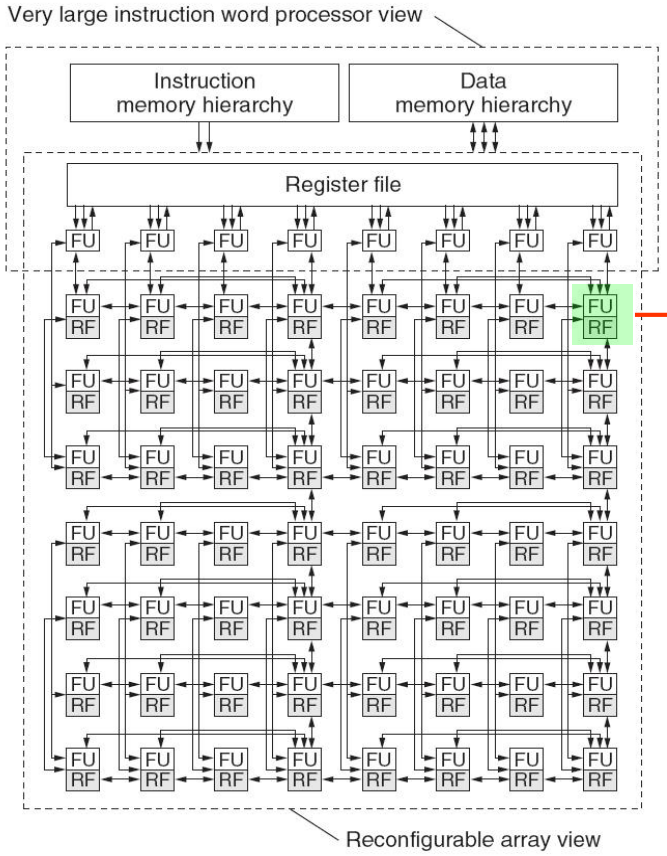


[XPP/LEON-based CSoC Architecture with Multi-Layer AMBA Interface]

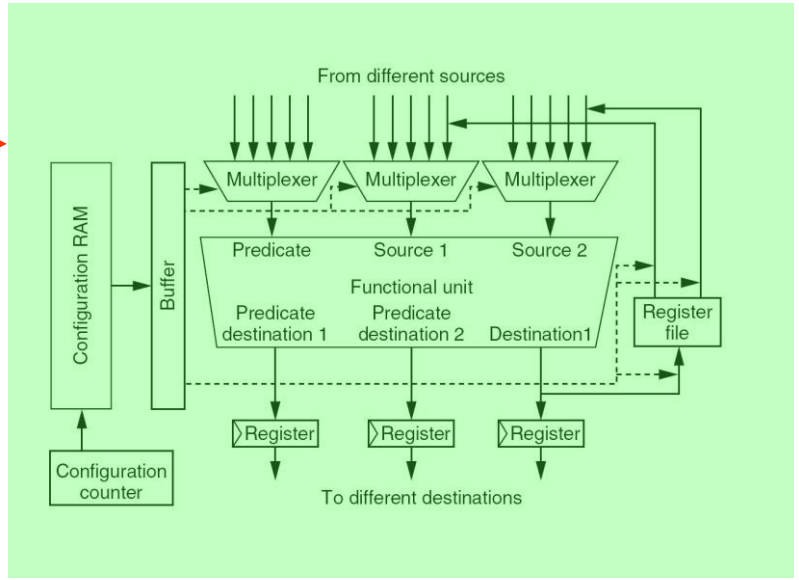
Trend in CGRA and Design Examples

- **ADRES**

Nation	Institution	Name	Year	DSE	Compiler	Chip
Belgium	IMEG	ADRES	2002	O	O	O



Overall Structure

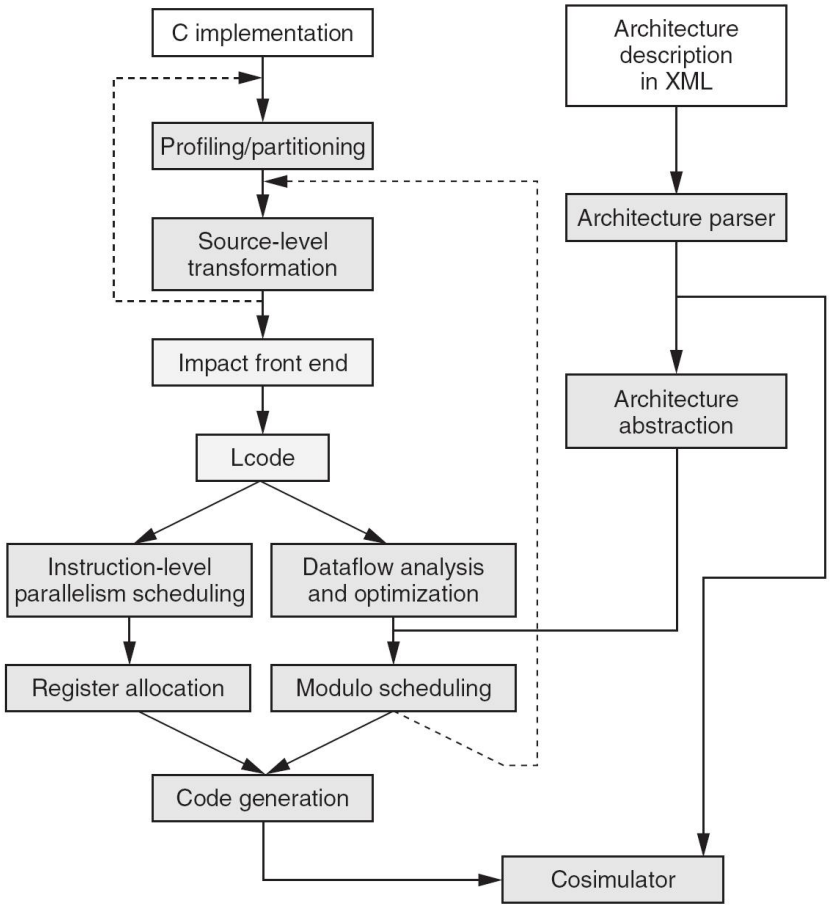


FU (Functional Unit) Inner Structure

Trend in CGRA and Design Examples

- **ADRES**
  - Its own compiler & architecture exploration framework
  - DRESC

    (Dynamically Reconfigurable Embedded System Compiler)



DRESC Framework

Trend in CGRA and Design Examples

- **ADRES**

Different ADRES Architectures and Their Characteristics

Template	Word Width	Number of FUs	Number of FUs with Load/Store capability	Number of RFs	Connections
4x4 multimedia	32	16	4	12	Nearest neighbor
4x4 wireless	64	16	4	13	mesh-plus routing
2x2 alternative	32	4	2	1 for 4words	MorphoSys routing
4x4_all alternative	32	16	4	12, connected to each FU	mesh-plus routing
4x4_4L alternative	32	16	4	4, each for 4 words	MorphoSys routing
4x4_16L alternative	32	16	4	4, each for 64 words	MorphoSys routing
8x8 alternative	32	64	8	16, each for 4 words	MorphoSys routing

Trend in CGRA and Design Examples

- **ADRES**

Comparison of the Image Processing Benchmarks for different ADRES architectures and the TI c64x.

Benchmark	Architecture	Cycles	Instructions	IPC	Utilization of the CGRA	Speedup to TI c64x
Tiff2BW	DSP TI c64x	4,547,947	26,529,081	6		1.00
	4x4 multimedia	2,273,967	24,729,217	10.9	68%	2.00
	4x4 wireless	3,410,949	26,718,897	7.8	49%	1.33
	2x2 alternative	8,811,517	24,444,802	2.8	70%	0.52
	4x4_all alternative	2,894,189	32,662,290	11.3	71%	1.57
	4x4_4L alternative	2,687,434	32,662,331	12.2	76%	1.69
	4x4_16L alternative	2,687,427	21,421,906	11.7	73%	1.69
	8x8 alternative	1,421,264	35,814,939	25.2	39%	3.20
Wavelet	DSP TI c64x	8,765,163	26,973,231	3.08		1.00
	4x4 multimedia	2,095,528	28,346,939	13.5	84%	4.18
	4x4 wireless	2,623,134	30,137,922	11.5	72%	3.34
	2x2 alternative	6,767,899	26,117,944	3.9	98%	1.30
	4x4_all alternative	3,091,148	35,451,612	11.5	72%	2.84
	4x4_4L alternative	2,885,585	35,022,239	12.1	76%	3.04
	4x4_16L alternative	2,792,529	34,202,225	12.2	76%	3.14
	8x8 alternative	1,627,106	46,342,177	28.5	45%	5.39

Trend in CGRA and Design Examples

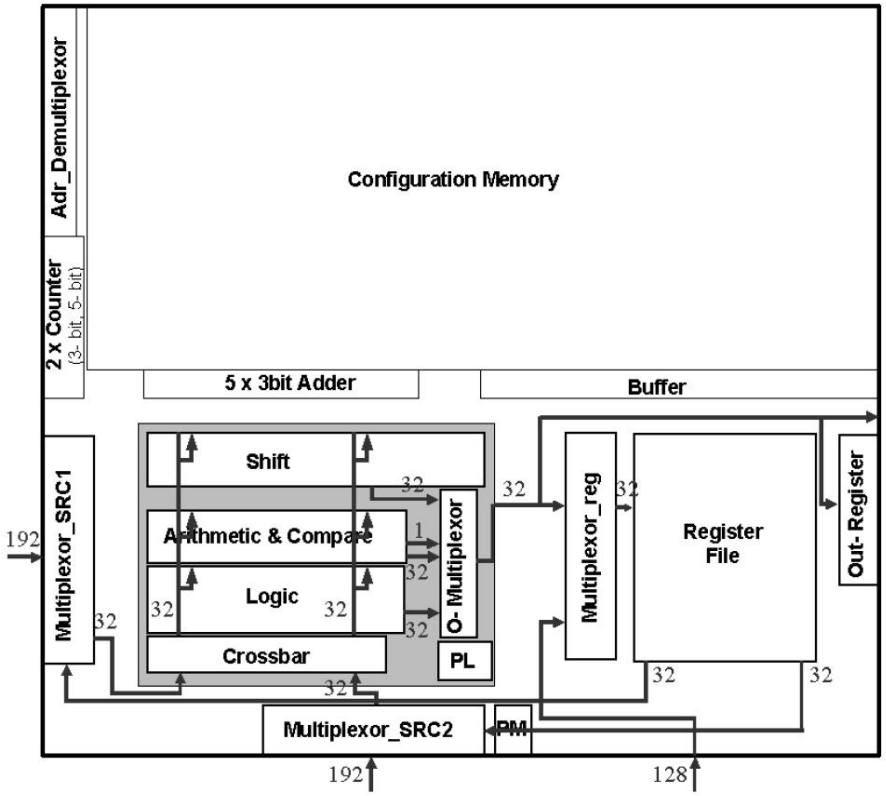
- **ADRES**

H.264 Decoder Performance Comparison

Bitstream	Frames/s (VLIW)	Frames/s (ADRES)	Speed-up (kernel/overall)
foreman_bf_264k	13.5	25.4	4.2/1.88
foreman_nobf_308k	15.0	26.2	4.2/1.75
mobile_nobf_320k	15.9	30.9	4.3/1.94
mobile_nobf_1366k	9.47	13.3	4.3/1.41

Trend in CGRA and Design Examples

- **ADRES**
  - Physical implementation (90nm, 400 MHz)



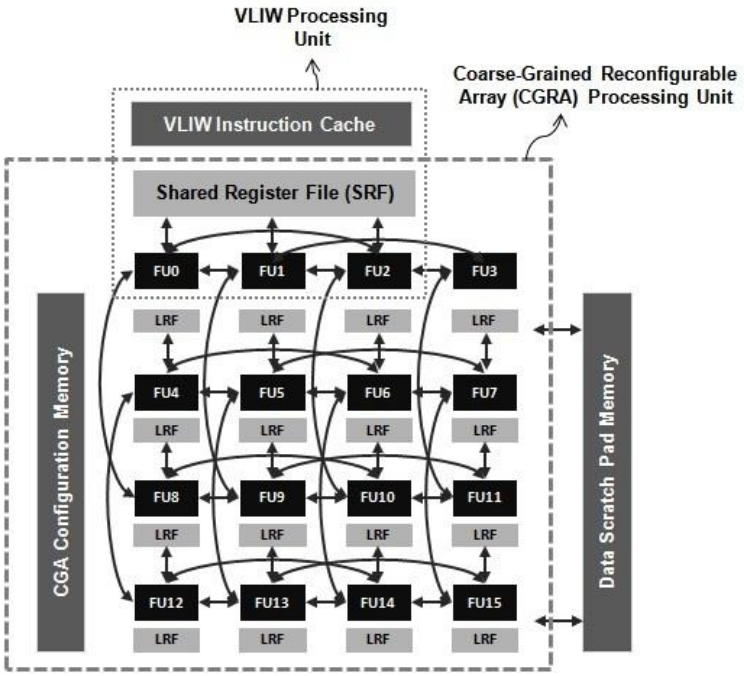
Floorplan of the Functional Unit in ADRES.

Trend in CGRA and Design Examples

- **SRP (Samsung Reconfigurable Processor)**

Nation	Institution	Name	Year	Design Space Exploration	Compiler	Chip
Korea	Samsung Electronics	SRP	2008	O	O	O

  - Actually, SRP is an extended version of ADRES



Conclusion

- CGRA for embedded systems
  - FPGA-like flexibility & ASIC-like performance
  - Diverse R&D works have results in various CGRAs.

CGRA Optimizations

Contents

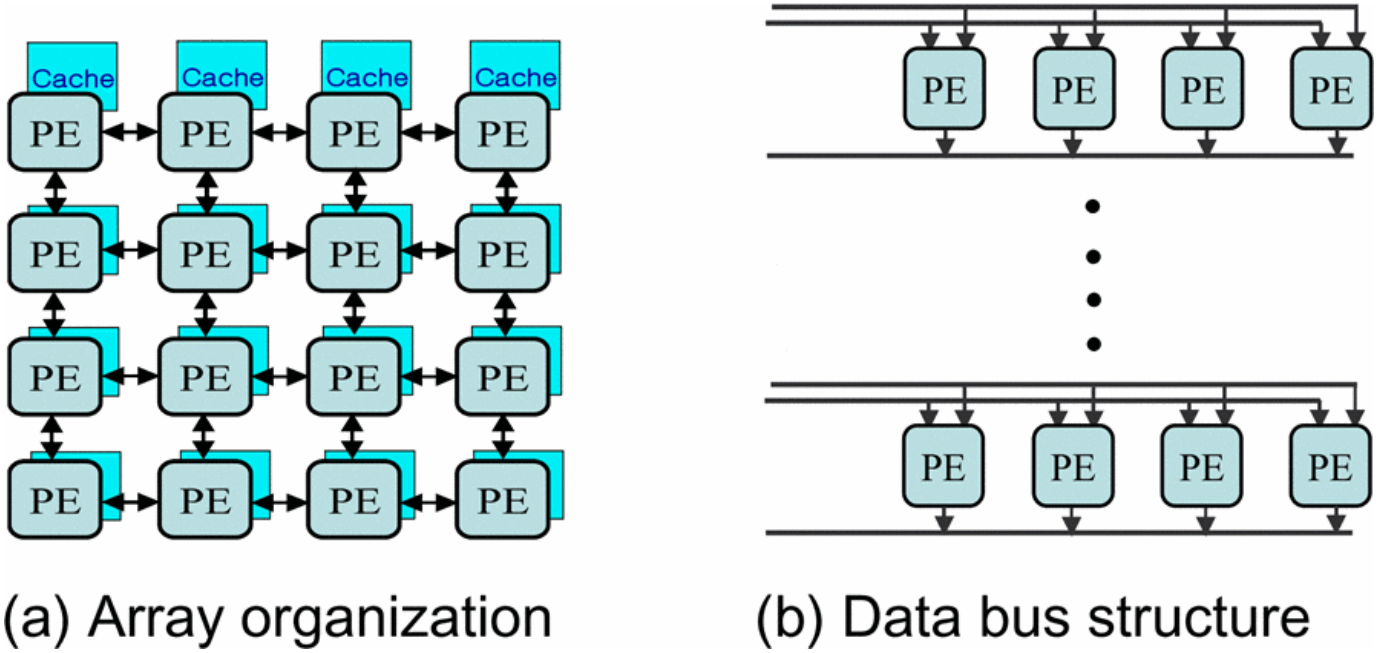
- Resource sharing and pipelining in CGRA for domain-specific optimization - DATE'05
- Power-conscious configuration cache structure and code mapping for CGRA - ISLPED'06
- Mapping multi-domain applications on CGRA - TCAD'11
- Software-level approaches for tolerating transient faults in CGRA - TDSC'13

Resource Sharing and Pipelining

- Reduce the area up to 42.8%
  - Area-critical resources are shared among the primitive PEs.
- Reduce the critical path delay up to 34.7%
  - Delay-critical resources are pipelined to curtail the overall critical path so as to increase the system clock frequency.
  - Performance enhancement up to 35.7%.

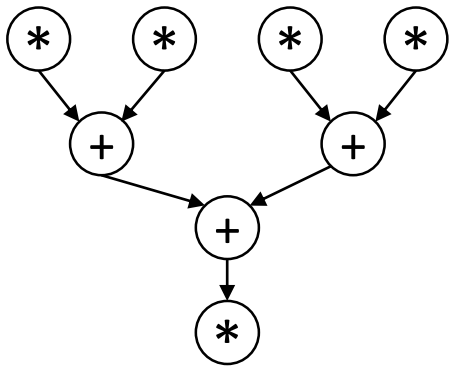
Resource Sharing and Pipelining

- 4x4 reconfigurable array example
  - Each row of the array shares R/W buses
  - 2 read and 1 write buses



Resource Sharing and Pipelining

- Loop pipelining of matrix multiplication of order 4
  - $C \times (a\ b\ c\ d)\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = C \times (ax + by + cz + dw)$

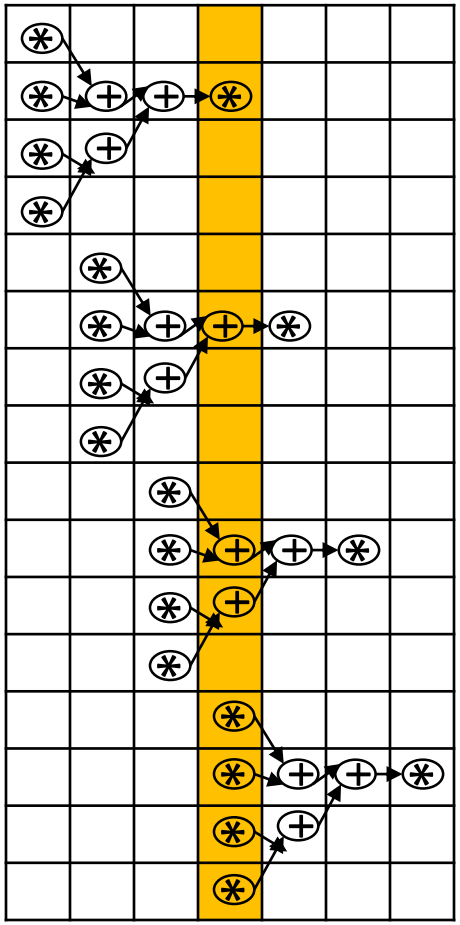


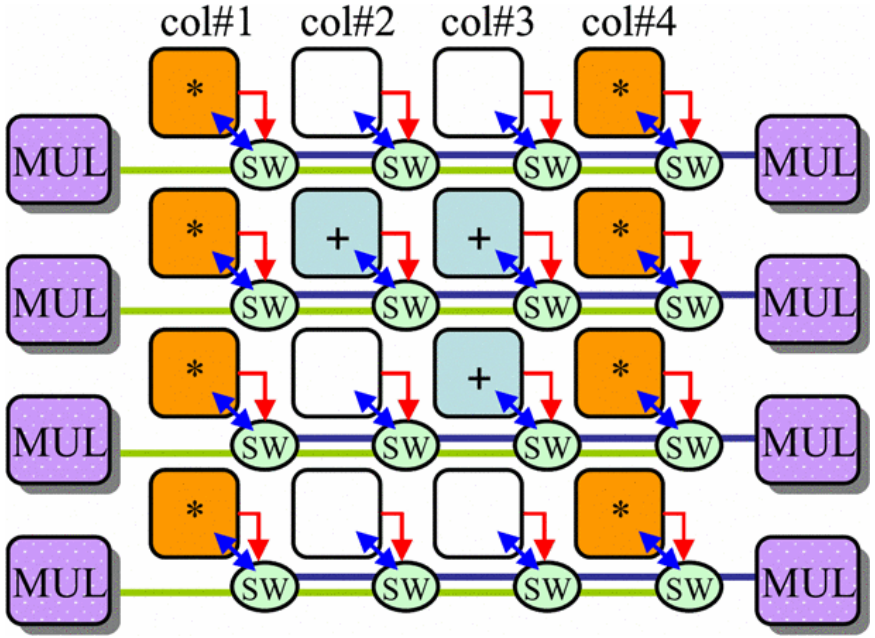
	1	2	3	4	5	6	7	8
col#1	Ld	*	+	+	*	St	Ld	*
col#2		Ld	*	+	+	*	St	Ld
col#3			Ld	*	+	+	*	St
col#4				Ld	*	+	+	*

Ld: load operation, St: store operation

Resource Sharing and Pipelining

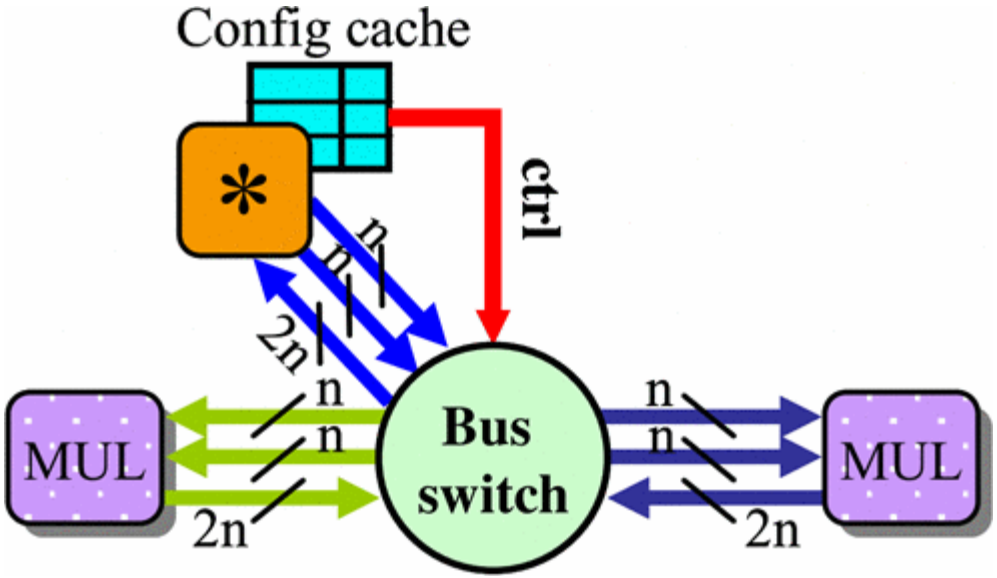
- Multipliers sharing among 16 PEs





Resource Sharing and Pipelining

- Connection between a PE and shared MULs
  - 16bit ALU
  - 8bit / 16 bit MUL



Resource Sharing and Pipelining

- Loop pipelining with pipelined multiplier
  - Requires only 4 multipliers without any stall whereas the non-pipelined scheduling requires
    8 multipliers without stall or
    4 multipliers with 1 cycle stall

	1	2	3	4	5	6	7	8	9	10
col#1	Ld	1*	2*	+	+	1*	2*	St	Ld	1*
col#2		Ld	1*	2*	+	+	1*	2*	St	Ld
col#3			Ld	1*	2*	+	+	1*	2*	St
col#4				Ld	1*	2*	+	+	1*	2*

1*/2* : first/second stage of pipelined multiplication

Resource Sharing and Pipelining

- Performance estimation according to the schedule
  - Latency (cycle * clk freq) by RS stall & RP delay
- Estimated HW cost for design space exploration
  -

$$HW_{cost} = \{n \times m \times (Sh\_PE_{area} + Reg_{area} + SW_{area}) + Sh\_Res_{area} \times (n \times shr + m \times shc)\} < n \times m \times PE_{area}$$

where

n, m : number of row, column

PE_{area} : area of a PE with shared resource

Sh_PE_{area} : area of a PE without shared resource

Reg_{area} : area of registers in a PE for pipelined operation

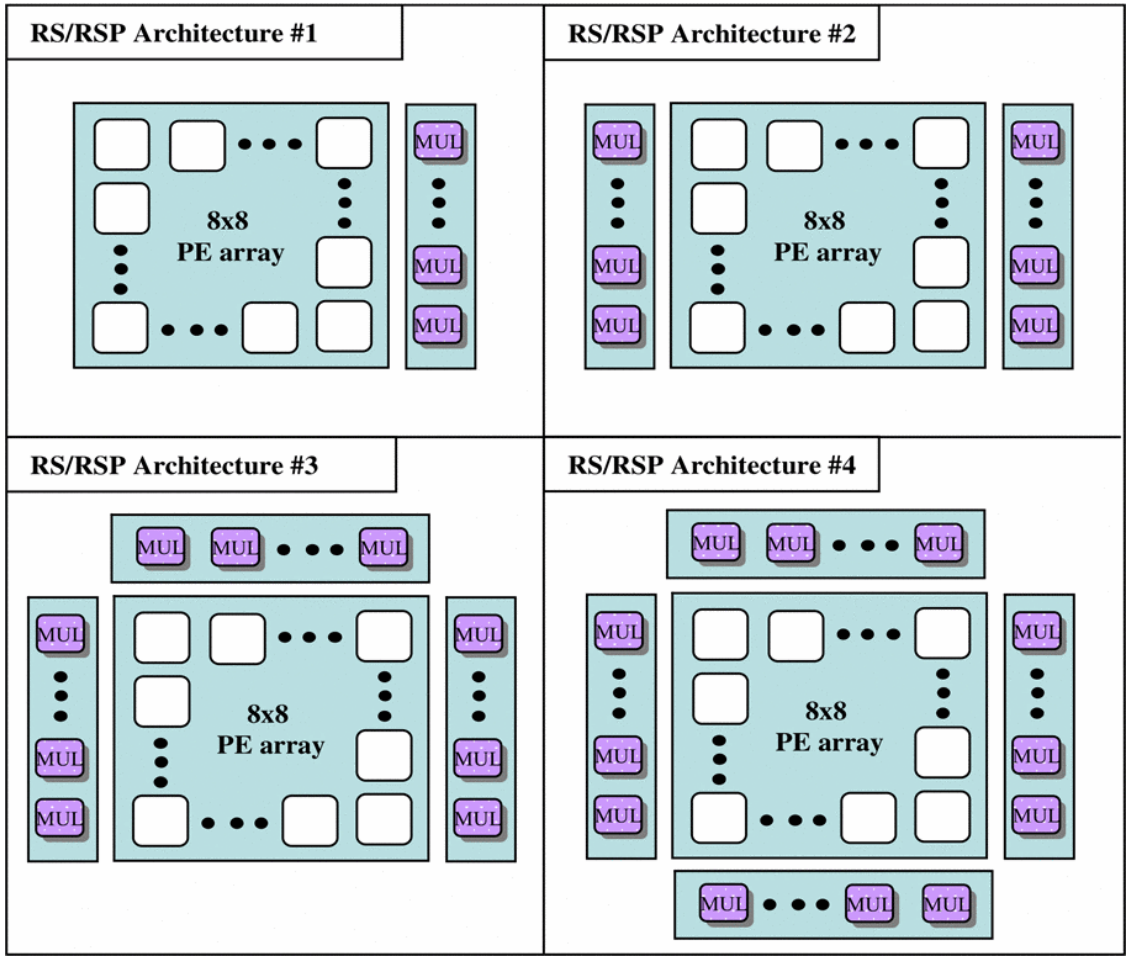
SW_{area} : area of a bus switch

Sh_Res_{area} : area of a shared resource

shr/shc : number of rows/columns of the shared resources

Resource Sharing and Pipelining

- Architecture evaluation



Resource Sharing and Pipelining

- Synthesis result of various architecture examples

Arch'	Area (No. of slices)				Critical path delay(ns)			
	PE	SW	Array	R(%)	PE	SW	Array	R(%)
Base	910	-	55739	0	25.6	-	26	0
RS#1	489	10	32446	**42.8**	15.3	0.7	26.85	-4.88
RS#2		34	36816	34.05		1.2	27.97	-9.25
RS#3		55	40577	27.02		1.8	28.89	-11.11
RS#4		68	44768	19.69		2.0	30.23	-16.27
RSP#1		10	33249	40.35		0.7	16.72	**34.69**
RSP#2		34	38422	31.07		1.2	17.26	32.58
RSP#3		55	42987	22.88		1.8	18.21	29.97
RSP#4		68	47981	13.92		2.0	18.83	27.58

R(%): Reduction ratio compared with Base architecture

SW: Bus switch

Resource Sharing and Pipelining

Table 4. Performance evaluation of the kernels in Livermore loop benchmark suite

Arch'	Hydro(32†)				ICCG(32†)				Tri-diagonal(64†)				Inner product(128†)				State(16†)			
	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall
Base	15	390	0	-	18	468	0	-	17	442	0	-	21	546	0	-	20	520	0	-
RS#1	19	510.15	-30.80	4	18	483.3	-3.26	0	17	456.45	-3.26	0	21	563.85	-3.26	0	35	939.75	-80.72	15
RS#2	15	419.55	-1.07	0	18	503.46	-7.58	0	17	475.49	-7.58	0	21	587.37	-7.58	0	20	559.4	-7.58	0
RS#3	15	433.35	-11.11	0	18	520.02	-11.11	0	17	491.13	-11.11	0	21	606.69	-11.11	0	20	577.8	-11.11	0
RS#4	15	453.45	-16.27	0	18	544.14	16.27	0	17	513.91	-16.27	0	21	634.83	-16.27	0	20	604.6	-16.27	0
RSP#1	21	351.12	10	2	19	317.68	**32.12**	0	18	300.96	**31.91**	0	22	367.84	**32.64**	0	37	618.64	-18.96	14
RSP#2	19	327.94	**15.92**	0	19	327.94	29.93	0	18	310.68	29.71	0	22	379.72	30.45	0	23	396.68	**23.65**	0
RSP#3	19	345.99	11.28	0	19	345.99	26.07	0	18	327.78	25.84	0	22	400.62	26.62	0	23	418.83	19.45	0
RSP#4	19	357.77	8.26	0	19	357.77	23.55	0	18	338.94	23.31	0	22	414.26	24.12	0	23	433.09	16.71	0

Kernel(No.†) : iteration number of the kernel, **ET(ns)** : Execution Time = cycle × Critical path delay(ns), **DR(%)** : Delay Reduction percentage, **stall** : stall number of resource lack, **bold number** : the largest amount of delay reduction %

Table 5. Performance evaluation of 2D-FDCT, SAD, MVM and FFT function

Arch'	2D-FDCT in H.263 enc				SAD in H.263 enc				*MVM(64†)				Multiplication Loop in FFT(32†)			
	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall	cycle	ET(ns)	DR(%)	stall
Base	32	832	0	-	39	1014	0	-	19	494	0	-	23	598	0	-
RS#1	56	1503.6	-80.72	24	39	1047.15	-3.26	0	19	510.15	-3.26	0	37	993.45	-66.12	14
RS#2	38	1062.86	-7.58	6	39	1090.83	-7.58	0	19	531.43	-7.58	0	23	643.31	-7.58	0
RS#3	32	924.48	-11.11	0	39	1126.7	-11.11	0	19	548.91	-11.11	0	23	664.47	-11.11	0
RS#4	32	967.36	-16.27	0	39	1178.97	-16.27	0	19	574.37	-16.27	0	23	695.29	-16.27	0
RSP#1	64	1070.08	-28.61	24	39	652.08	**35.7**	0	20	334.4	**32.31**	0	40	668.8	-11.83	13
RSP#2	40	690.4	**17.01**	0	39	673.14	33.61	0	20	345.2	30.12	0	27	466.02	**22.07**	0
RSP#3	40	728.4	12.45	0	39	710.19	29.96	0	20	364.2	26.27	0	27	491.67	17.78	0
RSP#4	40	753.2	9.47	0	39	734.37	27.57	0	20	376.6	23.76	0	27	508.41	14.98	0

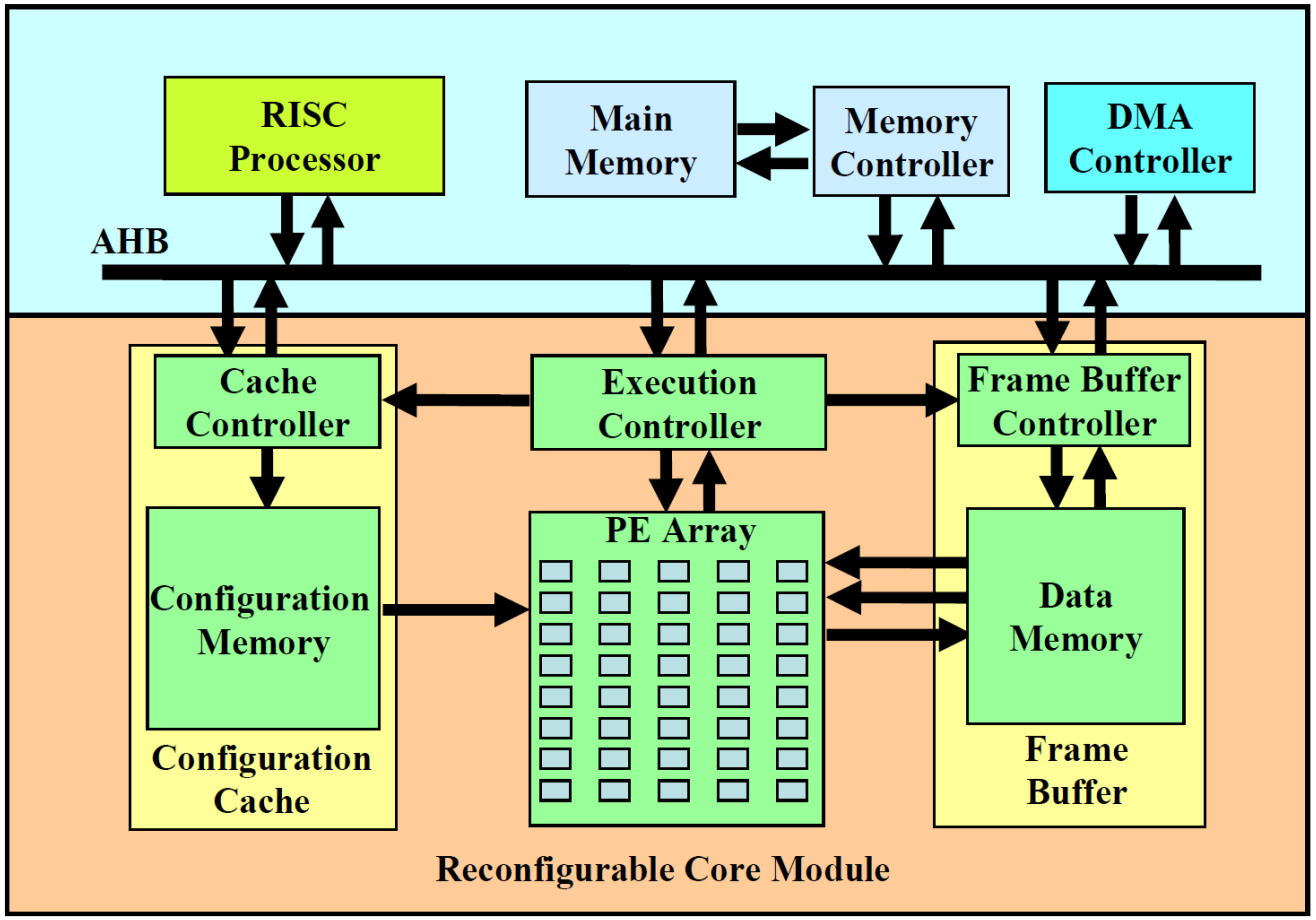
***MVM :** Matrix Vector Multiplication

Configuration Cache Structure

- Power reduction on average 32%
  - Reduce the configuration cache through context pipelining

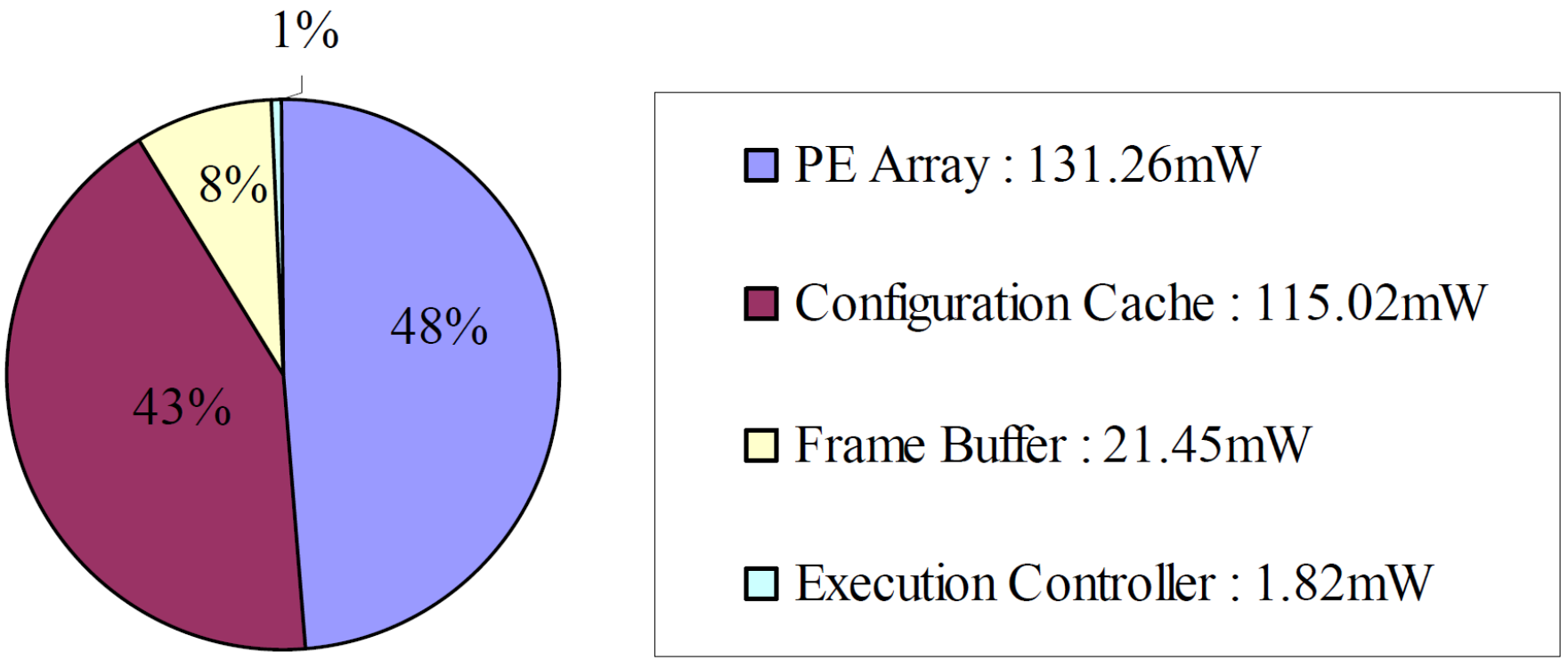
Configuration Cache Structure

- Block diagram of the typical CGRA



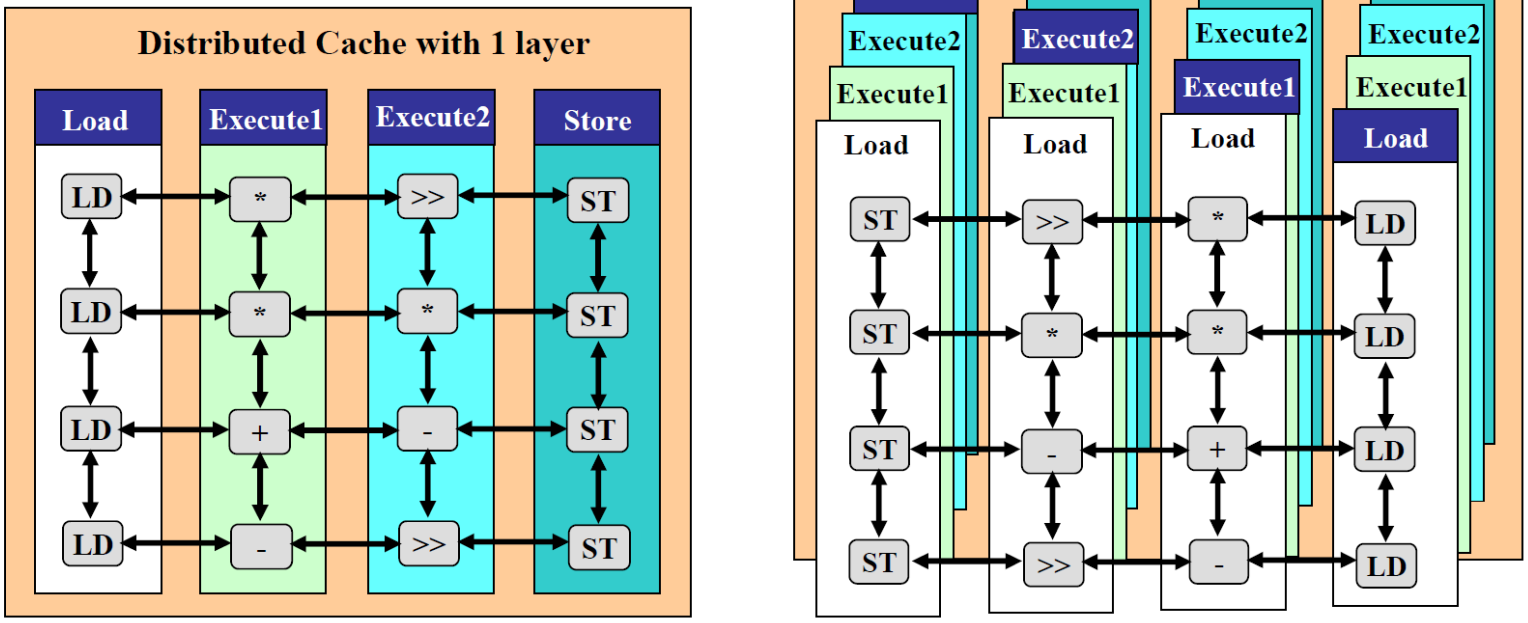
Configuration Cache Structure

- Power breakdown of CGRA
  - 2D-FDCT simulation result



Configuration Cache Structure

- Spatial mapping vs Temporal mapping

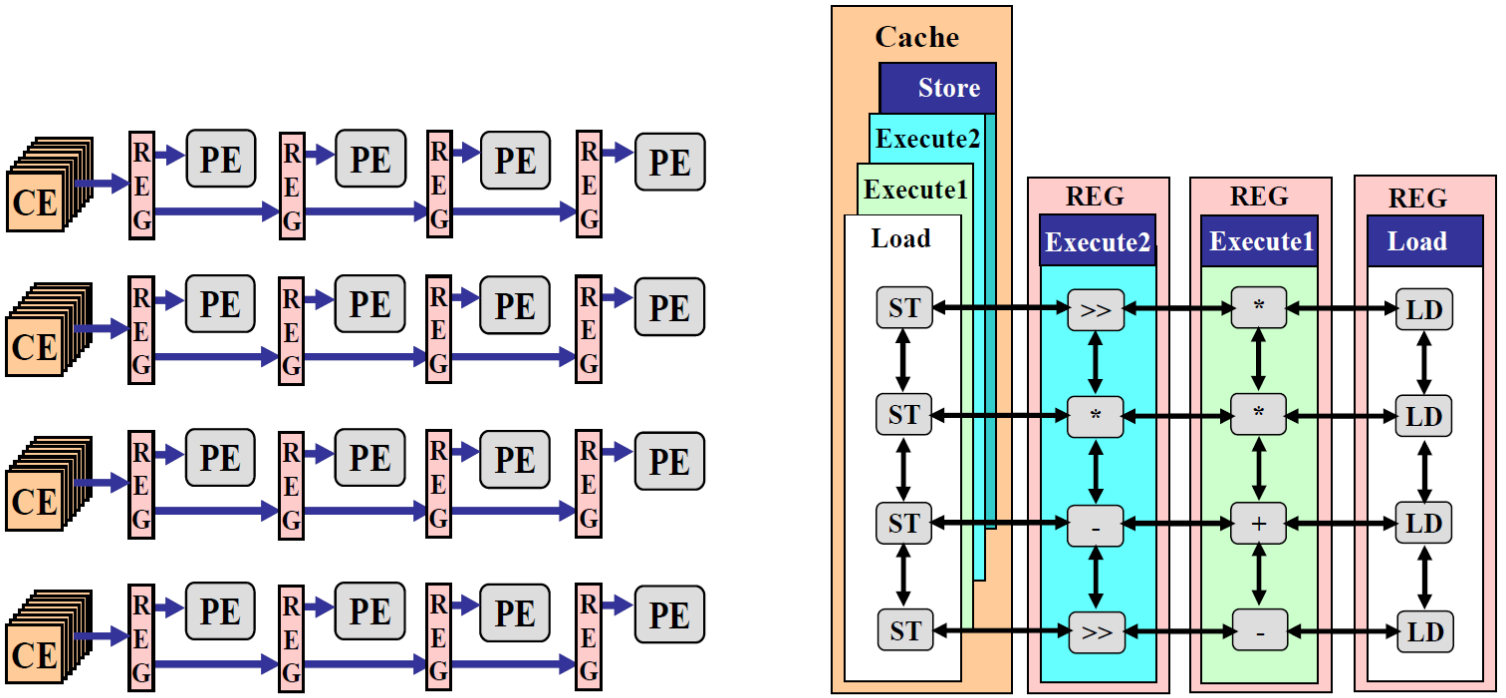


Configuration Cache Structure

- 2D Spatial mapping
  - Fixed operation with static configuration
  - Requires large PE array for large DFG
  - Limited data dependency
- 1D Temporal mapping
  - Change operations cycle by cycle
  - Requires large configuration cache for large DFG
  - Complex data dependency

Configuration Cache Structure

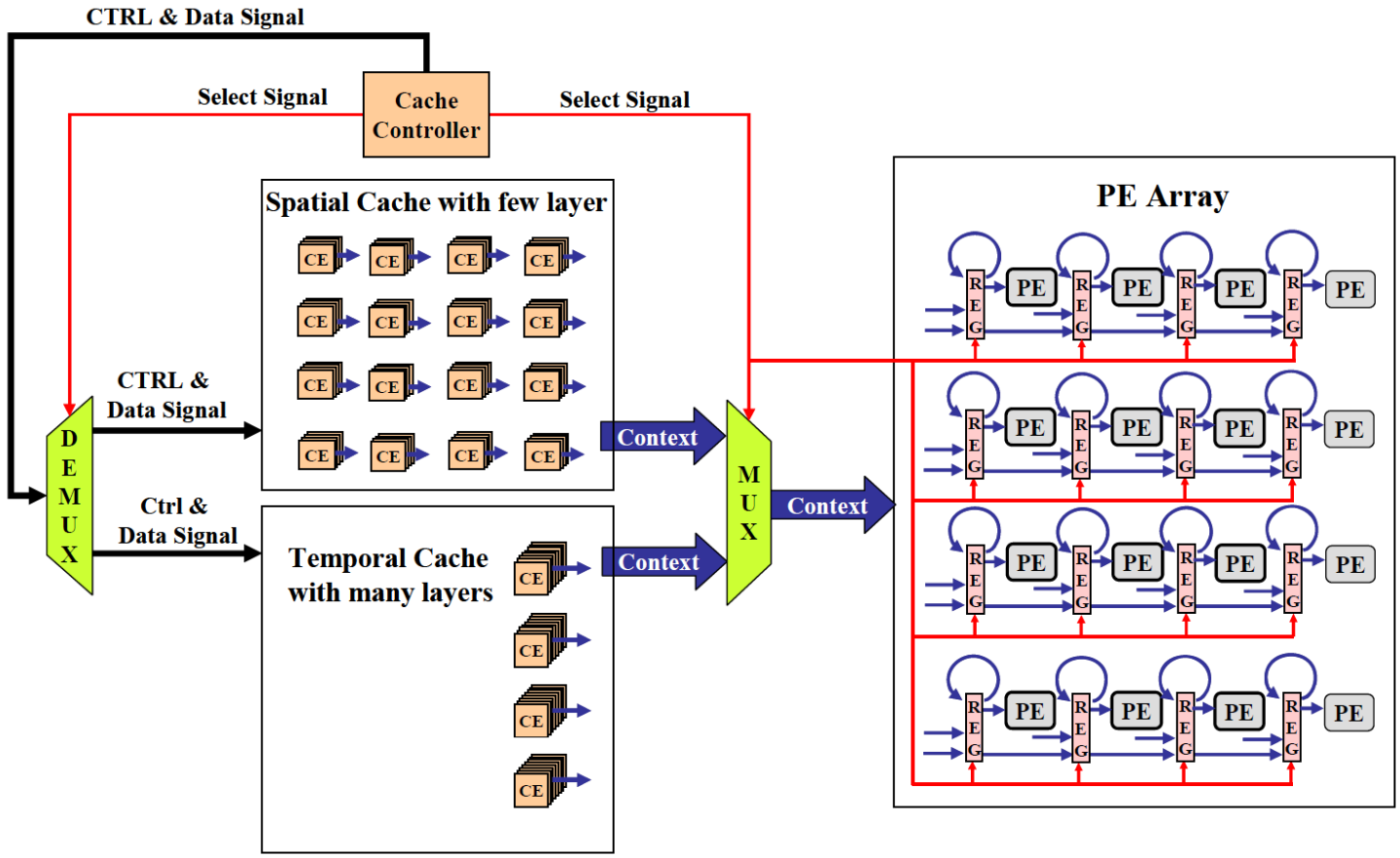
- Configuration cache structure for context pipelining
  - Context is pipelined from the left neighboring column



(a) Proposed cache structure (b) Context pipelining

Configuration Cache Structure

- Hybrid configuration cache structure
  - Context reuse and pipelining



Configuration Cache Structure

- Experimental result

Table 2. Power comparison between mapping techniques

Kernels	Prev	Prop					
	Spatial	Temporal			Spatial		
	Power (mW)	Power (mW)	Reduced(%)		Power (mW)	Reduced(%)	
			Cache	Entire		Cache	Entire
First_Diff	360.63	259.51	**71.3**	**28.0**	241.85	**83.7**	**32.9**
Tri-Diagonal	340.05	257.95	67.7	24.1	240.20	81.9	29.4
Dot_Product	321.05	253.78	65.9	21.0	244.90	74.7	23.7
Complex_Mult	371.13	285.75	67.8	23.0	269.13	81.9	27.5
Hydro	295.20	225.04	65.9	23.8	211.10	78.2	28.5

Table 4. Memory size evaluation

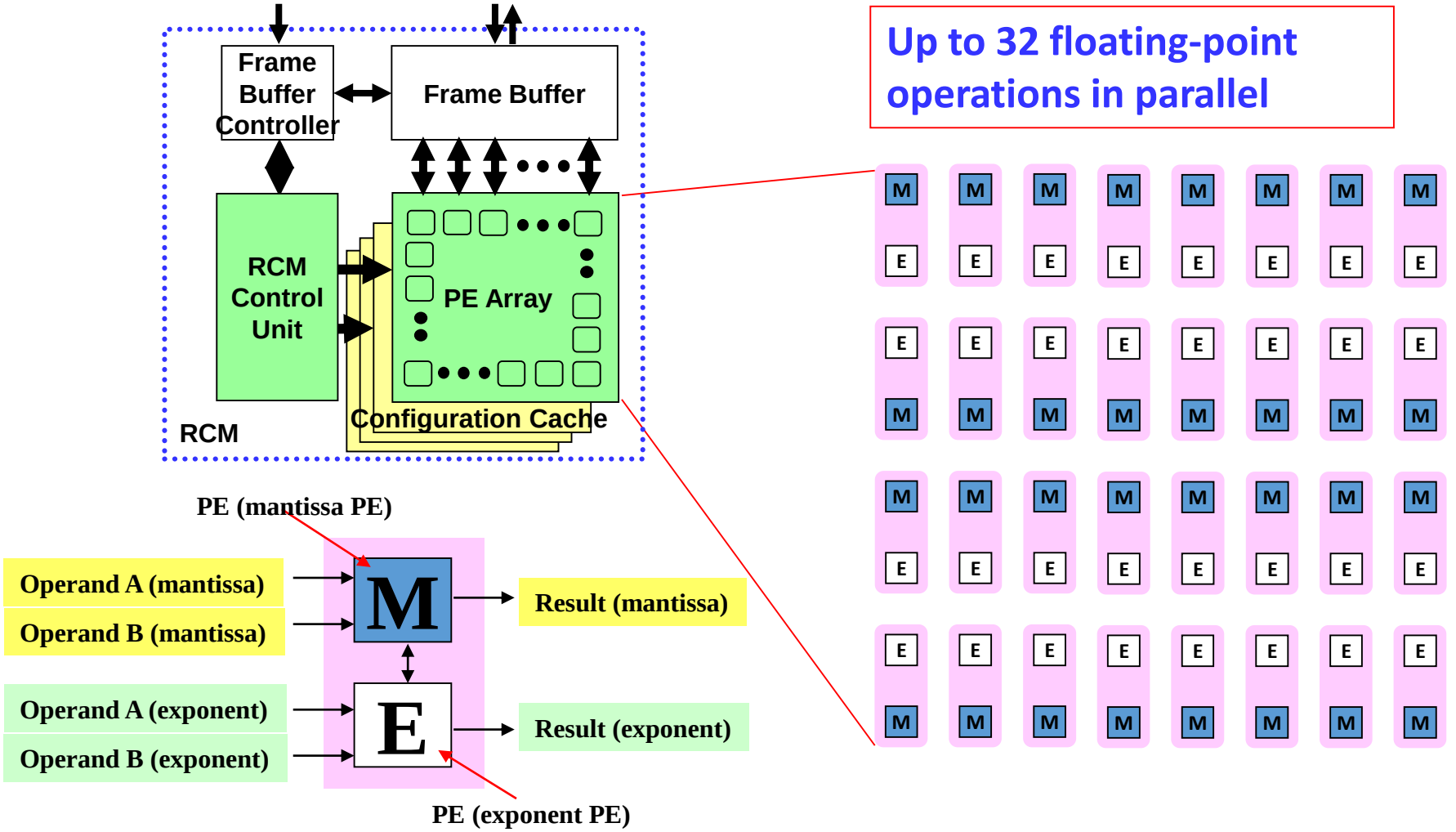
Architecture	Memory Element(Byte)	Reduced(%)
	SRAM in config' cache	
Previous	5120	0
Proposed	3584	**30.0**

Application Mapping onto CGRA

- FloRA Architecture
  - Support floating point operations on CGRA
- Automatic application mapping onto CGRA
  - Automated design flow from C to machine code
    - Support multi-domain applications (floating point operations as well as inter operations)
  - Develop fast heuristic algorithm
    - Around 100x faster but less than 3% differences to the optimal solution
  - Performance enhancement through loop unrolling and pipelining
    - Up to 120x performance improvement compared to software only implementation

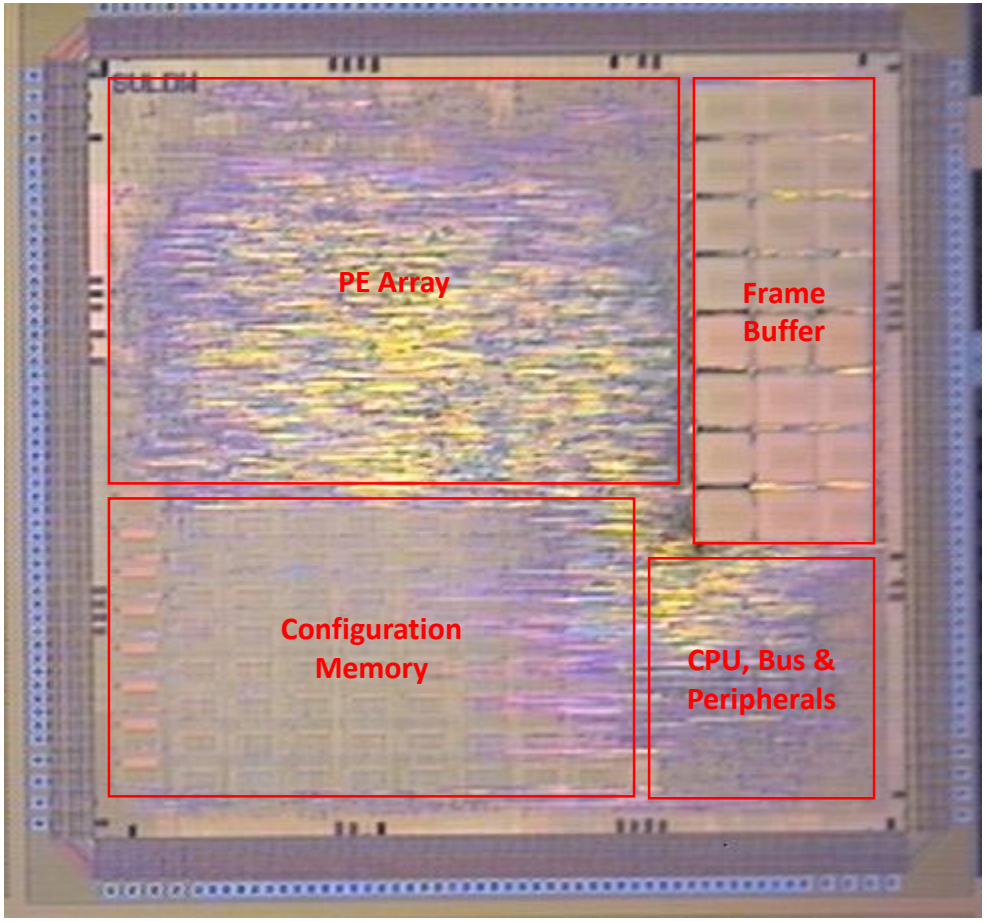
Application Mapping onto CGRA

- FloRA architecture



Application Mapping onto CGRA

- Physical chip design of FloRA



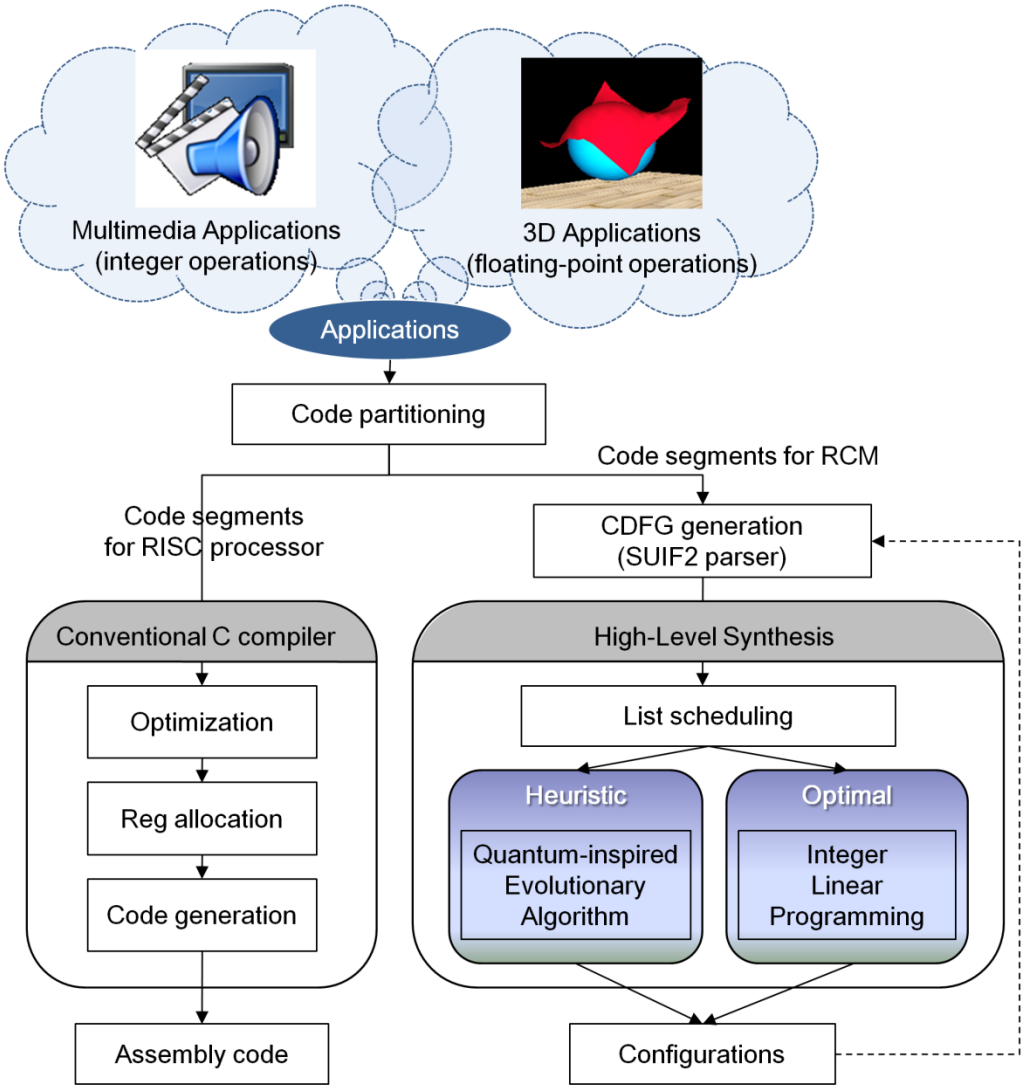
- **PE Array size: 8x8**
- **Configuration Memory:**
  (temporal) 2,560 bytes
  (spatial) 3,072 bytes
- **Frame Buffer: 6,144 bytes**
- **Technology: 130nm**
  (Dongbu HiTek thru IDEC)
- **Clock frequency: 125MHz**
  (typical case)
- **Area: 11.2 mm^2**

Application Mapping onto CGRA

- Chip test (JPEG and Fractal)

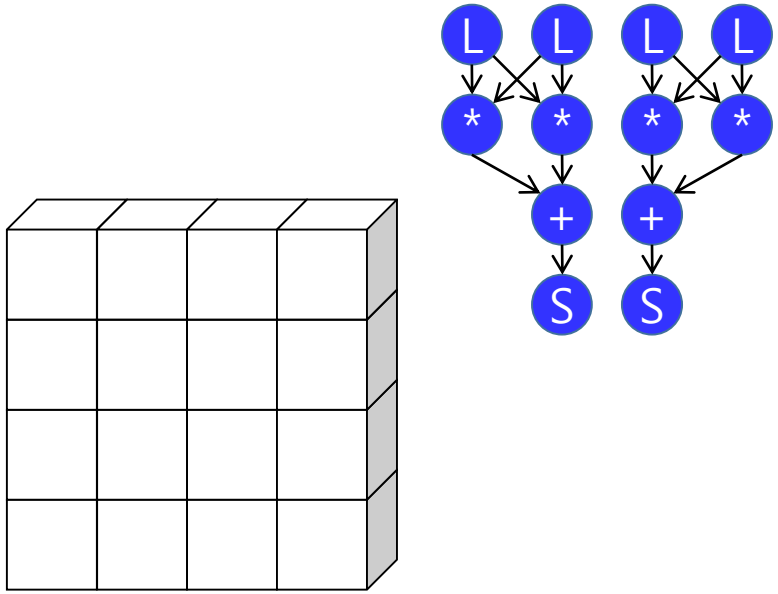
Application Mapping onto CGRA

- Overall flow



Application Mapping onto CGRA

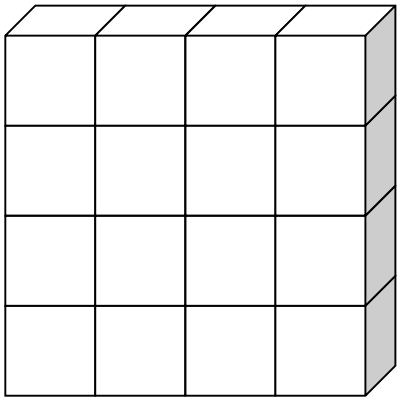
- Spatial mapping

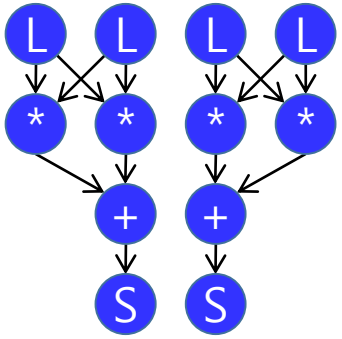


  - Each operation in a loop body is spatially mapped to a dedicated PE
  - Each PE executes a fixed operation with static configuration

Application Mapping onto CGRA

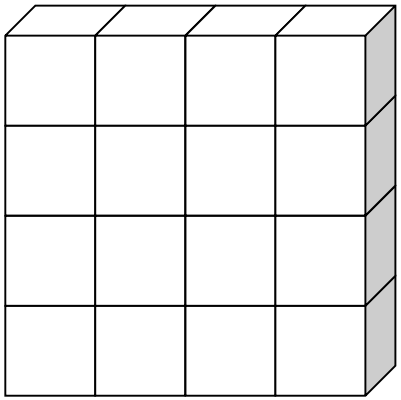
- Temporal mapping
  - A PE executes multiple operations in a loop by changing the configuration dynamically
  - Each column executes an iteration of the loop

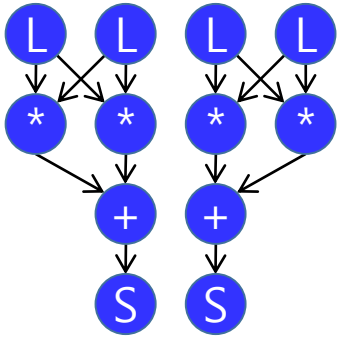




Application Mapping onto CGRA

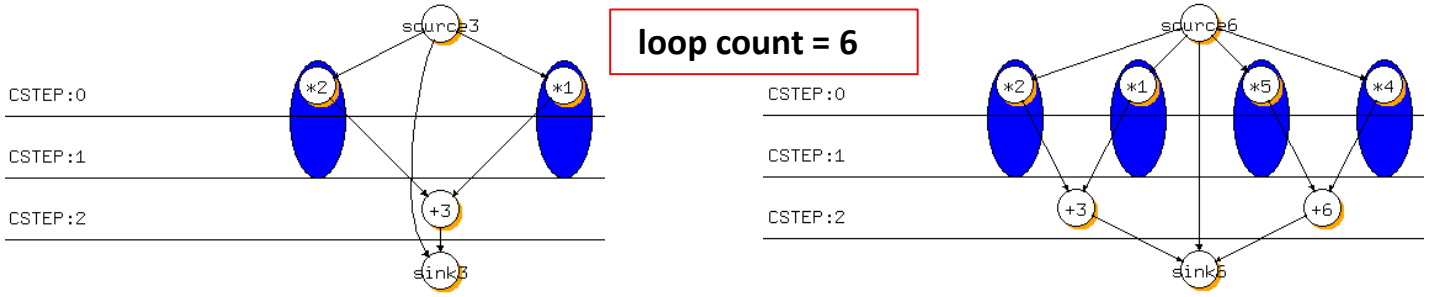
- Temporal mapping
  - Loop pipelining
  - Configuration is also pipelined





Application Mapping onto CGRA

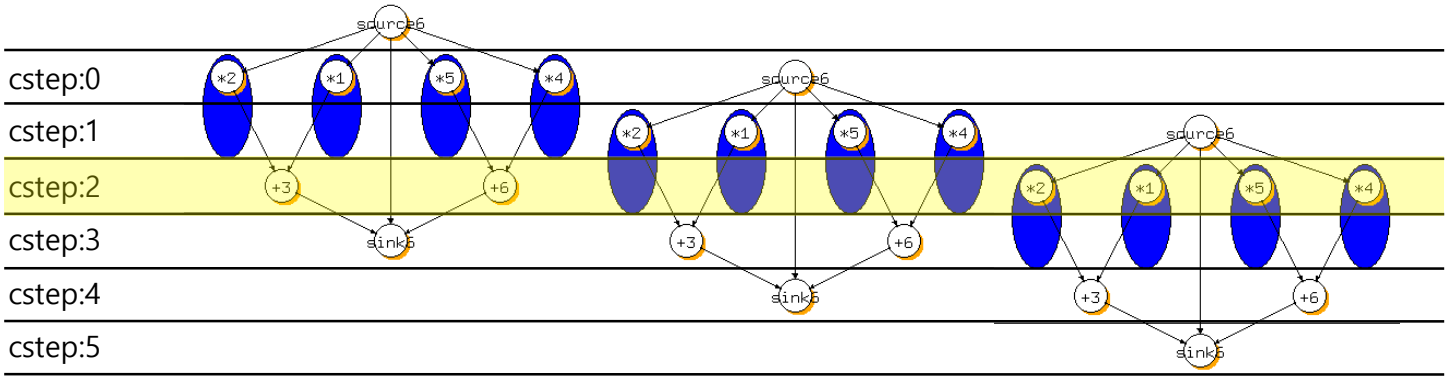
- Loop transformation
  - Loop unrolling



Total latency = 3*6 = 18 cycles

Total latency = 3*3 = 9 cycles

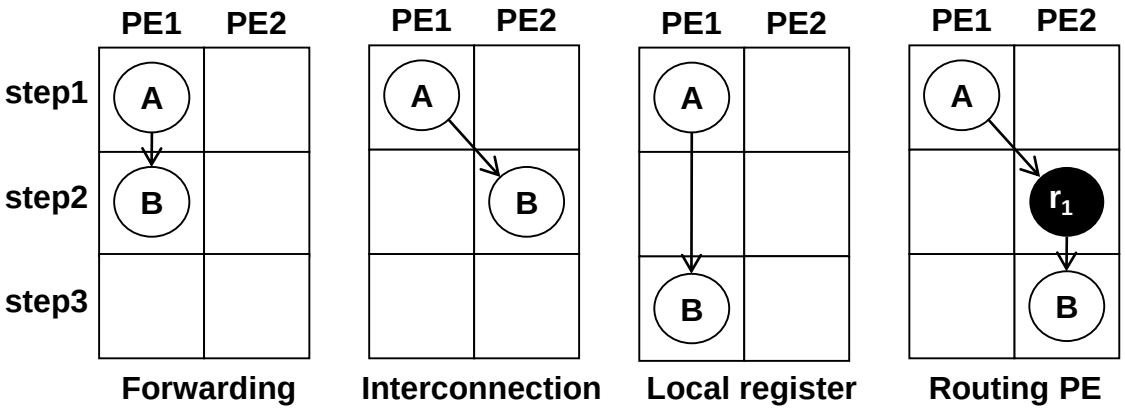
  - Loop pipelining



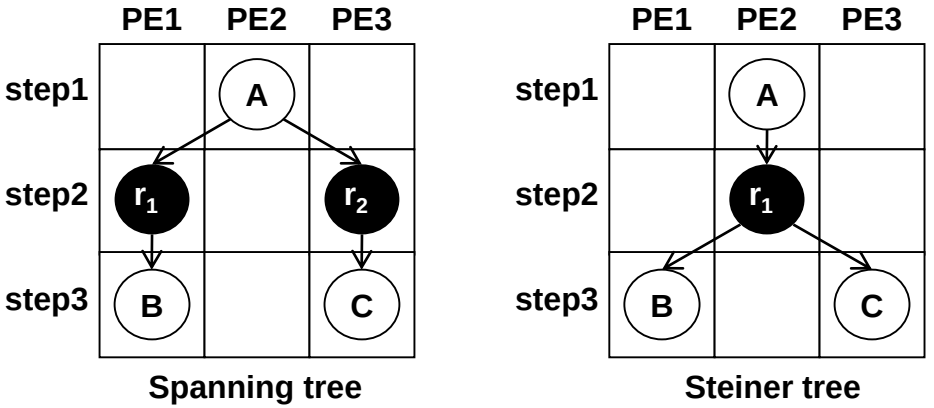
Total latency = 3 + 2*1 = 5 cycles

Application Mapping onto CGRA

- Finding routing paths
  - Single fanout

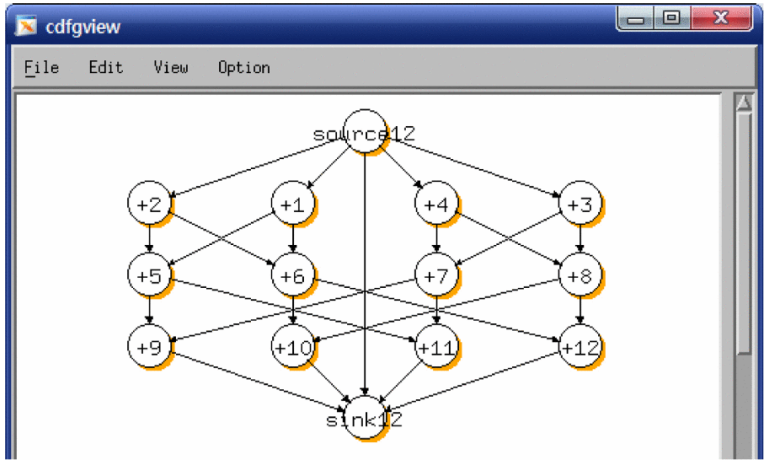


  - Multiple fanout

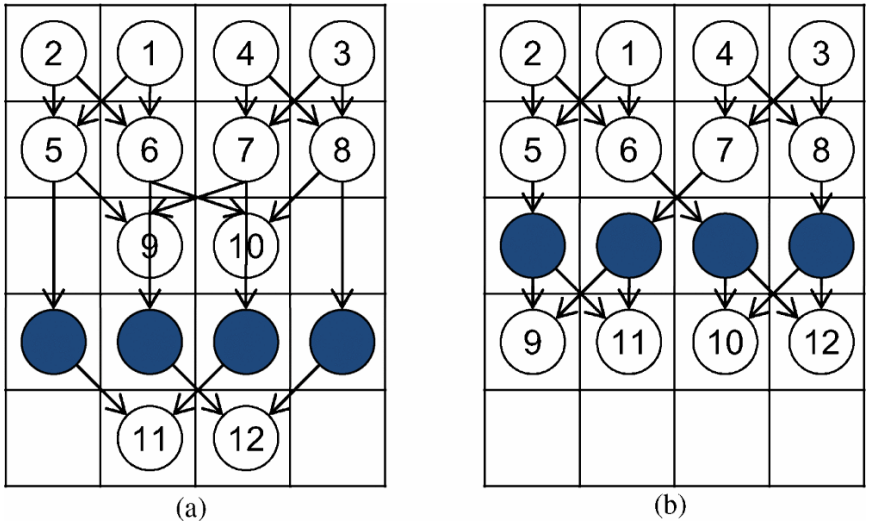


Application Mapping onto CGRA

- Routing problems in butterfly example



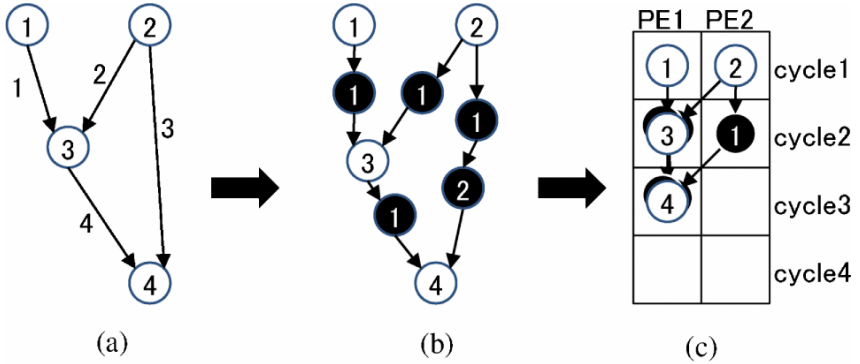
- Data flow graph of butterfly example



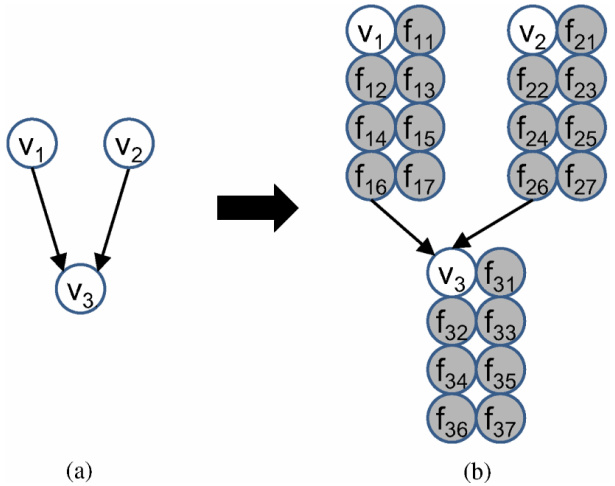
- Total execution cycle varies according to the usage of routing resources

Application Mapping onto CGRA

- Problem formulation
  - Routing resources



  - Floating-point operations

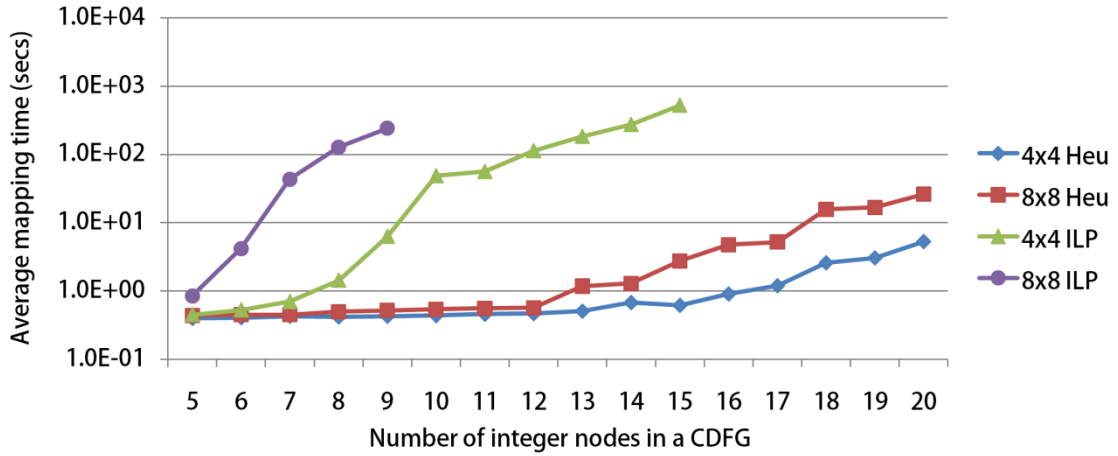


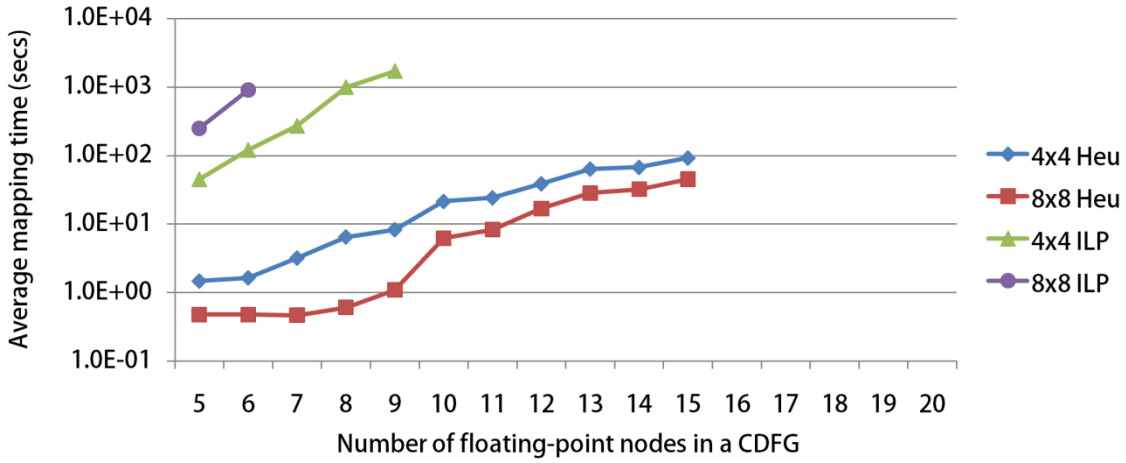
Application Mapping onto CGRA

- Approaches
  - Integer linear programming (ILP)
  - List scheduling
  - Evolutionary algorithm
    - Quantum-inspired Evolutionary Algorithm (QEA)
  - Mixed (proposed)
    - List scheduling + iterative improvement (QEA)

Application Mapping onto CGRA

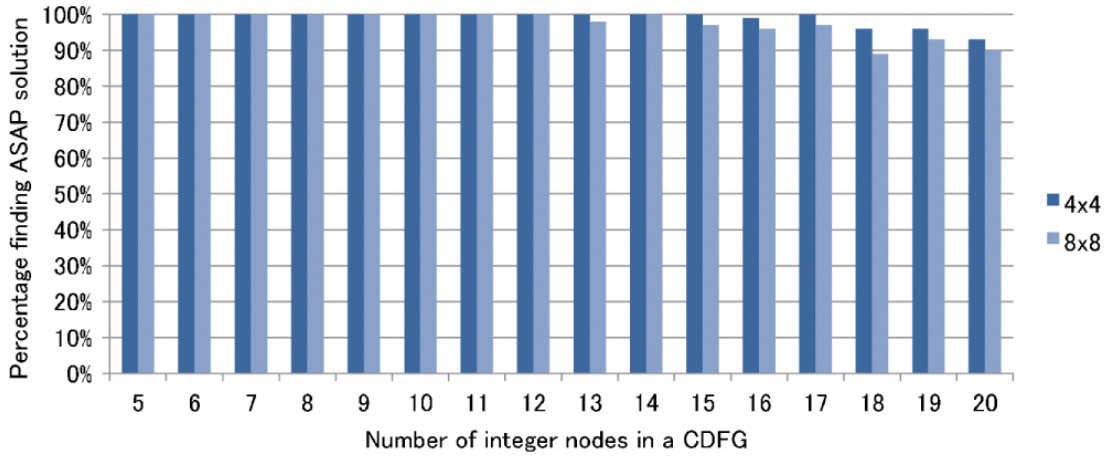
- Mapping time (randomly generated examples)

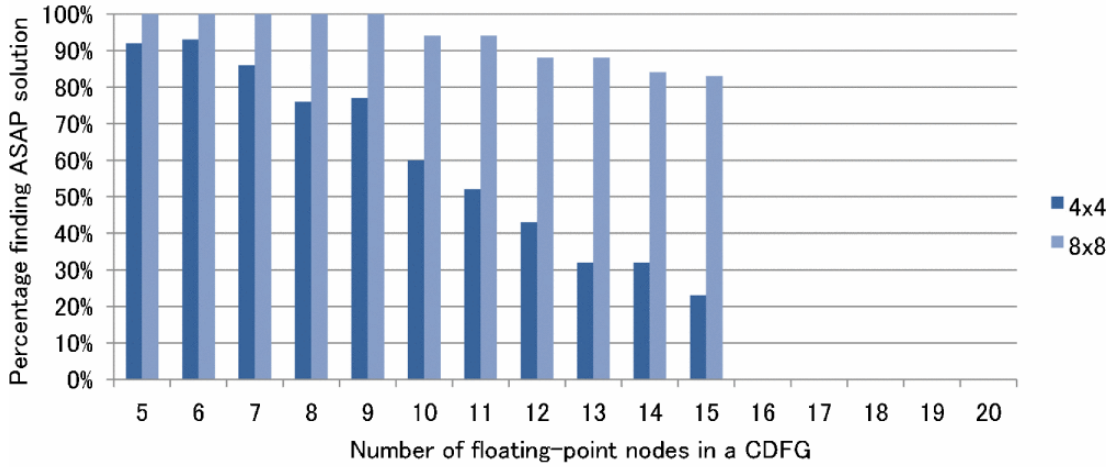




Application Mapping onto CGRA

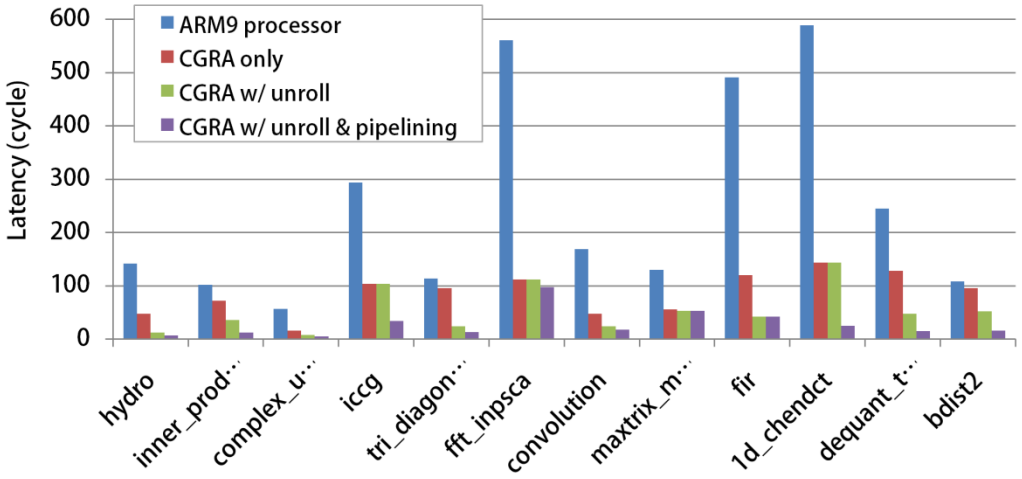
- Percentage finding optimal solutions



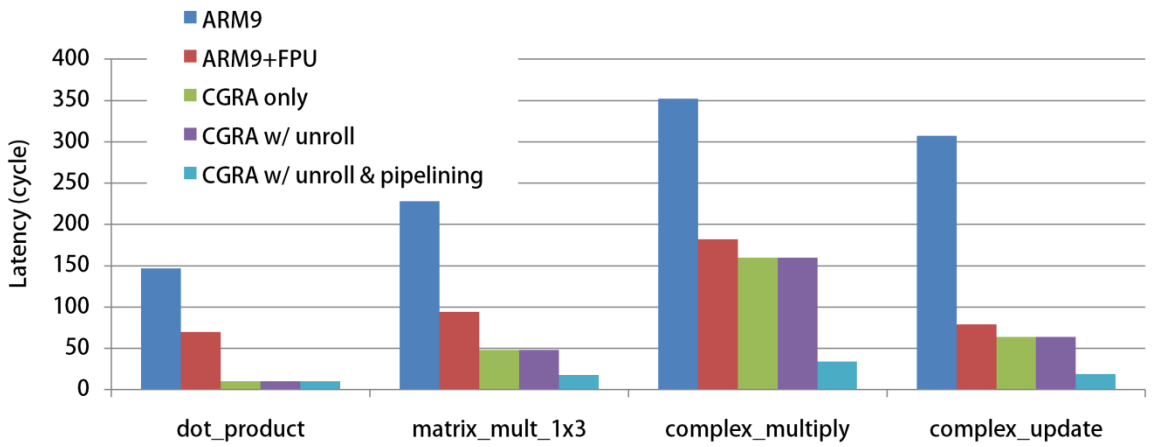


Application Mapping onto CGRA

- Performance for real benchmarks



Integer applications



Floating-point applications

Application Mapping onto CGRA

- Experimental result of integer benchmark examples

	# of Nodes	# of Iter.	Unroll Factor	Mapping Time (s)		Latency (Cycle)			Initiation Interval (Cycle)		
				Heu.	ILP	Heu.	ILP	ASAP	Heu.	ILP	recMII
hydro ($k = 4$)	20	8	4	1	1	6	6	6	1	1	0
inner_product	14	8	2	3	140	5	5	5	1	1	0
complex_update	16	4	2	1	46	4	4	4	1	1	0
iccg (k=8)	16	8	1	108	>86 400	13	–	13	3	–	1
tri_diagonal_elimination (k = 4)	36	8	4	1	1	12	12	12	1	1	0
fft_inpsca	12	16	1	15	>86 400	8	–	5	7	–	4
convolution	8	16	4	1	2	6	6	6	4	4	2
matrix_mult_4 × 4	145	4	4	124	>86 400	60	–	5	1	–	0
fir (k = 40)	240	40	40	228	>86 400	45	–	42	1	–	0
1d_chendct (jpeg_encoder)	60	8	1	221	>86 400	18	–	11	1	–	0
dequant_type1 (mpeg4_decoder)	40	16	4	2	>86 400	12	–	8	1	–	0
bdist2 (mpeg2_encoder)	64	8	2	3	10	12	12	12	3	3	3

- Experimental result of floating-point benchmark examples

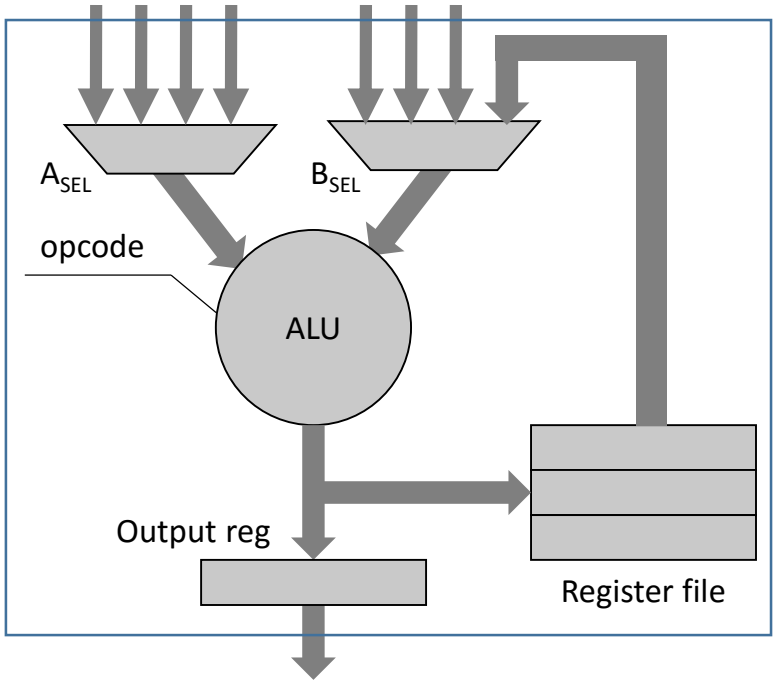
	# of Nodes	# of Iterations	Unroll Factor	Mapping Time (s)		Latency (Cycle)	
				Heuristic	ILP	Heuristic	ILP
dot_product	3	1	1	1	1	10	10
matrix_mult_1 × 3	5	3	1	1	235	18	18
complex_multiply	6	16	1	1	1	34	34
complex_update	8	4	1	1	581	19	19

Tolerating Transient Fault in CGRA

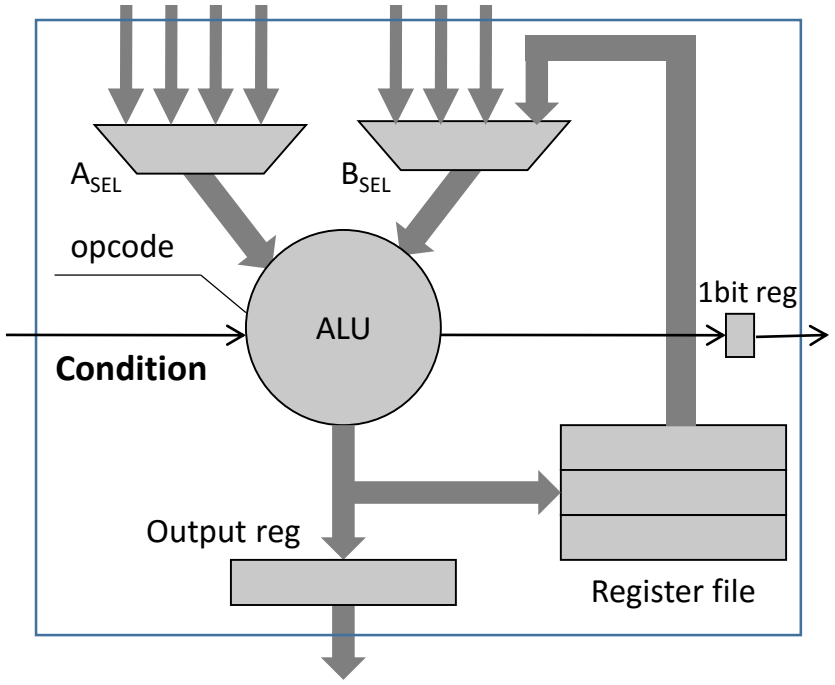
- Fault-tolerant system design through triple-modular redundancy in a software manner
  - Reliability and performance tradeoffs
  - Mathematical reliability analysis on CGRA

Tolerating Transient Fault in CGRA

- Supporting conditional execution
  - Predication



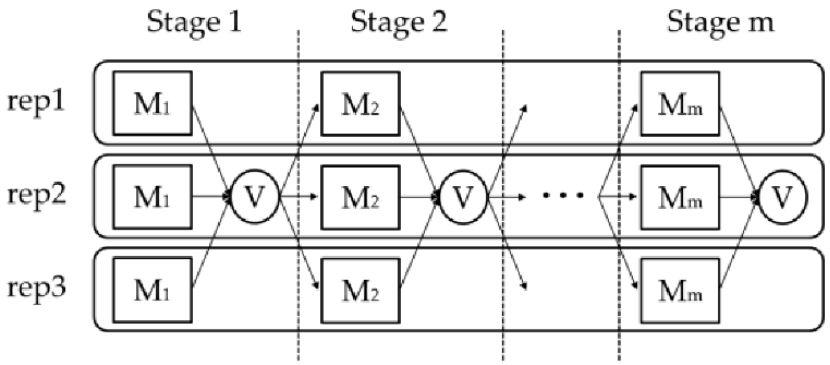
(a) Previous PE structure



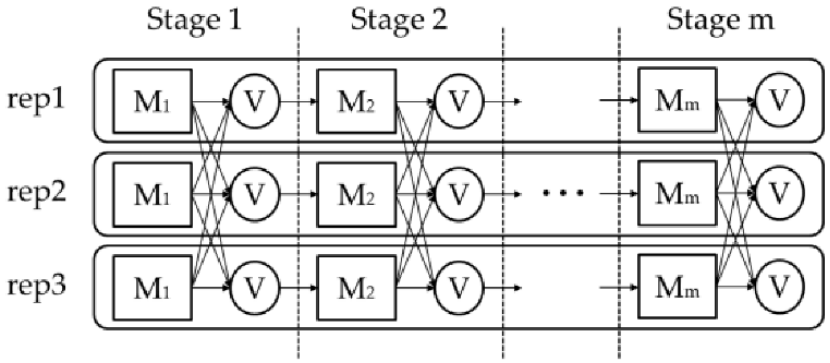
(b) Add 1 bit condition register

Tolerating Transient Fault in CGRA

- Two ways of integrating TMR into systems



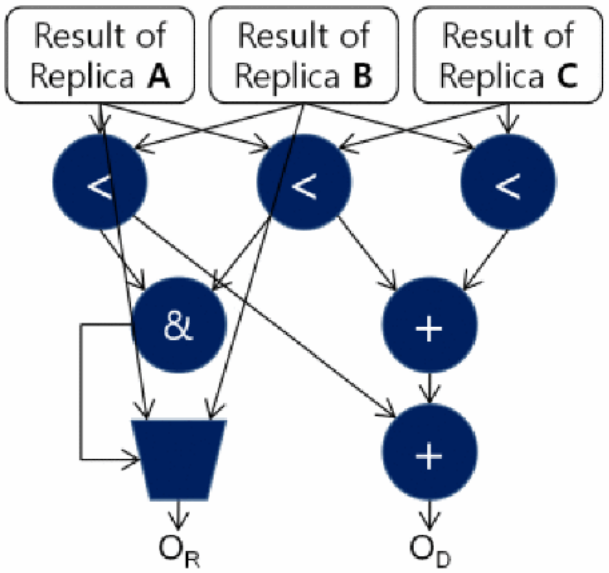
(a) TMR system



(b) AVTMR system

Tolerating Transient Fault in CGRA

- TMR voter design



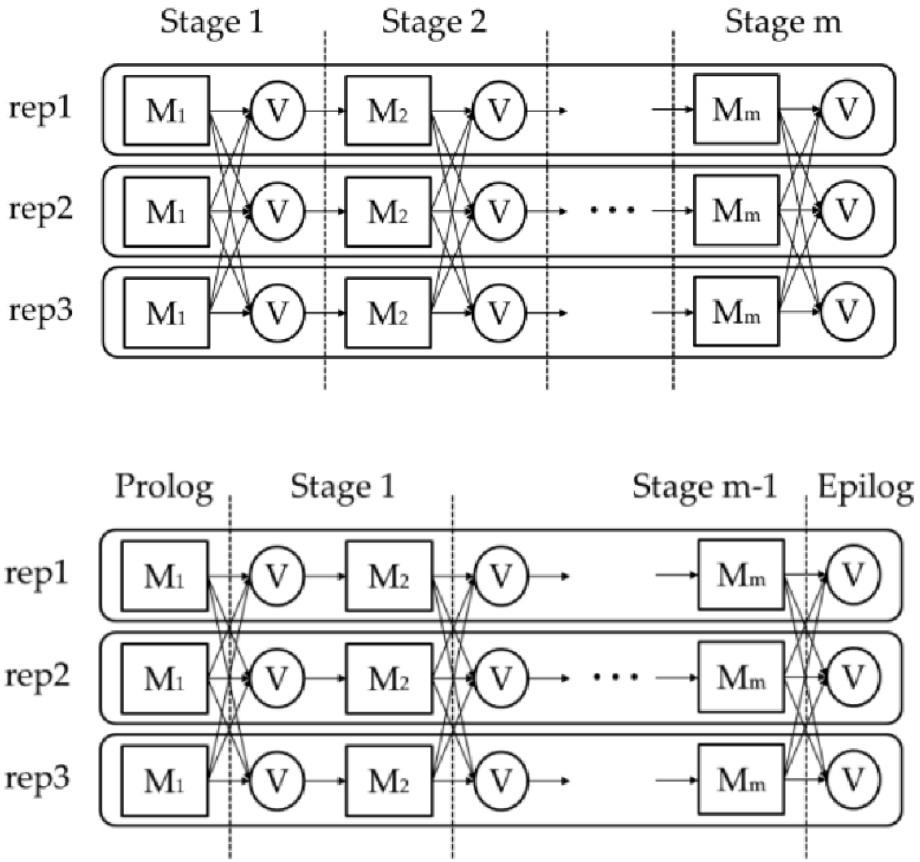
(a) DFG-level design

XYZ	O_D	O_R
000	00	A (or B or C)
001	01	error
010	01	error
011	10	A (or B)
100	01	error
101	10	A (or C)
110	10	B (or C)
111	11	error

(b) Output generation

Tolerating Transient Fault in CGRA

- Mathematical analysis of reliability



Tolerating Transient Fault in CGRA

(1) Reliability of a TMR system

$$R_{1TMR} = {R_O}^3 + 3{R_O}^2(1 - R_O) = 3{R_O}^2 - 2{R_O}^3$$

$$R_{mTMR} = \left(3{\mathrm{R_M}}^2 - 2{\mathrm{R_M}}^3\right)^{\mathrm{m}}$$

$$R_{mTMR} = \left[R_V\left(3{\mathrm{R_M}}^2 - 2{\mathrm{R_M}}^3\right)\right]^{\mathrm{m}}$$

(2) Reliability of a AVTMR system

$$R_{mAVTMR} = (3{\mathrm{R_M}}^2 - 2{\mathrm{R_M}}^3) \times [3(\mathrm{R_M}R_V)^2 - 2(\mathrm{R_M}R_V)^3]^{\mathrm{m-1}} \times \left(3{\mathrm{R_V}}^2 - 2{\mathrm{R_V}}^3\right)$$

$$= (R_M R_V)^{2m}(3 - 2R_M R_V)^{m-1}(3 - 2R_M)(3 - 2R_V)$$

(3) Adaptation to the CGRA

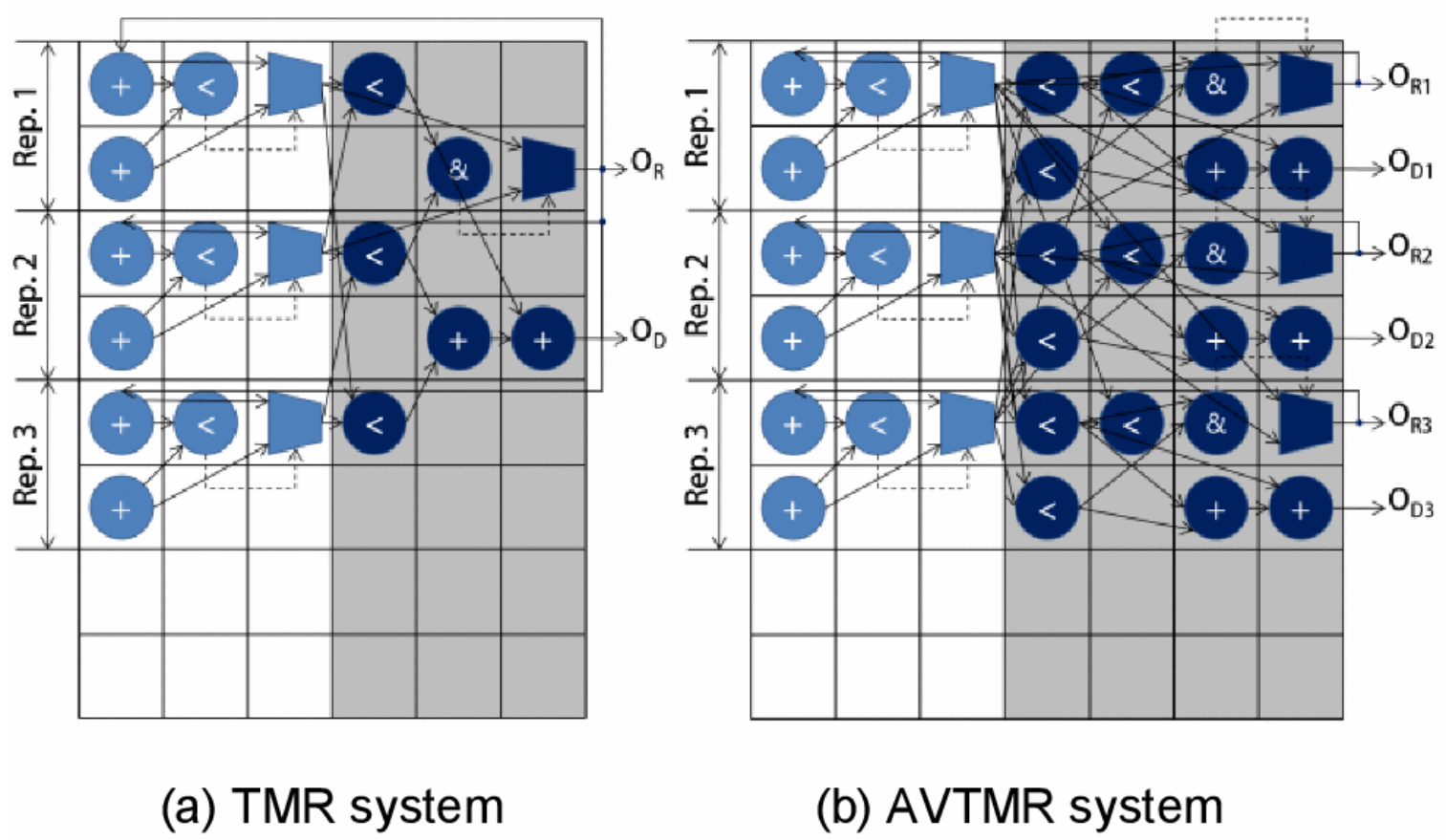
$$R_M = {R_{PE}}^k, \quad R_V = {R_{PE}}^l$$

$$R_{mTMR} = \left(3{R_{PE}}^{2k+l} - 2{R_{PE}}^{3k+l}\right)^m$$

$$R_{mAVTMR} = {R_{PE}}^{2m(k+l)}\left(3 - 2{R_{PE}}^{k+l}\right)^{m-1}\left(3 - 2{R_{PE}}^k\right)\left(3 - 2{R_{PE}}^l\right)$$

Tolerating Transient Fault in CGRA

- Application mapping with different reliability levels

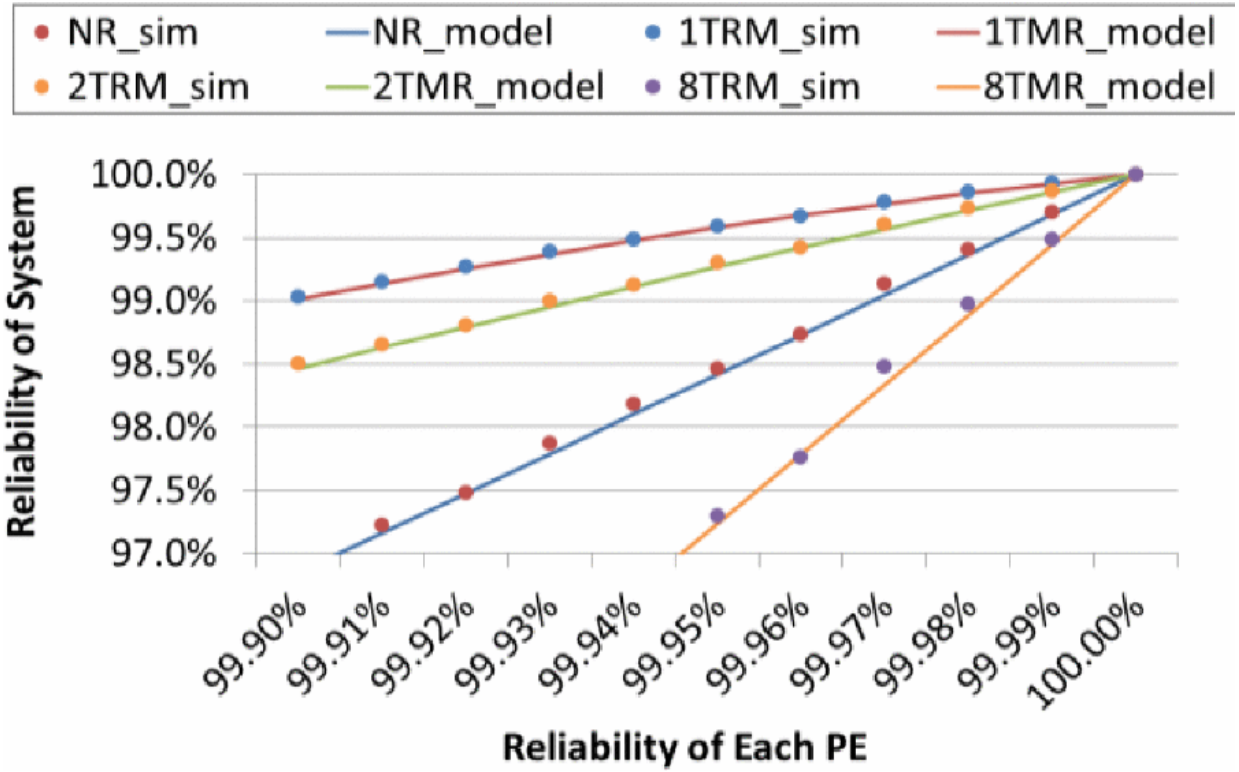


(a) TMR system

(b) AVTMR system

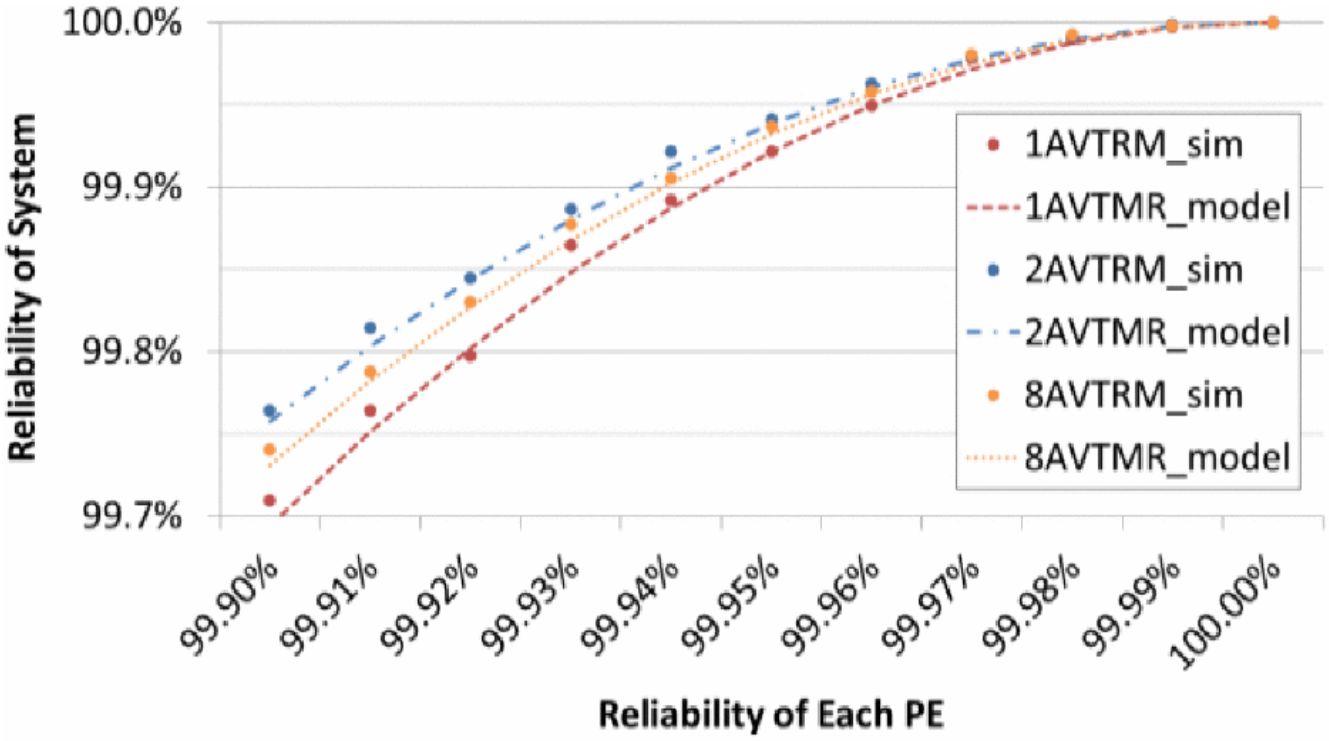
Tolerating Transient Fault in CGRA

- Validation of mathematical model - TMR

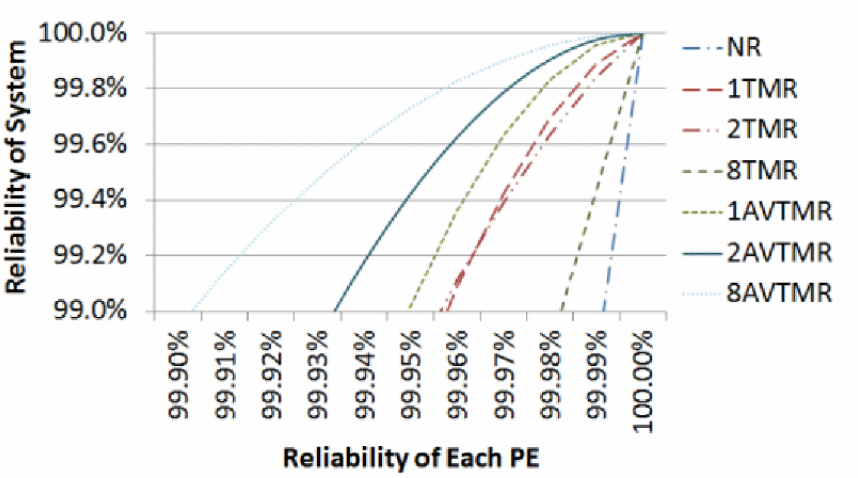


Tolerating Transient Fault in CGRA

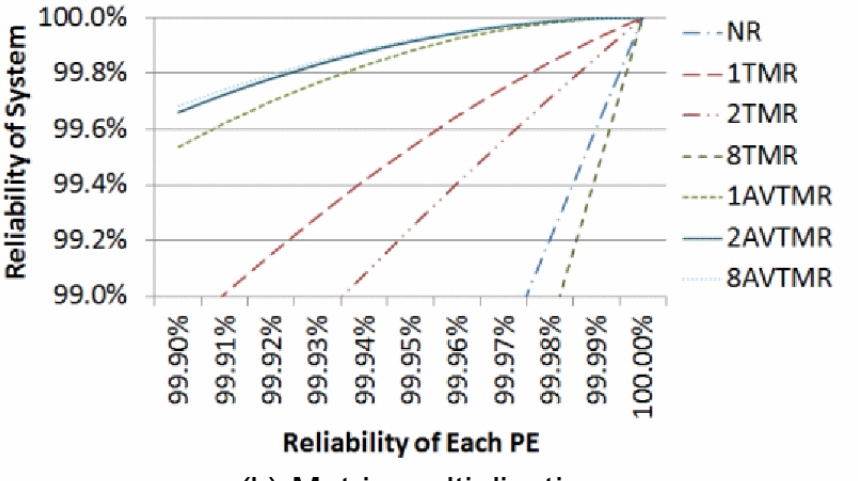
- Validation of mathematical model - AVTMR



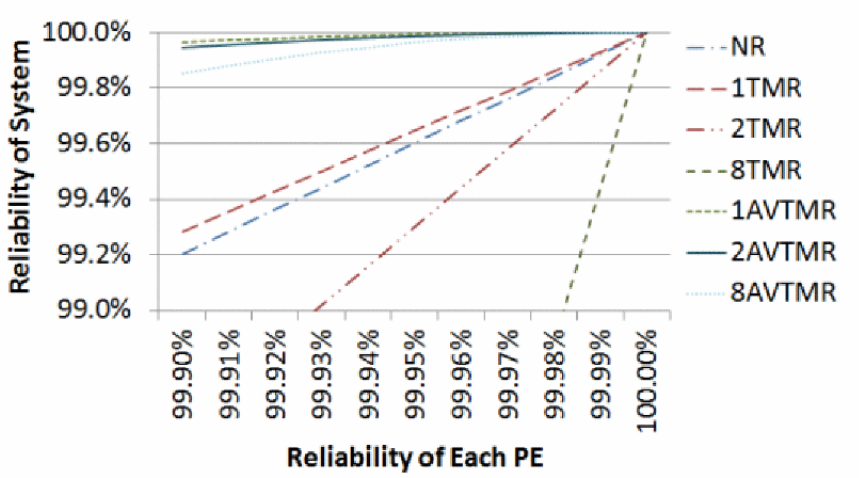
Tolerating Transient Fault in CGRA



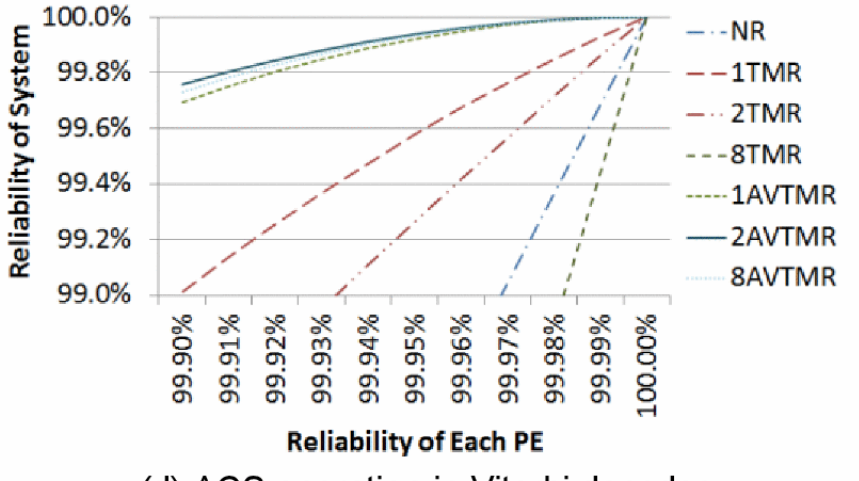
(a) Vector convolution



(b) Matrix multiplication



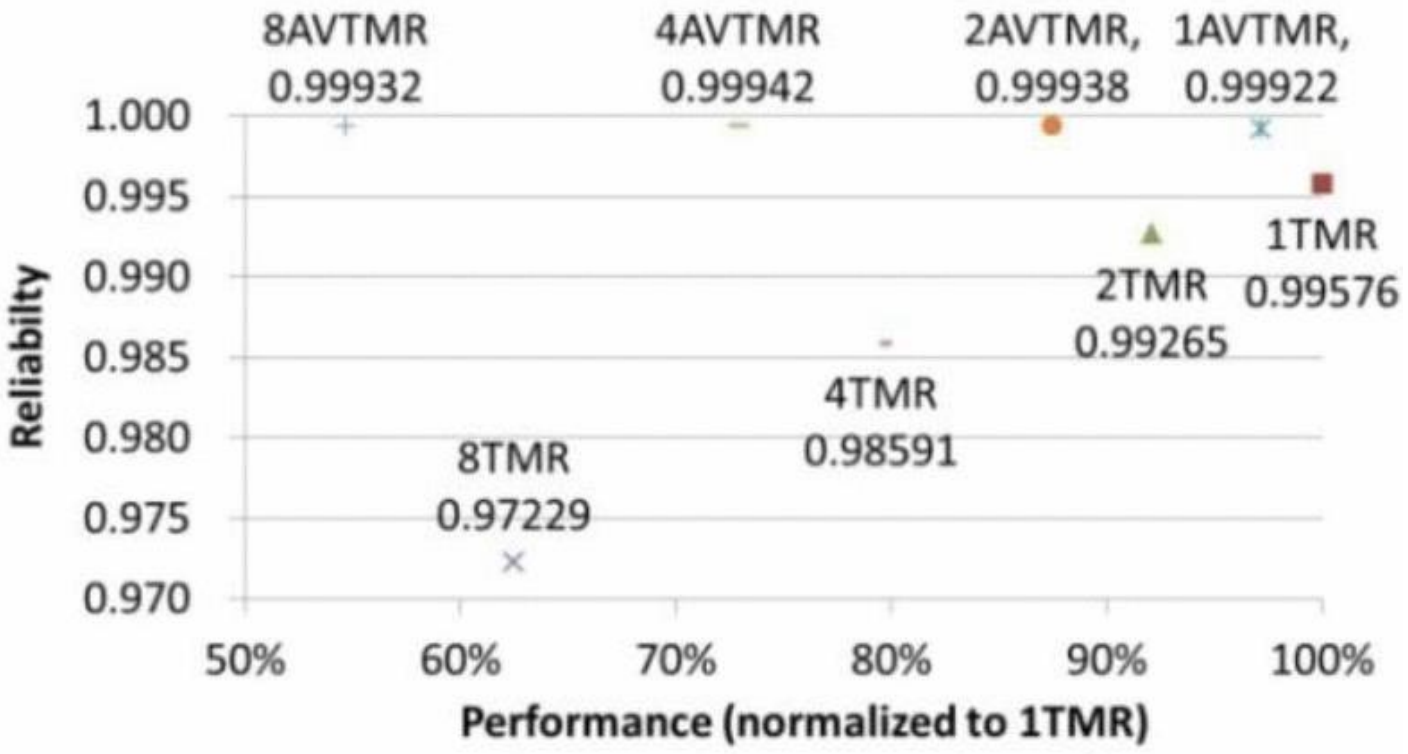
(c) Butterfly operation in FFT



(d) ACS operation in Viterbi decoder

Tolerating Transient Fault in CGRA

- Tradeoff between performance and reliability level



CGRA Optimizations

- Conclusion
  - PPA (Performance, Power, and Area) optimizations
    - Performance/area enhancement through resource sharing and pipelining
    - Power reduction through context pipelining in configuration cache
  - Application mapping
    - Fully automated fast mapping flow
  - Fault-tolerant system design using CGRA
    - Spatial redundancy of processing elements