プロセッサ設計演習

2023年6月26日

目次

- プロセッサ設計演習
  - 演習の目的
  - プロセッサ構造
  - ハザードと解決策
  - 性能評価
  - まとめ

- PIMに関する論文のディスカッション

演習の目的

- コンピュータアーキテクチャに関連する研究を 行うために、プロセッサの原理と設計プロセスを理解する必要がる

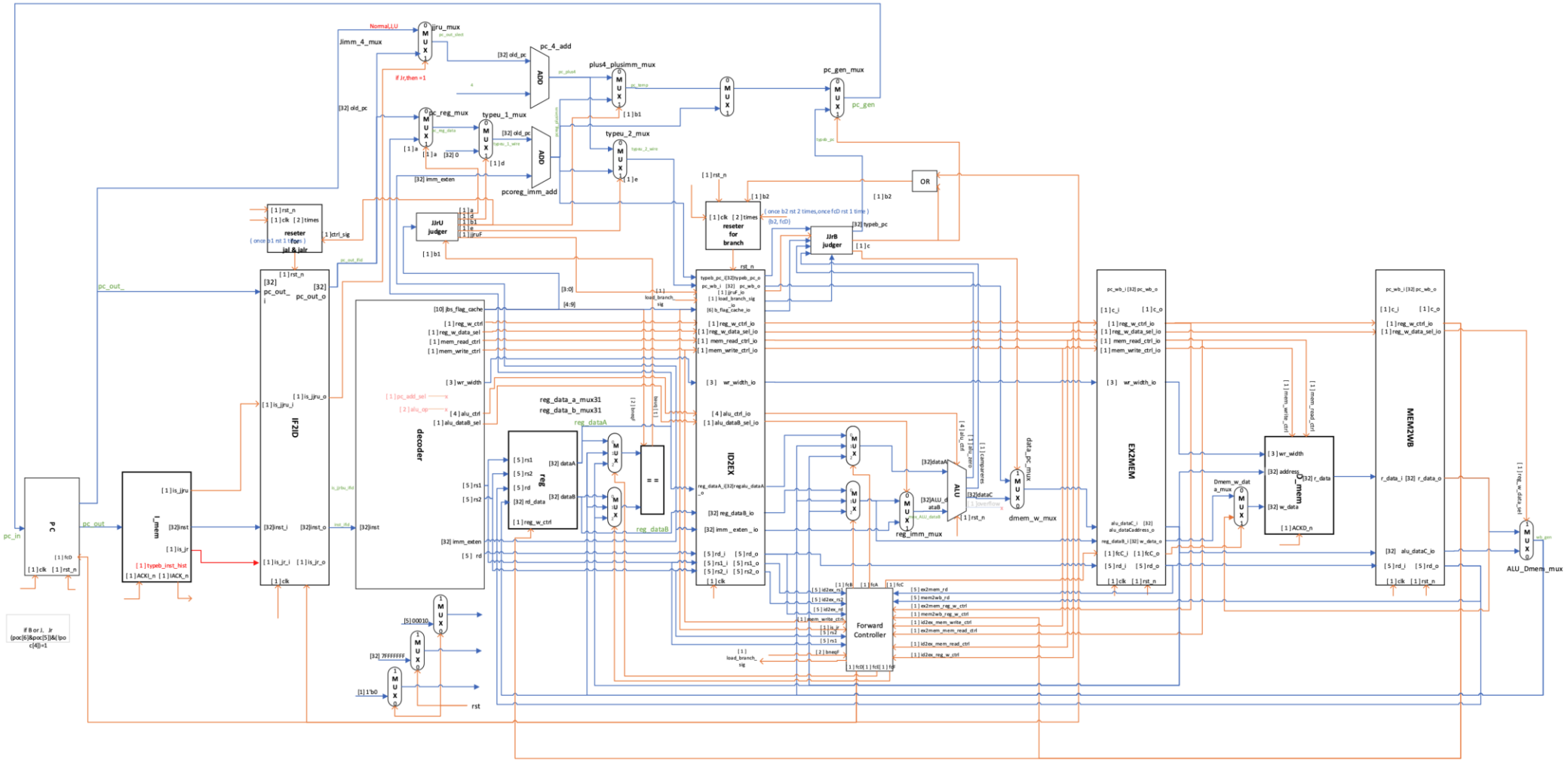
- したがって、 Verilog-HDL ハードウェア記述言語を使用して、 RISC-V プロセッサを設計した

プロセッサ構造

- 命令セット：
  RISC-V　RV32I　Base Integer Instruction
  （ecall, ebreak などが未実装）

- 構造：
  IF、ID、EX、MEM、WBを含む5段パイプライン

プロセッサ構造



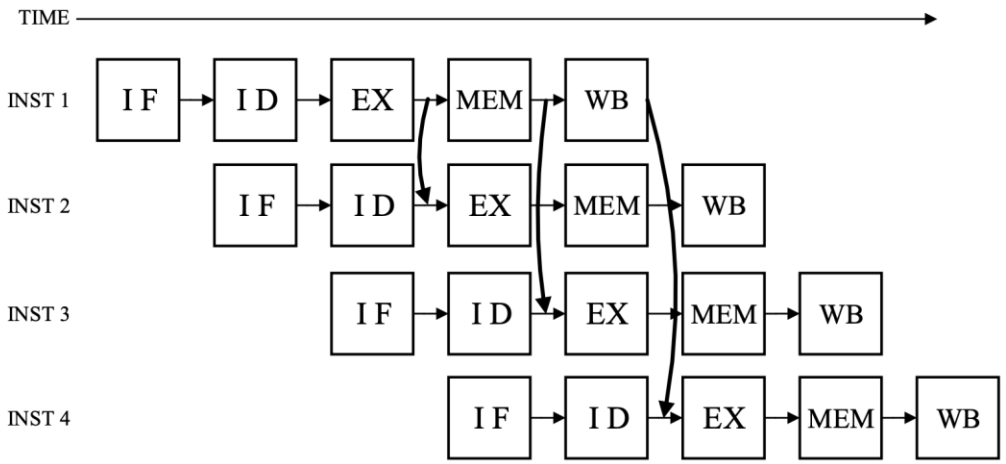
ハザードと解決策

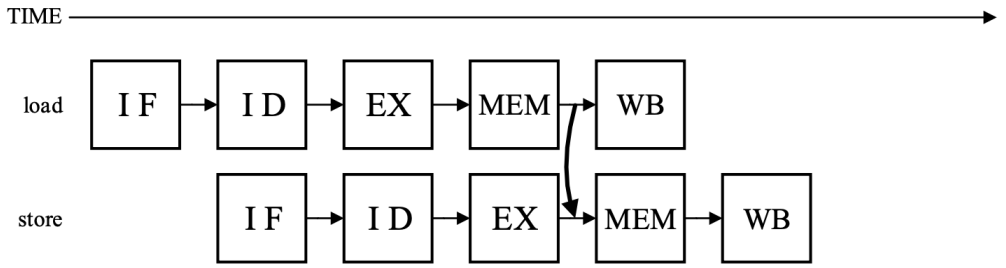
データハザード

プログラムを実行する際、ある命令が前の命令の結果に依存する場合がある。

パイプライン構造では各段階で異なる命令を実行するため、問題が起こる可能性がある。

データハザードには、データフォワーディングやデータの書き込みおよび 読み取りのタイミングを変更することで解決できる 4 つのケースがある。





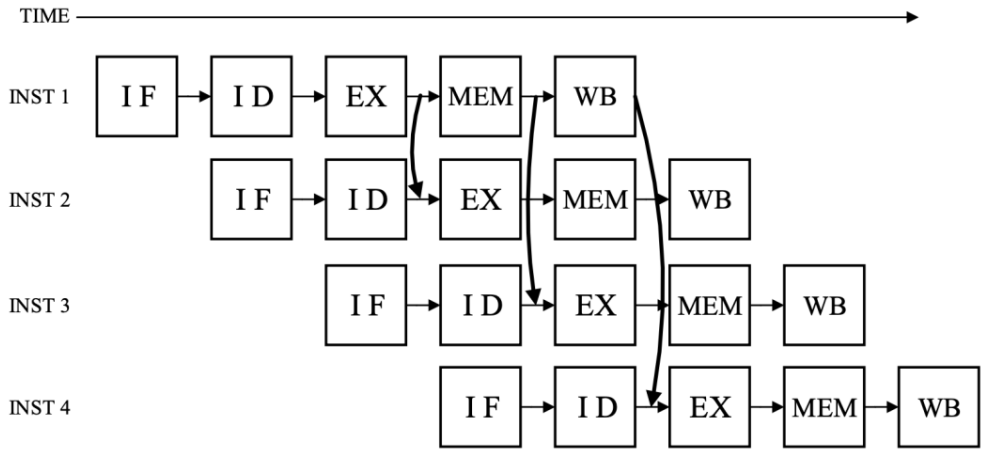
ハザードと解決策

データハザード

① 命令 2 が EX ステージで命令 1 が EX ステージで生成した実行結果を使用する場合、EX/MEM レジスタのデータをフォワードす る (INST1 ↔ INST2)

② 命令 3 が EX ステージで命令 1が MEM ステージでメモリから読み出したデータを使用する場合、MEM/WB レジスタのデータをフォワードする

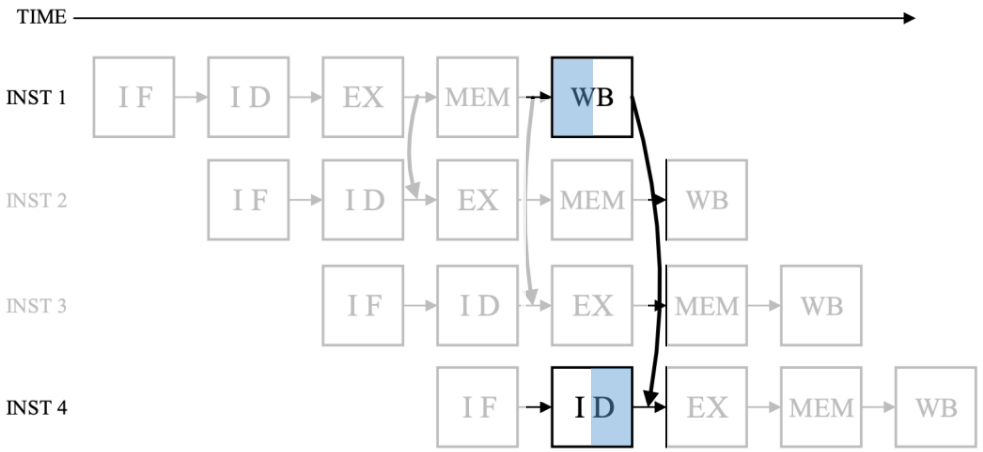
(INST1 ↔ INST3)



ハザードと解決策

データハザード

③　命令４が EX ステージで、命令１が WB ステージでレジスタに格納したデータを使用する場合



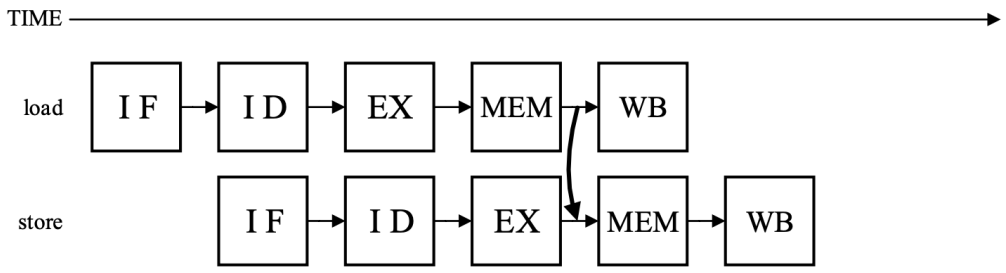
クロックサイクルの前半でレジスタにデータを格納し、クロックサイクルの後半でデータを読み出すため、データのフォワードは不要。

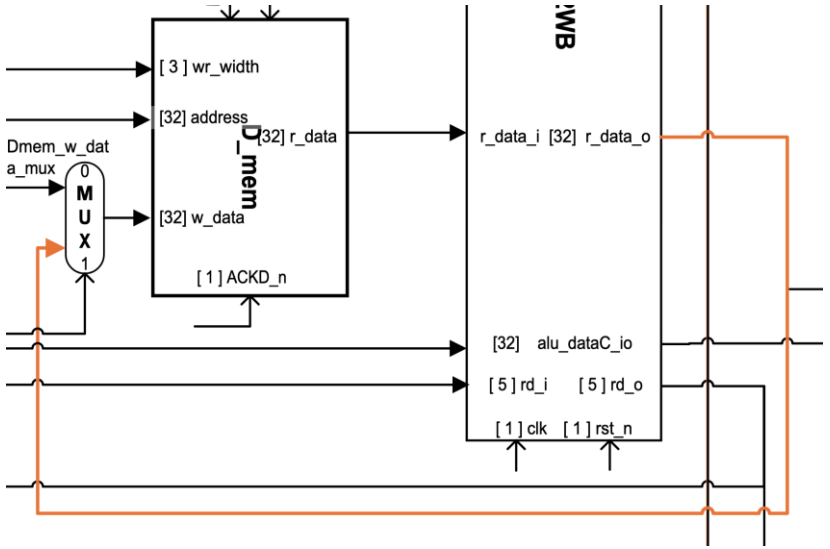
（INST1 ↔ INST4）

ハザードと解決策

データハザード

④　load- store 型のデータハザードが発生した 場合、load 命令の rd が store 命令の rs1 とは異な り、rs2 と同じであれば、ストールする必要はなく、MEM/WB レジスタのデータを MEM ステージにフォワードするだけでよい。





ハザードと解決策

制御ハザード

- 設計では分岐が発生しないと仮定されている
- B タイプの命令の判断は通常 EX ステージで実行される
- 予測失敗 が発生した場合は ID ステージと IF ステージの命令をフラッシュする必要があり、これには 2 つのサイクル が無駄になる

ハザードと解決策

制御ハザード / 一部の分岐判定の前倒し

- 分岐命令の beq と bne は、 判断プロセスが ALU を必要としなく、 beq と bne を実行する頻度も高いため、データの比較を ID ステージに前倒しになる

- この2つの 命令については、分岐予測失敗が発生した場合、1 つのサイクルの命令をフラッシュするだけでよい

	blt(u) , bge(u)	bne , beq
dijstra	≈ 0.70 %	≈ 16 %
bitcnts	≈ 0.01 %	≈ 18 %
stringsearch	≈ 0.17 %	≈ 19 %

(OPTIMIZE=3)

ハザードと解決策

制御ハザード / beqとbneの前倒しのデータハザード

- 一般的な load-use 型のデータハザード が発生した場合には、 1 サイクルの stall が必要

  （ EX ステージでデータを使う）

- ただし、 load-use(beq/bne) 型のデータハザードの場合、ID ステージでデータを取得するには 2 サイクルの stall が必要

ハザードと解決策

制御ハザード / beqとbneの前倒しのデータハザード

load-use(beq/bne) 型の場合、ID ステージでデータを取得するには 2 サイクルの stall が必要で、これは EX ステージで判断する場 合に無駄になるサイクル数と同じであるため 、bne または beq も EX ステージで判断する

time	IF	ID	EX	MEM	WB
1	inst_1	**beq**	**load**		
2	inst_1	**beq**	~~beq~~	**load**	
3	inst_1	**beq**	~~beq~~	~~beq~~	**load**
4	$\boldsymbol{inst_{tgt}}$	~~inst_1~~	**beq**	~~beq~~	~~beq~~

time	IF	ID	EX	MEM	WB
1	inst_1	**beq**	**load**		
2	inst_1	**beq**	~~beq~~	**load**	
3	inst_2	inst_1	**beq**	~~beq~~	**load**
4	$\boldsymbol{inst_{tgt}}$	~~inst_2~~	~~inst_1~~	**beq**	~~beq~~

性能評価

MiBenchフォルダ内のbitcnts、dijkstra、stringsearchの各プログラムで評価を行い、異なるサイズのプログラムを実行した結果と論理合成の結果を下表に示す。(OPTIMIZE=3)

	test	small	large
bitcnts	53158	38128163	570943108
dijkstra	4084141	37445372	189157568
stringsearch	10713	92120	2112971

最大遅延時間 / ns	消費電力/mW	面積 / μm^2
4.56	13.7230	277570

まとめ

できたこと

+5段パイプライン構造のプロセッサを設計し，正しく動けるようになった

＋プロセッサの構造や原理についての理解を深めた

＋ Verilogの書き方を復習し，深めた

まとめ

反省点

－最初に論理合成できないコードがたくさん書いてしまい、
　論理合成できるよになるため、コードを修正し、多くの時間を使った

－コードがi-VerilogとNC-Verilogで実行した結果は異なり、
　NC-Verilogの環境に動けるようのため、多くの時間を使った
　コードが異なる環境で実行結果が異なる原因が今までも不明

－分岐予測、例外処理などの機能が実現しなかった
（branch命令を実行する際、dijkstra には load-bne のハザードが多い、前倒ししたbeq/bne
　の判定の役割を大幅に弱めた。性能を上げるため、分岐予測が必要）

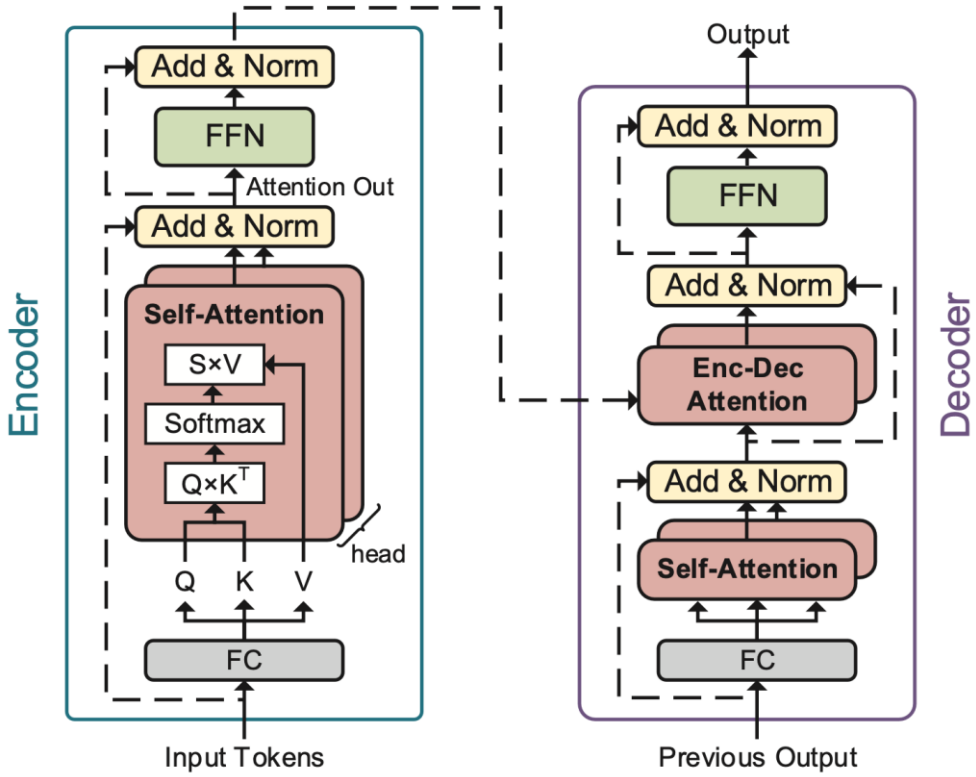
－設計では改善できるところがまだある
（機能が重複した構造、blt/bge判定の前倒し。。。）

TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer

背景

- Transformerモデルは、その並列性、低データ移動コストなどの特徴のため、大量のメモリ空間が必要

- 多くのメモリベースのニューラルネットワークアクセラレータが存在していますが、これらは Computationally intensiveのCNNsに最適化されており、Data IntensiveのTransformersに適用できないかもしれない



動機

- 過去研究によると、ソフトウェアレベルのスケジューリング（データフロー）およびハードウェアの設計は、ニューラルネットワークの加速において重要な役割を果たす

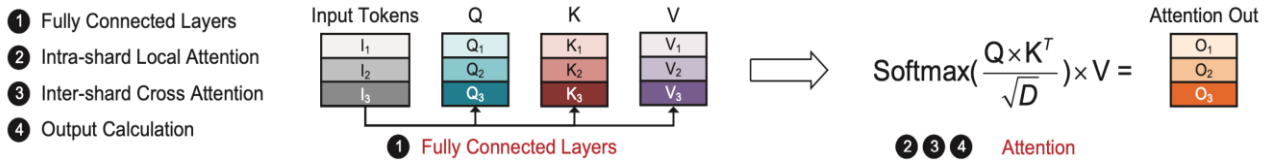
- Transformerは、中間変数を保存するための大量のスペースを必要し、また非常に高い計算の並列性が必要

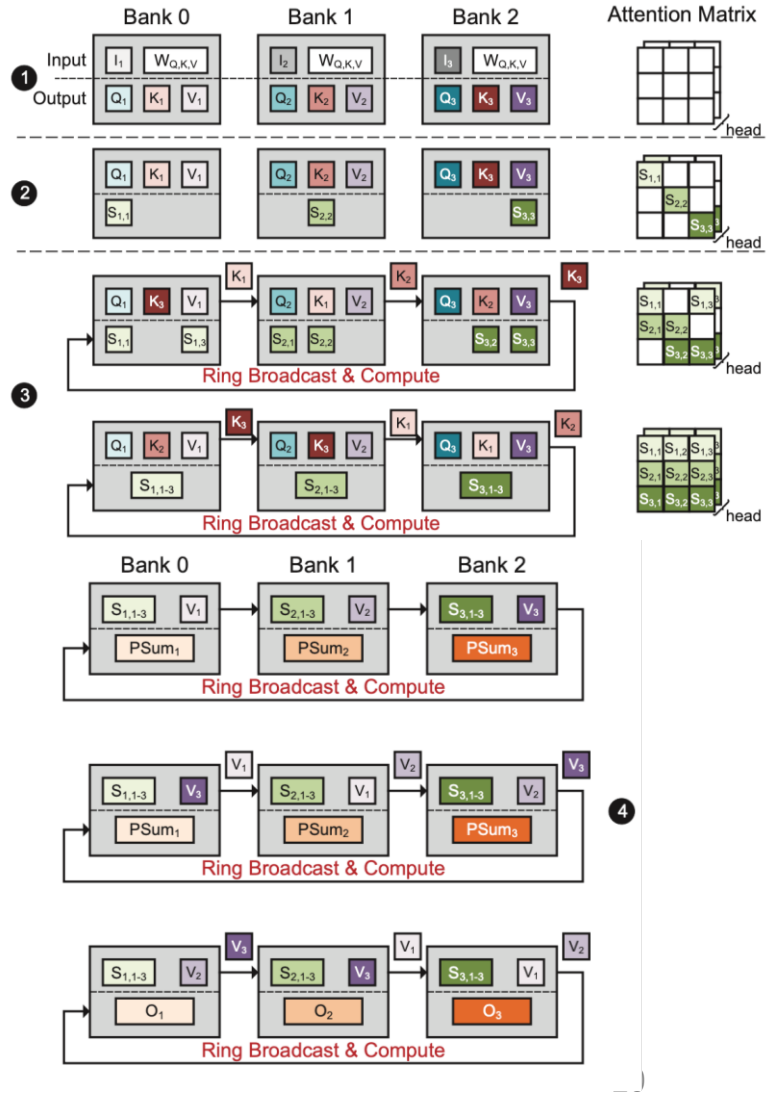
- ベクトルの長さがより長いため、Transformersモデルのデータ移動のオーバーヘッドは、CNNに比べて大きくなる

TransPIM構造

Attentionの計算プロセス

- Input 行列の各行 I_1 … I_N を各Bankに割り当てる
  W_K ， W_Q ， W_V を各Bankにコピーする
- 各BankはK，Q，V行列の部分行列 K_i , Q_i ,V_i を計算する
- S_{ii} を計算する （$S_{ii} = Q_i \times K_i^T$）





TransPIM構造

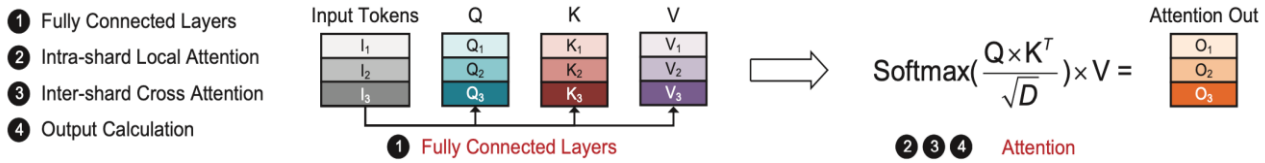
Attentionの計算プロセス

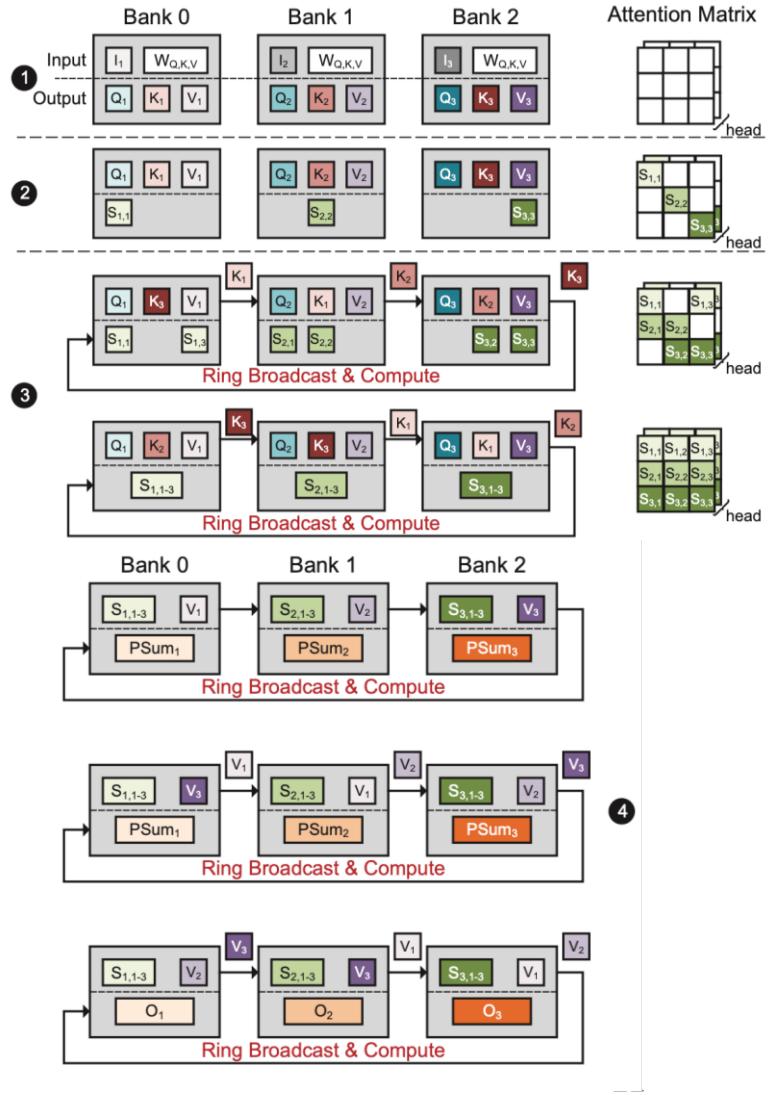
- 各BankがRing Broadcast Unitを通じて、自分の部分行列 K_i を交換し、他のBankから交換した K_j を用いて、 S_{ij} を計算する

（行列 S の計算が完了するまでN回繰り返す）

N：Memory Bank の数

- 行列 S の計算プロセスと似ている、出力行列 O を計算できる

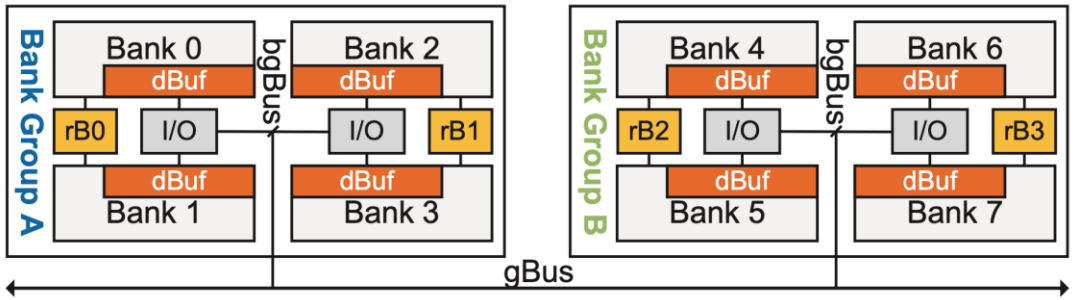




TransPIM構造

データの流れ

- Bank間のデータ転送とBank内の計算の効率を向上させるために、Ring Broadcast Unitを設置した
- 入力行列が8つに分割され、8つのBankに割り当てられると仮定する
- 伝統的なHBM アーキテクチャはバスを通じてRing Broadcastプロセスを実行するため、8つのサイクルが必要 (1→2→3 ... 7→0)
- Ring Broadcast Unitを使用すると、隣接するBank間のデータ移動が速くなれる。最終的にRing Broadcastプロセスに必要な時間を3サイクルに短縮できる。



(a) Memory Hierarchy in TransPIM

Time Step	1	2	3
Bank Group Bus A	3→4	7→0	1→2
Bank Group Bus B	3→4	7→0	5→6
Ring Broadcast Buffer	0→1, 6→7	2→3, 4→5	*idle*

(b) Timeline for Ring-based Broadcast

結果

- Transformerモデルの加速のために、 Software-Hardware Co-Designの設計を通じて、全体のデータの再利用を最大限に活用した
- 結果として、TransPIM アーキテクチャはメモリベースのアクセラレータより4.6倍速くなた
- TransPIMは、さまざまなTransformerモデルにおいて、GPUよりも22.1倍から114.9倍速くなた

future work

- PIMに関する知識や技術を調査し、もっと論文を読む
- オプティマイズできる方向を探し、PIMのアクセラレータを設計する

reference

H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, “Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, “Timeloop: A systematic approach to dnn accelerator evaluation,” in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, “Neural cache: Bit- serial in-cache acceleration of deep neural networks,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 383–396.

M. Imani, S. Gupta, Y. Kim, and T. Rosing, “Floatpim: In- memory acceleration of deep neural network training with high precision,” in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 802–815.