Operation and Data Mapping for CGRAs with Multibank Memory

Yongjoo Kim, Jongeun Lee, Aviral Shrivastava, Yunheung Park

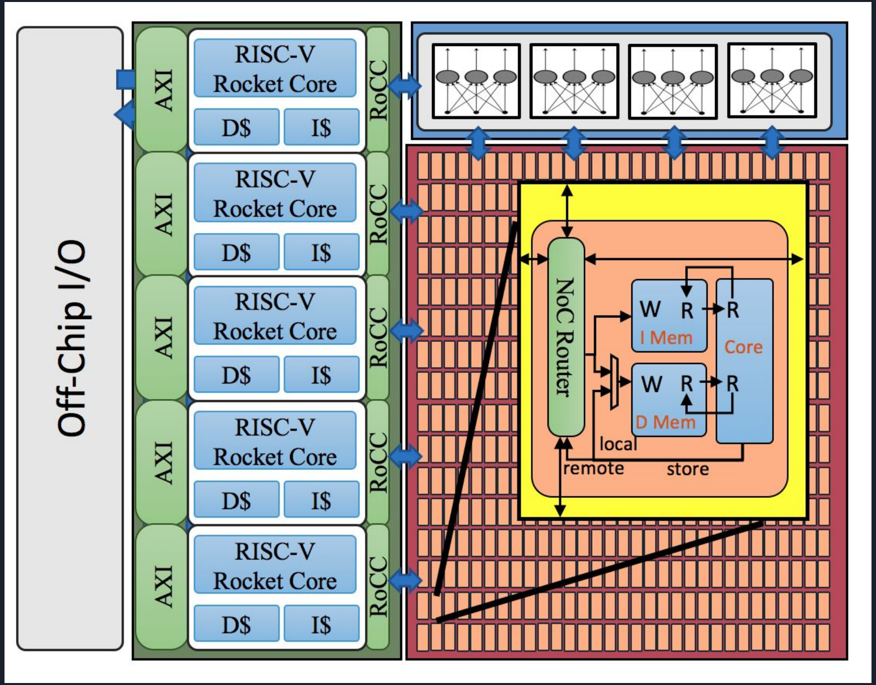
Presented by James Connolly, Jielun Tan, Pranav Srinivasan

Agenda

- What is CGRA/What can it do?
- CGRA Memory System
- Overview of Memory Aware Scheduling (MAS)
- Performance Analysis + Discussion of Paper

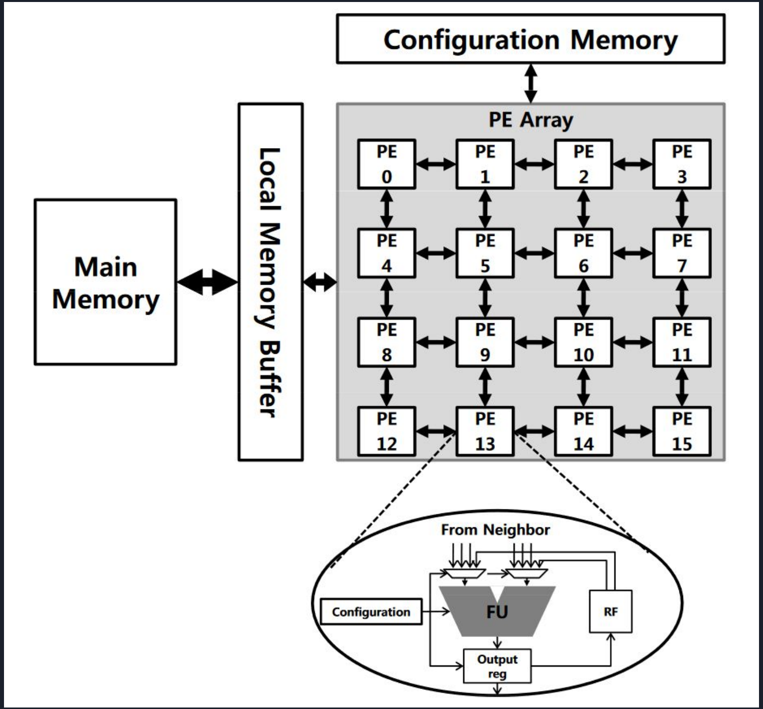
Dataflow Architectures

- Semantically: no PC
    - Data is processed as it is streamed in
- Not useful for generalized compute
- Powerful near the end of Moore's Law
    - Data driven applications require data driven architectures
    - TPU, Brainwave



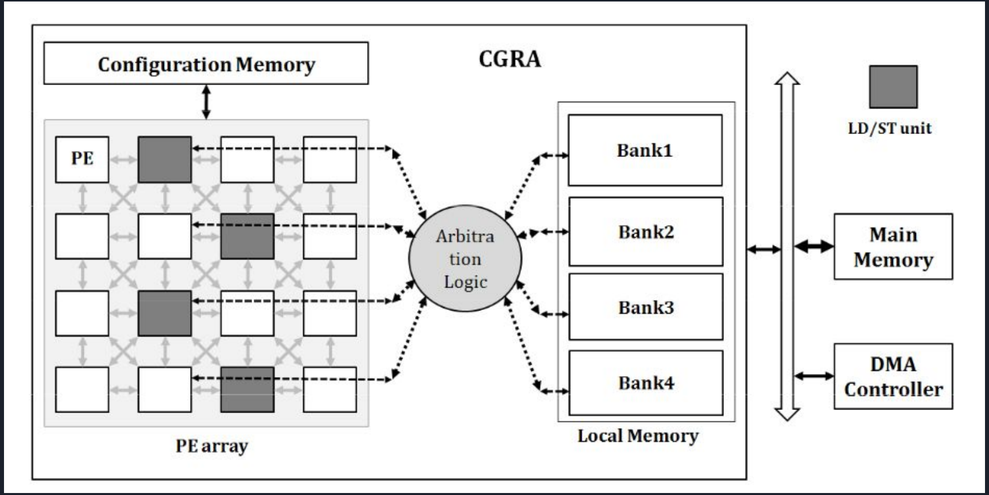
Coarse **G**rain **R**econfigurable **A**rchitecture

- A grid of **Processing Elements** (PEs)
  - Functional Unit (ALU, Multiplier, Load/Store Unit)
  - A small local Register File
  - Ability to send info to neighboring PEs
- Share Access to a single Local Memory
- Configurable (like FPGA) via Config Memory
- Flexibility of FPGA, Speed of an ASIC (Ideally)



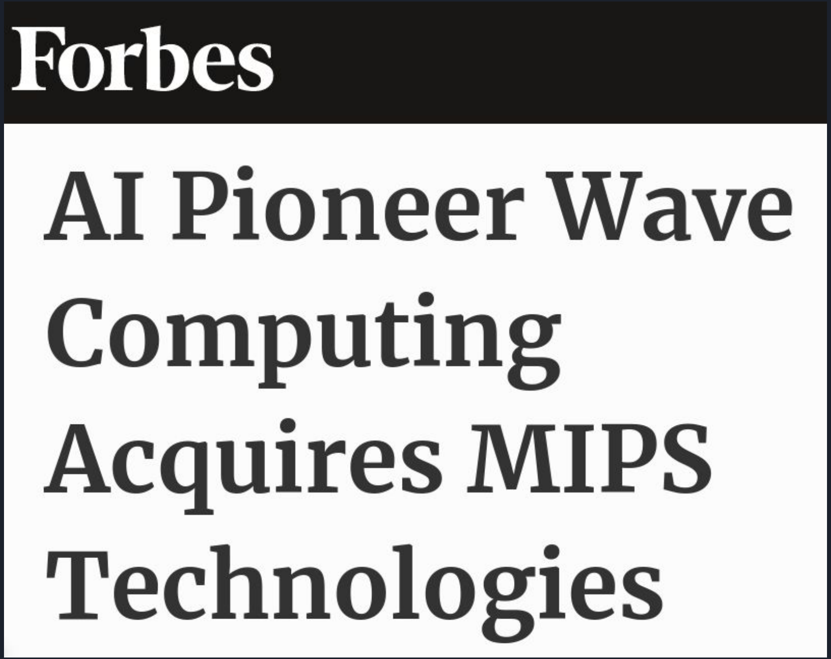
Local Memory (Scratchpad)

- Low Latency RAM that is typically populated with the arrays necessary to run loop
- Load/Store PEs access Local Memory for info
- Usual solution for multiple PEs - banking
  - MBA (multi-bank with arbitration): Hardware logic that allows any PE to access any bank
- Banking gives rise to *Bank Conflicts*
  - Two PEs can't access same bank on the same cycle



Why CGRA

- Excellent Balance of Performance vs Power vs Flexibility
  - Similar tradeoffs of VLIW to OoO
- Off-loading parallelizable loops to a CGRA component is a common use case
- Lines blurring between CGRA and many-core systems
  - Silicon is cheap, PEs are cores


Forbes

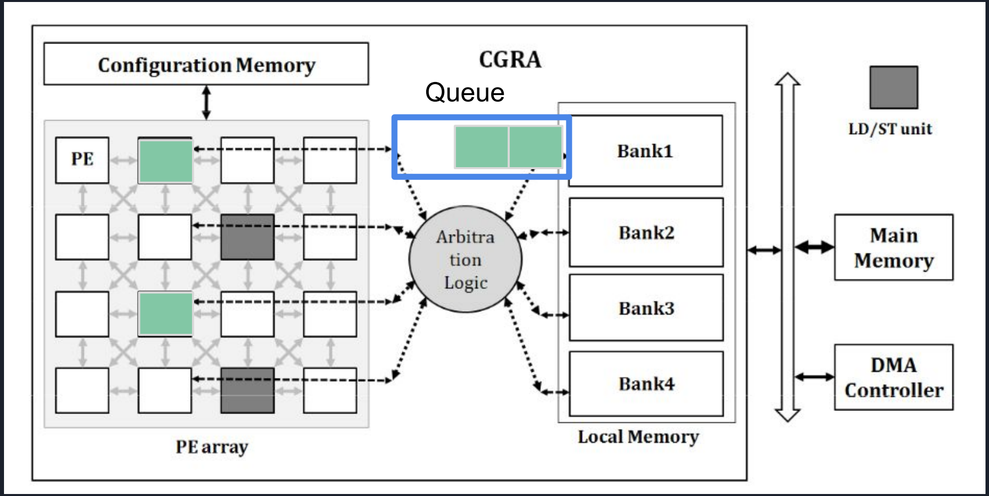
AI Pioneer Wave Computing Acquires MIPS Technologies

CGRA Scheduling: Previous Work

- Hardware is relatively simple -- onus is on the compiler
  - Analogous to VLIW
- Loop Level Parallelism is exploited
- Modulo scheduling is a clear choice
  - Added constraint: routing between PEs
  - How do we do this?
  - Two trains of thought
    - Node centric
    - Edge centric
  - More on this later...

Traditional MBA CGRA scheduling

- Memory Unaware Scheduling (MUS)
  - Deal with bank conflicts in hardware
- Regular MBA
  - In case of conflict, stall a PE
- Dynamic Multi Queue (DMQ)
  - Have a queue system of requests per bank
  - No stalls, but increase load/store latency
- Sequential vs Interleaving:
  - Should we leave an array contiguous in one bank?
  - Spread it so the following access is in a different bank?



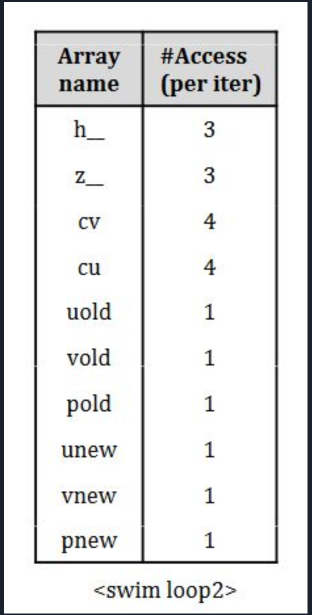
Memory Aware Scheduling (MAS)

- Compiler scheduling technique, used in conjunction with modulo scheduling to issue loads and stores to PEs to avoid bank conflicts
- High level idea: cluster arrays with distinct access patterns into groups
- Put groups in same bank to eliminate conflict

- Biggest Problem: both instruction scheduling and memory instruction scheduling are Hard problems
- Need to be done together so as to avoid conflicting schedules

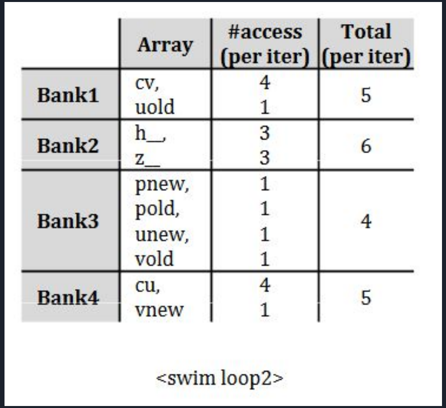
Memory Aware Scheduling

- Step 1: Array Clustering into banks
  - Compute priority for each array accessed in loop, based on several factors
  - (Size of Array / Size of Bank) + Sum over all loops ((Num accesses in loop) / II of Loop)
    - Intuition: Bigger Arrays have higher Priority, so do arrays that are accessed more
  - With this information, we cluster arrays based on cost of assigning to a bank
  - Gives rise to MemMII - related to number of accesses to a bank in one cycle
  - Combined with RecMII and ResMII to find MII for scheduling



Array name	#Access (per iter)
h__	3
z__	3
cv	4
cu	4
uold	1
vold	1
pold	1
unew	1
vnew	1
pnew	1

<swim loop2>

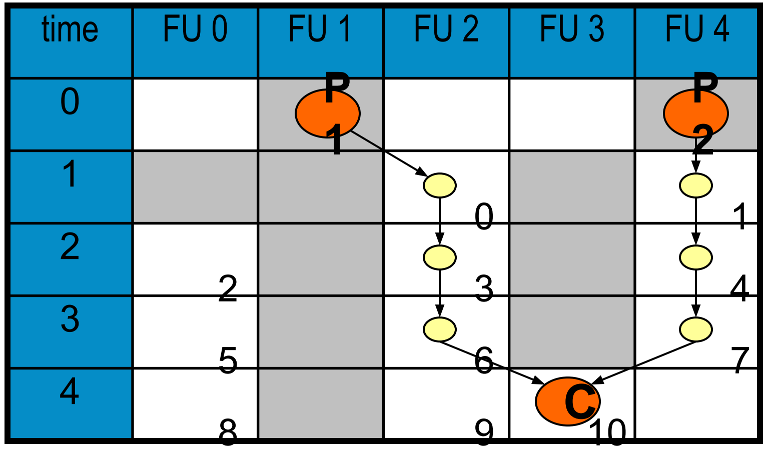


	Array	#access (per iter)	Total (per iter)
Bank1	cv, uold	4 1	5
Bank2	h__, z__	3 3	6
Bank3	pnew, pold, unew, vold	1 1 1 1	4
Bank4	cu, vnew	4 1	5

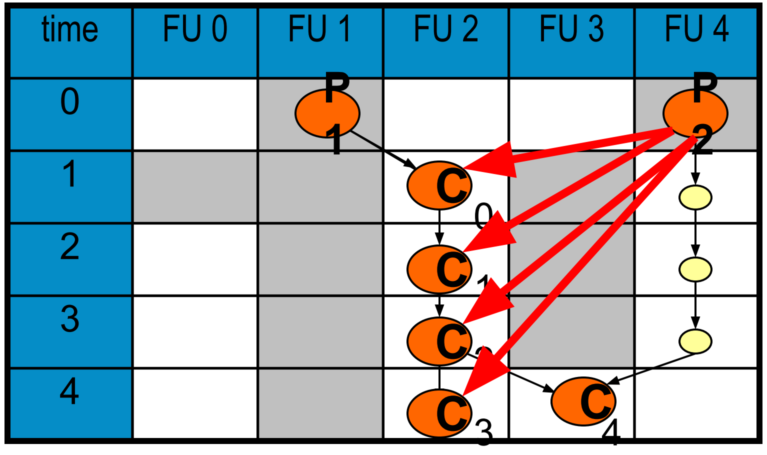
<swim loop2>

Flashback to the past

Scott's Approach : Edge-centric



Node-centric

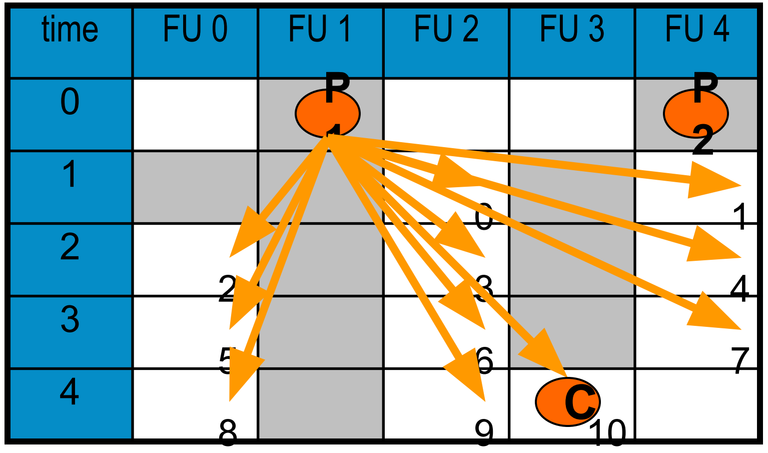


Edge-centric

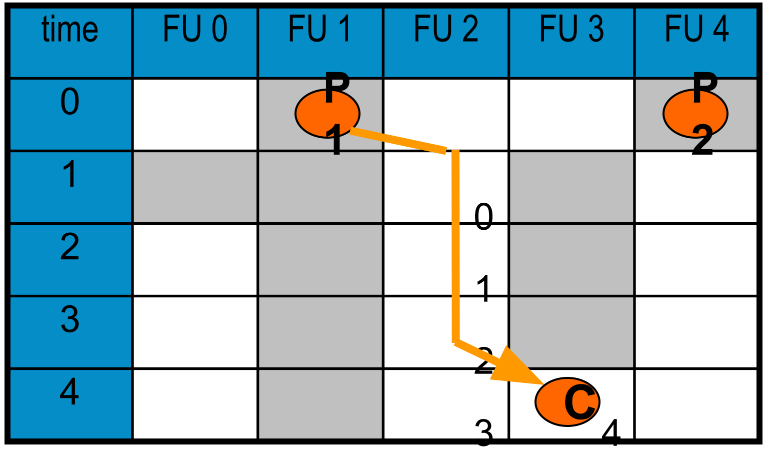
Start routing without placing the operation
Placement occurs during routing

Credit: [2]

Benefit 1 : Less Routing Calls



Node-centric



Edge-centric

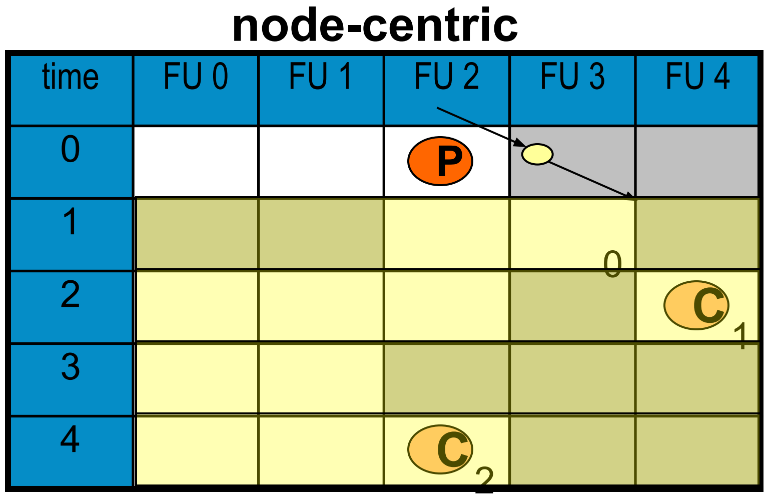
11 routing calls for P1 ➔ C

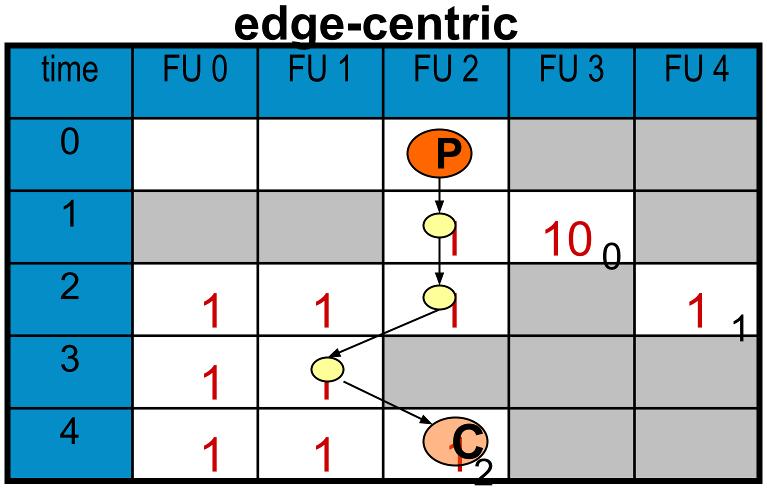
1 routing call for P1 ➔ C

Reduce compile time with less number of routing calls

Credit: [2]

Benefit 2 : Global View





- Assume slot 0 is a precious resource (better to save it for later use)
- Node-centric greedily picks slot 1
- Edge-centric can avoid slot 0 by simply assigning a high cost

Credit: [2]

Edge-centric Modulo Scheduling

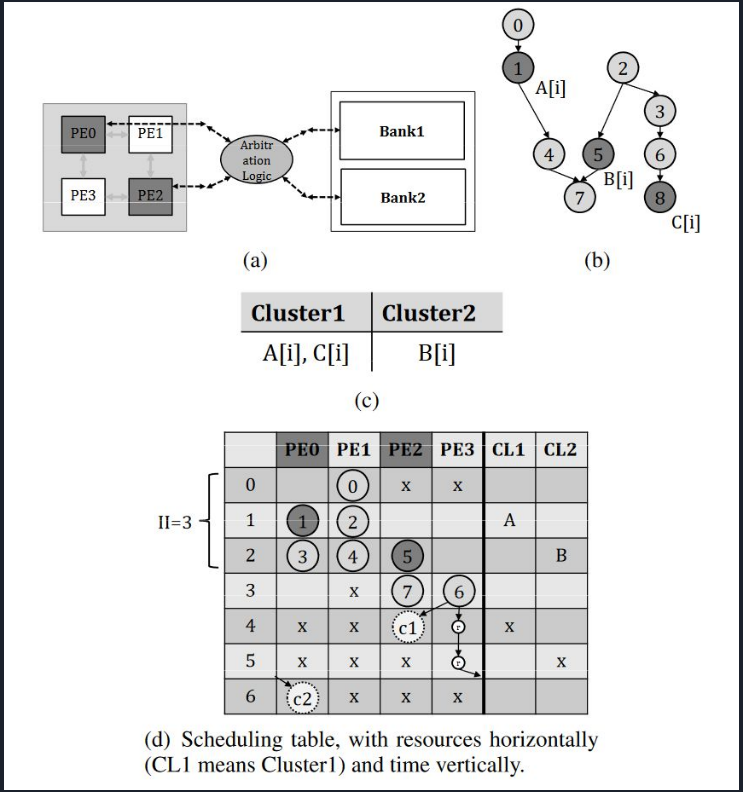
- It’s all about edges
  - Scheduling is constructed by routing ‘edges’
  - Placement is integrated into routing process
- Global perspective for EMS
  - Scheduling order of edges
    - Prioritize edges to determine scheduling order
  - Routing optimization
    - Develop contention model for routing resources

Credit: [2]

Memory Aware Scheduling

Step 2: Edge-Centric Modulo Scheduling with *memory bank awareness*

- Treat each PE *and bank* as a separate resource
- Factor in latency of sending information across PEs for dependent instructions in routing (done by EMS)



Cluster1	Cluster2
A[i], C[i]	B[i]

(c)

	PE0	PE1	PE2	PE3	CL1	CL2
0		0	x	x		
1	1	2			A	
2	3	4	5			B
3		x	7	6		
4	x	x	c1	r	x	
5	x	x	x	r		x
6	c2	x	x	x		

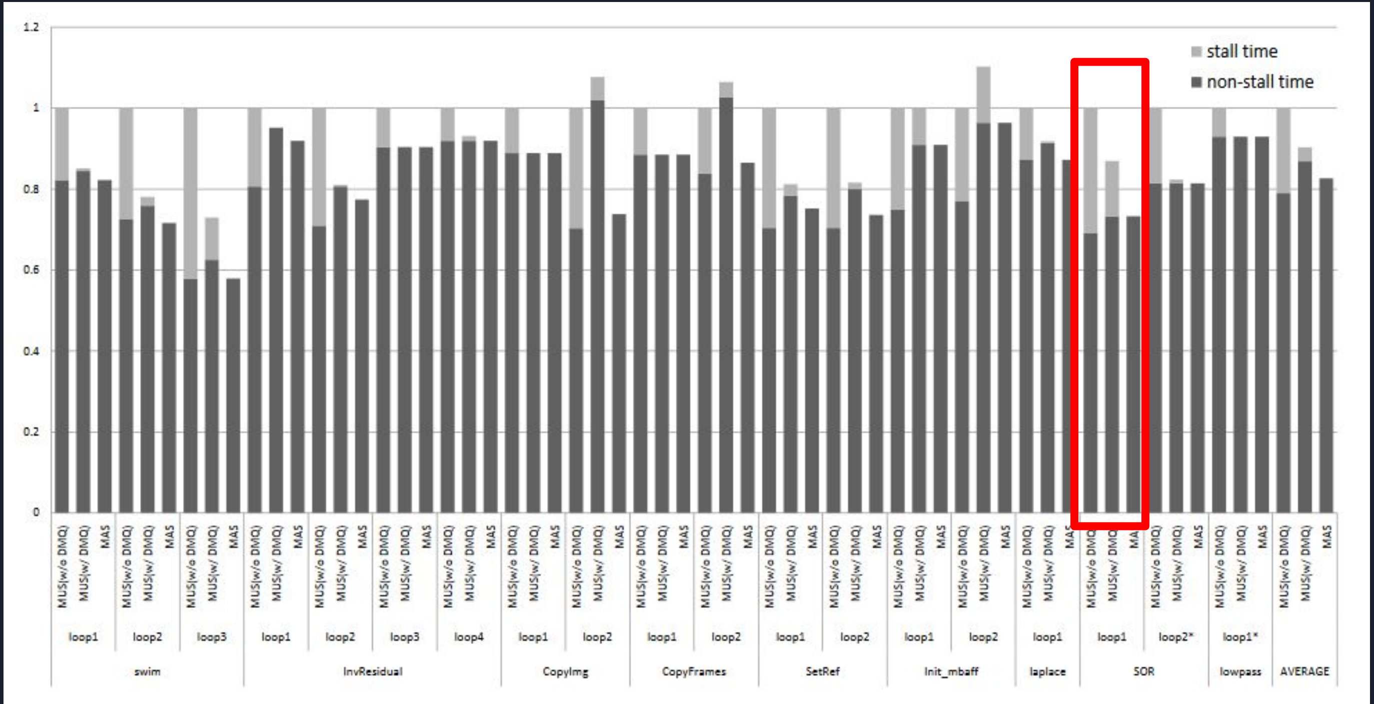
(d) Scheduling table, with resources horizontally (CL1 means Cluster1) and time vertically.

Performance Analysis

- Three Approaches to compare
  - MUS with stalls
  - MUS with queues
  - MAS
- Benchmarks: Multimedia Programs

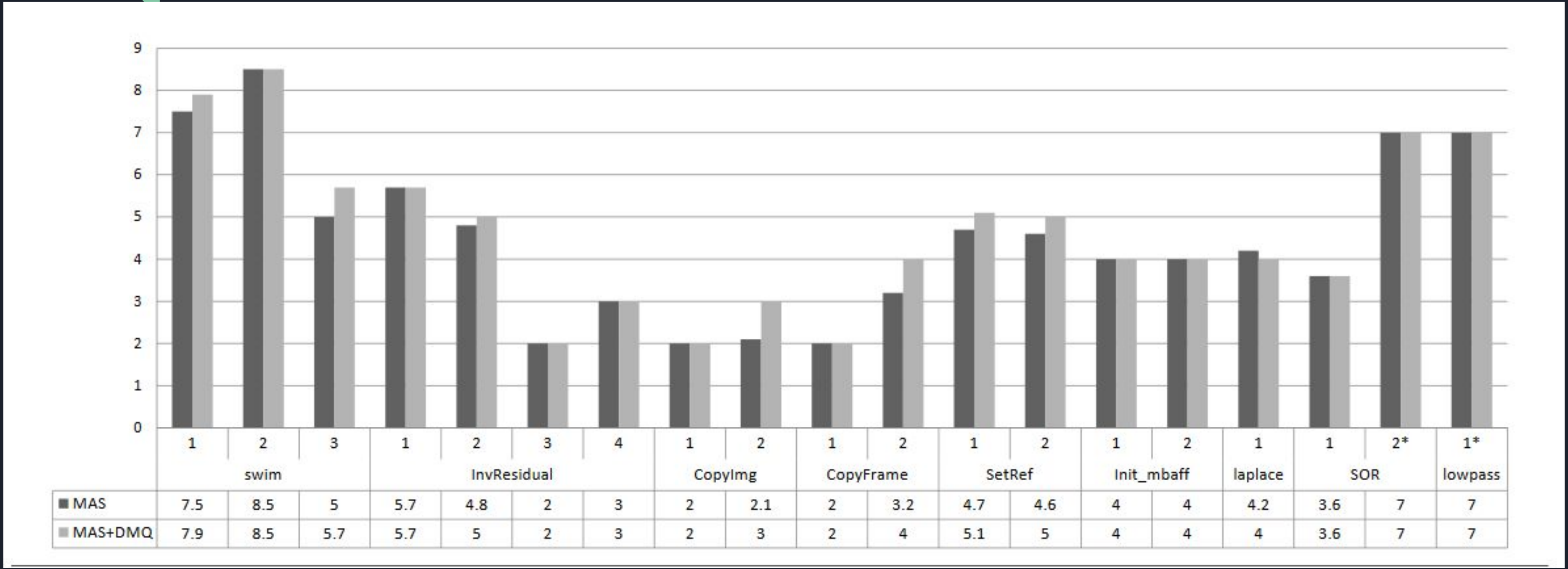
17.3% improvement on average over MUS

8.5% improvement over MUS + DMQ



MAS + Queues

Can adding the Queue to prevent stalling help in a Memory Aware Schedule?



	swim			InvResidual				CopyImg		CopyFrame		SetRef		Init_mbaff		laplace	SOR		lowpass
	1	2	3	1	2	3	4	1	2	1	2	1	2	1	2	1	1	2*	1*
■ MAS	7.5	8.5	5	5.7	4.8	2	3	2	2.1	2	3.2	4.7	4.6	4	4	4.2	3.6	7	7
▨ MAS+DMQ	7.9	8.5	5.7	5.7	5	2	3	2	3	2	4	5.1	5	4	4	4	3.6	7	7

Intuition: Queue effectively increases latency of loads/stores - doesn’t help if conflicts are rare

Strengths & Weaknesses

- Strengths:
  - Novel idea to reduce memory access overhead in CGRA mappings by being aware of access conflicts introduced by the Architecture of CGRA
  - Effective extension to pre-existing work in CGRA modulo scheduling
- Weaknesses:
  - Strong assumptions for scheduler to simplify problem
    - Assuming unlimited local memory
    - Assuming loop count is provided before mapping occurs during runtime
    - Array clustering is based on a greedy heuristic
- Y’all should read Scott’s paper though, for real

Food for thought

- Can the scheduler provide both an optimal performance vs optimal resource usage solution?
    - I.e. accomplish the same amount of work while using a subsection of the PEs
- How often does the optimal performance solution lead to optimal usage?
- Can this scheduler be replaced with better banking mechanics in hardware?

Thanks!

Any Questions?

References

[1] Yongjoo Kim, Jongeun Lee, Aviral Shrivastava, Yunheung Park. Operation and Data Mapping for CGRAs with Multibank Memory

[2] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-s. Kim. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. [