PULP PLATFORM

Open Source Hardware, the way it should be!

Efficient Systolic Execution on a Shared-Memory Manycore System

Samuel Riedel (sriedel@iis.ee.ethz.ch)

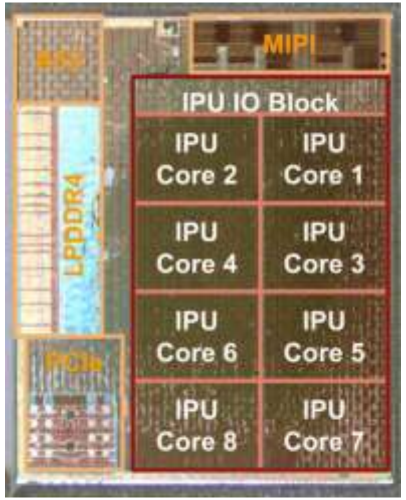
Matheus Cavalcante (matheus@iis.ee.ethz.ch)

Sergio Mazzola (smazzola@iis.ee.ethz.ch)

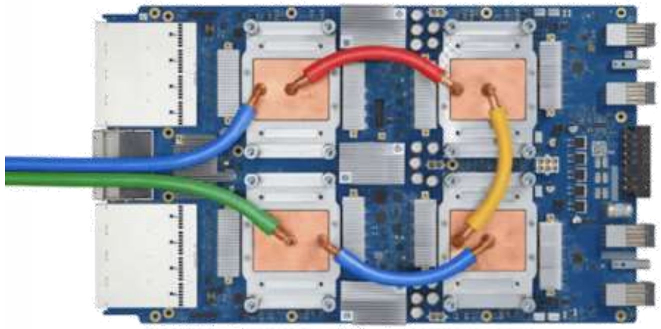
Luca Benini (lbenini@iis.ee.ethz.ch)

Systolic Architectures

- Network of tightly coupled processing untis
- Widely used for dedicated accelerators
  - Google's TPU and PVC
- \+ Highly efficient specific workloads
  - Machine learning & Image processing
- \- Very rigid execution scheme
  - Not all algorithms map nicely to the same topology



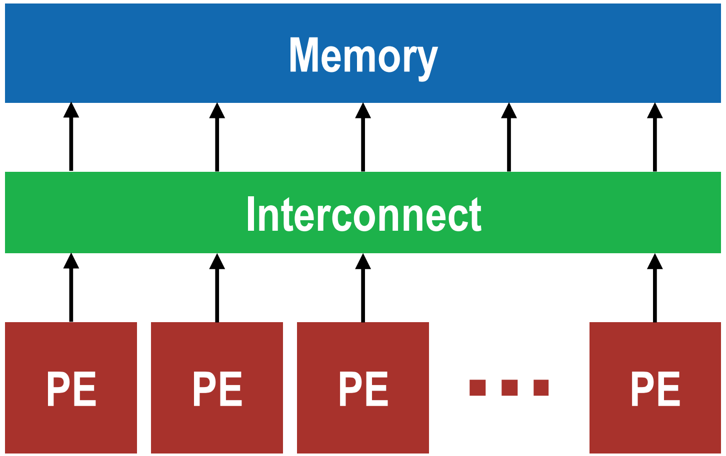
Source: https://blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/



Source: https://cloud.google.com/tpu

Shared-memory Manycore Systems

+ **General-purpose processing**
  - Very flexible execution scheme
  - Easy to program
+ **Widely used in CPUs, GPUs, accelerators**
- **Trade off throughput**
  - Communication overhead



Combine the Best of Both Worlds

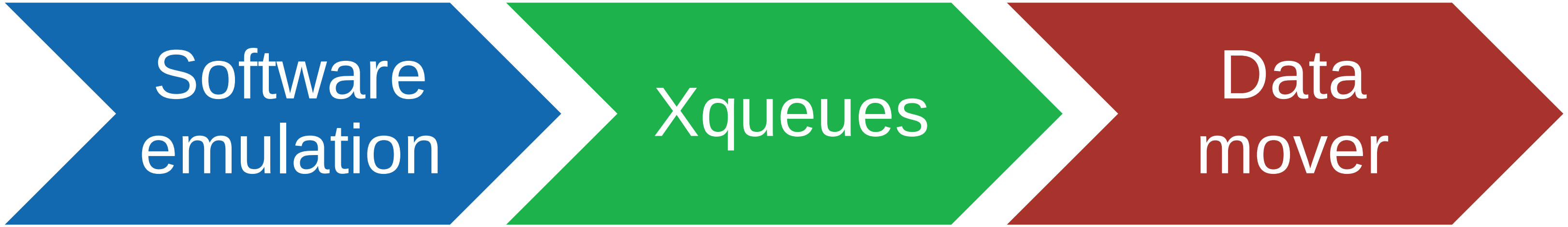
Systolic Array | Hybrid Architecture | Shared-memory System

High performance | High flexibility

- **Efficient systolic execution on a shared-memory system**
  - Extend a shared-memory manycore system with a systolic operation mode
  - Get performance of a systolic array for suitable workloads
  - Keep the flexibility of a shared-memory system

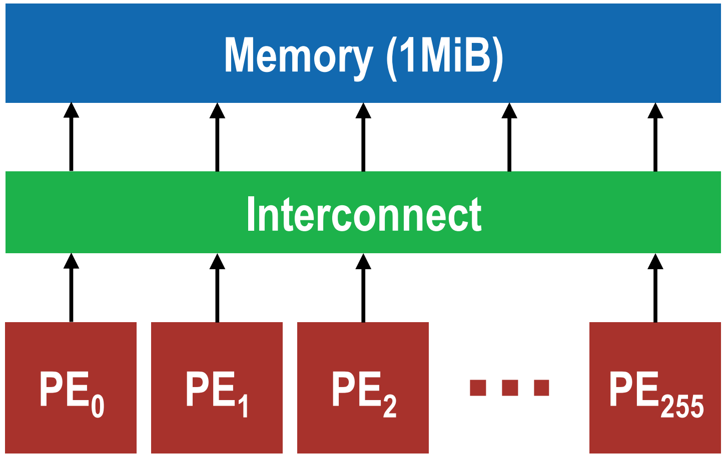
Our Approach

- **Emulate systolic behavior through software**
  - Allows exploring systolic topologies
- **Explore hybrid programming model**
  - Merge systolic and classical programming to boost performance
- **Add lightweight hardware extensions**
  - Reduce communication overhead through a custom ISA extension
  - Completely hide communication with an autonomous data mover

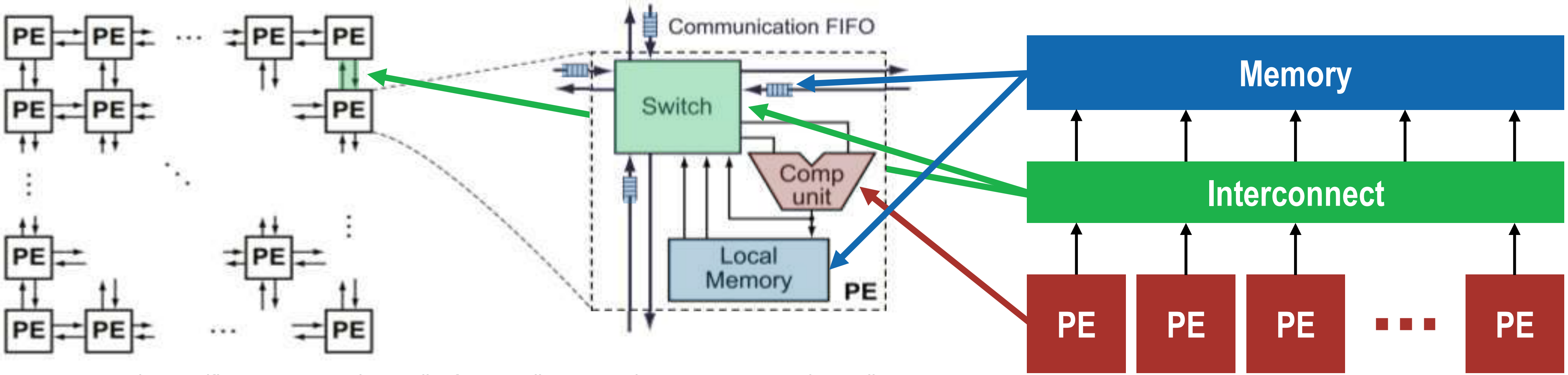


MemPool

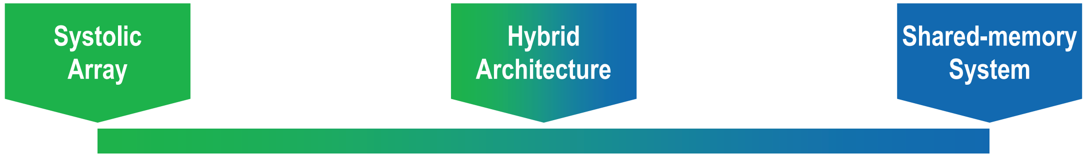
- **Scaled-up shared-L1 manycore system**
  - 256 32-bit RISC-V cores
  - 1 MiB of shared L1 data memory in 1024 banks
  - ≤ 5 cycles latency (without contention)
- **Full flexibility**
  - Individually programmable cores
- **Open source**
  - https://github.com/pulp-platform/mempool



Emulate Systolic execution



Source: *Domain-Specific Language and Compiler for Stencil Computation on FPGA-Based Systolic Computational-Memory Array - https://link.springer.com/chapter/10.1007/978-3-642-28365-9_3*




Software emulation
Xqueues
Data mover

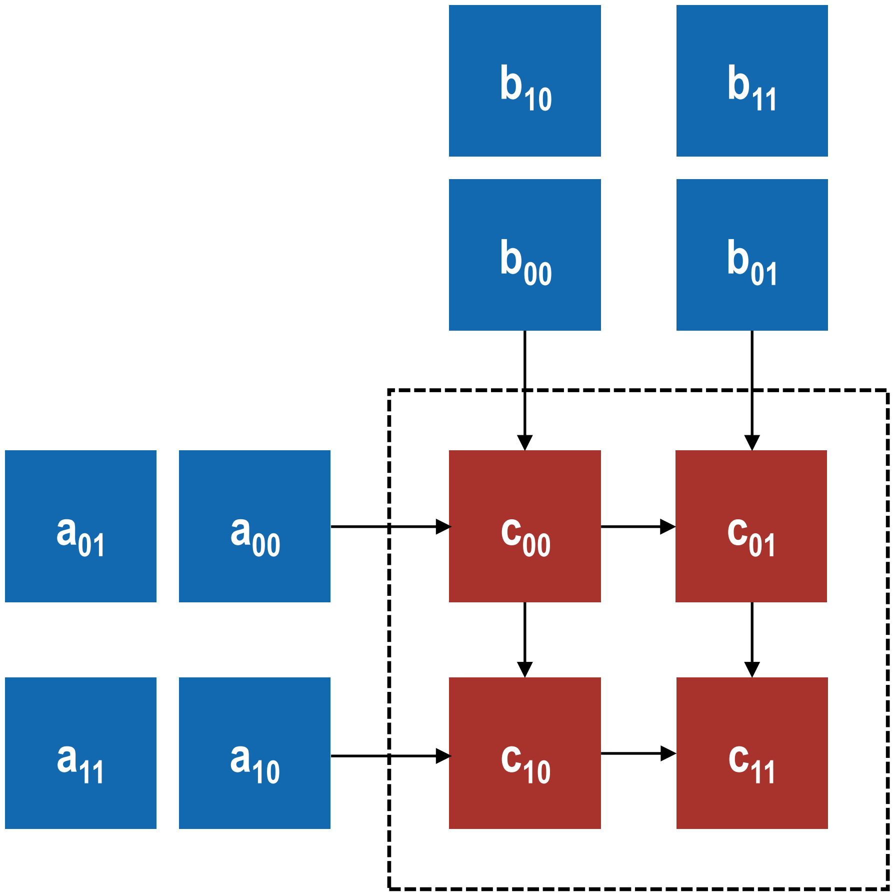
Emulate Systolic in Software

- Emulate all communication queues in software
- Explore systolic topologies
  - Arbitrary number of queues
  - Arbitrary interconnect topology

```
// Baseline
c = 0;
for (i=0; i<N; i++) {
  a = queue_pop(qa_in);
  b = queue_pop(qb_in);
  c += a * b;
  queue_push(a, qa_out);
  queue_push(b, qb_out);
}
```

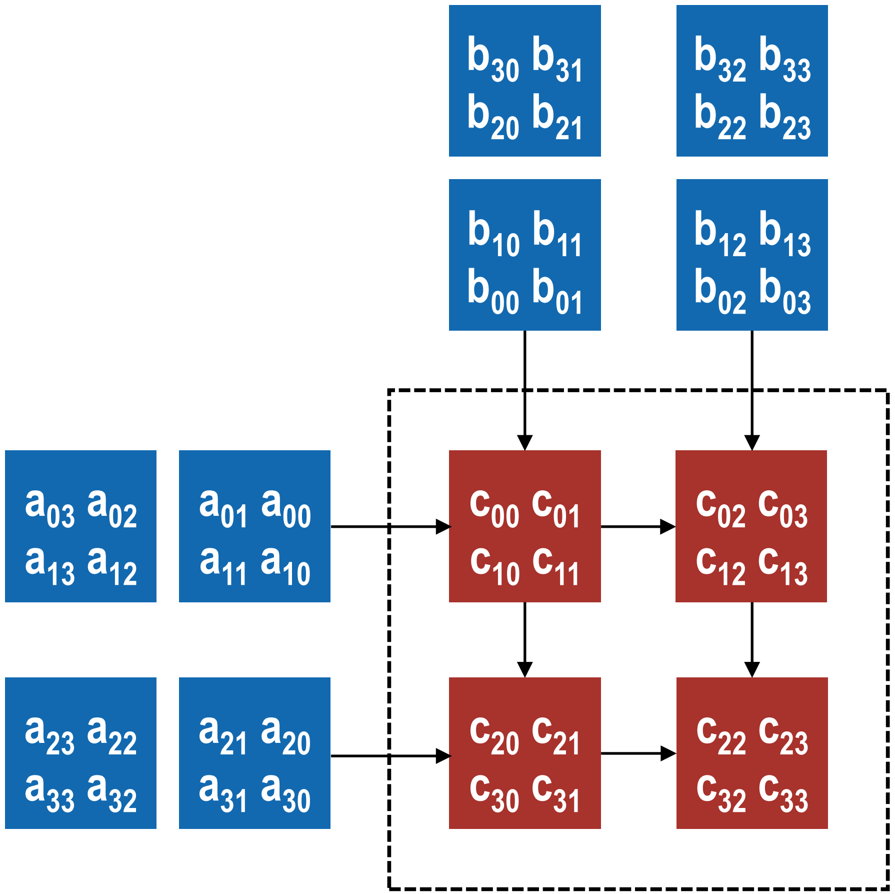
Matrix Multiplication

- **Systolic 2D grid**
  - Feed inputs from left and top
  - Outputs are stationary
- **MemPool's 256 cores form a 16x16 grid**
  - Two pushes and pops per MAC
- **Can we better utilize the cores?**



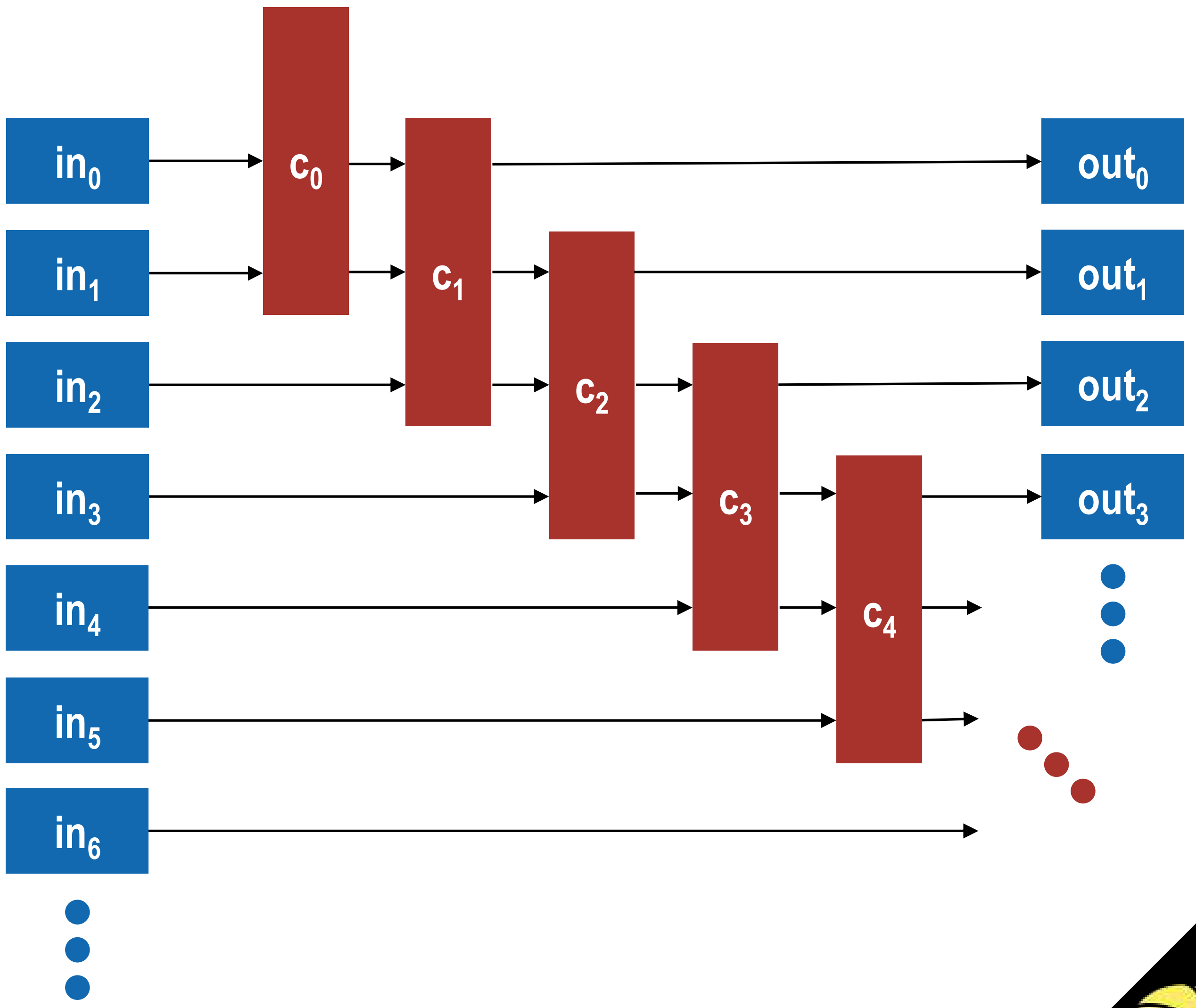
Matrix Multiplication

- **Utilize programmable cores**
  - Reuse data in register file
  - Allows for 32x32 tiles → more computation
  - 8 MACs for the same number of push and pop operations
- **5x faster than baseline topology**
  - Hybrid approach allows exploring topologies



2D Convolution

- **Different topology**
  - One long chain of PEs computing on input rows
  - Maximize input reuse
  - Weights can be stationary or streamed in
- **Our hybrid approach allows for flexible topologies**



Emulate Systolic in Software

- **Software emulation gives us flexibility**
- **At the cost of performance**
  - Software queue push and pop take tens to hundreds of cycles

```
// Baseline
c = 0;
for (i=0; i<N; i++) {
  a = queue_pop(qa_in);
  b = queue_pop(qb_in);
  c += a * b;
  queue_push(a, qa_out);
  queue_push(b, qb_out);
}
```

→ Function calls take up to hundreds of cycles

Software emulation → Xqueues → Data mover

ISA Extension: Xqueue pop and push

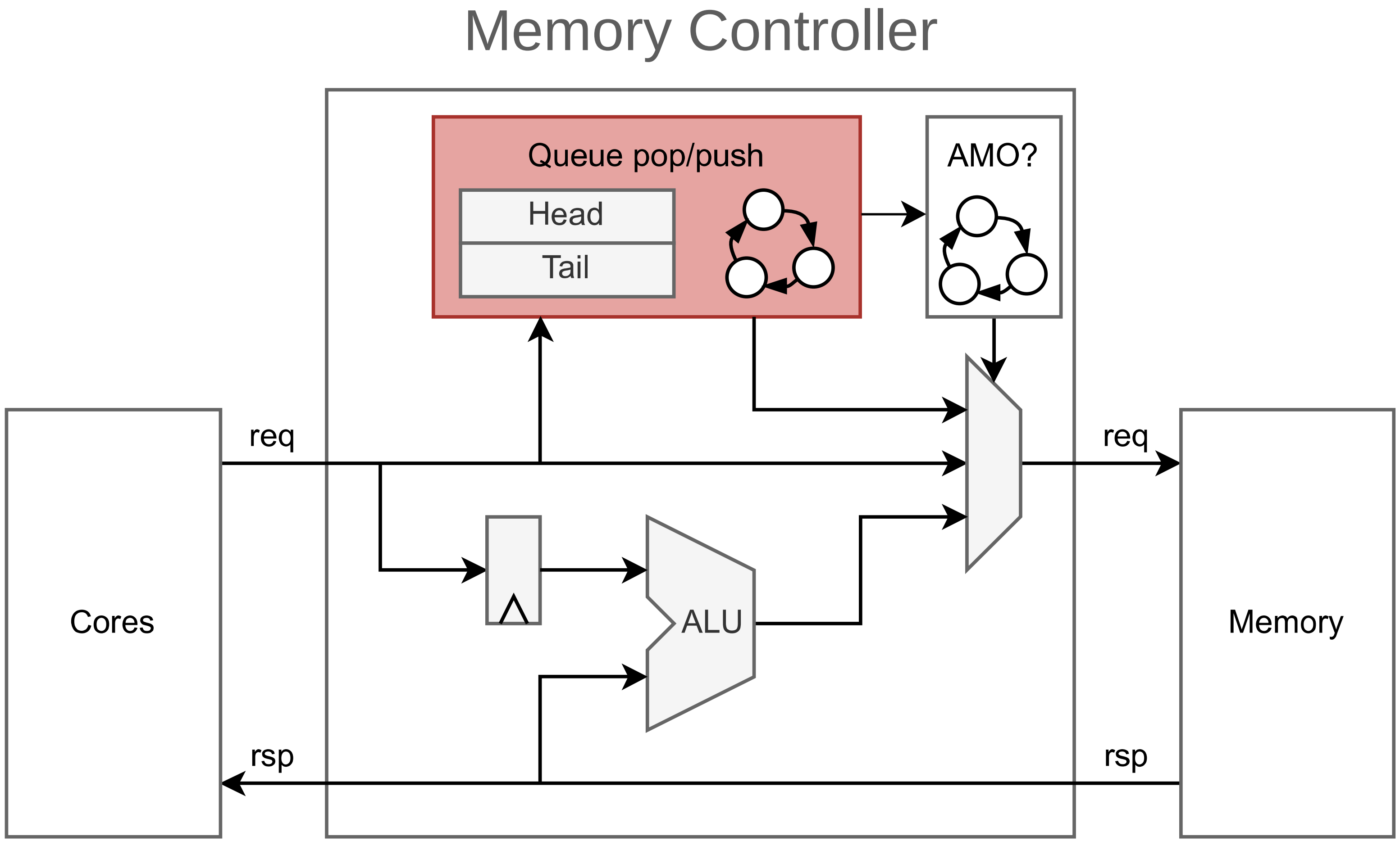
- Reduce queue access to a single instructions
  - Keep the benefits of queues in the memory
- Similar implementation to atomics
  - Extension in core and memory controller

Eliminate tens of instructions

```
// Baseline
c = 0;
for (i=0; i<N; i++) {
  a = queue_pop(qa_in);
  b = queue_pop(qb_in);
  c += a * b;
  queue_push(a, qa_out);
  queue_push(b, qb_out);
}
```

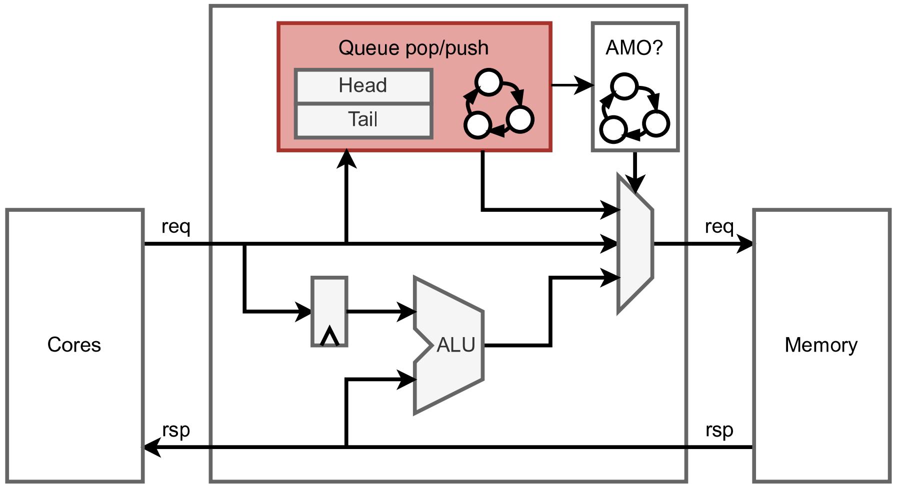
```
// +Xqueue pop/push extension
c = 0;
for (i=0; i<N; i++) {
  a = __builtin_pop(qa_in);
  b = __builtin_pop(qb_in);
  c += a * b;
  __builtin_push(a, qa_out);
  __builtin_push(b, qb_out);
}
```

Xqueue pop and push hardware



Queue pop and push in hardware

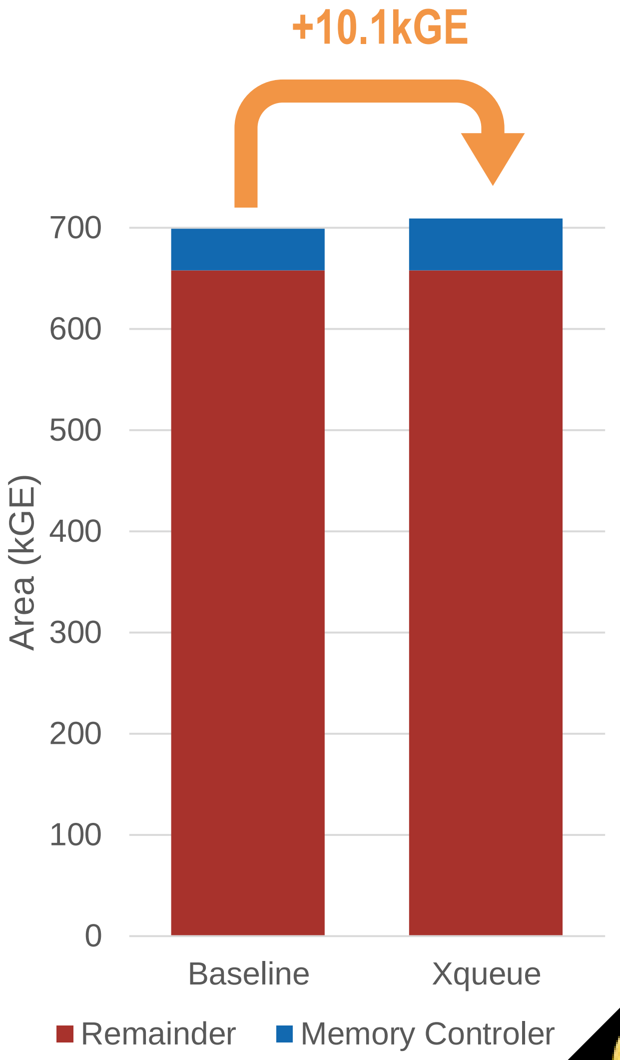
- **Fully parametrizable**
  - Number of queues per bank
  - Queue size
- **One queue per bank is enough**
  - 4 queues per core in MemPool



Area Cost Breakdown

Stage	Module Category	Total Area[1] [kGE]	Percent [%]
Baseline	Memory controller	41.3	5.9
	Remainder	657.9	94.1
	Total Tile	699.2	100.0
Xqueues	Memory controller	51.3	7.2
	Remainder	658.0	92.8
	Total Tile	709.3	100.0

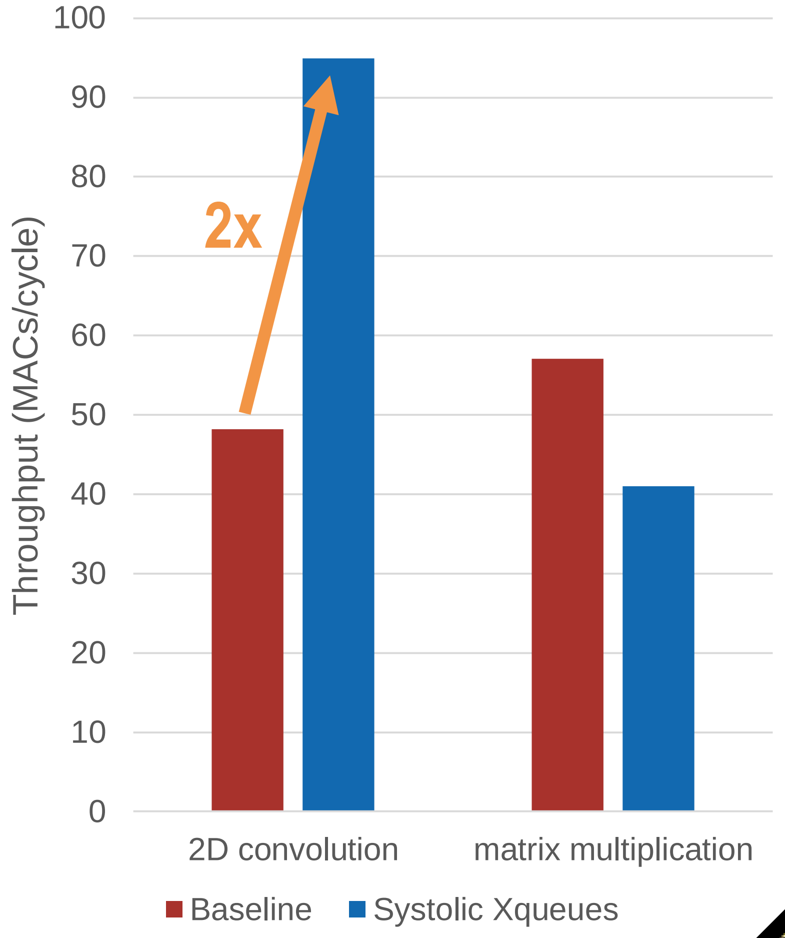
[1]*post-synthesis area in **22FDX** at **worst-case corner (0.72 V, 125°C)** targeting 500 MHz*



➔ Minimal hardware impact

Performance Evaluation

- Shared-memory vs systolic
- Double the performance on 2D convolution
- Baseline matmul is still faster
  - Limited by explicit queue operations
- How can we do even better?
  - Eliminate explicit communication




Software emulation
Xqueues
Data mover

Automatically push and pop

- Eliminate the explicit push/pop instructions
  - Stream-like behavior
  - Do communication in parallel
- Core focuses on computation
  - Extension to core

Eliminate explicit communication

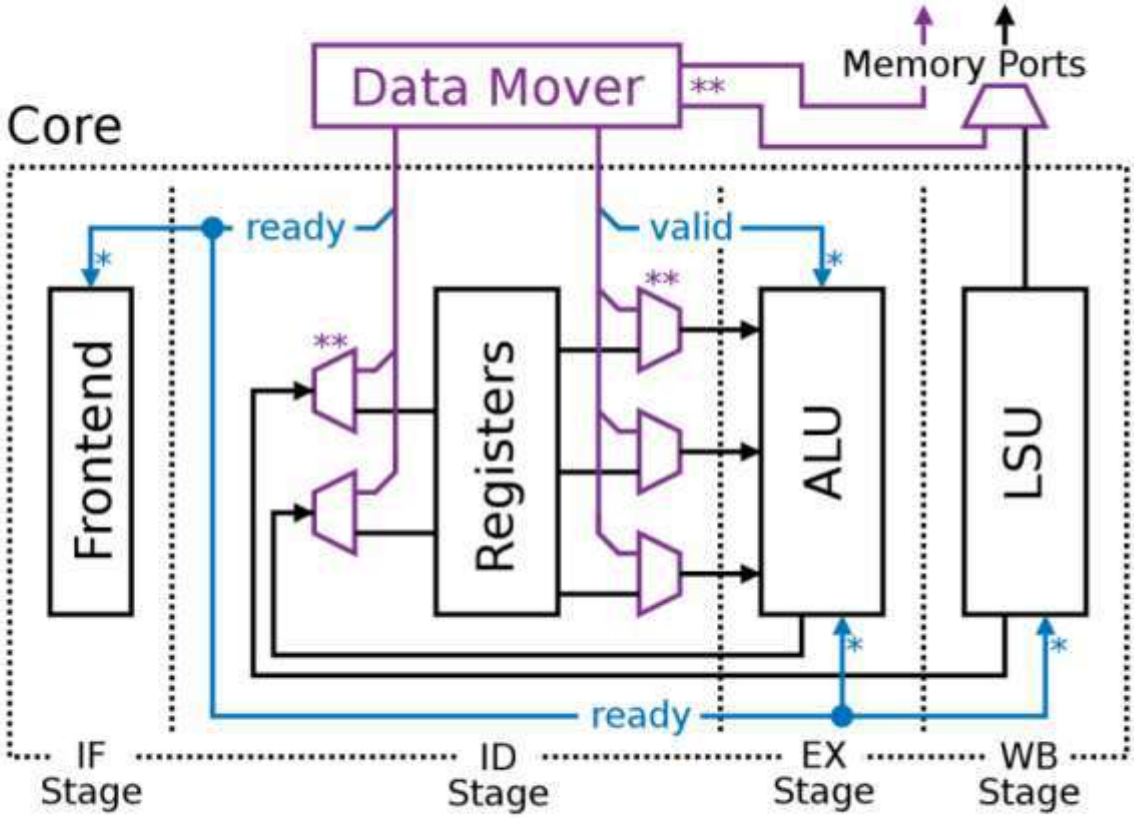
```
// Baseline
c = 0;
for (i=0; i<N; i++) {
  a = queue_pop(qa_in);
  b = queue_pop(qb_in);
  c += a * b;
  queue_push(a, qa_out);
  queue_push(b, qb_out);
}
```

```
// +queue pop/push extension
c = 0;
for (i=0; i<N; i++) {
  a = __builtin_pop(qa_in);
  b = __builtin_pop(qb_in);
  c += a * b;
  __builtin_push(a, qa_out);
  __builtin_push(b, qb_out);
}
```

```
// +Stream-like extension
c = 0;
setup_stream(a, qa);
setup_stream(b, qb);
for (i=0; i<N; i++) {
  c += a * b;
}
```

SSR extension

- 'Data Mover' can be configured to read/write data streams
  - Registers are refilled automatically
  - Data mover performs queue pop/push
  - Could increase memory ports
- Future work



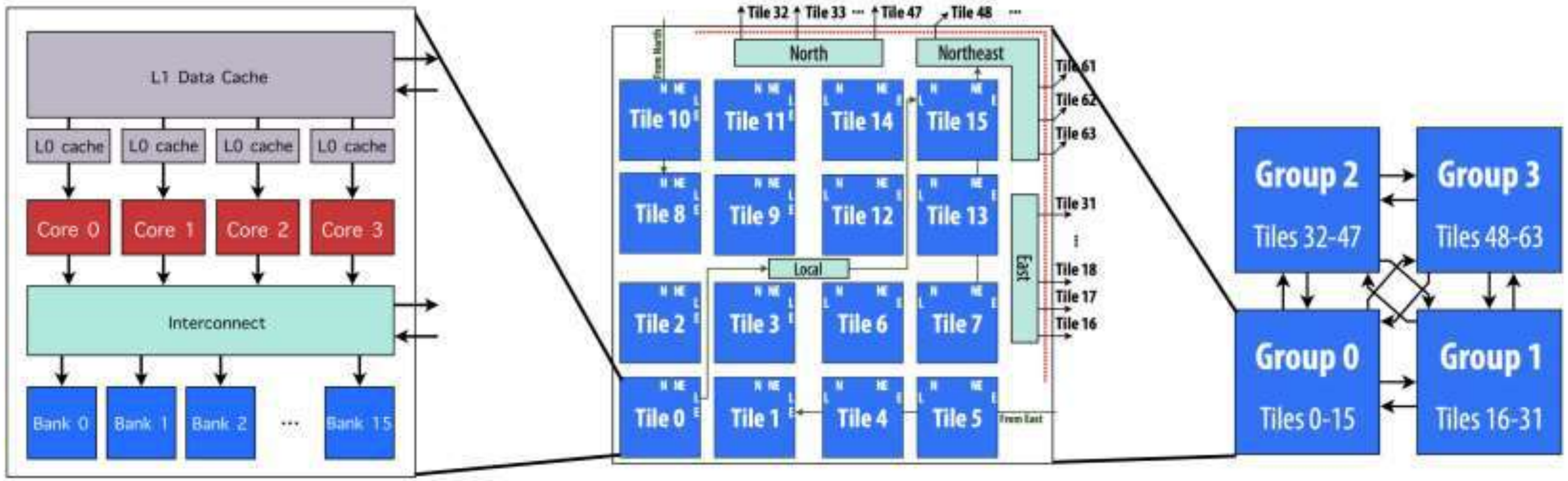
Source: SCHUIKI et al.: STREAM SEMANTIC REGISTERS - http://htor.inf.ethz.ch/publications/img/schuiki-ssr.pdf

Conclusion

- **Hybrid systolic shared-memory system**
  - Efficiently execute systolic workloads on a shared-memory system
  - Keep the flexibility of the shared-memory system
- **Explore systolic topologies**
  - Mix systolic and shared-memory programming
- **ISA extension: Xqueue**
  - 2x speedup for 1% hardware overhead
- **Future optimization with autonomous data mover**
  - Potentially double the performance

MemPool's Hierarchy

- Tile:
  - 4 32-bit cores
  - 16 banks
  - Single cycle memory access
- Group:
  - 64 cores
  - 256 banks
  - 3 cycles latency
- Cluster
  - 256 cores
  - 1 MiB of memory (1024 banks)
  - 5 cycles of latency



MemPool Tile

Group

Cluster