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 units
- 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
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

- 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

High performance

High flexibility
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
  - \( \leq 5 \) 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**

![Diagram of 2D Convolution]
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

```c
// 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 \rightarrow \text{Xqueues} \rightarrow \text{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

---

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

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

<table>
<thead>
<tr>
<th>Stage</th>
<th>Module Category</th>
<th>Total Area [kGE]</th>
<th>Percent [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Memory controller</td>
<td>41.3</td>
<td>5.9</td>
</tr>
<tr>
<td></td>
<td>Remainder</td>
<td>657.9</td>
<td>94.1</td>
</tr>
<tr>
<td></td>
<td>Total Tile</td>
<td>699.2</td>
<td>100.0</td>
</tr>
<tr>
<td>Xqueues</td>
<td>Memory controller</td>
<td>51.3</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td>Remainder</td>
<td>658.0</td>
<td>92.8</td>
</tr>
<tr>
<td></td>
<td>Total Tile</td>
<td>709.3</td>
<td>100.0</td>
</tr>
</tbody>
</table>

1post-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

Throughput (MACs/cycle)

Baseline vs Systolic Xqueues

2D convolution vs matrix multiplication

2x improvement
Automatically push and pop

- Eliminate the explicit push/pop instructions
  - Stream-like behavior
  - Do communication in parallel

- Core focuses on computation
  - Extension to core

// Baseline

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

// +queue pop/push extension

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

// +Stream-like extension

```c
int c = 0;
setup_stream(a, qa);
setup_stream(b, qb);
for (int i = 0; i < N; i++) {
    int a = __builtin_pop(qa_in);
    int b = __builtin_pop(qb_in);
    c += a * b;
    __builtin_push(a, qa_out);
    __builtin_push(b, qb_out);
}
```
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

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