Pipelining a Triggered Processing Element

Thomas J. Repetti, João P. Cerqueira, Martha A. Kim, Mingoo Seok
Columbia University
MICRO 2017
Outline

1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion
Spatial Accelerators

- Cryptography
- Computer Vision
- Data Analysis
- Scientific Computing
- Signal Processing

Wide variety of applications:

- Some general purpose
- Many domain workload specific
Figures of Merit

\[
\frac{\text{Instructions}}{\text{Time}} \times \frac{\text{Program}}{\text{Instructions}} = \frac{\text{Program}}{\text{Time}}
\]

\[
\frac{\text{Instructions}}{\text{Energy}} \times \frac{\text{Program}}{\text{Instructions}} = \frac{\text{Program}}{\text{Energy}}
\]
Figures of Merit

Spatially-Programmed Architectures, ISCA 2013

Parashar, Pellauer, et al. Triggered Instructions: A Control Paradigm for Program and ISA

Dynamic Instruction Count

\[
\frac{\text{Instruction \ Program}}{\text{Time}} \times \frac{\text{Instruction \ Program}}{\text{Energy}} = \frac{\text{Program}}{\text{Time}} \times \frac{\text{Program}}{\text{Energy}}
\]
Microarchitecture and Circuit Support for deep pipelines gives fine-grain control over energy and delay at the microarchitectural and circuit level.
Prerequisite for non-trivial, high-latency functional units
Benefits of Deeper Pipelines

- Prerequisite for non-trivial, high-latency functional units

For a given design, the increased timing slack gives you options:

- Decrease supply voltage
- Increase frequency (save energy)

Pre-requisite for non-trivial, high-latency functional units
Outline

1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion
Background on Triggered Instruction Architectures

- Locally autonomous
- No program counter
- PEs communicate through queues

4

Locally autonomous

PEs instruct PEs

4

Triggereg instruction Architectures

Background on
Background on Triggered Instruction Architectures

Actions: function of program

- Locally autonomous
- No program counter
- PEs communicate through queues

Actions: function of program

- halt
- when %p == X111
  - when %p == X010
    - set %p = ZZ00
  - when %p == X011
    - set %p = ZZ11
  - when %p == X110
    - set %p = ZZ10
  Other actions...

Triggered Instruction Architectures
when \( p == 00 \) with \( i_0, i_1 \):
\[
\text{mul } r_1, i_0, i_1; \text{ deq } i_0, i_1; \text{ set } p = 01
\]

when \( p == 01 \):
\[
\text{add } r_0, r_0, r_1; \text{ set } p = 10
\]

when \( p == 10 \):
\[
\text{ugt } p_3, r_0, r_2; \text{ set } p = 11
\]

when \( p == 11 \):
\[
\text{mov } o_1.0, r_0; \text{ set } p = 00
\]

when \( p == 111 \):
\[
\text{halt};
\]

Background on Triggered Instruction Architectures

- Locally autonomous
- No program counter
- PEs communicate through queues

Actions: function of program

Guards: control flow

Processing Element

Interconnected insensitive latency

• PEs independently

Triggeoked Instruction Architectures on Background
when %p == XX00 with %i0.0, %i1.0:
  mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
  add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
  ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
  mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
  halt;

Background on Triggered Instruction Architectures

- Locally autonomous
- No program counter
- PEs communicate through queues

Actions: function of program

Guards: control flow

Predicate update: $\sim \Phi = PC + 4$

- Interconnection insensitive
- Latency
- Element
- Processing

Triggered Instruction Architectures on Background
when %p == XX00 with %i0.0, %i1.0:
  mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
  add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
  ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
  mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
  halt;

---

Background on Triggered Instruction Architectures

- Locally autonomous
- No program counter
- PEs communicate through queues
- Queue operations: Spatial
- Guards: control flow
- Actions: function of program
- Predicate update: ~PC = PC + 4
- Actions: function of program
- Guards: control flow
- Queue operations: Spatial
- Locally autonomous
when %p == XX00 with %i0.0, %i1.0:
mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
halt;

Actions: function of program
Guards: control flow
Predicatc update: ~ PC = PC + 4
Datapath predicate write: data-communication
Queue operations: spatial
Dependent control flow

- PEs communicate through queues
- No program counter
- Locally autonomous

Actions: function of program
Guards: control flow
Predicatc update: ~ PC = PC + 4
Datapath predicate write: data-communication
Queue operations: spatial
Dependent control flow

- PEs communicate through queues
- No program counter
- Locally autonomous

Background on Triggered Instruction Architectures

● Locally autonomous
● No program counter
● Interconnected

Element
Processing
Lateney
Insensitive
3. Challenges of Triggered Pipelines
Pipeline Stages and State

<table>
<thead>
<tr>
<th>Stage</th>
<th>Register File</th>
<th>Input Queues</th>
<th>Output Queues</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>N</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trigger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Architectural state: atomically state updated
Predicate state: updated, predicate read, predicate read
## Pipeline Stages and State

<table>
<thead>
<tr>
<th></th>
<th>N Execute</th>
<th>I Execute</th>
<th>Decode</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Queues</td>
<td></td>
<td></td>
<td></td>
<td>read</td>
</tr>
<tr>
<td>Output Queues</td>
<td>read</td>
<td></td>
<td></td>
<td>update</td>
</tr>
<tr>
<td>Register File</td>
<td>pop</td>
<td>read</td>
<td></td>
<td>update</td>
</tr>
<tr>
<td>Predicates</td>
<td>read</td>
<td>read</td>
<td>read</td>
<td>tokens consumed</td>
</tr>
</tbody>
</table>

- N Execute
- I Execute
- Decode
- Trigger
Pipeline Stages and State

Data-dependent predicate update, register file and output queue retirement.

Predicates
Register File
Input Queues
Output Queues
Triggered Pipeline Hazards
Triggered Pipeline Hazards

1. Traditional RAW hazard: Solutions well known
1. Traditional RAW hazard: solutions well known
2. Predicate hazard: data-dependent write to predicate register
1. Traditional RAW hazard: solutions well known.
3. Input queue pop.
1. **Traditional RAW hazard:**
   Solutions well known

2. **Predicate hazard:**
   Data-dependent write to predicate register

3. **Input queue pop**
   Output queue push

4. **Output queue push**
   Input queue pop

Trigged Pipeline Hazards
1. Traditional RAW hazard:
   Solutions well known

2. Predicate hazard:
   Data-dependent write to predictor register

3. Input queue pop

4. Output queue push
1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion
Predicate Prediction

- Speculate on predicate
- Forbid updates to datapath result
- Forbid updates to architectural state
during speculation
- Memory write
- Pending dequeue
- Flush via pipeline

Saturating Predictors

Registers

Predicate Predication
Surprisingly effective

- Predicate indices encode branch locality
- Predictable primary control structure (loop-like or low-entropy)
Predicate Prediction

- Surprisingly effective
- Encode branch locality
- Unpredictable branch nested within loop
- Unpredictable primary control
- Surprisingly effective
- Encode branch locality
- Unpredictable indices
- Surprisingly effective

Predicate indices encode branch locality

Predicatable primary control

Unpredictable branch nested within loop

Structure (loop-like or low-entropy)
Predicate Prediction

- Surprisingly effective
- High-entropy data
- Unpredictable branch
- ked within loop
- High-entropy data

Unpredictable branch nested within loop
- Structure: loop-like or low
- Predictable primary control

Encode branch locality
- Predictable indices
- Encode branch locality

Workload:
- Merge
- Filter
- Blt
-udy
- String search
- Arg max
- Stream
- Red mean
- Predicte Primary control
- Encode branch locality
- Predictable indices
- Surprisingly effective
53.6 → 64.3 FO4
+7.0% Power
+0.5% Area

Δ CPI: -17%
Effective Queue Status

- Processor pipelines are FIFO queues
- Trigger stage needs an up-to-date view of the world:
  - Output queues
  - The pipeline itself as a queue
  - Input queues
Trigger stage must account for:

- Input queue empty state and head semantic tag information
- Output queue full state

Effective Queue Status

Input Queues: %i0, %i1
Output Queues: %o0, %o1

Tag of head empty, full
Problem: Dequeueing from an input queue has a single cycle of latency.

Effective Queue Status
Effective Queue Status

**Solution:** Expose head and "neck" or queue and calculate if empty pending the dequeue.

```
deq %i0
```
Effective Queue Status

Solution: Expose head and "neck" of queue and calculate if empty pending dequeue.

\[
\text{deq} \%i0 \rightarrow \%i0 \text{ next token visible}
\]
Problem: Insight enqueues can effect the future state of the output queues.
Solution: Expose output queue counts, inspect the pipeline for input enqueues and calculate effective status for trigger stage.

Right enqueue and calculate effective status for trigger stage.
Effective Queue Status

Input Queues

Output Queues

Solution: expose output queue counts, inspect the pipeline for input enqueues and calculate effective status for trigger stage.
**Effective Queue Status**

**Input Queues**

**Output Queues**

**Effect of optimizations:**

- Producing data tokens
- Consuming data tokens
- Maintain high instruction throughput when consuming data tokens

**Count**
Effective Queue Status Performance

Δ CPI: -17%

+P

+Q

No Effect

0% Power

+2.0% Area

on Timing

pipeline depth

0

0.5

1

1.5

2

2.5

3

3.5

CPI

Default

+P

+Q

4

3

2

1
Combined Effect of Optimizations

Effect on CPI is Additive

Δ CPI: -35%

Combined Effect of Optimizations
Outline

1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion
Experimental Infrastructure

workload.s

Assembler (Python)

workload.bin

Userspace (C)

Driver (C)

Spatial Accelerator

FPGA

ARM Host

(Zynq Board)

array.sv

workload.bin

workload.s

(Charm)

Assembler (Python)
Experimental Infrastructure

Assembler (Python)

workload.bin

Userspace (C)

Driver (C)

Spatial Accelerator

ARM Host

FPGA

(Zynq Board)

array.sv

{ string-search, dot-product, filter, arg-max, bst, merge, ... }

workloads.s
Experimental Infrastructure

- Assembler (Python)
- Workload.bin
- Userspace (C)
- Driver (C)
- Spatial Accelerator
- ARM Host
- FPGA (Zynq Board)

*Array.sv*
*parameters.yaml*
*array.sv*
*workload.bin*
*workload.s*

- String-search
- dot-product
- filter, arg-max,
- best, merge,
Experimental Infrastructure

- workload.s
- Assembler (Python)
- workload.bin
- Userspace (C)
- Driver (C)

Spatial Accelerator

ARM Host

FPGA

Zynq Board

Spatial Accelerator

array.sv

parameters.yaml

- word width
- num. instructions per PE
- num. data registers
- pipeline µarch
- etc...

workloads.bin

workload.s

(Assembler)

Python

14

Parameters.yaml

string-search

best, merge,

filter, arg-max,
Experimental Infrastructure

- Userspace (C)
- Driver (C)
- Assembler (Python)

- bst, merge, filter, arg_max, dot_product, string_search...

- workload.s
- workload.bin
- array.sv
- parameters.yaml

Performance Counters

- word width
- num. instructions per PE
- num. data registers
- pipeline µarch
- etc...
Experimental Infrastructure

Performance Counters

Available: github.com/arcade-lab/tia-infrastructure

-bst, merge, %lter, arg_max, dot_product, string_search...

array.sv

parameters.yaml

Spatial Accelerator

ARM Host

Driver (C)

Userspace (C)

(ARM Board)

FPGA

(word width)

(num. instructions per PE)

(num. data registers)

(pipeline arch)

etc...

_workload.s

_workload.bin

_Assembler

_Python

_workload.s

array.sv
Evaluation Methodology

Assembler

bst.s

bst.bin
Evaluation Methodology

- Assembler
- bst.bin
- Post-Synthesis
  - Gate-Level Simulation
  - Synthesized Netlist
    - 8 different Pipelines with and without pipeline optimizations
    - (T|DX, T|DX1|X2, etc.)

Input
- bst.vcd
- bst.bin

Simulation

Assembler

Data
- bst.s

Post-Synthesis

Synthesized Netlist

Input Data
- bst.s

Diagram:
- Assembler
- bst.bin
- Post-Synthesis
  - Gate-Level Simulation
  - Synthesized Netlist
    - 8 different Pipelines with and without pipeline optimizations
    - (T|DX, T|DX1|X2, etc.)
Evaluation Methodology

- Synthesized Netlist
  - Fully Annotated
  - Std. Cell Library

- Post-Synthesis
  - Gate-Level Simulation
  - Simulator
  - bst.vcd

- Input Data
  - bst.s

- Power Performance Area
  - Pipelines with and without optimizations
    - 8 different pipelines
    - 8 differ pipelines

- Simulator
  - Post-Synthesis
  - bst.bin

- Assembly
  - bst.s
  - bst.vcd

(\text{V}_{\text{DD}} = 1.0 - 0.4 \text{ V})
Evaluation Methodology

TSMC 65 nm low-, high- and standard-Vt GP standard cells

Std. Cell Library (Vdd = 1.0 - 0.4 V)

Fully Annotated Netlist in PrimeTime

Post-Synthesis Gate-Level Simulation

Synthesized Netlist

- 8 diff. Pipelines (TDX, TDX1X2, etc.)
- with and without pipeline optimizations

Power Performance Area

Input Data

Assembler

bst.s

bst.bin

bst.vcd
Outline

1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion
Microarchitectural and Circuit Design Space

- 225x span in delay
- 71x span in energy
- 4,000+ design points
- 1.37 - 3.09, 0.3 - 0.69, 47.59 - 47.03 ns / inst.
Microarchitectural and Circuit Design Space

- 4,000+ design points
- 71x span in delay
  - 1.37 - 3.09 ns / inst.
- 71x span in energy
  - 0.69 - 47.59 pJ / inst.
- 225x span in aggressive supply voltage scaling

Pipelines enable aggressive supply voltage scaling
Proposed optimizations especially effective in the balanced region of the Pareto frontier.
Microarchitectural and Circuit Design Space

Proposed optimizations especially effective in the balanced region of the Pareto frontier.

Predicate prediction (+p)
Microarchitectural and Circuit Design Space

- Proposed optimizations especially effective in the balanced region of the Pareto frontier
- Effective Queue Status (+Q)
- Predicate Prediction (+P)
- The Pareto frontier
Effective Queue Status (+Q)

Predicate Prediction (+p)

Effect is additive

The Pareto Frontier

The balanced region of especially effective in proposed optimizations

Microarchitectural and Circuit Design Space
Proposed optimizations especially effective in the balanced region of the Pareto frontier.

- Predicate prediction (+P)
- Effective Queue Status (+Q)
- Predicate prediction

Example: with an energy budget of \(1.7\) pJ / inst. \(+P+Q\) improves throughput by \(2\times\).
Pipelining a triggered processing element presents new but tractable microarchitectural hazards. Work well in a wide range of regimes: low-power to high-performance. Demonstrated two techniques:

- Effective Queue Status
- Predicate Prediction

Pipelining a triggered processing element.