Practical Acceleration of Irregular Applications on Reconfigurable Architectures

Quan M. Nguyen and Daniel Sanchez
Massachusetts Institute of Technology

MICRO-54
Live session: Session 9A (Graph Processing)
October 21, 2021 at 2:15 PM EDT
Irregular applications are difficult to accelerate

- Irregular applications’ unpredictable data reuse & control flow are hard to accelerate on today’s architectures
  - CPUs: poor latency tolerance, high instruction execution overheads
  - Dedicated accelerators: not flexible

- **Reconfigurable spatial architectures** offer circuit-level control of distributed computation, but...
  - still cannot extract enough parallelism
  - only benefit *regular* memory/compute patterns
Fifer enables accelerating irregular applications

• Insight: accelerate irregular applications by exploiting pipeline parallelism

• Create **dynamic temporal pipelines**: time-multiplexing stages of a pipeline on reconfigurable fabric

• Fifer’s speedups: over gmean 17x over OOO multicore and 2.8x over reconfigurable spatial architectures without time-multiplexing
Agenda

Intro ➔ Background ➔ Fifer ➔ Evaluation
Irregular applications are difficult to accelerate

• Characterized by **unpredictable reuse**:
  • Caches, scratchpads capture some locality
  • But, irregular applications generally have poor locality, large data structures

```python
def bfs(src):
    ...
    for v in current fringe:
        start, end = offsets[v], offsets[v+1]
        for ngh in neighbors[start:end]:
            dist = distances[ngh]
            if dist is not set:
                set distance; add to next fringe
    ...
```
General-purpose cores handle irregular applications poorly

- Modern cores have expensive latency tolerance mechanisms:
  - Out-of-order execution
  - Multithreading
- General-purpose cores are temporal architectures: they change operations (instructions) over time
- Unit of work is small; high fetch/decode overheads
Spatial architectures improve computational intensity...

- Map operations *spatially* to array of *functional units* (FUs)
- **Switches** set to pass operands between FUs
- Input/output ports feed values to/from fabric
- FUs operate at machine word width: *coarse-grain reconfigurable array* (CGRA)
Anatomy of a CGRA-based system

- Many processing elements (PEs) with fabric and private cache
- Data flow within a CGRA: rigid pipelines
- Inter-PE communication: decoupled
- Control core for system interactions, setup/teardown
... but not flexible enough for irregularity

- Only transform inner loop
- Some approaches highly specialized to application
- Irregular applications have unpredictable latencies and variable computational intensity as they execute
- Current spatial architectures would either stall or suffer poor utilization (many PEs idle)
Time-multiplexing on CGRAs: Triggered Instructions [ISCA’13]

• Triggered Instructions PEs can choose among many instructions
• **Limited number of instructions** (16)
• **Complex scheduling** to keep PEs active
• Fifer’s approach: **coarse-grain reconfiguration on coarse-grain sets of operations**

![Figure 6: A triggered-instruction based PE.](image-url)
Irregular applications can be decoupled and mapped to spatial architectures

```python
def bfs(src):
    ...
    
    for v in current fringe:
        start, end = offsets[v], offsets[v+1]
        for ngh in neighbors[start:end]:
            dist = distances[ngh]
            if dist is not set:
                set distance; add to next fringe
    ...
```

- Process current fringe
- Enumerate neighbors
- Visit neighbors
- Update data, next fringe
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures

Static pipeline
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures

Dynamic temporal pipeline

Queues
Insight: Create dynamic temporal pipelines on reconfigurable spatial architectures
Agenda

Intro ➔ Background ➔ Fifer ➔ Evaluation
Fifer is flexible yet performant

- **Coarse-grain reconfigurable array** (CGRA) increases compute density over general-purpose cores
- **Buffering** between PEs and *within* PEs provides latency tolerance
- **Time-multiplexed fabric** keeps throughput and utilization high
Mapping applications to Fifer

Serial code:

```python
for e in range(start, end):
    ngh = neighbors[e]
```

Pseudo-assembly:

```assembly
mov %r_neighbors, ...;
deq %r_e, $q_start;
deq %r_end, $q_end;
loop:
    lea %r_addr, (%r_neighbors,%r_e,2);
    ld %r_ngh, (%r_addr);
    enq $q_ngh, %r_ngh;
    addi %r_e, %r_e, 1;
    blt %r_e, %r_end, loop
done:
    ...
```
Fifer needs reconfigurations to be...

**Rare**
- Amortize reconfiguration cost over hundreds of cycles
- **Round-robin** scheduling switches too often
- Fifer’s **scheduler** in each PE keeps a stage scheduled until queues become full or empty
- Prioritize stages with most work in input queues.

**Fast**
- Quickly tolerate variations in the amount of work between stages
- Prior techniques (e.g., scan chains) reconfigure in ~microseconds; we need cycles
- Fifer’s **double-buffered configuration cells** make reconfiguration fast at low hardware cost
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches
Configuration signals

ALU

Current Config.

Config. Slot A
Config. Slot B

Config. data from L1 cache
Config. load & select signals

Double-buffered Configuration Cell
Fast reconfiguration with double-buffering

**Functional Unit**
- Connections from switches
- Configuration signals

**Config. Cell**
- Current Config.

**Double-buffered Configuration Cell**
- Config. data from L1 cache
- Config. load & select signals

**Config. Slot A**
- Config. Slot B

**ALU**
- To switches

**Time**
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches

Configuration signals

> A
> B
> const

ALU

Double-buffered Configuration Cell

Config. data from L1 cache

Config. load & select signals

Config. Slot A

Config. Slot B

Configuration signals

Connections from switches

To switches

Current Config.

Green steady-state execution

Config. Slot A

Config. Slot B

Time

Config. Cell

Config. data from L1 cache

Current Config.

Time
Fast reconfiguration with double-buffering

Double-buffered Configuration Cell

Scheduler decides to switch

Config. Slot A
Green steady-state execution

Config. Slot B

Functional Unit

Connections from switches

Configuration signals

ALU

A
B
const

Current Config.

Config. data from L1 cache

Config. load & select signals

To switches

Connections to switches

Config. Cell

Config. Slot A

Config. Slot B

Time
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches

Configuration signals

Config. Cell

To switches

Double-buffered Configuration Cell

Config. data from L1 cache

Config. load & select signals

Current Config.

Scheduler decides to switch

Config. Slot A

Green steady-state execution

Config. Slot B

ALU

A

B

const

Config. data

Config. load & select signals

Config. Slot A

Config. Slot B

Time
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches
Configuration signals

A
B
const

ALU

To switches

Double-buffered Configuration Cell

Scheduler decides to switch

Config. Slot A
Config. Slot B

Config. load & select signals

Config. data from L1 cache

Current Config.

Load & buffer new config.

Green steady-state execution

Config. Slot A

Time
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches

Configuration signals

ALU

A

B

const

Config. Cell

Double-buffered Configuration Cell

Config. data from L1 cache

Config. Slot A

Config. Slot B

Config. load & select signals

Current Config.

Scheduler decides to switch

Config. Slot A

Green steady-state execution

Config. Slot B

Load & buffer new config.

To switches

Config. data

Time
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches

Configuration signals

Config. Slot A

Config. Slot B

Current Config.

Scheduler decides to switch

Config. data from L1 cache

Config. load & select signals

Double-buffered Configuration Cell

ALU

Config. Cell

To switches

Load & buffer new config.

Green steady-state execution

Load & buffer new config.

Activate Config B

Time
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches
Configuration signals

Connections to switches

ALU

Double-buffered Configuration Cell

Config. data from L1 cache
Config. load & select signals

Current Config.

Config. Slot A
Configs. data from L1 cache
Config. Slot B

Scheduler decides to switch

Load & buffer new config.

Time

Functional Unit

A
B
const

Config. Slot A

Config. Slot B

Green steady-state execution

Scheduler decides to switch
Fast reconfiguration with double-buffering

Config. Slot A

Config. Slot B

Connections from switches

To switches

Config. data from L1 cache

Config. load & select signals

Scheduler decides to switch

Green steady-state execution

Orange steady-state execution

Load & buffer new config.

Time

Functional Unit

Double-buffered Configuration Cell

Config. Cell

Current Config.

Config. data

Connections

ALU

A

B

const

Config. load & select signals
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches

Configuration signals

A
B
const

Config. Cell

Double-buffered Configuration Cell

Config. data from L1 cache

Config. Slot A
Config. Slot B

Config. load & select signals

Current Config.

Scheduler decides to switch

Config. Slot A

Green steady-state execution

Activate Config B

Config. Slot B

Orange steady-state execution

Load & buffer new config.

Time

Functional Unit

Connections from switches

Configuration signals

A
B
const

Config. Cell

Double-buffered Configuration Cell

Config. data from L1 cache

Config. Slot A
Config. Slot B

Config. load & select signals

Current Config.

Scheduler decides to switch

Config. Slot A

Green steady-state execution

Activate Config B

Config. Slot B

Orange steady-state execution

Load & buffer new config.

Time
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches
Configuration signals

A
B
const

ALU

To switches

Current Config.

Configuration Cell

Config. data from L1 cache
Config. load & select signals

Config. Slot A
Config. Slot B

Double-buffered Configuration Cell

Scheduler decides to switch
Load & buffer new config.

Green steady-state execution
Orange steady-state execution

Time

Drain in-flight operations

18
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches

Configuration signals

ALU

To switches

Config. Slot A

Config. Slot B

Current Config.

Scheduler decides to switch

Activate Config A

Activate Config B

Orange steady-state execution

Drain in-flight operations

Load & buffer new config.

Config. data from L1 cache

Config. load & select signals

Double-buffered Configuration Cell

Config. data

Config. load & select signals

Connections from switches

Configuration signals

To switches
Fast reconfiguration with double-buffering

Functional Unit

Connections from switches
Configuration signals

Connections to switches

ALU

Config. Cell

Config. Slot A
Config. Slot B

Double-buffered Configuration Cell

Scheduler decides to switch

Green steady-state execution
Activate Config A
Activate Config B

Orange steady-state execution

Load & buffer new config.

Blue steady-state execution

Drain in-flight operations

Time

Config. load & select signals

Config. data from L1 cache

Current Config.

A

B

const

A

B

const

Config. data from L1 cache

Config. load & select signals

Double-buffered Configuration Cell
Exploiting pipeline and data parallelism together

- Process current fringe
- Enumerate neighbors
- Visit neighbors
- Update data, next fringe
Exploiting pipeline and data parallelism together

PE0

Proc. cur. fringe → Enum. neighs. → Fetch dists. → Upd. data, next fringe
Exploiting pipeline and data parallelism together

- Replicate pipelines using many PEs
- Careful partitioning avoids synchronization through shared memory
- Increase PE utilization & parallelism by performing multiple units of work in one stage (SIMD-style)
  - e.g., enumerate multiple neighbors per cycle
See paper for more

• Accelerating memory accesses with decoupled reference machines
• Evaluating other decoupling strategies
• Handling control flow
• Energy, area estimates for Fifer’s major components
Agenda

Intro ➔ Background ➔ Fifer ➔ Evaluation
Evaluation

• Baseline system: 16-PE system with static stage mappings (*no* intra-PE queues)
• Other comparison systems: serial and 4-core OOO cores
• A Fifer PE is 1.34 mm² at 45 nm; core count set for roughly iso-area comparison
• Benchmarks evaluated:
  • Graph analytics (BFS, Connected Components, PageRank-Delta, Radii)
  • Sparse linear algebra (SpMM)
  • Databases (Silo)
Other application pipelines

Graph analytics
- Process current fringe
- Enumerate neighbors
- Visit neighbors
- Update data, next fringe

SpMM (linear algebra)
- Stream rows of A
- Merge-intersect
- Stream cols of B
- Accumulate

Silo (database)
- Query stage
- Lookup stage
- Traverse internal node
- Process leaf node
Fifer outperforms the baseline

- Fifer achieves consistent speedups over the baseline

![Bar chart showing speedup comparison between Multicore OOO, Static pipeline, and Fifer for various benchmarks: BFS, CC, PRD, Radii, SpMM, and Silo. Fifer outperforms the baseline significantly.]
Fifer effectively hides latency

- Fifer reduces waiting on queues, even when reconfiguring
Fifer achieves infrequent and fast reconfiguration

- Stages remain resident for hundreds of cycles
- SpMM has very short residence time, necessitating double-buffering

<table>
<thead>
<tr>
<th>Application</th>
<th>BFS</th>
<th>CC</th>
<th>PRD</th>
<th>Radii</th>
<th>SpMM</th>
<th>Silo</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average residence time (cycles)</td>
<td>140</td>
<td>279</td>
<td>927</td>
<td>564</td>
<td>30</td>
<td>1490</td>
<td>448</td>
</tr>
<tr>
<td>Average reconfiguration period (cycles)</td>
<td>12.5</td>
<td>13.9</td>
<td>20.4</td>
<td>27.7</td>
<td>12.6</td>
<td>60.1</td>
<td>19.7</td>
</tr>
</tbody>
</table>
Fifer helps accelerate irregular applications on reconfigurable architectures

- Transform irregular applications into dynamic temporal pipelines and efficiently execute them on reconfigurable spatial architectures
- Fast reconfiguration is enabled by Fifer’s scheduler and double-buffered configuration cells
- Fifer makes executing irregular applications on reconfigurable spatial architectures practical
Thank you!

Fifer:
Practical Acceleration of Irregular Applications on Reconfigurable Architectures

Quan M. Nguyen and Daniel Sanchez
qmn@csail.mit.edu and sanchez@csail.mit.edu

MICRO-54
Live session: Session 9A (Graph Processing)
October 21, 2021 at 2:15 PM EDT

This presentation and recording belong to the authors. No distribution is allowed without the authors' permission.