Efficient and Easily Programmable Accelerator Architectures

Tor Aamodt
University of British Columbia

PPL Retreat, 31 May 2013
Decreasing cost per unit computation

1971: Intel 4004

1981: IBM 5150

2007: iPhone

2012: Google Datacenter
Heterogeneity helps…
Review: Amdahl’s Law

Hard to accelerate

Easy to accelerate

\[
\text{Improvement}_{\text{overall}} = \frac{1}{\text{Fraction}_{\text{hard}} + \frac{1 - \text{Fraction}_{\text{hard}}}{\text{Improvement}_{\text{easy}}}}
\]
What defines division between hard and easy?

Fraction_{\text{hard}} = f(\text{problem, prog. model, SW budget})
Goal:

- easy to accelerate (Acc. Arch1)
- easy to accelerate (Acc. Arch2)
Ease of Programming

Hardware Efficiency

Better

?
Increase Accelerator Efficiency (x-axis)
- Control Flow
- Memory

Improve Accelerator Programmability (y-axis)
- Easier coding
- Fewer bugs
- Easier debugging
SIMT Execution (MIMD on SIMD)
(Levinthal SIGGRAPH’84)

foo[] = {4,8,12,16};
A: n = foo[tid.x];
B: if (n > 10)
C:    …;
else
D:    …;
E:    …

Branch Divergence

<table>
<thead>
<tr>
<th>PC</th>
<th>Active Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1111</td>
</tr>
<tr>
<td>D</td>
<td>1100</td>
</tr>
<tr>
<td>C</td>
<td>0011</td>
</tr>
</tbody>
</table>
Dynamic Warp Formation
(Fung: MICRO 2007, HPCA 2011)

Warp 0

A 1 2 3 4
B 1 2 3 4
C 1 2 -- --
D -- -- 3 4
E 1 2 3 4

Warp 1

A 5 6 7 8
B 5 6 7 8
C 5 -- 7 8
D -- 6 -- --
E 5 6 7 8

Warp 2

A 9 10 11 12
B 9 10 11 12
C 9 10 11 12
D 9 10 -- --
E 9 10 11 12

Time

Reissue/Memory Latency

SIMD Efficiency 58 → 88%

Pack

22% average [HPCA’11]
Memory
Scheduler affects access pattern

Round Robin Scheduler

Greedy then Oldest Scheduler

Warp Scheduler

Cache
Use scheduler to shape access pattern

Greedy then Oldest Scheduler

Warp\_0

Id A,B,C,D

Id A,B,C,D

Warp\_1

Id Z,Y,X,W

Warp Scheduler

Cache

Cache-Conscious Wavefront Scheduling (Rogers: MICRO 2012, Top Picks 2013)

W

X

Y

Z

W

X

Y

Z

D

C

B

A

working set size per warp

63% perf. improvement
Easier coding
Accelerator Coherence Challenges

- Challenges of introducing coherence messages on a GPU
  1. Traffic: transferring messages
  2. Storage: tracking message
  3. Complexity: managing races between messages

- GPU cache coherence without coherence messages?
  - YES – using global time
Temporal Coherence (Singh: HPCA 2013)
Related: Library Cache Coherence

Global Timestamp
< Global Time ⇒ NO L1 COPIES

Local Timestamp
> Global Time ⇒ VALID

Core 1
L1D
A=0

Core 2
L1D

Interconnect

L2 Bank
A=0
Temporal Coherence Example

No coherence messages
## Complexity

### Non-Coherent L1

<table>
<thead>
<tr>
<th>Load</th>
<th>L1 WTthru</th>
<th>L1 Atomic</th>
<th>L1 Replacement</th>
<th>Data</th>
<th>WB+ Atomic</th>
<th>WB+ Atomic Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>h k</td>
<td>p c</td>
<td>g w d a k L1</td>
<td>g w</td>
<td>d a k L1</td>
<td></td>
</tr>
<tr>
<td>LS</td>
<td>p c</td>
<td>g w d a k L1</td>
<td></td>
<td>g w</td>
<td>d a k L1</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>p c</td>
<td>g w d a k L1</td>
<td></td>
<td>g w</td>
<td>d a k L1</td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>p c</td>
<td>g w d a k L1</td>
<td></td>
<td>g w</td>
<td>d a k L1</td>
<td></td>
</tr>
</tbody>
</table>

### Non-Coherent L2

<table>
<thead>
<tr>
<th>L1 GETS</th>
<th>WB Data</th>
<th>L2 Atomic</th>
<th>L2 Replacement</th>
<th>Mem Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>l b t i s j k l m</td>
<td>g h f d a m i j k l m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td>l b t i s j k l m</td>
<td>g h f d a m i j k l m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>l b t i s j k l m</td>
<td>g h f d a m i j k l m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
<tr>
<td>IMA</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
</tbody>
</table>

### TC-Weak L1

### TC-Weak L2

---

This section contains tables and diagrams related to the complexity and state transitions of various cache coherence protocols, including Non-Coherent L1, Non-Coherent L2, TC-Weak L1, and TC-Weak L2. The tables detail various states and transitions under different conditions, with columns for Load, L1 WTthru, L1 Atomic, L1 Replacement, Data, WB+ Atomic, and WB+ Atomic Done for L1, and similarly for L2 and Mem Data. The diagrams illustrate the state transitions and interactions between different cache states.
Interconnect Traffic

- Reduces traffic by 53% over MESI and 23% over GPU-VI
- Lower traffic than 16x-sized 32-way directory

Do not require coherence
Performance

- TC-Weak with simple predictor performs 85% better than disabling L1 caches
Fewer bugs
Lifetime of Accelerator Application Development

- Functionality
- Performance

Fine-Grained Locking

Transactionals Memory
Are TM and GPUs Incompatible?

GPU uarch very different from multicore CPU.

**KILO TM** [Fung MICRO’11, Top Picks’12]

- Hardware TM for GPUs
- Half performance of fine grained locking
- Chip area overhead of 0.5%
Hardware TM for GPUs
Challenge #1: SIMD Hardware

- On GPUs, scalar threads in a warp/wavefront execute in lockstep
KILO TM – Solution to Challenge #1: SIMD Hardware

- Transaction Abort
  - Like a Loop
  - Extend SIMT Stack

```asm
...  
TxBegin
LD r2,[B]
ADD r2,r2,2
ST r2,[A]
TxCommit
...
```
Hardware TM for GPUs
Challenge #2: Transaction Rollback

CPU Core

@ TX Abort

@ TX Entry

Register File

Checkpoint Register File

10s of Registers

GPU Core (SM)

Warp

Warp

Warp

Warp

Warp

Warp

Register File

32k Registers

2MB Total On-Chip Storage

Checkpoint?
KILO TM – Solution to Challenge #2: Transaction Rollback

- SW Register Checkpoint
  - Most TX: Reg overwritten first appearance (idempotent)
  - TX in Barnes Hut: Checkpoint 2 registers

```
TxBegin
LD r2, [B]
ADD r2, r2, 2
ST r2, [A]
TxCommit
```

Overwritten

Abort
Hardware TM for GPUs

Challenge #3: Conflict Detection

Existing HTMs use Cache Coherence Protocol
- Not Available on (current) GPUs
- No Private Data Cache per Thread

Signatures?
- 1024-bit / Thread
- 3.8MB / 30k Threads
KILO TM: Value-Based Conflict Detection

- Self-Validation + Abort:
  - Only detects existence of conflict (not identity)
Easier debugging
```c
__global__ void BFS_step_kernel(...) {
    if( active[tid] ) {
        active[tid] = false;
        visited[tid] = true;
        foreach(int id = neighbour_nodes){
            if( visited[id] == false ){
                level[id] = level[tid] + 1;
                active[id] = true;
            }
        }
    }
}
```

### BFS algorithm
Published in HiPC 2007

---

**Result Variation (Kepler)**

- **# edges**
- **different results over multiple executions**

- level = 1
  - active = 1
- level = 2
  - active = 1
GPUDet (Jooybar: ASPLOS 2013)

1. Instruction Count
2. Atomic Operations
3. Memory Fences
4. Workgroup Barriers
5. Execution Complete

2x Slowdown

Global Memory

Read Only

Wavefront

Store Buffers

Local Memory

Parallel Mode

Commit Mode

Serial Mode

Core0

W0
W1
W2

Core1

W0
W1
W2

time

Quantumₙ

Commit

Atomic Op
Summary

- Start from efficient architecture and try to improve programmability
  - Get efficiency and keep programmers happy
    - reasonably
Thanks!
Questions?