Simultaneous Branch and Warp Interweaving for Sustained GPU Performance

Nicolas Brunie	Sylvain Collange	Gregory Diamos

by
Ravi Godavarthi
Outline

• Introduction
  • ISCA'39 (*International Society for Computers and Their Applications*) Portland, June 11, 2012
  • History & Motivation
• GPU architecture
• Simultaneous branch interweaving
• Simultaneous warp interweaving
• Results
• Conclusion
Introduction

• Nicolas Brunie
  • Currently developer in the project FloPoCo(Floating-Point Cores)
  • Project Developer in CMS (content management system)

• Sylvain Collange
  • Research Scientist in the ALF project-team at Inria in Rennes, France

• Gregory Diamos
  • Research Scientist currently employed by Nvidia
Introduction - motivation

- GPUs group threads into warps to run them in lockstep.
- Applications having irregular memory access under utilize GPU
- Goal = utilize the wasted simd units without effecting regular GPU applications
- Claim: improves performance by 23% on a set of regular GPGPU applications and by 40% on irregular applications.

<table>
<thead>
<tr>
<th>Schedulers</th>
<th>ALUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2:6</td>
<td>6 6 6 6</td>
</tr>
<tr>
<td>W1:6</td>
<td>6 6 6 6</td>
</tr>
<tr>
<td>W2:5</td>
<td>5 5 5 5</td>
</tr>
<tr>
<td>W1:5</td>
<td>4 4 4 4</td>
</tr>
<tr>
<td>W2:4</td>
<td>3 3 3 3</td>
</tr>
<tr>
<td>W1:4</td>
<td>5 5 5 5</td>
</tr>
<tr>
<td>W2:3</td>
<td>2 2 2 2</td>
</tr>
<tr>
<td>W1:3</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>W2:2</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>W1:2</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>W1:1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primary scheduler</th>
<th>ALUs</th>
<th>Secondary scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2:6</td>
<td>6 6 6 6</td>
<td>-</td>
</tr>
<tr>
<td>W1:6</td>
<td>6 6 6 6</td>
<td>-</td>
</tr>
<tr>
<td>W1:4</td>
<td>4 4 4 4</td>
<td>W2:4</td>
</tr>
<tr>
<td>W2:3</td>
<td>3 3 3 3</td>
<td>W2:3</td>
</tr>
<tr>
<td>W2:2</td>
<td>5 2 5 2</td>
<td>W2:5</td>
</tr>
<tr>
<td>W1:1</td>
<td>1 1 1 1</td>
<td>W1:5</td>
</tr>
<tr>
<td>W2:1</td>
<td>1 1 1 1</td>
<td>-</td>
</tr>
<tr>
<td>W1:1</td>
<td>1 1 1 1</td>
<td>-</td>
</tr>
</tbody>
</table>
GPU architecture — Multi-thread SPMD execution

• SIMD execution Model
  • Fetch 1 instruction for a *warp* of lockstepping threads.
  • Execute them in lock-step on SIMD units.
  • Optimized for regular workloads.

![Diagram of GPU architecture](image-url)
Control Divergence

- Loss of efficiency
  - 2 simd units are not utilized
  - current SIMT architectures execute each branch sequentially
  - Have to run T1 & T3 in different cycle causing extra power usage

```cpp
1: if(!tid%2) {
  2:    a+b;
  3: else {
  4:      a*b;
  5:    }
}
```
Baseline architecture

- Warps are split into two warp pools based on even or odd Identifier.
- Each pool has independent scheduling resources
- Each cycle – one ready instruction per pool is fetched
- Dependencies are tracked using a scoreboard mechanism.
Simultaneous Branch Interweaving

- Double warp size than baseline architecture
- Add a second fetch unit

Figure 3: Simultaneous Branch Interweaving micro-architecture.

```
1: if(!tid%2) {
  2:   a+b;
  3:   else {
  4:     a*b;  5: }
```

```
T0
PC= 2
Fetch @ 2
mul
mul
Warp

T1
PC= 4
Fetch @ 4
add
mul
mul

T2
PC= 2

T3
PC= 4
```
Re-convergence Mechanism

• Standard way is Stack based reconvergence
  • Each warp has a mask with bits set for threads ready to run an instruction
  • Runs branches sequentially

• Thread Frontier reconvergence
  • By default runs branches sequentially but can have constraints
  • for parallelism
  • Policy : CPC = min(PC)
  • Earliest reconvergence with code laid out in Thread Frontiers order
  • For two branches CPC1= min{PC} & CPC2 = min{PC, PC ≠ MPC1}
Reconvergence Mechanism

• Issues with Greedy Scheduling
  i) letting threads run ahead may increase memory-level parallelism and allow data prefetching
  ii) more instructions are issued, increasing power consumption
  iii) opportunities of memory coalescing may be missed
  iv) warp-splits may conflict for memory resources
Reconvergence Mechanism

Control-flow graph

Greedy scheduling

Instruction 6, 7 broken down, issued twice

Earliest reconvergence

Synchronize before instruction 6
Enforcing Reconvergence

- Between `Pcdiv` & `Pcrec`, wait for further diverging threads
- Keep pointer to immediate dominator at convergence points.

T0 and T2 (at F) wait for T1 (in D).

T3 (in B) can proceed in parallel.
Implementation

• sorted heap based implementations to store warp splits
• Each warp split context is a tuple (CPC, m, v)
  • Where m – activity mask & v – valid bit
• Keep Common PCs + activity masks in sorted heap
• HCT register has top two Context entries in it.
• Other entries are in CCT as Linked List (in incremental order or branching)

(a) General architecture
(b) HCT sorter
Simultaneous Warp Interweaving

• SBI limitations
  • No benefit with unbalanced thread workloads (eg: only if’s blocks & no else)
• SWI is to combine threads of different warps where all Tid’s are different.
  • i.e. Predicate mask of both warps are non over lapping.
  • Eg: 1 0 1 0 & 0 0 0 1 ; SWI = 1 0 1 1
Simultaneous Warp Interweaving

• Warp Subdivision – future work, currently resulting in performance loss
  • Warp subdivision is when no warp fits with primary warp (mask), a unfitting warp is subdivided to fit in the primary warp so as to increase throughput

• Unbalanced divergence introduce conflicts
  • Types: Under-occupancy, reduction, Triangular Domain
  • Eg:

```
  warp 0 1 2 3  warp 1 0 1 2 3  warp 2 0 1 2 3  warp 3 0 1 2 3

Warp 0 is never compatible with warp 2:
```
Simultaneous Warp Interweaving

• Solution is Lane Shuffling!
  • Apply thread to lane mapping permutation for each warp
  • Inter-thread memory locality Is preserved by mapping functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>tid</td>
<td></td>
</tr>
<tr>
<td>MirrorOdd</td>
<td>n – tid if wid odd, tid otherwise</td>
<td></td>
</tr>
<tr>
<td>MirrorHalf</td>
<td>n – tid if wid &gt; m/2, tid otherwise</td>
<td></td>
</tr>
<tr>
<td>Xor</td>
<td>tid ⊕ wid</td>
<td></td>
</tr>
<tr>
<td>XorRev</td>
<td>tid ⊕ bitrev(wid)</td>
<td></td>
</tr>
</tbody>
</table>

• Table 1: Lane shuffle functions. The physical lane id is computed from the thread-in-warp ID tid and warp ID wid. ⊕ is the XOR operator and bitrev is the bit-reversal function. The diagrams on the right illustrate the effect on 4 warps of 4 threads each by plotting the lane ID as a function of $4 \times \text{wid} + \text{tid}$
Simultaneous Warp Interweaving

• **Limited Associativity**: Finding an instruction whose mask is a subset of free lane mask

• **Achieved using CAM**
  • Bit-inclusion test
  • Set-associative Lookup

• **Bit Inclusion Test**:
  • Takes lot of power for computation

• **Set-associative Lookup**:
  • Warps are divided into sets for lookup
  • Power efficient
Simultaneous Warp Interweaving

Bit Inclusion Test

Set-associative Lookup
Results

Figure 2: Comparison of the contents of the execution pipeline using classic SIMT, Simultaneous Branch Interleaving with optional constraints, Simultaneous Warp Interleaving, and both.
Results

Speedup of 15% - regular 41% - irregular

Baseline  SBI  SWI

SBI+SWI  Warp 64

Regular applications

Irregular applications
Results

Figure 9: Slowdown of SWI lookup set-associativity compared to fully-associative lookup.
Simulation Platform

• Barra: functional GPU simulator modeled after NVIDIA Tesla GPUs
• Timing-power model
Advantages & Disadvantages

• Full dynamic scheduling and require minimal compiler involvement

• set-associative mask lookup and warp affinity using lane shuffling

• SBI works best on irregular workloads, regular workloads benefit most from SWI

• Overheads of SBI, SWI and both are 3.0%, 2.9% and 3.7% area requirement for overhead hardware

• Reconvergence policy and constraints proposed may be applied to both DWF and DWS

• Flexibility may be improved further by allowing more decoupling between lanes, without compromising efficiency
Conclusion

• The paper was very descriptive about & clear about their goals.
• They followed up with clear diagrams & tables to explain their ideas.
• They’ve mentioned how it is different from other warp scheduling mechanisms like DWF.

• This paper is aimed towards improving throughput of irregular GPGPU applications & the authors say it may or may not increase for regular workloads.
References

• G. Diamos et al. *SIMD re-convergence at thread frontiers*. MICRO 44, 2011.
• W. Fung et al. *Thread block compaction for efficient SIMT control flow*. 