Matching with Non-Uniformity: A Key to Efficiency on Multicore and GPU

Xipeng Shen

Eddy Z Zhang, Yunlian Jiang, Ziyu Guo, Kai Tian

Computer Science Department
The College of William and Mary
Appearance of Non-Uniformity

- non-uniform execution model
- non-uniform communication
- non-uniform resource sharing

Matching with the non-uniformity is key to computing efficiency.
GPU

a SIMD group (warp)

Regularity Within a Warp is Key to GPU Efficiency
Dynamic Irregularities

Degrade throughput by up to \((\text{warp size} - 1)\) times.

\((\text{warp size} = 32 \text{ in modern GPUs})\)
Performance Impact

- Applications: dynamic programming, fluid simulation, image reconstruction, data mining, ...

Potential Speedup

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMMER</td>
<td>5.27</td>
</tr>
<tr>
<td>3D-LBM</td>
<td>1.46</td>
</tr>
<tr>
<td>CUDA-EC</td>
<td>1.5</td>
</tr>
<tr>
<td>NN</td>
<td>2.51</td>
</tr>
<tr>
<td>CFD</td>
<td>2.75</td>
</tr>
<tr>
<td>CG</td>
<td>1.8</td>
</tr>
<tr>
<td>Unwrap</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Host: Xeon 5540.
Device: Tesla 1060.
Previous Work on GPU

• Mostly on Static Memory Irregularities
  ❖ [Baskaran+, ICS’08], [Lee+, PPoPP’09],[Yang+, PLDI’10], etc.

• Dynamic Irregularity
  ❖ Remain unknown until runtime
  ❖ Mostly through hardware extensions.
    ❖ [Meng+, ISCA’10], [Tarjan+, SC’09], [Fung+, MICRO’07], etc.
Open Questions for Dyn. Irreg. Removal

• Is it possible \textit{w/o hardware extentions}?

• More fundamentally
  • Relations among data, threads, \& irregularities?
  • What layout or thread-data mappings minimize the irreg.? How to find them? Complexity?
  • How to resolve conflicts among irregularities? Dependences?
Streamlining On the Fly

- No profiling or hw ext.
- Transparent removal on the fly
- Jeopardize no basic efficiency
- Treat both types of irreg. holistically

G-Streamline

[Zhang+: ASPLOS’11]

[Ding+: PLDI’99, Chilimbi+: PLDI’06, Tseng+: TPDS’06, etc.]
- NP-Complete
  - Layout: 3D matching
  - Mapping: Partition Problem
- Approx.
  - Duplication
  - Clustered sorting

How to determine optimal layouts / thread-data mapping?

See Zhang+:ASPLOS’11 for details.
After Transformation

- Benchmark Suites: Rodinia, Tesla Bio, and etc.
- Host: Xeon 5540. Device: Tesla 1060

How to minimize or hide overhead?
- CPU-GPU pipelining
- Kernel splitting
- Partial transf. and overlap.
- Two-level adaptive control

- Transparent, on-the-fly
- No perf. degradation
- Automatically balance benefits and overhead

- Adaptive to pattern changes
- Resilient to dependence

G-Streamline

overhead hiding & minimization

Efficiency control
Adaptive CPU-GPU pipelining
Three-level efficiency-driven adaptation
CPU-GPU Pipelining

- Utilize Idle CPU Time
  - Transform on CPU while computing on GPU
  - Automatic shutdown when necessary

```matlab
for i=1:n
    async_transform (i+2);
    async_copy (i+2);
    gpu_kernel(i);
end
```

cpu_transform()
copy_to_gpu
gpu_kernel
Kernel Splitting

- Also useful for loops with no dependences
  - Enable partial transformation
Efficiency control

- CPU-GPU pipelining
- Kernel splitting
- Partial transf. and overlap.
- Two-level adaptive control

- Transparent, on-the-fly
- Adaptive to pattern changes
- No perf. degradation
- Resilient to dependence
- Automatically balance benefits and overhead

See Zhang+:ASPLOS’11 for details.
Final Speedup

- HMMER: 2.5 (Basic transformation), 2.5 (w/ efficiency control), 5.27 (full potential)
- 3D-LBM: 1.4 (Basic transformation), 1.46 (w/ efficiency control), 1.8 (full potential)
- CUDA-EC: 1.1 (Basic transformation), 1.2 (w/ efficiency control), 1.5 (full potential)
- NN: 0.45 (Basic transformation), 0.38 (w/ efficiency control), 1.08 (full potential)
- CFD: 0.3 (Basic transformation), 0.3 (w/ efficiency control), 2.75 (full potential)
- CG: 0.27 (Basic transformation), 0.27 (w/ efficiency control), 1.8 (full potential)
- Unwrap: 0.27 (Basic transformation), 0.27 (w/ efficiency control), 3.6 (full potential)
Non-Uniform Cache Sharing on Multicore

- Cache sharing is a double-edged sword
  - Reduces communication latency
  - But causes conflicts & contention
Optimal Job Co-Scheduling

[Jiang+: PACT’08, Jiang+: TPDS’11]

- Minimum-weight perfect matching problem & a $O(N^4)$ solution.
- NP-completeness for K-core ($K>2$) & some approximation algorithms.
On Threads of a Multithreading Application

[Zhang+:PPoPP’10]

• Observation
  • Insignificant effects from shared cache (PARSEC)

• Reasons
  • Three mismatches with non-uniform shared cache
    • Limited data sharing among threads
    • Large working set size
    • Uniform inter-thread relations
Cache-Sharing-Aware Transformations

[Zhang+:PPoPP’10]

- Increase data sharing among siblings
- Decrease data sharing otherwise

Non-uniform data sharing

Non-uniform cache sharing

- >50% cache miss reduction.
- 5-33% speedup.
Summary

Matching with non-uniformity is the key.

Job co-sch.

Exploiting job non-uniformity

Cache-sharing-aware transformation

Creating thread non-uniformity

G-Streamline

Eliminating warp non-uniformity

Shared cache on multicore

Execution model on GPU
Thanks!