A Characterization and Analysis of PTX Kernels

Andrew Kerr*, Gregory Diamos, and Sudhakar Yalamanchili

School of Electrical and Computer Engineering
Georgia Institute of Technology

October 5, 2009

IEEE International Symposium on Workload Characterization 2009
Introduction

Workload Characterization Goals
NVIDIA’s Parallel Thread Execution (PTX) ISA
  • CUDA Programming Language
Ocelot Infrastructure
Application Workloads
Metrics and Workload Characteristics
Summary
Workload Characterization Goals

Understand

- Control flow behavior of SIMD kernels
- Memory demand
- Available parallelism within and across SIMD kernels

To provide insights for

- Compiler optimizations
- Application restructuring
- Architectural optimizations
- Dynamic optimizations
Parallel Thread Execution (PTX) Model

PTX Thread Hierarchy
- Grid of cooperative thread arrays
  - Coarse-grain parallelism

- Cooperative Thread Array
  - Fine-grain parallelism

PTX Virtual ISA
- RISC Instruction Set
- Defined by NVIDIA - target of CUDA compiler

Multiprocessor Architecture
- Multiprocessor
  - register file
  - shared memory
  - param memory
  - const memory
  - texture memory
- n-way SIMD
- local memory
- Global Memory
## CUDA SDK: Basic Characteristics

<table>
<thead>
<tr>
<th>Applications</th>
<th>Kernels</th>
<th>CTA Size</th>
<th>Average CTAs</th>
<th>Instructions</th>
<th>Branches</th>
<th>Branch Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bicubic Texture</td>
<td>27</td>
<td>256</td>
<td>1024</td>
<td>222,208</td>
<td>5120</td>
<td>3</td>
</tr>
<tr>
<td>Binomial Options</td>
<td>1</td>
<td>256</td>
<td>4</td>
<td>725,280</td>
<td>68,160</td>
<td>8</td>
</tr>
<tr>
<td>Black-Scholes Options</td>
<td>1</td>
<td>128</td>
<td>480</td>
<td>3,735,550</td>
<td>94230</td>
<td>4</td>
</tr>
<tr>
<td>Box Filter</td>
<td>3</td>
<td>32</td>
<td>16</td>
<td>1,273,808</td>
<td>17,568</td>
<td>4</td>
</tr>
<tr>
<td>DCT</td>
<td>9</td>
<td>70.01</td>
<td>2,446</td>
<td>1,898,752</td>
<td>25,600</td>
<td>3</td>
</tr>
<tr>
<td>Haar wavelets</td>
<td>2</td>
<td>479.99</td>
<td>2.5</td>
<td>1,912</td>
<td>84</td>
<td>5</td>
</tr>
<tr>
<td>DXT Compression</td>
<td>1</td>
<td>64</td>
<td>64</td>
<td>673,676</td>
<td>28,800</td>
<td>8</td>
</tr>
<tr>
<td>Eigen Values</td>
<td>3</td>
<td>256</td>
<td>4.33</td>
<td>9,163,154</td>
<td>834,084</td>
<td>13</td>
</tr>
<tr>
<td>Fast Walsh Transform</td>
<td>11</td>
<td>389.94</td>
<td>36.8</td>
<td>32,752</td>
<td>1216</td>
<td>4</td>
</tr>
<tr>
<td>Fluids</td>
<td>4</td>
<td>36.79</td>
<td>32.6</td>
<td>151,654</td>
<td>3,380</td>
<td>5</td>
</tr>
<tr>
<td>Image Denoising</td>
<td>8</td>
<td>64</td>
<td>25</td>
<td>4,632,200</td>
<td>149,400</td>
<td>6</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>2</td>
<td>256</td>
<td>40</td>
<td>6,136,566</td>
<td>614,210</td>
<td>26</td>
</tr>
<tr>
<td>Mersenne twister</td>
<td>2</td>
<td>128</td>
<td>32</td>
<td>1,552,704</td>
<td>47,072</td>
<td>7</td>
</tr>
<tr>
<td>Monte Carlo Options</td>
<td>2</td>
<td>243.54</td>
<td>96</td>
<td>1,173,898</td>
<td>76,512</td>
<td>8</td>
</tr>
<tr>
<td>Threaded Monte Carlo</td>
<td>4</td>
<td>243.54</td>
<td>96</td>
<td>1,173,898</td>
<td>76,512</td>
<td>8</td>
</tr>
<tr>
<td>Nbody</td>
<td>2</td>
<td>256</td>
<td>4</td>
<td>82,784</td>
<td>1,064</td>
<td>5</td>
</tr>
<tr>
<td>Ocean</td>
<td>4</td>
<td>64</td>
<td>488.25</td>
<td>390,786</td>
<td>17,061</td>
<td>7</td>
</tr>
<tr>
<td>Particles</td>
<td>16</td>
<td>86.79</td>
<td>29.75</td>
<td>277,234</td>
<td>26,832</td>
<td>16</td>
</tr>
<tr>
<td>Quasirandom</td>
<td>2</td>
<td>278.11</td>
<td>128</td>
<td>3,219,609</td>
<td>391,637</td>
<td>8</td>
</tr>
<tr>
<td>Recursive Gaussian</td>
<td>2</td>
<td>78.18</td>
<td>516</td>
<td>3,436,672</td>
<td>41,088</td>
<td>8</td>
</tr>
<tr>
<td>Sobel Filter</td>
<td>12</td>
<td>153.68</td>
<td>426.66</td>
<td>2,157,884</td>
<td>101,140</td>
<td>6</td>
</tr>
<tr>
<td>Volume Render</td>
<td>1</td>
<td>256</td>
<td>1,024</td>
<td>2,874,424</td>
<td>139,061</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table:** CUDA SDK Application Statistics
### Applications: Basic Characteristics

#### Benchmarks

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Kernels</th>
<th>Average CTA Size</th>
<th>Average CTAs</th>
<th>Instructions</th>
<th>Branches</th>
<th>Branch Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td>10</td>
<td>128</td>
<td>256</td>
<td>430,261,760</td>
<td>10,245,120</td>
<td>3</td>
</tr>
<tr>
<td>MRI-FHD</td>
<td>7</td>
<td>256</td>
<td>110.571</td>
<td>9,272,268</td>
<td>198,150</td>
<td>5</td>
</tr>
<tr>
<td>MRI-Q</td>
<td>4</td>
<td>256</td>
<td>97.5</td>
<td>7,289,604</td>
<td>393,990</td>
<td>5</td>
</tr>
<tr>
<td>PNS</td>
<td>112</td>
<td>256</td>
<td>17.85</td>
<td>683,056,349</td>
<td>33,253,961</td>
<td>11</td>
</tr>
<tr>
<td>RPES</td>
<td>71</td>
<td>64</td>
<td>64,768.7</td>
<td>1,395,694,886</td>
<td>95,217,761</td>
<td>13</td>
</tr>
<tr>
<td>SAD</td>
<td>3</td>
<td>61.42</td>
<td>594</td>
<td>4,690,521</td>
<td>87,813</td>
<td>7</td>
</tr>
<tr>
<td>TPACF</td>
<td>1</td>
<td>256</td>
<td>201</td>
<td>1,582,900,869</td>
<td>230,942,677</td>
<td>18</td>
</tr>
</tbody>
</table>

#### Table: Parboil Application Statistics

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Kernels</th>
<th>Average CTA Size</th>
<th>Average CTAs</th>
<th>Instructions</th>
<th>Branches</th>
<th>Branch Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK</td>
<td>145</td>
<td>217.64</td>
<td>457.25</td>
<td>55,884,066</td>
<td>3,504,904</td>
<td>26</td>
</tr>
<tr>
<td>RIAA</td>
<td>10</td>
<td>64</td>
<td>16</td>
<td>322,952,484</td>
<td>23,413,125</td>
<td>16</td>
</tr>
<tr>
<td>RDM</td>
<td>2237</td>
<td>174.558</td>
<td>63.0595</td>
<td>46,448,530</td>
<td>4,082,425</td>
<td>6</td>
</tr>
<tr>
<td>Parboil</td>
<td>208</td>
<td>177.238</td>
<td>9,435.09</td>
<td>4,113,166,257</td>
<td>370,339,472</td>
<td>11</td>
</tr>
</tbody>
</table>

#### Table: Aggregate Workload Statistics
Metrics

Control flow
- Branch Divergence
- Activity Factor

Global memory and data flow
- Memory Intensity
- Memory Efficiency
- Interthread Data Flow

Parallelism
- MIMD Parallelism
- SIMD Parallelism
Ocelot serializes execution of CTAs

- Each instruction executed for active threads
- Warp size is equal to CTA size
- Divergent control flow splits active context

Metrics averaged over all dynamic instructions for all kernels in an application

- PC
- Activity mask
- Memory references
Branch Divergence

Fraction of branches that are divergent

\[ BD = \frac{\# \text{divergent branches}}{\# \text{branches}} \]

Computed on dynamic instruction stream
Post Dominator versus Barrier Reconvergence

Pseudocode

```
barrier;
s0;
if ( cond_0 ) {
    s1;
    if ( cond_1 ) {
        s2;
    } else {
        s3;
    }
} else {
    s4;
}
s5;
barrier;
s6;
barrier;
s7;
```

Branch Divergence Results

- Branches correlated (in time within the same thread) result in differences in ideal-vs-barrier reconvergence
- Frequent handling of special cases results in high overall divergent control flow
- Recommendation:
  - Correlation of branches suggests restructuring of threads to reduce divergence
  - If warp split costs are high, use barrier synchronization reconvergence method
Activity Factor

Average number of active SIMD ways

Activity Factor

$$AF = \frac{1}{N} \sum_{i=1}^{N} \frac{active(i)}{CTA(i)}$$

- $active(i)$: active threads executing dyn. instruction $i$
- $CTA(i)$: threads in CTA executing $i$
- $N$: number of dynamic instructions
**Recommendation:**

- Compiler use of predication to reduce control flow for short divergent paths
- Placement of bar.sync earlier to increase AF
- Hardware support for p-dom reconvergence
Memory Intensity

Fraction of loads or stores to global memory per dynamic instruction

\[ I_M = \times \frac{\sum_{i=1}^{\text{kernels}} A_f M_i}{\sum_{i=1}^{\text{kernels}} D_i} \]

\( A_f \): activity factor
\( M_i \): global memory instructions
\( D_i \): dynamic instructions
Texture samples counted as global memory accesses
Memory Intensity Results

- CUDA SDK, RDM, Parboil have low average memory intensities (3.5%)
  - Efficient applications strive to be compute bound
  - Statistic ignores shared and local memory operations
  - Memory intensity not same as bandwidth

- RIAA application has relatively high memory intensity
  - Consequence of application: large hash table, pointer chasing
Memory Efficiency

Coalesced gather - 1 transaction

Uncoalesced scatter - 4 serialized transactions

Average number of transactions needed to satisfy a load or store to global memory

\[ E_M = \sum_{i=1}^{kernels} \sum_{j=1}^{CTAs} \frac{2W_{i,j}}{T_{i,j}} \]

\( W_{i,j} \): warps issuing memory instructions
\( T_{i,j} \): transactions required
Recommendation:

- Opportunity for compiler, hardware, runtime to trade off Activity Factor and Memory Efficiency
Interthread Data Flow

Cooperative Thread Array

Thread: 0 1 2 3

<table>
<thead>
<tr>
<th>bar.sync</th>
<th>ld.global</th>
<th>st.shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar.sync</td>
<td>ld.shared</td>
<td>mad.f32</td>
</tr>
<tr>
<td>bar.sync</td>
<td>st.shared</td>
<td></td>
</tr>
</tbody>
</table>

Output: 0 1 2 3

<table>
<thead>
<tr>
<th>mad.f32</th>
<th>st.global</th>
</tr>
</thead>
</table>

Intensity of producer-consumer relationships within a CTA

- Ignore st.shared if value to store was loaded from global memory.
- Otherwise, st.shared annotates words in shared memory with writer’s thread ID.
- ld.shared compares thread ID with annotated thread ID.
- Count number of ld.shared with annotation ≠ thread ID.

Interthread Data Flow

\[
IDF = \frac{X_i}{S_i}
\]

- \(X_i\): words loaded by inter-thread ld.shared
- \(S_i\): ld.shared instructions
Interthread Data Flow Results

- Shared memory used as: a cache, and as producer-consumer conduit
- Data dependencies between threads inform scheduling decisions and thread placement
- Recommendation:
  - Improve efficiency of data sharing among threads
  - Support smaller synchronization domains
Parallelism Scaling

Average speedup of MIMD/SIMD machine with infinite parallelism

**MIMD Parallelism**

\[
MIMD_{\text{kernel}} = \frac{\sum_{i=1}^{\text{ctas}} D_i}{\max_{i=1}^{\text{ctas}}(D_i)}
\]

\[
MIMD_{\text{application}} = \frac{\sum_{i=1}^{\text{kernels}} D_i \times MIMD_{\text{kernel},i}}{\sum_{i=1}^{\text{kernels}} D_i}
\]

- \(D_i\): dynamic instructions
- \(A_f\): activity factor

**SIMD Parallelism**

\[
SIMD_{\text{kernel}} = \frac{\sum_{i=1}^{\text{ctas}} A_f \times D_i}{\sum_{i=1}^{\text{ctas}} D_i}
\]

\[
SIMD_{\text{application}} = \frac{\sum_{i=1}^{\text{kernels}} D_i \times SIMD_{\text{kernel},i}}{\sum_{i=1}^{\text{kernels}} D_i}
\]

- \(D_i\): dynamic instructions
- \(A_f\): activity factor
Parallelism Results

* semi-log plot warning

Applications should express as much possible parallelism to enable performance scaling

Recommendation:

- Efficiently mapping parallel code to collections of serial processors is crucial
- Overheads: redundancy, context switching, locality of memory references
GPGPU-Sim
- Derived from SimpleScalar to support GPU constructs
- Extended to include PTX as an instruction set
- Assesses impact of architectural parameters

Barra
- Virtual machine for SASS - native GPU instruction set
- Captures calls to CUDA driver API
- Results are detailed but specific to particular architecture implementation
Future Work

PTX to PTX
- Ocelot’s PTX internal representation to produce executable PTX kernels
- Optimizations and transformations

PTX to LLVM to Multicore
- Translate PTX to Low-Level Virtual Machine
- Leverage existing optimization passes and code generators
- Target many existing multicore ISAs
Characteristics of PTX applications motivate compiler optimizations, adaptive runtimes, architectural support.

- Thread restructuring reduces divergent control flow.
- Reconvergence methods tradeoff warp splitting with activity factor.
- Balance activity factor with memory efficiency.
- Data dependencies among threads suggest smaller synchronization domains.
- Data parallel kernels must be serialized efficiently.

Ocelot provides a unique approach to observing characteristics independently of particular architectures.
Acknowledgements

The authors gratefully acknowledge the generous support of this work by LogicBlox Inc., IBM Corp., and NVIDIA Corp. both through research grants, fellowships, as well as technical interactions, and equipment grants from Intel Corp. and NVIDIA Corp.