An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness

Sunpyo Hong, Hyesoon Kim
Outline

- Background
- Model
- Results
- Conclusion
Overview of GPU Architecture

- Software-managed cache
- SIMD Execution Unit inside SM
Warp

- **Warp is the basic unit of execution**
  - A group of threads (e.g. 32 threads for the Tesla GPU architecture)

**Warp Execution**

- Programmer specifies the # of threads
- Sources ready

Finite number of streaming processors

**SIMD Execution Unit**
Occupancy

- Shows how many warps are assigned to the SM
- Warps are assigned at block granularity
- Programmer specifies the number of threads per block

100% Occupancy

Only one block is allocated
Higher Occupancy

- Better processor utilization
- Hide the memory latency

Processor is not utilized

Warp 1

Warp 2

Warp 3

Warp 4

Warp 5

Better utilization!
High Occupancy = High Performance?

- Programmers try to optimize programs for occupancy

![Graph showing increase in occupancy and performance improvement](image)
High Occupancy ≠ High Performance

- Programmers try to optimize programs for occupancy
- No performance improvement from increased occupancy
Motivation of the Work

- Propose analytical model that can estimate performance

- Why?
  - Optimizing for occupancy may not have impact on the performance
  - Occupancy does not consider the application behavior
  - To understand the GPU performance and bottlenecks

- Other benefits
  - Prediction for faster performance simulation
Outline

- Background
- Model
- Results
- Conclusion
How is Performance Determined?

- Memory accesses can be overlapped between warps
  - Performance *significantly* depends on the memory-level parallelism

No Parallelism

Infinite Parallelism

Finite Parallelism (Two)

- Performance can be *predicted* by knowing the amount of memory-level parallelism

Saved cycles

Additional Cycles
MWP

- Memory Warp Parallelism

- Metric of memory-level parallelism

  Four warps are overlapped during memory accesses

- Maximum number of warps that can overlap memory accesses

- Tightly coupled with DRAM system
  - Memory latency, bandwidth, memory access type
Memory Access Type

One warp generates a memory request

Coalesced memory access type

One memory transaction

Uncoalesced memory access type

- More processing cycles for the uncoalesced case
Each SM has a simple queue and consumes an equal bandwidth.

MWP is determined by #Active SMs, #Active warps, Bandwidth, Types of memory accesses (Coalesced, Uncoalesced)
CWP

- Computation Warp Parallelism

- Analogous concept to MWP

MWP = 2

CWP = 4

- Number of warps that execute instructions during one memory access period

- Three scenarios can occur depending on the MWP and CWP relationship
(1) When MWP ≤ CWP

MWP=2, N = 8 (Number of warps)

- Computation cycles are hidden by memory waiting periods
- Overall performance is dominated by the memory cycles

\[
\text{Exec\_cycles} = \text{Mem\_cycles} \times \frac{N}{MWP} + \text{Comp\_p} \times MWP
\]  

\( (MWP=2, \ N = 8) \)
(2) When MWP > CWP

- Memory accesses are mostly hidden due to high MWP
- Overall performance is dominated by the computation cycles

\[ \text{Exec}_{\text{cycles}} = \text{Mem}_p + \text{Comp}_{\text{cycles}} \times N \]

(MWP=8,  N = 8)
(3) Not Enough Warps

- Increasing the number of warps will increase the processor utilization
- MWP is limited by the number of active warps per SM
- The analytical model is inside the paper
Outline

- Background
- Model
- Results
- Conclusion
Evaluation Methodology

- Micro benchmarks are devised to obtain the memory parameters
  - Memory latency, departure delay

- Model inputs
  - Number of instructions, memory type, thread/block configuration, memory parameters

- Merge benchmarks
  - Execution time, CPI compared

<table>
<thead>
<tr>
<th>Evaluated Systems</th>
<th>8800GTX</th>
<th>FX5600</th>
<th>8800GT</th>
<th>GTX280</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of SMs</td>
<td>16</td>
<td>16</td>
<td>14</td>
<td>30</td>
</tr>
<tr>
<td>(SP) Processor Cores</td>
<td>128</td>
<td>128</td>
<td>112</td>
<td>240</td>
</tr>
<tr>
<td>Processor Clock</td>
<td>1.35 GHz</td>
<td>1.35GHz</td>
<td>1.5 GHz</td>
<td>1.3 GHz</td>
</tr>
<tr>
<td>Memory Size</td>
<td>768 MB</td>
<td>1.5 GB</td>
<td>512 MB</td>
<td>1 GB</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>86.4 GB/s</td>
<td>76.8 GB/s</td>
<td>57.6 GB/s</td>
<td>141.7 GB/s</td>
</tr>
<tr>
<td>Computing Version</td>
<td>1</td>
<td>1</td>
<td>1.1</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Micro Benchmarks

- Ratio of memory to computation instructions is varied
- Coalesced, uncoalesced memory types

### Memory Model Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>FX5600</th>
<th>GTX280</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory latency</td>
<td>420</td>
<td>450</td>
</tr>
<tr>
<td>Departure delay uncoalesced</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>Departure delay coalesced</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Merge Benchmarks

- Merge benchmark performance estimation
- The prediction closely follows the actual execution
  - Two types of execution behavior are predicted
CPI Comparison

- CPI comparison between the model and the actual execution

Overall 13.3 % error
Outline

- Background
- Model
- Results
- Conclusion
Conclusions

- Introduced **MWP, CWP** metrics that determine the performance

- **Simplified** the complex memory operations

- **Prediction**
  - For Micro benchmarks, the prediction error is 5.4%
  - For Merge benchmarks, the prediction error is 13.3%

- First analytical model that calculates the *execution cycles* for GPU

- **Better** understanding of the performance aspects of the GPU architecture

- Future research
  - Help providing more systematic approaches for **optimizing** GPGPU applications
Thank you
Questions ?
Insights on MWP (Motivation Example)

Warps < MWP

Warps > MWP

No performance improvement
The model provides the upper limit of # of active warps for a given application that fully utilizes the processor resources.

- **Increasing** the # of warps when N is smaller than MWP, CWP

- **Trade-off**
  - More register allocation vs. More computation instructions

Calculated CPI value indicates how optimized the code is.

- CPI per warp near 4 is the upper-bound

- Traditionally, if the optimization on the thread decreases the occupancy, that optimization is unlikely to be performed.

  \[
  CPI = \frac{\text{Exec Cycles app}}{\text{Total insts} \times \text{#Threads per block} \times \text{#Blocks each SM}}
  \]

- However, if the model predicts that higher occupancy does not improve the performance, then that optimization can be applied without performance degradation.

We are currently developing more systematic approach by using the metrics that we proposed in this work.
Limitations of the Model

- **Cache misses**
  - Current analytical model does not consider cache miss penalties

- **Graphics Applications**
  - Not modeling texture cache, texture processing

- **Divergent branches**
  - Double counting the number of instructions in both path
  - Provides the upper limit for the execution time

- **Data transfer time between CPU and GPU**
  - The analytical work models the GPU kernel execution only

- **Considers total average execution time**
  - No time-phase behavior
How to use the model (I)

- Inputs to the model
  - Thread/block configuration
  - Register/shared memory usage
  - Number of Instructions
  - Memory access type

- Micro benchmarks
  - Exact number of instructions for different arithmetic intensity is known

- Merge benchmarks
  - Source and PTX (virtual ISA) analysis
  - Currently, GPU emulator is available
  - Dynamic number of PTX instructions is calculated

Programmer specifies in the source code
Available in the CUDA compiler output (.cubin file)
Source code analysis
PTX file (compiler output)
How to use the model (II)

- Inputs to the model
  - Thread/block configuration
  - Register/shared memory usage
  - Number of Instructions
  - Memory access type

Analyzing memory access pattern

- Analyze the memory access pattern
  - By using the index to the memory function
  - Devised the algorithms for determining the memory access type, and the number of memory transactions.

\[
CPI = \frac{\text{Exec}_{\text{cycles}}_{\text{app}}}{\text{Total}_{\text{insts}} \times \text{Threads}_{\text{per}}_{\text{block}} \times \text{Blocks}} \times \text{Threads}_{\text{per}}_{\text{warp}} \times \text{Active}_{\text{SMs}}
\]
Memory System

\[ MWP = \min(MWP_{\text{Without BW}}, MWP_{\text{peak BW}}, N) \]

- Broken down the memory system from high-level view
  - Maximum possible MWP without bandwidth consideration
  - MWP with bandwidth consideration (Considers #warps, #Transactions, ...)
  - Effects of active warps
  - Captured high-level concepts with careful interactions

Coalesced
- Transaction #1 (Warp1)
- Transaction #1 (Warp 2)
- Departure delay

Uncoalesced
- Transaction #1 (Warp1)
- Transaction #2 (Warp1)
- Departure delay

- Transaction #N (Warp1)
  - Transaction #1 (Warp 2)
  - Departure delay
Synchronization effects

- Barrier instruction causes extra waiting cycles
- Warps inside one SM are synchronized

- Extra cycles are calculated by knowing the value of MWP

No synchronization

Additional delay

Extra cycles are calculated by knowing the value of MWP