Moka: Model-based Concurrent Kernel Analysis

Leiming Yu, Xun Gong, Yifan Sun, Qianqian Fang, Norm Rubin and David Kaeli

Northeastern University, Boston, MA

IISWC, Oct 2017
Outline

Part 1: Motivation
Part 2: Concurrent Kernel Execution
Part 3: Moka
Part 4: Evaluation
Part 5: Summary
Part 1

Motivation
GPU Computing

Optimized for serial tasks

Optimized for parallel tasks
GPU Computing

Single Precision Performance

[Graph showing performance trends of different GPU models over years, with labels for NVIDIA, INTEL Xeon CPUs, NVIDIA Titan X, and INTEL Xeon Phis.]
GPU Computing

Top500 from June 2017 - 91 out of 500 (18%) systems are using GPUs
GPU Computing

- High FLOPS
- Thousands of computing cores on the chip
- Single Instruction Multiple Thread (SIMT)
- Faster memory, high bandwidth
- Special instructions (e.g., SHFL)
GPU Computing

Concurrent Kernel Execution (CKE) for better device utilization

- Fermi: 16 hardware supported queues
- Kepler and after: 32 hardware supported queues (Hyper-Q)
GPU Computing

What we would like…

What we actually get – argh!
Programming CKE

cudaMemcpyAsync(dev_ptr, host_ptr, size, cudaMemcpyHostToDevice, stream)

Kernel<<<gridSize, blockSize, sharedMemSize, stream>>> (dev_ptr)

cudaMemcpyAsync(host_ptr, dev_ptr, size, cudaMemcpyDeviceToHost, stream)

- Leverage pinned memory `cudaMallocHost`
- Utilize non-blocking CUDA API `cudaMemcpyAsync`
- Issue the data transfer to each stream, then return to the host immediately
Concurrency

Stream 1: H2D, Kern, Kern, D2H
Stream 2: H2D, Kern, Kern, D2H
Stream 3: H2D, Kern, Kern, D2H
Stream 4: H2D, Kern, Kern, D2H

CPU
Factors for Concurrency

**Number of copy engines**
- GPU with 1 copy engine
  - No overlap between data transfers
- GPUs with 2 copy engines
  - Simultaneous data transfer between two directions: D2H and H2D

**Coding Style**
- Interleave:
  - Execute all operations from one stream to the next
- Batch:
  - Dispatch the same operation for all streams, then move to the next
GPUs with two copy engines provide more opportunities for concurrency!
Challenges to tune CKE

- Block size matters
- Concurrent Kernel Execution could lead to performance degradation
- When to use CKE?
Part3  Model-based Concurrent Kernel Analysis (Moka)
Moka Overview

- User Apps
  - CUDA API Info
  - SASS Info
  - Kernel Configuration
  - Kernel Metrics
- GPU Info
- Block Size Tuning
  - Kernel Classification
  - Kernel Matching
- Multi-Stream Timing Trace
  - Select launch order
  - Predict runtime
  - Maximize performance
- Data Transfer Model
- Kernel Execution Model
- Resource Contention Model
Moka

Block Size Tuning

Data Transfer Model

Kernel Execution Model

Contention Model

Stream Launch
Block Size Tuning

Step 1: Classify kernels based on the warp characteristics

Memory SASS Instructions per warp (LD / ST )
Compute SASS Instructions per warp (FP32 / FP64 / INT)
Micro-benchmark SASS instruction cycles
## Block Size Tuning

### Step 2: Build a training set (CUDA SDK)

- Classify training set kernels
- Benchmark the best block size for individual kernel

#### Compute-intensive Kernels

<table>
<thead>
<tr>
<th>Kernel Dims</th>
<th>Kernel Name</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>fwbatch2kernel</td>
<td>fastWalshTransform</td>
</tr>
<tr>
<td></td>
<td>mergeranksandindiceskernel</td>
<td>mergeSort</td>
</tr>
<tr>
<td></td>
<td>mergesortsharedkernel</td>
<td>mergeSort</td>
</tr>
<tr>
<td></td>
<td>dwhaard1d</td>
<td>dwtHaar1D</td>
</tr>
<tr>
<td></td>
<td>blackscholesgpu</td>
<td>BlackScholes</td>
</tr>
<tr>
<td></td>
<td>bitonicisortshared</td>
<td>sortingNetworks</td>
</tr>
<tr>
<td></td>
<td>sproprocess2d_kernel</td>
<td>convolutionFFT2D</td>
</tr>
<tr>
<td></td>
<td>sproprocess2d_kernel</td>
<td>convolutionFFT2D</td>
</tr>
<tr>
<td></td>
<td>test_interval_newton</td>
<td>interval</td>
</tr>
<tr>
<td></td>
<td>computeangles_kernel</td>
<td>lineOfSight</td>
</tr>
<tr>
<td></td>
<td>binomialoptionskernel</td>
<td>binomialOptions</td>
</tr>
<tr>
<td></td>
<td>inverseCNDKernel</td>
<td>quasirandomGenerator</td>
</tr>
</tbody>
</table>

#### Memory-intensive Kernels

<table>
<thead>
<tr>
<th>Kernel Dims</th>
<th>Kernel Name</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>scalarProdGPU</td>
<td>scalarProd</td>
</tr>
<tr>
<td></td>
<td>reduce6</td>
<td>reduction</td>
</tr>
<tr>
<td></td>
<td>vectoradd</td>
<td>vectorAdd</td>
</tr>
<tr>
<td></td>
<td>modulateAndNormalize_kernel</td>
<td>convolutionFFT2D</td>
</tr>
<tr>
<td></td>
<td>modulatekernel</td>
<td>fastWalshTransform</td>
</tr>
<tr>
<td></td>
<td>generatesamplerankskernel</td>
<td>mergeSort</td>
</tr>
<tr>
<td></td>
<td>scanExclusiveShared2</td>
<td>scan</td>
</tr>
<tr>
<td></td>
<td>scanExclusiveShared</td>
<td>scan</td>
</tr>
<tr>
<td></td>
<td>uniformupdate</td>
<td>scan</td>
</tr>
<tr>
<td></td>
<td>mergeHistogram256Kernel</td>
<td>histogram</td>
</tr>
<tr>
<td></td>
<td>histogram64Kernel</td>
<td>histogram</td>
</tr>
<tr>
<td></td>
<td>mergeHistogram64Kernel</td>
<td>histogram</td>
</tr>
<tr>
<td></td>
<td>reduceMultiPass</td>
<td>threadFenceReduction</td>
</tr>
<tr>
<td></td>
<td>reduceSinglePass</td>
<td>threadFenceReduction</td>
</tr>
<tr>
<td></td>
<td>uniform_add</td>
<td>shfl_scan</td>
</tr>
<tr>
<td></td>
<td>computeVisibilities_kernel</td>
<td>lineOfSight</td>
</tr>
</tbody>
</table>

#### 2D Kernels

<table>
<thead>
<tr>
<th>Kernel Dims</th>
<th>Kernel Name</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>transposeCoarseGrained</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeNoBankConflicts</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeCoalesced</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeFineGrained</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeNaive</td>
<td>transpose</td>
</tr>
</tbody>
</table>

### 2D Kernels

<table>
<thead>
<tr>
<th>Kernel Dims</th>
<th>Kernel Name</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>transposeCoarseGrained</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeNoBankConflicts</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeCoalesced</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeFineGrained</td>
<td>transpose</td>
</tr>
<tr>
<td></td>
<td>transposeNaive</td>
<td>transpose</td>
</tr>
</tbody>
</table>
Block Size Tuning

Step 3: Recommend the block size from the most similar kernel

- t-SNE Analysis: dimension reduction and visualization
- Find the most similar kernel using the majority vote
- Recommend the block size to the incoming kernel

Performance Counters for t-SNE Analysis

<table>
<thead>
<tr>
<th>Memory-Intensive</th>
<th>Compute-intensive</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2m_ratio</td>
<td>gld_ratio</td>
</tr>
<tr>
<td>ipc</td>
<td>gst_ratio</td>
</tr>
<tr>
<td>achieved_occupancy</td>
<td>sld_ratio</td>
</tr>
<tr>
<td>issue_slot_utilization</td>
<td>sst_ratio</td>
</tr>
<tr>
<td>eligible_warps_per_cycle</td>
<td>shared_utilization</td>
</tr>
<tr>
<td></td>
<td>achieved_occupancy</td>
</tr>
<tr>
<td></td>
<td>issue_slot_utilization</td>
</tr>
<tr>
<td></td>
<td>eligible_warps_per_cycle</td>
</tr>
</tbody>
</table>

MNIST dataset

Two-dimensional embedding of 70,000 handwritten digits with t-SNE
Block Size Tuning

cuobjdump → SASS → Application

profiler → Trace → Metrics

Kernel Classification

Memory Intensive

Training Kernels

Top-K Kernels

Compute Intensive

Recommend Block Size
Data Transfer Model

- LogGP [Van Werkhoven, 2014]
- Linear Regression
- Share PCI-e bandwidth when the transfer is in the same direction
Moka

Block Size Tuning

Data Transfer Model

Kernel Execution Model

Contention Model

Stream Launch
Kernel Execution Model

- GPU kernels are formed of parallel thread block
- Blocks are scheduled in a round-robin fashion
- Device resources limit the max concurrent thread blocks per SM
  - Registers, Shared Memory, Max Threads per Streaming Multiprocessor

\[ \text{maxBlkPerSM} = \min\{\text{BlkLmt\_Reg}, \text{BlkLmt\_Shared}, \text{BlkLmt\_Thread}, \text{BlkLmt\_Dev}\} \]

- Average Block Execution Time, assuming each block has an identical life span

\[ \text{KernBlksPerIter} = \text{GPU\_SMs} \times \text{maxBlkPerSM} \]
\[ \text{Iters} = \text{KernelBlks}/\text{KernBlksPerIter} \]
\[ \text{AvgBlkTime} = \text{KernelTime}/\text{Iters} \]
Kernel Execution Model

- Based on the scheduling policy, the kernel runtime is the runtime of the most time-consuming SM

\[ SM\_Time(K_i) = BLK\_Start(K_i) - BLK\_End(K_i) \]

\[ Kernel\_Time(K_i) = MAX\{SM\_Time(K_i)\} \]

- Apply contention model to update kernel runtime for concurrent kernel execution
## Contention Model

- Resource contention

<table>
<thead>
<tr>
<th>Metrics Name</th>
<th>Resource Type</th>
<th>Metrics Name</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>eligible_warps_per_cycle</td>
<td>occupancy</td>
<td>dram_utilization</td>
<td>memory</td>
</tr>
<tr>
<td>cf_fu_utilization</td>
<td>function unit</td>
<td>tex_utilization</td>
<td>memory</td>
</tr>
<tr>
<td>tex_fu_utilization</td>
<td>function unit</td>
<td>shared_utilization</td>
<td>memory</td>
</tr>
<tr>
<td>ldst_fu_utilization</td>
<td>function unit</td>
<td>l2_utilization</td>
<td>memory</td>
</tr>
<tr>
<td>single_precision_fu_utilization</td>
<td>function unit</td>
<td>sysmem_utilization</td>
<td>memory</td>
</tr>
<tr>
<td>double_precision_fu_utilization</td>
<td>function unit</td>
<td>flop_sp_efficiency</td>
<td>compute</td>
</tr>
<tr>
<td>special_fu_utilization</td>
<td>function unit</td>
<td>flop_dp_efficiency</td>
<td>compute</td>
</tr>
</tbody>
</table>
Contention Model

- Apply Max to quantify the interference

$$C_i = \left( \sum_{j=0}^{kern} Util_i^j > 1 \right) ? \sum_{j=0}^{kern} Util_i^j : 1$$

$$C_{type} = MAX \left\{ C_i^{type} \right\}$$

$$C' = MAX \left\{ C_{occupancy}, C_{fu}, C_{mem}, C_{cmp} \right\}$$

- Update average block execution time

$$AvgBlkTime'(K_i) = AvgBlkTime(K_i) \times C'$$
Stream Launch

- Benchmark the average stream launch overhead

\[ StreamStart_i = StreamStart_{i-1} + StreamLaunchOvhd \]

- Comparison with the actual launch overhead on a NVIDIA GTX 950 GPU
Stream Launch

Interesting Observation

- **Overlapped** data transfers (NVIDIA gaming **GPUs**)

![Diagram of data transfers for different GPU models](image-url)
Stream Launch

Interesting Observation

- **No overlapped** data transfers (NVIDIA Computing GPUs)
## Model Consolidation

### Algorithm: Modeling Concurrent Kernel Execution

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize CUDA call for each stream</td>
</tr>
<tr>
<td>Initialize trace table based on <em>Stream Launch Model</em></td>
</tr>
<tr>
<td>Sort trace table by the starting time of CUDA calls</td>
</tr>
<tr>
<td><strong>while</strong> not all CUDA calls are done</td>
</tr>
<tr>
<td>wake a call from sleep</td>
</tr>
<tr>
<td>check concurrency among the wake list</td>
</tr>
<tr>
<td><strong>if</strong> concurrent data transfer</td>
</tr>
<tr>
<td><strong>then</strong></td>
</tr>
<tr>
<td>Apply <em>Data Transfer Model</em></td>
</tr>
<tr>
<td><strong>else if</strong> concurrent kernel execution</td>
</tr>
<tr>
<td><strong>then</strong></td>
</tr>
<tr>
<td>Apply <em>AvgBlkExe</em> and <em>Contention Model</em></td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td>Update wake list and trace table</td>
</tr>
<tr>
<td><strong>end while</strong></td>
</tr>
</tbody>
</table>
Part 4

Evaluation
Monte Carlo eXtreme

- Block Size Tuning
Monte Carlo eXtreme

- CKE Performance Prediction

(a) Native: 434 (ms)

(b) Moka: 417 (ms)

(c) Comparison

Time (ms)
Hidden Markov Model

- Block Size Tuning
Hidden Markov Model

- CKE Performance Prediction

(a) Native: 61 (us)

(b) Moka: 68 (us)

(c) Comparison

![Diagram showing performance comparison between Native and Moka for different stream numbers and CKE configurations.](image-url)
Concurrent Kernel Scheduling

- Three Applications:
  1. vector add
  2. matrix multiplication
  3. pathfinder
- Six different scheduling orders
- Compare the native execution time with the model predicted execution time
- Achieving a close-to-optimal solution
Part5

Summary
Moka: an empirical model for concurrent kernel execution

**Prediction Accuracy**
- Avg 12% estimation error

**Stream Scheduling**
- Suggest dispatching order
- Close-to-optimal solution

**Future Work**
- Develop non-linear model for data transfer contention
- Improve concurrent workloads scheduling
THANK YOU