Scalable Performance Analysis on Heterogeneous Architectures with HPCToolkit

Milind Chabbi, Karthik Murthy, Mike Fagan, and John Mellor-Crummey

Rice University

CScADS 2012
Snowbird, Utha
June 26, 2012
HPCToolkit

- Performance measurement using statistical sampling of timers and performance counters
- Attribution to hierarchical calling context
- Works on multilingual, fully-optimized, statically or dynamically linked applications (no source modification)
  - Pthread, OMP, MPI, and any combination
- Low overhead (under 5%) for both profiling and tracing
- Scales to large parallel systems
- Analysis of execution costs, inefficiencies, and scaling characteristics
Supporting Heterogeneity in HPCToolkit

- **Heterogeneous** doesn’t mean just GPU kernel
- Most work on the performance analysis of heterogeneous architectures deals with
  - Identifying GPU-kernel-level issues, and improving via: kernel fusion, unrolling, memory access reordering, etc.
- They ignore other parts of heterogeneous systems viz.
  - Nodes with several GPUs and CPUs, and CPUs with several threads
  - GPUs shared by multiple ranks, and concurrent kernel executions
  - Inter-node, and intra-node communication
Should Measure, Analyze and Present

Performance of

• A standalone GPU kernel
  - Timing, and hardware counter values
• Concurrently executing GPU kernels on multiple graphics cards
  - Challenges: concurrent streams, multiple threads, multiple contexts, GPU sharing between threads and processes
• Data communication between CPUs and GPUs
• Multi-threaded processes
• Multiple MPI processes
And It Should Scale

- Should be able to gather data from thousands of nodes
  - Each with several CPUs, Cores, and multiple GPU cards
- Should not distort original execution overlap
- Should have low runtime overhead
- Should produce manageable profile and trace files
Focus on Resource (under) Utilization

- Heterogeneous systems have multiple resources each with disparate capabilities
- Classical “hot-spot” analysis is insufficient
  - Focuses on “most consumed” resources
  - Provides only symptoms of problems
  - Does not indicate causes of problems
- Key to achieving peak performance on heterogeneous systems is to keep all compute resources working simultaneously
  - Overlap computations on multiple resources
Work Balance Between CPU and GPU

- Offloading entire computation to GPUs wastes CPU compute power
- Offloading entire computation to CPUs wastes GPU compute power

Matrix multiplication on Nvidia 8800 GTX (575 Mhz) and Intel Core2 Quad (2.4Ghz)

Figure credit: Qilin Exploiting Parallelism on Heterogeneous Multiprocessors
Root Cause Analysis with Blame Shifting

- If GPU is idle, code executing on CPU is responsible for not offloading (enough) work to GPU
  ✦ Attribute blame to CPU code executing while GPU is idle

- If CPU is idle waiting for GPU kernel(s) to finish, executing GPU kernel(s) are responsible for CPU idleness
  ✦ Attribute proportional blame to each such kernels

- Credit codes that are well overlapped
Performance Expectations for Heterogeneous Systems with Blame Shifting

Top GPU-kernel may not be the best candidate for tuning
Performance Expectations for Heterogeneous Systems with Blame Shifting

CPU

WORK

SYNC

CPU

WORK

SYNC

Kernel A

Kernel B

5% expected gain by tuning Kernel A

Top GPU-kernel may not be the best candidate for tuning
Performance Expectations for Heterogeneous Systems with Blame Shifting

Top GPU-kernel may not be the best candidate for tuning
Performance Expectations for Heterogeneous Systems with Blame Shifting

- 5% expected gain by tuning Kernel A
- 40% expected gain by tuning Kernel B

Top GPU-kernel may not be the best candidate for tuning

Tuesday, June 26, 12
Performance Expectations for Heterogeneous Systems with Blame Shifting

5% expected gain by tuning Kernel A

40% expected gain by tuning Kernel B

Visa-versa is also true

Top GPU-kernel may not be the best candidate for tuning

Hot spot analysis

Blame shifting

Tuesday, June 26, 12
Advantages of Blame Shifting on Heterogeneous Systems

• Pinpoints codes (both GPU kernels and CPU contexts) that benefit most from tuning
  ✦ Improves developer productivity
  ✦ Full calling context to distinguish same kernel, different callpath

• Provides an expectation for the upper bound of performance gain when tuning

• Sampling-based approach keeps overhead low and provides scalability

• Extends naturally to any shared resource
  ✦ GPU, communication network, I/O network
Proxy Sampling of GPU Activities

CPU thread

GPU stream

Tuesday, June 26, 12
Proxy Sampling of GPU Activities
Proxy Sampling of GPU Activities

CPU thread

GPU stream

Tuesday, June 26, 12
Proxy Sampling of GPU Activities

![Diagram of CPU thread and GPU stream with kernel execution and overlap]

CUDA Device Synchronize()
Proxy Sampling of GPU Activities

CPU thread

GPU stream

Kernel Execution

Overlap

cudaDevice Synchronize()
Proxy Sampling of GPU Activities

Event Query

Kernel Execution

cudaDevice Synchronize()
Proxy Sampling of GPU Activities

CPU thread

GPU stream

Overlap

cudaDevice Synchronize()

Kernel Execution

GPU ONLY
Proxy Sampling of GPU Activities

CPU thread

GPU stream

Kernel Execution

Overlap

cudaDevice Synchronize()

CPU IDLE

GPU ONLY

Tuesday, June 26, 12
Proxy Sampling of GPU Activities

- CPU thread
- GPU stream
- Event Query
- cudaDevice Synchronize()
- Overlap
- CPU IDLE
- GPU ONLY
- Blame

Kernel Execution

Tuesday, June 26, 12
Proxy Sampling of GPU Activities

- CPU thread
- GPU stream
- EventQuery
- EventQuery
- EventQuery
- EventQuery
- cudaDevice Synchronize()
- CPU IDLE
- CPU ONLY
- GPU idle
- GPU idle
- GPU ONLY
- Kernel Execution
- Blame

Tuesday, June 26, 12
Proxy Sampling of GPU Activities
Proxy Sampling of GPU Activities

- **Kernel Execution**
  - **CPU ONLY**
  - **GPU ONLY**
  - **GPU IDLE**
  - **CPU IDLE**

- **Overlap**
  - EventQuery

- **CUDA Device Synchronize**

- **Blame**
  - CPU thread
  - GPU stream

**Tuesday, June 26, 12**
Implementation Challenges

- No sampling support from GPUs
  ✦ Would have liked timer/counter-based signals from GPUs

- CUPTI has several limitations (some fixed in 5.0RC)
  ✦ Kernel serialization when using CUPTI
  ✦ Serialization of CPU threads simultaneously using CUPTI
  ✦ Activity API is more tracing style, not suitable for profiling

- CUDA limitations (supposed to be fixed in Kepler 2)
  ✦ Kernel serialization when using events for querying/timing

- Can’t poke GPU with cudaEventQuery() from a signal handler when thread is inside a CUDA API call
Workarounds

- CUDA Function wrapping to inject events
  - Eliminates CPU threads serialization
  - Waiting for Kepler-2 to fix kernel serialization when using events

- Disable calling cudaEventQuery() from signal handler when CPU is inside CUDA API
  - Deferred blaming of kernels
Workarounds: Deferred Blaming

- **CPU thread**
- **GPU stream**

**T₁**

- **Overlap**
- **Kernel Execution**

- **cudaDevice Synchronize()**

**CPU IDLE**

**CPU ONLY**

**GPU IDLE**

**GPU ONLY**

- **Blame**
Workarounds: Deferred Blaming

- CPU thread
- GPU stream

**Overlap**: EventQuery

**Kernel Execution**: EventQuery

**CUDA Device Synchronize()**: CPU IDLE

**GPU IDLE**: GPU ONLY

**CPU ONLY**: CPU ONLY

**Blame**: GPU idle

**Tuesday, June 26, 12**
Workarounds: Deferred Blaming

- **CPU thread**
  - Kernel
  - Overlap
  - EventQuery
  - EventQuery

- **GPU stream**
  - Kernel Execution
  - Event
  - Event

- **CUDA Device**
  - Synchronize()

- **CPU ONLY**
  - CPU idle

- **GPU ONLY**
  - GPU idle
  - GPU idle

- **GPU IDLE**

- **Blame**
Workarounds: Deferred Blaming

CPU thread

GPU stream

Kernel Execution

Overlap

cudaDevice Synchronize()

CPU ONLY

GPU IDLE

Blame

GPU idle

GPU idle

Blind zone

K

e

r

n

l

r

3

event

event

event

event

event

event

event

event

T_1

T_2

T_3

Tuesday, June 26, 12
Workarounds: Deferred Blaming

- **CPU thread**
  - Event
  - Kernel
  - Event

- **GPU stream**
  - Event
  - Kernel Execution
  - Event

- **Overlap**
  - EventQuery

- **Blind zone**
  - cudaDevice Synchronize()

- **Post blind zone**
  - Basic zone
  - GPU only
  - CPU only
  - CPU idle

- **T1**, **T2**, **T3**

- **GPU idle**
  - GPU idle

- **Blame**
  - GPU only
  - CPU only
**Workarounds: Deferred Blaming**

- **CPU thread**
- **GPU stream**

**CPU ONLY**
- **GPU IDLE**

**Overlap**
- **EventQuery**
- **Kernel Execution**
- **Multiple kernels**

**T_1**
- **Event**

**T_2**
- **Blind zone**
- **cudaDeviceSynchronize()**

**T_3**
- **Post blind zone**
- **Blame**
- **CPU ONLY**
- **GPU idle**
- **GPU idle**
Workarounds: Deferred Blaming

CPU thread

GPU stream

Kernel Execution

Multiple kernels

Overlap

EventQuery

EventQuery

cudaDevice Synchronize()

Post blind zone

T₁

T₂

T₃

Blind zone

GPU IDLE

CPU ONLY

GPU idle

GPU idle

Blame

Blame

Blame

Tuesday, June 26, 12
Workarounds: Deferred Blaming

- **CPU thread**
- **GPU stream**

**Kernel Execution**

- Event Query
- Event Query

**Overlap**

- Event Query

**cudaDevice Synchronize()**

**Multiple kernels**

**Post blind zone**

**GPU IDLE**

**CPU ONLY**

**GPU ONLY**

**Blame**

**Deferred Blaming**

**Tuesday, June 26, 12**
# HPCToolkit vs. TAU & Vampir Time Overhead

Keeneland: Intel Westmere hex-core CPUs@2.8GHz, 24GB, NVIDIA 6GB Tesla M 2090 GPUs, and a Qlogic QDR InfiniBand interconnect

<table>
<thead>
<tr>
<th>Program</th>
<th>Base runtime</th>
<th>HPCToolkit</th>
<th>TAU</th>
<th>Vampir</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAMMPS</td>
<td>26.8264 sec</td>
<td>8.9% (29.2059s)</td>
<td>10% (29.5081s)</td>
<td>3.1x (83.6458s)</td>
</tr>
<tr>
<td>rhodopsin protein in solvated lipid bilayer (32procs, 6 nodes, 6ppn, 3 gpu/node)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LULESH</td>
<td>17.4887 sec</td>
<td>4.1% (18.2031s)</td>
<td>5.8% (18.5003s)</td>
<td>47% (25.7486s)</td>
</tr>
<tr>
<td>(1 node, 1 proc, 1 gpu)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# HPCToolkit vs. TAU & Vampir
## Data Volume

<table>
<thead>
<tr>
<th>Program</th>
<th>HPCToolkit</th>
<th>TAU</th>
<th>Vampir</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Profiling</td>
<td>Tracing</td>
<td>Profiling</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>16MB</td>
<td>57MB</td>
<td>43x (693MB)</td>
</tr>
<tr>
<td>(32procs, 6 nodes, 6ppn, 3 gpu/node)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LULESH</td>
<td>268KB</td>
<td>4MB</td>
<td>3.5x (948KB)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simulations involving complex multi-material motion are one of the most CPU time consuming applications.

- **LULESH**: classic hydro-dynamics code, solves Sedov blast wave problem with “leap frog” time integration scheme.
- CUDA version available from LLNL
- DEMO
LULESH CUDA Memory Allocation

Total bytes allocated

Memory request with time
Replaced repeated memory allocation/free with a global allocation: 30% running time improvement
LAMMPS on LJ

- Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS): Classical molecular dynamics code

- Two CUDA versions
  - **GPU**
    - Designed to exploit common GPU hardware configurations with atom-based data (e.g. coordinates, forces) moves back-and-forth between the CPU(s) and GPU every timestep.
    - Neighbor lists can be constructed on the CPU or on the GPU
    - The charge assignment and force interpolation portions of PPPM can be run on the GPU. The FFT portion runs on the CPU.
    - Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU.
  - **USER-CUDA (all on GPU)**
    - Many timesteps, to run entirely on the GPU
Conclusions

• Hybrid CPU/GPU blame shifting with HPCToolkit
  ✦ Provides novel and practical technique for performance analysis of heterogeneous systems
  ✦ Pinpoints code fragments (CPU and GPU) worth tuning
    ★ Improves developer productivity
  ✦ Provides scalable performance measurement and analysis with low space and time overhead compared to state-of-the-art tools

• Several implementation challenges
  ✦ Better API/hardware support from vendor can eliminate workarounds in all tools
Conclusions

• Hybrid CPU/GPU blame shifting with HPCToolkit
  ✦ Provides novel and practical technique for performance analysis of heterogeneous systems
  ✦ Pinpoints code fragments (CPU and GPU) worth tuning
    ★ Improves developer productivity
  ✦ Provides scalable performance measurement and analysis with low space and time overhead compared to state-of-the-art tools

• Several implementation challenges
  ✦ Better API/hardware support from vendor can eliminate workarounds in all tools