Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs

Heike Jagode

Paper:
Malony, A., Biersdorff, S., Shende, S., Jagode, H., Tomov, S., Juckeland, G., Dietrich, R., Poole, D., Lamb, C.
ICPP 2011 (submitted), Taipei, Taiwan, 2011.
OUTLINE

1. Heterogeneous Computational Model

2. CPU-GPU Measurement Approaches
   • Synchronous method
   • Event queue method
   • Callback Method

3. Heterogeneous Performance Tools
   • PAPI CUDA Component (used by TAU and VAMPIR)

4. Experiments
OUTLINE

1. Heterogeneous Computational Model

2. CPU-GPU Measurement Approaches
   - Synchronous method
   - Event queue method
   - Callback Method

3. Heterogeneous Performance Tools
   - PAPI CUDA Component (used by TAU and VAMPIR)

4. Experiments
Heterogeneous Computation Model

- A heterogeneous computing architecture based on GPUs: can be characterized as scalable cluster of shared memory nodes with multicore processors and PCI-attached GPUs
- Cluster is interconnected by a high-performance network fabric for fast, high-bandwidth inter-node communication

That’s the model that is targeted by the tools
Heterogeneous Computation

3 types of operational interactions:

- Interactions between nodes that take place through communication between processes: MPI, global address space
- Intra-node interactions between threads as part of the node’s CPU multicore parallel execution: shared memory, multi-threads
- Interactions between a node’s CPU and the attached GPU devices: DMA memory transfers to/from GPU device over a PCI bus; launching of GPU kernels, etc.

Ideal: Parallel performance tools provide a comprehensive heterogeneous performance perspective that represents all the different operational aspects of the computation in a unified way
1. **Heterogeneous Computational Model**

2. **CPU-GPU Measurement Approaches**
   - Synchronous method
   - Event queue method
   - Callback Method

3. **Heterogeneous Performance Tools**
   - PAPI CUDA Component (used by TAU and VAMPIR)

4. **Experiments**
CPU-GPU Measurement Approaches

Three sources of information available:

1. **GPU Programming Language**: CUDA and OpenCL are the current standards for development on GPUs each provides support for observing *events* associated with GPU use

2. **GPU Driver**: GPU device interface makes host-to-device operations visible to tools (if accessible)

3. **GPU device**: hardware performance measurements associate GPU metrics with kernel execution

**Note**: there is no guarantee that all the information is available on all platforms
(A) Synchronous method
(WLOG, consider measurement of execution time)

- Time measurement is predicted when the kernel begins and ends
- CPU-host observes the begin and end event (red)
- CPU immediately waits on kernel termination after launch  
  → kernel measurement is synchronized with kernel execution
- Assume kernel executes immediately  → time between begin and end  
  event will accurately reflect kernel performance
(A) Synchronous method
(WLOG, consider measurement of execution time)

**CONS**
- Leads to inaccuracies with more flexible kernel execution modes
- There might be a long time between kernel launch and sync → results in poor estimates of actual kernel execution time
- Multiple kernels can be launched before sync point is encountered

**PROS**
- Time measurement is predicted when the kernel begins and ends
  - CPU host observes the begin and end event (red)
- Doesn't require any additional performance measurement mechanisms beyond what's presently available in tools
- It works fine when a synchronous library such as CUBLAS is used
  - Assume kernel executes immediately → time between begin and end event will accurately reflect kernel performance
Event queue method
(WLOG, consider measurement of execution time)

**Event kernel** records state of GPU when it’s executed
- add event kernel before and after computational kernel
  → performance data more closely with kernel begin and end
- Host: generate event kernels, queue them, read results
- Device: make measurement
(B) Event queue method
(WLOG, consider measurement of execution time)

**CONS**
- Relies on device manufacturer to provide event kernel support
- Depends on implementations on how events can be used and what performance information is returned
- Requires host to process and read events
- If timestamps are used to merge GPU and CPU events → times need to be synchronized between host and device

**PROS**
- Addresses case where multiple kernels are launched in a stream
- Host: generate event kernels, queue them, read results
- Device: make measurement

*Event kernel* records state of GPU when it’s executed
(C) Callback method
(WLOG, consider measurement of execution time)

- relies on mechanism in device layer that triggers callbacks on host for registered actions (e.g. begin and end of kernel execution)
- Allows more immediate kernel performance measurement since control can be given to CPU process via callback
(C) Callback method
(WLOG, consider measurement of execution time)

CONS
- Heavy dependence on the device manufacturer to provide support in the device layer
- Not supported (yet) in CUDA 4.0 for kernels (only runtime fcts)
  ➔ currently, immediate callback is only received after CUDA_SYNC

NOTE: kernel completion callbacks are planned for CUDA 4.1

PROS
- Allows more immediate kernel performance measurement since control over beginning and end of kernel execution
- More flexible: wider range of callbacks can be provided
  ➔ performance measurement can be specific to callback type
OUTLINE

1. Heterogeneous Computational Model

2. CPU-GPU Measurement Approaches
   - Synchronous method
   - Event queue method
   - Callback Method

3. Heterogeneous Performance Tools
   - PAPI CUDA Component (used by TAU and VAMPIR)

4. Experiments
Heterogeneous Performance Tools

Requirement:
• measurement methods for GPUs kernel execution and CPU-GPU interaction can be integrated with existing parallel performance tools → heterogeneous performance measurement

Tools:
• **PAPI**: support of performance counter measurement
• **TAU**: portable profiling and tracing toolkit for performance analysis of parallel programs
• **VAMPIR**: consists of VampirTrace part for instrumentation, monitoring and recording + VampirServer part for visualization and analysis of runtime event traces
PAPI CUDA Component

• HW performance counter measurement technology for NVIDIA CUDA platform
• Access to HW counters inside the GPUs
• Based on CUPTI (CUDA Performance Tool Interface) (CUDA 4.0rc)
• In any environment with CUPTI, PAPI CUDA component can provide detailed performance counter info regarding execution of GPU kernel
• Initialization, device management and context management is enabled by CUDA driver API
• Domain and event management is enabled by CUPTI
• Name of events is established by the following hierarchy: Component.Device.Domain.Event
### Portion of CUDA events available on IG (GeForce GTX, Tesla C870)

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x44000000</td>
<td>CUDA.GeForce_GTX_480.gpc0.local_load</td>
<td># executed local load instructions per warp on a multiprocessor</td>
</tr>
<tr>
<td>0x44000001</td>
<td>CUDA.GeForce_GTX_480.gpc0.local_store</td>
<td># executed local store instructions per warp on a multiprocessor</td>
</tr>
<tr>
<td>0x44000002</td>
<td>CUDA.GeForce_GTX_480.gpc0.gld_request</td>
<td># executed global load instructions per warp on a multiprocessor</td>
</tr>
<tr>
<td>0x44000003</td>
<td>CUDA.GeForce_GTX_480.gpc0.get_request</td>
<td># executed global store instructions per warp on a multiprocessor</td>
</tr>
<tr>
<td>0x44000004</td>
<td>CUDA.GeForce_GTX_480.gpc0.shared_load</td>
<td># executed shared load instructions per warp on a multiprocessor</td>
</tr>
<tr>
<td>0x44000005</td>
<td>CUDA.GeForce_GTX_480.gpc0.shared_store</td>
<td># executed shared store instructions per warp on a multiprocessor</td>
</tr>
<tr>
<td>0x44000006</td>
<td>CUDA.GeForce_GTX_480.gpc0.branch</td>
<td># branches taken by threads executing a kernel</td>
</tr>
<tr>
<td>0x44000007</td>
<td>CUDA.GeForce_GTX_480.gpc0.divergent_branch</td>
<td># divergent branches within a warp</td>
</tr>
<tr>
<td>0x4400000b</td>
<td>CUDA.GeForce_GTX_480.gpc0.active_cycles</td>
<td># cycles a multiprocessor has at least one active warp</td>
</tr>
<tr>
<td>0x4400000c</td>
<td>CUDA.GeForce_GTX_480.gpc0.sm_cta_launched</td>
<td># thread blocks launched on a multiprocessor</td>
</tr>
<tr>
<td>0x4400000d</td>
<td>CUDA.GeForce_GTX_480.gpc0.1_l_local_load_hit</td>
<td># local load hits in L1 cache</td>
</tr>
<tr>
<td>0x4400000e</td>
<td>CUDA.GeForce_GTX_480.gpc0.1_l_local_load_miss</td>
<td># local load misses in L1 cache</td>
</tr>
<tr>
<td>0x44000011</td>
<td>CUDA.GeForce_GTX_480.gpc0.1_l_global_load_hit</td>
<td># global load hits in L1 cache</td>
</tr>
<tr>
<td>0x4400002e</td>
<td>CUDA.Tesla_C870.domain_a.tex_cache_hit</td>
<td># texture cache misses</td>
</tr>
<tr>
<td>0x4400002f</td>
<td>CUDA.Tesla_C870.domain_a.tex_cache_miss</td>
<td># texture cache hits</td>
</tr>
<tr>
<td>0x44000034</td>
<td>CUDA.Tesla_C870.domain_b.local_load</td>
<td># local memory load transactions</td>
</tr>
<tr>
<td>0x44000037</td>
<td>CUDA.Tesla_C870.domain_b.branch</td>
<td># branches taken by threads executing a kernel</td>
</tr>
<tr>
<td>0x44000038</td>
<td>CUDA.Tesla_C870.domain_b.divergent_branch</td>
<td># divergent branches within a warp</td>
</tr>
<tr>
<td>0x44000039</td>
<td>CUDA.Tesla_C870.domain_b.instructions</td>
<td># instructions executed</td>
</tr>
</tbody>
</table>
Tool interoperability

CUDA OpenCL

TAU

PAPI

VampirTrace

ParaProf

Event queue

Callback

Vampir
1. Heterogeneous Computational Model

2. CPU-GPU Measurement Approaches
   - Synchronous method
   - Event queue method
   - Callback Method

3. Heterogeneous Performance Tools
   - PAPI CUDA Component (used by TAU and VAMPIR)

4. Experiments
Experiments: MAGMA versus CUBLAS library

- Symmetry exploitation more challenging → computation would involve irregular data access
- How well is symmetry exploited? What about bank conflicts and branching?
- SYMV implementation: Access each element of lower (or upper) triangular part of the matrix only once → $N^2/2$ element reads (vs. $N^2$)
- Since SYMV is memory-bound, exploiting symmetry is expected to be twice as fast
- To accomplish this, additional global memory workspace is used to store intermediate results
- We ran experiments using CUBLAS_dsymv (general) and MAGMA_dsymv (exploits symmetry) to observe the effects of cache behavior on Tesla S2050 (Fermi) GPU
CUDA performance counters for read behavior (as measured by PAPI)

# of read requests from L1 to L2 (green), which is equal to # of read misses in L2 (orange); number of read requests from L2 to DRAM (black) for CUBLAS_dsymv (left) and MAGMA_dsymv (right)
CUDA performance counters for write behavior (as measured by PAPI)

# of write requests from L1 to L2 (green), which is equal to # of write misses in L2 (orange); # of write requests from L2 to DRAM (black) for CUBLAS_dsymv (left) and MAGMA_dsymv (right)
CUDA performance counter for L1 behavior (as measured by PAPI)

# of L1 shared bank conflicts in the MAGMA_dsymv kernel for medium to large matrix sizes (left); Performance of MAGMA_dsymv kernel with and without shared bank conflicts (right)
Left: also due to the exploitation of the symmetry of the matrix, the # of branches executes in the MAGMA_dsymv kernel is reduced by a factor of ~30 compared to the # of braches in the CUBLAS_dsymv kernel

Right: We also see that MAGMA is more efficient in instruction access
VAMPIR display of Stencil2D execution on 4 MPI processes with 4 GPUs. Time synchronized GPU counter rates convey important performance characteristics of the kernel execution.
Conclusion and Future Work

• Performance of scalable heterogeneous parallel systems and applications depends on addressing new challenges of
code instrumentation
measurement
analysis
of heterogeneous components

• Support for GPU performance measurement with CUDA (and OpenCL) in PAPI (Vampir and TAU)

• Computation and measurement models (with respect to present technology)

• Further optimizations necessary:
  • CUPTI currently supports callbacks for CUDA runtime but not kernels → limits timing of CUDA kernel in asynchronous manner
  • Kernel code instrumentation is currently not supported
Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs

Heike Jagode

Paper:
Malony, A., Biersdorff, S., Shende, S., Jagode, H., Tomov, S., Juckeland, G., Dietrich, R., Poole, D., Lamb, C.
ICPP 2011 (submitted), Taipei, Taiwan, 2011.