Performance Measurement and Analysis of Heterogeneous Parallel Systems: Tasks and GPU Accelerators

Allen D. Malony, Sameer Shende, Shangkar Mayanglambam, Scott Biersdorff, Wyatt Spear
{malony,sameer, smeitei,scottb,wspear}@cs.uoregon.edu

Computer and Information Science Department
Performance Research Laboratory
University of Oregon
Outline

- What’s all this about heterogeneous systems?
- Heterogeneity and performance tools
- Beating up on TAU
- Task performance abstraction and good ‘ol master/worker
- What’s all this about GPGPU’s?
  - Accelerator performance measurement in PGI compiler
  - TAU CUDA performance measurement
- Final thoughts
Heterogeneous Parallel Systems

- What does it mean to be heterogeneous?
  - New Oxford America, 2nd Edition: 
    diverse in character or content
  - Prof. Dr. Felix Wolf, Sage of Research Centre Juelich: 
    not homogeneous

- Diversity in what?
  - Hardware
    - processors/cores, memory, interconnection, …
    - different in computing elements and how they are used
  - Software (hybrid)
    - how the hardware is programmed
    - different software models, libraries, frameworks, …

- Diversity when? Heterogeneous implies combining together
Why Do We Care?

- Heterogeneity has been around for a long time
  - Have different programmable components in computer systems
  - Long history of specialized hardware

- Heterogeneous (computing) technology more accessible
  - Multicore processors
  - Manycore accelerators (e.g., NVIDIA Tesla GPU)
  - High-performance processing engines (e.g., IBM Cell BE)

- Performance is the main driving concern
  - Heterogeneity is arguably the only path to extreme scale

- Heterogeneous (hybrid) software technology required
  - Greater performance enables more powerful software
    - Will give rise to more sophisticated software environments
Implications for Performance Tools

- Tools should support parallel computation models
- Current status quo is comfortable
  - Mostly homogeneous parallel systems and software
  - Shared-memory multithreading – OpenMP
  - Distributed-memory message passing – MPI
- Parallel computational models are relatively stable (simple)
  - Corresponding performance models are relatively tractable
  - Parallel performance tools are just keeping up
- Heterogeneity creates richer computational potential
  - Results in greater performance diversity and complexity
- Performance tools have to support richer computation models and broader (less constrained) performance perspectives
Current TAU Performance Perspective

- TAU is a direct measurement performance systems
  - Event stack performance perspective for “threads of execution”
  - Message communication performance
- TAU measures two general types of events
  - Interval event: coupled begin and end events
  - Atomic events
- TAU also maintains an event stack during execution
  - Events can be nested
  - Top of event stack the event context
  - Used to generate callpath performance measurements
  - Events can not overlap! (TAU enforces this requirement)
- What about events that are not event stack compatible?
MPI and Performance View

- TAU measures MPI events through the MPI interface
  - Standard PMPI approach (same as other tools)
  - Performance for interval events plus metadata

- Consider a paired message send/receive between P1 and P2
  - Suppose we want to measure the time on P1 from:
    - when P1 sends a message to P2
    - to when P1 receives a message from P2
  - TAU MPI events will not do this
  - Can create a TAU user-level interval event \((s-r)\)
    - \(s-r\) begin and \(s-r\) end must have the same event context
    - no other events can overlap (nested events are ok)
  - What if these requirements can not be maintained?
Conflicting Contexts in Send-Receive MPI Scenario
Supporting Multiple Performance Perspectives

- Need to support alternative performance views
  - Reflect execution logic beyond standard actions
  - Capture performance semantics at multiple levels
  - Allow for compatible perspectives that do not conflict
- TAU event stack (nesting) perspective somewhat limited
- TAU’s performance mapping can partially address need
- Some frameworks have own performance (timing) packages
  - Cactus, SAMRAI, PETSc, Charm++
  - Want to leverage/integrate/layer on TAU infrastructure
- Need also to incorporate views of external performance
TAU ProfilerCreate API

- Exposes TAU measurement infrastructure
- Software packages can easily access TAU profiler objects
  - Control completely determined by package
  - Can use to translate performance measures
  - Can access and set any part of the profiler information
- Goal of simplicity
  - API had to be easy to integrate in existing packages!
- Allows for multiple, layered performance measurements
  - Simultaneous to TAU (internal) measurement system
ProfilerCreate API

#include <TAU.h>

// TAU_PROFILER_CREATE(void *ptr, char *name, char *type, TauGroup_t tau_group);

TAU_PROFILER_CREATE(ptr, "main", "int (int, char**)", TAU_USER);

TAU_PROFILER_START(ptr);
    // work
TAU_PROFILER_STOP(ptr);

#include <TAU.h>

TAU_PROFILER_GET_INCLUSIVE_VALUES(handle, data)
TAU_PROFILER_GET_EXCLUSIVE_VALUES(handle, data)
TAU_PROFILER_GET_CALLS(handle, data)
TAU_PROFILER_GET_CHILD_CALLS(handle, data)
TAU_PROFILER_GET_COUNTER_INFO(counters, numcounters)
Use of TAU ProfilerCreate API in Cactus

- Cactus has its own performance evaluation interface
- Developers prefer to use TAU’s interface
- Need a runtime performance assessment interface
- Layered Cactus API on top of new ProfilerCreate API
- Created a TAU scoping profiler for capturing top-level performance event (equivalent to main)
Cactus Performance (Full Profile)

- Events under Cactus control
- Use TAU to capture timing and hardware measures
Performance Views of External Execution

- Heterogeneous applications can have concurrent execution
  - Main “host” path and “external” external paths
  - Want to capture performance for all execution paths
  - External execution may be difficult or impossible to measure
- “Host” creates measurement view for external entity
  - Maintains local and remote performance data
  - External entity may provide performance data to the host
- What perspective does the host have of the external entity?
  - Determines the semantics of the measurement data
- Consider the “task” abstraction
Task-based Performance Views

- Host regards external execution as a task
  - Tasks operate concurrently with respect to the host
  - Requires support for tracking asynchronous execution

- Host keeps measurements for external task
  - Host-side measurements of task events
  - Performance data received external task
  - Tasks may have limited measurement support
  - May depend on host for performance data I/O

- Need an task performance API
  - Capture abstract (host-side) task events
  - Populate TAU’s performance data structures for task
  - Derived from ProfilerCreate API to address these concerns
#include <TAU.h>

TAU_CREATE_TASK(taskid);

// TAU_PROFILER_CREATE(void *ptr, char *name, char *type,
//                        TauGroup_t tau_group);

TAU_PROFILER_CREATE(ptr, "main", "int (int, char**)", TAU_USER);

TAU_PROFILER_START_TASK(ptr, taskid);
// work
TAU_PROFILER_STOP_TASK(ptr, taskid);
#include <TAU.h>

TAU_PROFILER_GET_INCLUSIVE_VALUES_TASK(ptr, data, taskid);
TAU_PROFILER_SET_INCLUSIVE_VALUES_TASK(ptr, data, taskid);

TAU_PROFILER_GET_EXCLUSIVE_VALUES_TASK(ptr, data, taskid);
TAU_PROFILER_SET_EXCLUSIVE_VALUES_TASK(ptr, data, taskid);

TAU_PROFILER_GET_CALLS_TASK(ptr, data, taskid);
TAU_PROFILER_SET_CALLS_TASK(ptr, data, taskid);

TAU_PROFILER_GET_CHILD_CALLS_TASK(ptr, data, taskid);
TAU_PROFILER_SET_CHILD_CALLS_TASK(ptr, data, taskid);
**Master-Worker Scenario with TAU Task API**

- Master sends tasks to $N$ workers
- Workers report back their performance to master
  - Done for each piece of work
- Build a worker performance perspective in the master
- TAU will only output a performance profile from the master
  - Each work task will appear as a separate “thread” of the master
- In general, the external performance data can be arbitrary
  - Single time value
  - More complete representation of external performance
Master-Worker with Task API: 32 Workers
CPU – GPU Execution Scenarios
PGI Compiler for GPUs

- Accelerator programming support
  - Fortran and C
  - Directive-based programming
  - Loop parallelization for acceleration on GPUs
  - PGI 9.0 for x64-based Linux (preview release)

- Compiled program
  - CUDA target
  - Synchronous accelerator operations

- Profile interface support
TAU with PGI Accelerator Compiler

- Supports compiler-based instrumentation for PGI compilers
- Track runtime system events as seen from the host processor
- Show source information associated with events
  - Routine name
  - File name, source line number for kernel
  - Variable names in memory upload, download operations
  - Grid sizes
- Any configuration of TAU with PGI supports tracking of accelerator operations
  - Tested with PGI 8.0.3, 8.0.5, 8.0.6 compilers
  - Qualification and testing with PGI 9.0-1 complete
Wrapping PGI Accelerator Runtime System Calls

- Wrapping performed using *performance interface*
  - Append “_p” to runtime calls of interest to measure

```c
void __pgi_cu_module_p(void *image);
void __pgi_cu_module(void *image) {
    TAU_PROFILE("__pgi_cu_module","",TAU_DEFAULT);
    __pgi_cu_module_p(image);
}
```

- Provided in calls for:
  - Init
  - Launching kernels (synchronous execution)
  - Upload and download
PGI Accelerator Runtime Measurement API

__pgi_cu_sync
__pgi_cu_fini
__pgi_cu_module
__pgi_cu_module_function
__pgi_cu_module_file
__pgi_cu_module_unload
__pgi_cu_paramset
__pgi_cu_launch
__pgi_cu_free
cuda_deviceptr __pgi_cu_alloc

__pgi_cu_download
__pgi_cu_download1
__pgi_cu_download2
__pgi_cu_download3
__pgi_cu_downloadp
__pgi_cu_upload
__pgi_cu_upload1
__pgi_cu_upload2
__pgi_cu_upload3
__pgi_cu_uploadc
__pgi_cu_uploadn
Matrix Multiply (MM) Example

- Test with simple matrix multiply
- Vary the matrix sizes
- Demonstrate TAU integration

```fortran
module mymm
contains
subroutine multiply_matrices(a, b, c, m)
  real, dimension(:, :) :: a, b, c
  i = 0

!$acc region
  do j = 1, m
    do i = 1, m
      a(i, j) = 0.0
    enddo
  enddo
  do k = 1, m
    do i = 1, m
      a(i, j) = a(i, j) + b(i, k) * c(k, j)
    enddo
  enddo
!$acc end region
end subroutine
end module
```

"mm2.f90" 23 lines --95%-- 22.5
Build with Compiler-based Instrumentation

```bash
[ ~/mm]$ export TAU_MAKEFILE=/opt/tau-2.18.2/x86_64/lib/Makefile.tau-pgi
[ ~/mm]$ export TAU_OPTIONS='-optCompInst -optVerbose'
[ ~/mm]$ export PATH=/opt/tau-2.18.2/x86_64/bin:$PATH
[ ~/mm]$ cat Makefile
  #F90=pgf90
  # To profile with TAU, set TAU_MAKEFILE, TAU_OPTIONS and use
  # tau_f90.sh as the compiler
  F90=tau_f90.sh

  #FFLAGS=-03
  FFLAGS=-ta=nvidia

  mm: mm2.o mmdrv.o
      $(F90) $(FFLAGS) mm2.o mmdrv.o -o mm $(LIBS)
  %.o: %.f90
      $(F90) -c $< $(FFLAGS)
clean:
    /bin/rm -rf mm *.o profile.* MULT* *.mod
[ ~/mm]$ make

20.0-1 All
```
MM Profile (3000 x 3000, ~22 Gflops)
MM Program on Different Array Sizes

- Parameter study of MM to evaluate GPU
  - Array sizes: 100, 500, 1000, 2000, 5000
  - 10 iterations
  - Results uploaded to performance database
- Want to observe the effects on PGI accelerator runtime routines
  - `__pgi_cu_launch`
### MM Callpath Profiling – Tree Table View

![TAU: ParaProf: Thread Statistics: n,c,t, 0,0,0 - Application 1, Experiment 3, Trial 15.]

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>program</td>
<td>0.738</td>
<td>119.23</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>pgf accelerator region</td>
<td>0</td>
<td>118.492</td>
<td>10</td>
<td>180</td>
</tr>
<tr>
<td>__pgi_cu_alloc</td>
<td>0.092</td>
<td>0.092</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_download2</td>
<td>0</td>
<td>2.845</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>__pgi_cu_download1</td>
<td>2.845</td>
<td>2.845</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_free</td>
<td>0.093</td>
<td>0.093</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_init</td>
<td>3.835</td>
<td>3.835</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_launch</td>
<td>108.844</td>
<td>108.844</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_module</td>
<td>0.003</td>
<td>0.003</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_module_function</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_module_unload</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_paramset</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_upload2</td>
<td>0</td>
<td>2.78</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>__pgi_cu_upload1</td>
<td>2.78</td>
<td>2.78</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
MM Array Size Comparison with PerfExplorer

- Show effects of array size variation (log scale)
  - Init is significant, but constant
  - Launch grows with size because of computation
  - Upload and download do also, as determined by algorithm
MM Trace View with Jumpshot
CUDA Programming for GPGPU

☐ PGI compiler represents GPGPU programming abstraction
  ☐ Performance tool uses runtime system wrappers
    ➢ essentially a synchronous call performance model!!!

☐ In general, programming of GPGPU devices is more complex

☐ CUDA environment
  ☐ Programming of multiple streams and GPU devices
    ➢ multiple streams execute concurrently
  ☐ Programming of data transfers to/from GPU device
  ☐ Programming of GPU kernel code
  ☐ Synchronization with streams
  ☐ Stream event interface
  ☐ CUDA profiling tool
**CPU – GPU Execution Scenarios**

1. **Scenario 1 (S1)**: CPU syncs with GPU before waiting (left).
2. **Scenario 2 (S2)**: CPU syncs with GPU after waiting (right).

---

**DOE CSCaDS 2009**  
Performance Measurement and Analysis of Heterogeneous Parallel Systems
TAU CUDA Performance Measurement

- Build on CUDA event interface
  - Allow “events” to be placed in streams and processed
    - events are timestamped
  - CUDA runtime reports GPU timing in event structure
  - Events are reported back to CPU when requested
    - use begin and end events to calculate intervals
- Want to associate TAU event context with CUDA events
  - Get top of TAU event stack at begin
- CUDA kernel invocations are asynchronous
  - CPU does not see actual CUDA “end” event
  - CPU retrieves events in a non-blocking and blocking manner
- Want to capture “waiting time”
**TAU CUDA Measurement API**

`void tau_cuda_init(int argc, char **argv);`
- To be called when the application starts
- Initializes data structures and checks GPU status

`void tau_cuda_exit();`
- To be called before any thread exits at end of application
- All the CUDA profile data output for each thread of execution

`void* tau_cuda_stream_begin(char *event, cudaStream_t stream);`
- Called before CUDA statements to be measured
- Returns handle which should be used in the end call
- If `event` is new or the TAU context is new for the event, a new CUDA event profile object is created

`void tau_cuda_stream_end(void * handle);`
- Called immediately after CUDA statements to be measured
- Handle identifies the stream
- Inserts a CUDA event into the stream
TAU CUDA Measurement API (2)

vector<Event> tau_cuda_update();
- Checks for completed CUDA events on all streams
- Non-blocking and returns # completed on each stream

int tau_cuda_update(cudaStream_t stream);
- Same as tau_cuda_update() except for a particular stream
- Non-blocking and returns # completed on the stream

vector<Event> tau_cuda_finalize();
- Waits for all CUDA events to complete on all streams
- Blocking and returns # completed on each stream

int tau_cuda_finalize(cudaStream_t stream);
- Same as tau_cuda_finalize() except for a particular stream
- Blocking and returns # completed on the stream
Scenario Results – One and Two Streams

- Run simple CUDA experiments to test TAU CUDA
- Tesla S1070 test system

<table>
<thead>
<tr>
<th>CPU Load</th>
<th>GPU Load</th>
<th>Event</th>
<th>Inclusive Time</th>
<th>Exclusive Time</th>
<th>Wait Time</th>
<th>Finalize Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-0)</td>
<td>75222.4922</td>
<td>75222.4922</td>
<td>75134.7656</td>
<td>87.8906</td>
</tr>
<tr>
<td>0</td>
<td>2X</td>
<td>Interpolate (C-[main—]#D-0#S-0)</td>
<td>150097.7031</td>
<td>150097.7031</td>
<td>149995.6094</td>
<td>102.0508</td>
</tr>
<tr>
<td>0</td>
<td>3X</td>
<td>Interpolate (C-[main—]#D-0#S-0)</td>
<td>225034.2031</td>
<td>225034.2031</td>
<td>224915.5312</td>
<td>118.6523</td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-0)</td>
<td>74985.6953</td>
<td>74985.6953</td>
<td>64097.1680</td>
<td>10888.6719</td>
</tr>
<tr>
<td>2Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-0)</td>
<td>75058.5234</td>
<td>75058.5234</td>
<td>42563.9648</td>
<td>32494.6289</td>
</tr>
<tr>
<td>10Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-0)</td>
<td>75032.9609</td>
<td>75032.9609</td>
<td>0.0000</td>
<td>108114.7500</td>
</tr>
</tbody>
</table>

Table 1: TAUCUDA Profiles for a single stream (Time measured in microseconds)

<table>
<thead>
<tr>
<th>CPU Load</th>
<th>GPU Load</th>
<th>Event</th>
<th>Inclusive Time</th>
<th>Exclusive Time</th>
<th>Wait Time</th>
<th>Finalize Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2X</td>
<td>Interpolate (C-[main—]#D-0#S-1)</td>
<td>149982.8750</td>
<td>149982.8750</td>
<td>149858.8906</td>
<td>124.0234</td>
</tr>
<tr>
<td>0</td>
<td>2X</td>
<td>Interpolate (C-[main—]#D-0#S-2)</td>
<td>74929.6953</td>
<td>74929.6953</td>
<td>74909.6719</td>
<td>20.0195</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-1)</td>
<td>74993.2188</td>
<td>74993.2188</td>
<td>74869.6250</td>
<td>123.5352</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-2)</td>
<td>150055.8750</td>
<td>150055.8750</td>
<td>150019.0469</td>
<td>36.6211</td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-1)</td>
<td>75054.0156</td>
<td>75054.0156</td>
<td>53687.0117</td>
<td>21367.1875</td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-2)</td>
<td>74989.4688</td>
<td>74989.4688</td>
<td>53708.9844</td>
<td>21280.2734</td>
</tr>
<tr>
<td>2Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-1)</td>
<td>74899.1406</td>
<td>74899.1406</td>
<td>32293.9453</td>
<td>42604.9805</td>
</tr>
<tr>
<td>2Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-2)</td>
<td>74948.7344</td>
<td>74948.7344</td>
<td>32429.6875</td>
<td>42519.0430</td>
</tr>
<tr>
<td>5Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-1)</td>
<td>75007.4219</td>
<td>75007.4219</td>
<td>0.0000</td>
<td>106393.0625</td>
</tr>
<tr>
<td>5Y</td>
<td>X</td>
<td>Interpolate (C-[main—]#D-0#S-2)</td>
<td>75008.5469</td>
<td>75008.5469</td>
<td>0.0000</td>
<td>106305.6641</td>
</tr>
</tbody>
</table>

Table 2: TAUCUDA Profiles for two streams
### Scenario Results – Two Devices, Two Contexts

<table>
<thead>
<tr>
<th>CPU Load</th>
<th>GPU Load</th>
<th>Event</th>
<th>Inclusive Time</th>
<th>Exclusive Time</th>
<th>Wait Time</th>
<th>Finalize Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-0</td>
<td>D-1</td>
<td>2X</td>
<td>75068.2500</td>
<td>75068.2500</td>
<td>74855.4688</td>
<td>212.8906</td>
</tr>
<tr>
<td>0 0</td>
<td>X</td>
<td>Interpolate (C-[main--]#D-0#S-0)</td>
<td>149795.0156</td>
<td>149795.0156</td>
<td>149698.7344</td>
<td>96.1914</td>
</tr>
<tr>
<td>0 0</td>
<td>X</td>
<td>Interpolate (C-[main--]#D-1#S-0)</td>
<td>150171.8750</td>
<td>150171.8750</td>
<td>150054.6875</td>
<td>117.1875</td>
</tr>
<tr>
<td>0 0</td>
<td>2X</td>
<td>Interpolate (C-[main--]#D-0#S-0)</td>
<td>74969.5625</td>
<td>74969.5625</td>
<td>74892.5781</td>
<td>77.1484</td>
</tr>
<tr>
<td>0 0</td>
<td>2X</td>
<td>Interpolate (C-[main--]#D-1#S-0)</td>
<td>75121.7266</td>
<td>75121.7266</td>
<td>53530.7617</td>
<td>21590.8203</td>
</tr>
<tr>
<td>2Y</td>
<td>Y</td>
<td>Interpolate (C-[main--]#D-0#S-0)</td>
<td>75864.0938</td>
<td>75864.0938</td>
<td>18769.0430</td>
<td>57095.2148</td>
</tr>
<tr>
<td>2Y</td>
<td>Y</td>
<td>Interpolate (C-[main--]#D-1#S-0)</td>
<td>75119.8750</td>
<td>75119.8750</td>
<td>53557.1289</td>
<td>21562.9883</td>
</tr>
<tr>
<td>Y</td>
<td>2Y</td>
<td>Interpolate (C-[main--]#D-0#S-0)</td>
<td>75123.8984</td>
<td>75123.8984</td>
<td>18204.1016</td>
<td>56919.9219</td>
</tr>
<tr>
<td>Y</td>
<td>2Y</td>
<td>Interpolate (C-[main--]#D-1#S-0)</td>
<td>75123.8984</td>
<td>75123.8984</td>
<td>18204.1016</td>
<td>56919.9219</td>
</tr>
</tbody>
</table>

Table 3: TAUCUDA Profiles for two Devices

<table>
<thead>
<tr>
<th>Event</th>
<th>calls</th>
<th>Inclusive Time</th>
<th>Exclusive Time</th>
<th>Wait Time</th>
<th>Finalize Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-Interpolate (C-[FirstWarepper—]#D-0#S-0)</td>
<td>1</td>
<td>300019.9375</td>
<td>65.3992</td>
<td>0.0000</td>
<td>8380799</td>
</tr>
<tr>
<td>InterpolateA (C-[FirstWarepper—]#D-0#S-0)</td>
<td>10</td>
<td>150013.6250</td>
<td>150013.6250</td>
<td>0.0000</td>
<td>83806752</td>
</tr>
<tr>
<td>InterpolateB (C-[FirstWarepper—]#D-0#S-0)</td>
<td>10</td>
<td>149940.8750</td>
<td>149940.8750</td>
<td>0.0000</td>
<td>83806616</td>
</tr>
<tr>
<td>All-Interpolate (C-[SecondWarepper—]#D-0#S-0)</td>
<td>1</td>
<td>300111.6250</td>
<td>65.0635</td>
<td>0.0000</td>
<td>467571.2812</td>
</tr>
<tr>
<td>InterpolateA (C-[SecondWarepper—]#D-0#S-0)</td>
<td>10</td>
<td>150018.1719</td>
<td>150018.1719</td>
<td>0.0000</td>
<td>4674823</td>
</tr>
<tr>
<td>InterpolateB (C-[SecondWarepper—]#D-0#S-0)</td>
<td>10</td>
<td>150028.3750</td>
<td>150028.3750</td>
<td>0.0000</td>
<td>4674740</td>
</tr>
</tbody>
</table>

Table 4: TAUCUDA Profiles With Two TAU Contexts and Nested Events
TAU CUDA in NAMD

- TAU integrated in Charm++ (another talk)
- NAMD is a molecular dynamics application using Charm++
- NAMD has been accelerated with CUDA
- Test out TAU CUDA with NAMD
  - Two processes with one Tesla GPU for each

**CPU profile**

**GPU profile (P0)**
- dev_nonbonded (C-[cuda_nonbonded_forces]#D-0#S-0)
- dev_sum_forces (C-[cuda_nonbonded_forces]#D-0#S-0)

**GPU profile (P1)**
- dev_nonbonded (C-[cuda_nonbonded_forces]#D-0#S-0)
- dev_sum_forces (C-[cuda_nonbonded_forces]#D-0#S-0)
Conclusions

- Heterogeneous parallel computing will challenge parallel performance technology
  - Must deal with diversity in hardware and software
  - Must deal with richer parallelism and concurrency

- Performance tools should support parallel execution and computation models
  - Understanding of “performance” interactions
    - between integrated components
    - control and data interactions
  - Might not be able to see full parallel (concurrent) detail

- Need to support multiple performance perspectives
  - Layers of performance abstraction