Porting PAPI to the Cloud
(and Other Places)

Dan Terpstra
with a little help from:
Vince Weaver,
Heike Jagode,
James Ralph,
& Kiran Kasichayanula
Alphabet Soup

- Component PAPI
- PAPI in the Cloud
- PAPI on GPUs
- PAPI and User Defined Events
- PAPI Component Repository

- PAPI-C
- PAPI-V
- PAPI-G
- PAPI-U
- PAPI-R??
Piotr Luszczek, Eric Meek, Shirley Moore, Dan Terpstra, Vincent M.Weaver, Jack Dongarra

Evaluation of the HPC Challenge Benchmarks in Virtualized Environments

VHPC’11, Bordeaux, France

August 30, 2011

PAPI in the Cloud
PAPI and the Cloud Computing Future

- Much work is being done to investigate the practicality of moving High Performance Computing to the “cloud”
- Before such a move is made, the tradeoffs of moving to a cloud environment must be investigated
- PAPI is the ideal tool for making such measurements, but it will need enhancements before it works in a virtualized cloud environment
Obstacles with PAPI and Virtualization

- Virtualization makes time measurements difficult; virtualized time can run faster or slower than wall-clock time in unpredictable ways.
- Hardware performance counter readings require the cooperation of both the operating system and hypervisor. Support for this is still under development.
- Virtualized hardware (such as network cards and disk) may require new PAPI components to be written.
Virtual Time vs Wall Clock

Variation in percentage difference between the measured CPU and wall clock times for MPIRandomAccess test of HPC Challenge. The vertical axis has been split to offer a better resolution for the majority of data points.
Variation in percentage difference between the measured wall clock times for HPL (a computationally intensive problem) for ascending and descending orders of problem sizes during execution.
HPL: Compute Intensive

% Of Bare Metal Performance

Problem Size

VMware Player
VirtualBox
KVM
MPIRandomAccess: Communication Intensive

% Of Bare Metal Performance

Log of Problem Size

VMware Player
VirtualBox
KVM
PAPI-V Future Plans

- Support for enhanced timing support, including access to real wall-clock time (if available)
- Provide components for collecting performance of virtualized hardware, such as virtual network, infiniband, GPU, and disk devices
- Provide transparent access to virtualized hardware performance counters


http://portal.acm.org/citation.cfm?doid=1952682.1952687
Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs
Allen Malony, Scott Biersdorff, Sameer Shende, Heike Jagode, Stanimire Tomov, Guido Juckeland, Robert Dietrich, Duncan Poole and Christopher Lamb
ICPP 2011, Taipei, Taiwan, 2011.

PAPI on GPUs
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) in CUDA 4.0
- 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.gst_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.l1_local_load_hit</td>
<td># local load hits in L1 cache</td>
</tr>
<tr>
<td>0x4400000e</td>
<td>CUDA.GeForce_GTX_480.gpc0.l1_local_load_miss</td>
<td># local load misses in L1 cache</td>
</tr>
<tr>
<td>0x44000011</td>
<td>CUDA.GeForce_GTX_480.gpc0.l1_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
MAGMA versus CUBLAS: SYMV

- 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)
SHOC Benchmarks – Stencil2D

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.
PAPI-G Future Goals

- Implement CUPTI callbacks for kernel information
- Provide multiple PAPI GPU component instantiations through a single PAPI meta component
- Measure performance *inside* the kernel
PAPI and User Defined Events
PAPI User Defined Events

- PAPI has a built-in low overhead RPN event parser
- Allow users access to define their own metrics
- Example -- Memory bandwidth on Intel Core2:
  - BUS_TRANS:SELF | 64 | * | core_frequency | * | PAPI_TOT_CYC | /

<table>
<thead>
<tr>
<th>MB/s</th>
<th>STREAM output</th>
<th>Counters</th>
<th>% Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>2227</td>
<td>2204</td>
<td>-1%</td>
</tr>
<tr>
<td>Scale</td>
<td>2332</td>
<td>2333</td>
<td>0%</td>
</tr>
<tr>
<td>Add</td>
<td>2471</td>
<td>2326</td>
<td>-6%</td>
</tr>
<tr>
<td>Triad</td>
<td>2473</td>
<td>2312</td>
<td>-6%</td>
</tr>
</tbody>
</table>
Specification of User Defined Events

- Event specification file
  - parsed at `PAPI_library_init` time with `PAPI_USER_EVENTS_FILE` environment variable
  - anytime afterwards with `PAPI_set_opt` call
  - Static definition at PAPI compile time

- Events defined by
  - `EventName, OPERATION_STRING`

- Can include predefined constants
  - `#define Mem_lat 450`

- Multiplexing or multiple runs if necessary
- PAPI Utilities for enumerating / post-processing
PerfExpert LCPI

PAPI User Defined Event
Future Work

- Official release of user-defined events RealSoonNow™
- Power modeling
- Detailed cycle accounting
  - \( CPU_{CLK\_UNHALTED\_CORE} = Retired + Non_{retired} + Stalls \)
- Runtime Roofline Models
- Cross – Component User Events
PAPI Component Repository
A PAPI Component Repository

• We want user contributions  
  • We don’t want to maintain them
• Users want to know what’s available  
  • And often want to contribute
• Why not a web-based Repository?  
  • Registration form to submit and track components  
  • Link to a tarball or RCS repository  
    • Sourceforge, GitHub, Google code, private repository  
  • Public page to view current components & descriptions  
  • Private page for author updates  
  • Admin page to monitor / control submissions
Performance Application Programming Interface

PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. PAPI enables software engineers to see, in near real time, the relation between software performance and processor events.
Components for PAPI-C generally consist of a named folder containing source files and various other support files. For more details on creating a component of your own, see the documentation here and examples of other components distributed with PAPI here.

If you want to view information about other available components, click one of the component names at the bottom of this page.

**Add a Component**

You must be logged into a pre-approved account to add or edit component information. To contribute a component that you’ve written, enter a descriptive name for your component in the box below, and press the button. This will become the name of the page that describes your component. If that page already exists, you will be directed to a form to edit that page.

For an illustration of what your Component page will look like, visit the Component Example page or other Component pages.

**Component Categories**
# All Component Contributions

Please refresh the page to see the changes

<table>
<thead>
<tr>
<th>Component</th>
<th>Last modified</th>
<th>Component overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPI</td>
<td>15 June 2010</td>
<td>Advanced Configuration and Power Interface Component</td>
</tr>
<tr>
<td>CUDA</td>
<td>18 March 2011</td>
<td>Provides access to hardware counters inside NVIDIA GPUs through the CUDA / CUPTI interface</td>
</tr>
<tr>
<td>Component Example</td>
<td>26 May 2011</td>
<td>This is a short description of what your component does.</td>
</tr>
<tr>
<td>CoreTemp</td>
<td>11 March 2011</td>
<td>Access hardware sensors through the coretemp sysfs interface</td>
</tr>
<tr>
<td>Coretemp freebsd</td>
<td>11 March 2011</td>
<td>Access hardware temperature sensors on FreeBSD</td>
</tr>
<tr>
<td>Example</td>
<td>23 May 2011</td>
<td>Example component code with 3 counters</td>
</tr>
<tr>
<td>Infiniband</td>
<td>18 June 2010</td>
<td>Infiniband Network Component</td>
</tr>
<tr>
<td>Lm-sensors</td>
<td>4 April 2011</td>
<td>Component interface for Lm-sensors system health measurement</td>
</tr>
<tr>
<td>Lustre</td>
<td>4 April 2011</td>
<td>Measure performance data on a Lustre filesystem</td>
</tr>
</tbody>
</table>
**Description**

The CUDA component is a hardware performance counter measurement technology for the NVIDIA CUDA platform which provides access to the hardware counters inside the GPU. PAPI CUDA is based on CUPTI support - shipped with CUDA 4.0rc - in the NVIDIA driver library. In any environment where the CUPTI-enabled driver is installed, the PAPI CUDA component can provide detailed performance counter information regarding the execution of GPU kernels.

**Implementation**

Use the "Download tarball" link on the provided page to download a tarball of the source code folder to your computer. Untar the folder and place it in the components directory of your PAPI source tree. Configure using "--with-component=cuda". Rebuild PAPI.

**Usage**

Use as directed :)

**Supported Platforms**

Linux platforms with CUDA 4.0 or greater and NVIDIA GPU cards installed.
For more information

• PAPI Website: http://icl.eecs.utk.edu/papi/
  • Software
  • Release notes
  • Documentation
  • Links to tools that use PAPI
  • Mailing/discussion list
• ICL Website: http://icl.eecs.utk.edu/
  • Job openings!
• Questions?