Porting PAPI to the Cloud
(and Other Places)

Dan Terpstra

with a little help from:

Vince Weaver,

Heike Jagode,

James Ralph,

& Kiran Kasichayanula

ICL UT INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE

Center for Scalable Application Development Software
Performance Tools Aug 1 – 4, 2011

Alphabet Soup

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

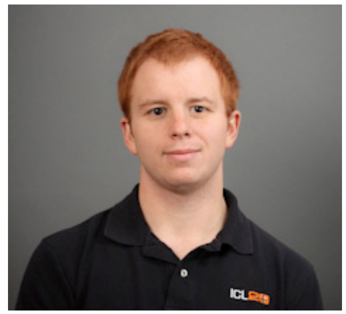
- **PAPI-C**
- **PAPI-V**
- **PAPI-G**
- **PAPI-U**

- **PAPI-R??**

PAPI Team

Vince Weaver
Post Doc

Heike Jagode
PhD Student



James Ralph
Staff

Kiran
Kasichayanula
Masters Student

Piotr Luszczek
Gadfly

Jack Dongarra

Shirley Moore

Dan Terpstra

Phil Mucci

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

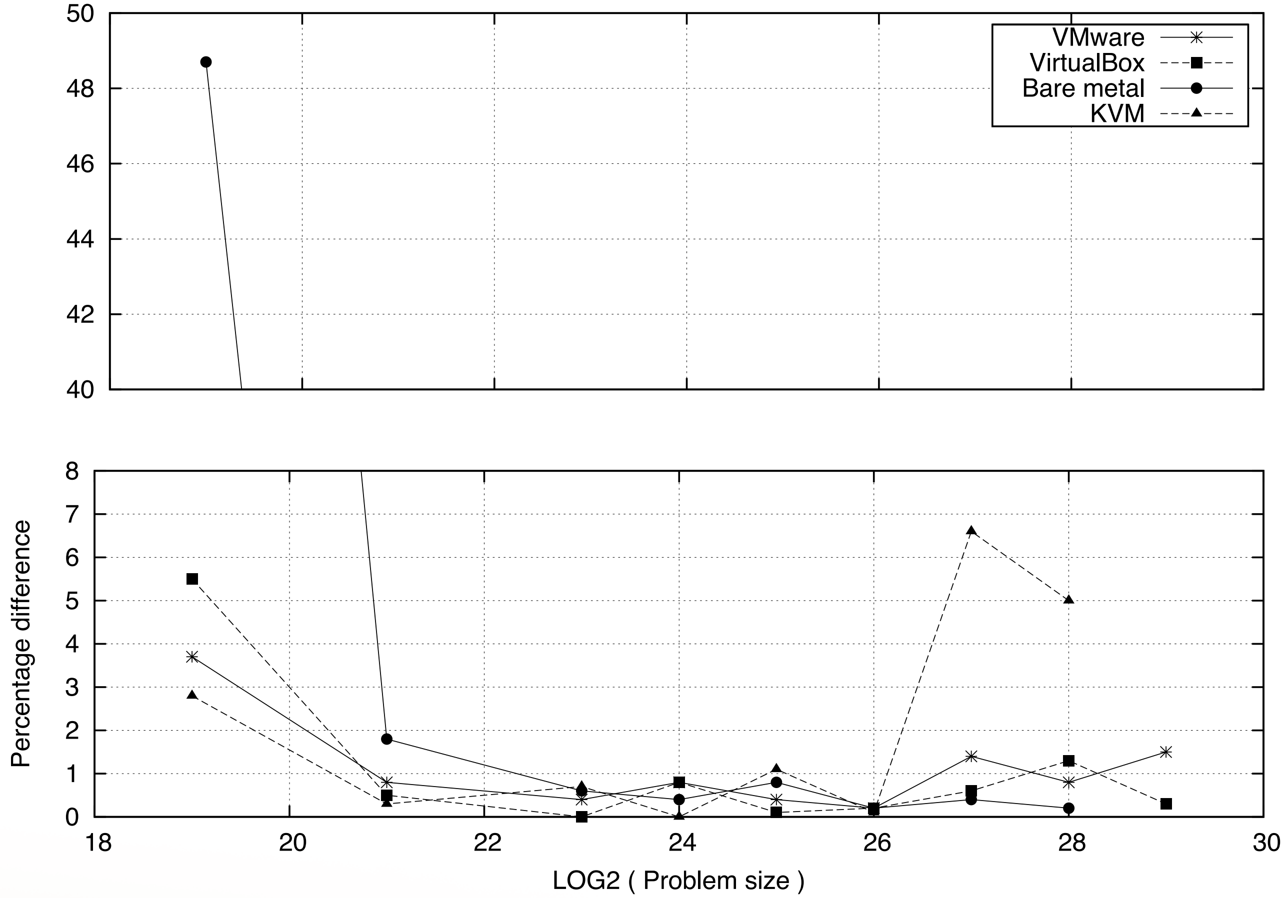
PAPI and the Cloud Computing Future

- 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

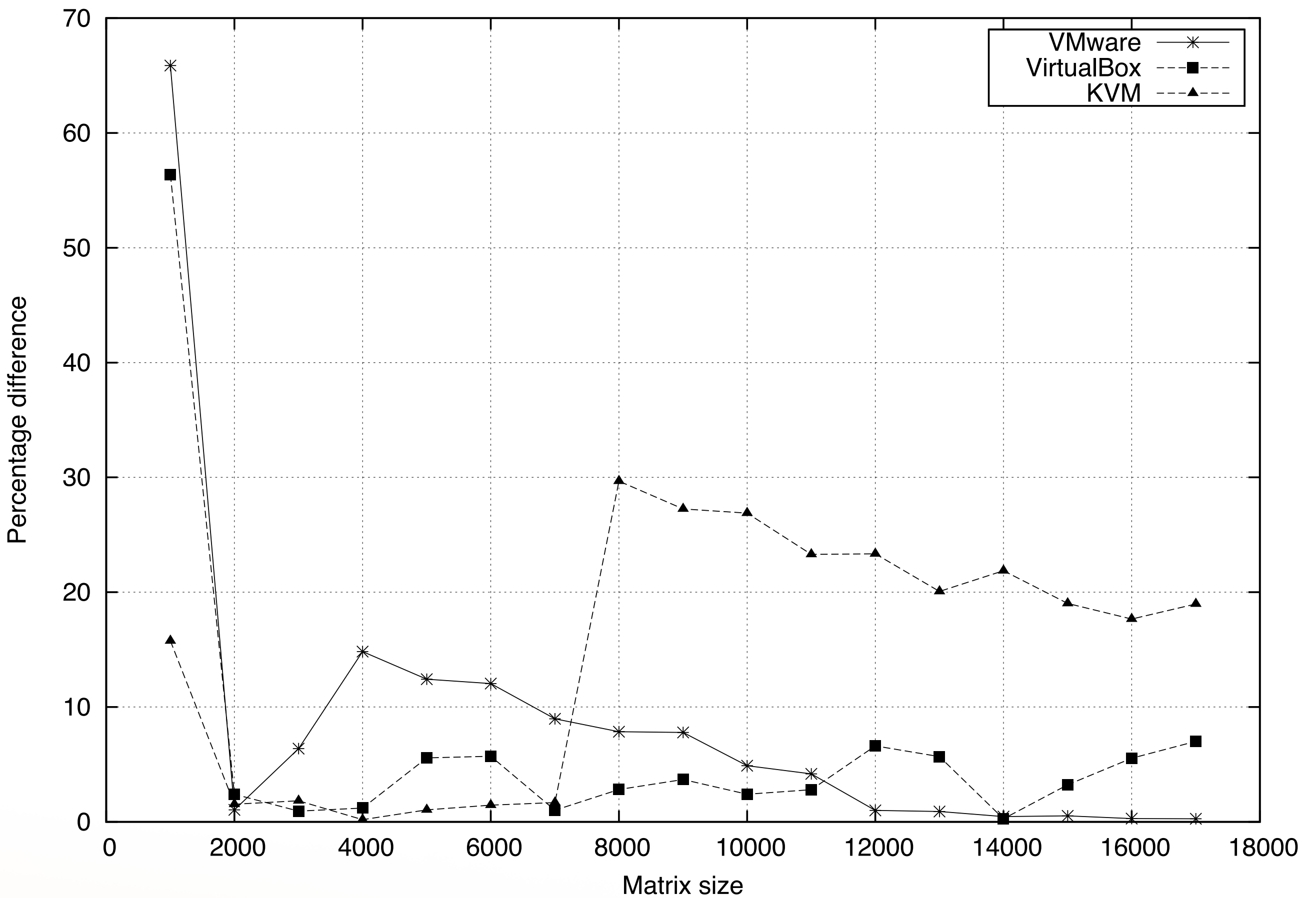
- Virtualization makes time measurements difficult; virtualized time can run faster or slower than wall-clock time in unpredictable ways
- Hardware performance counter readings require the co-operation of both the operating system and hypervisor. Support for this is still under development.
- Virtualized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

Virtual Time vs Wall Clock



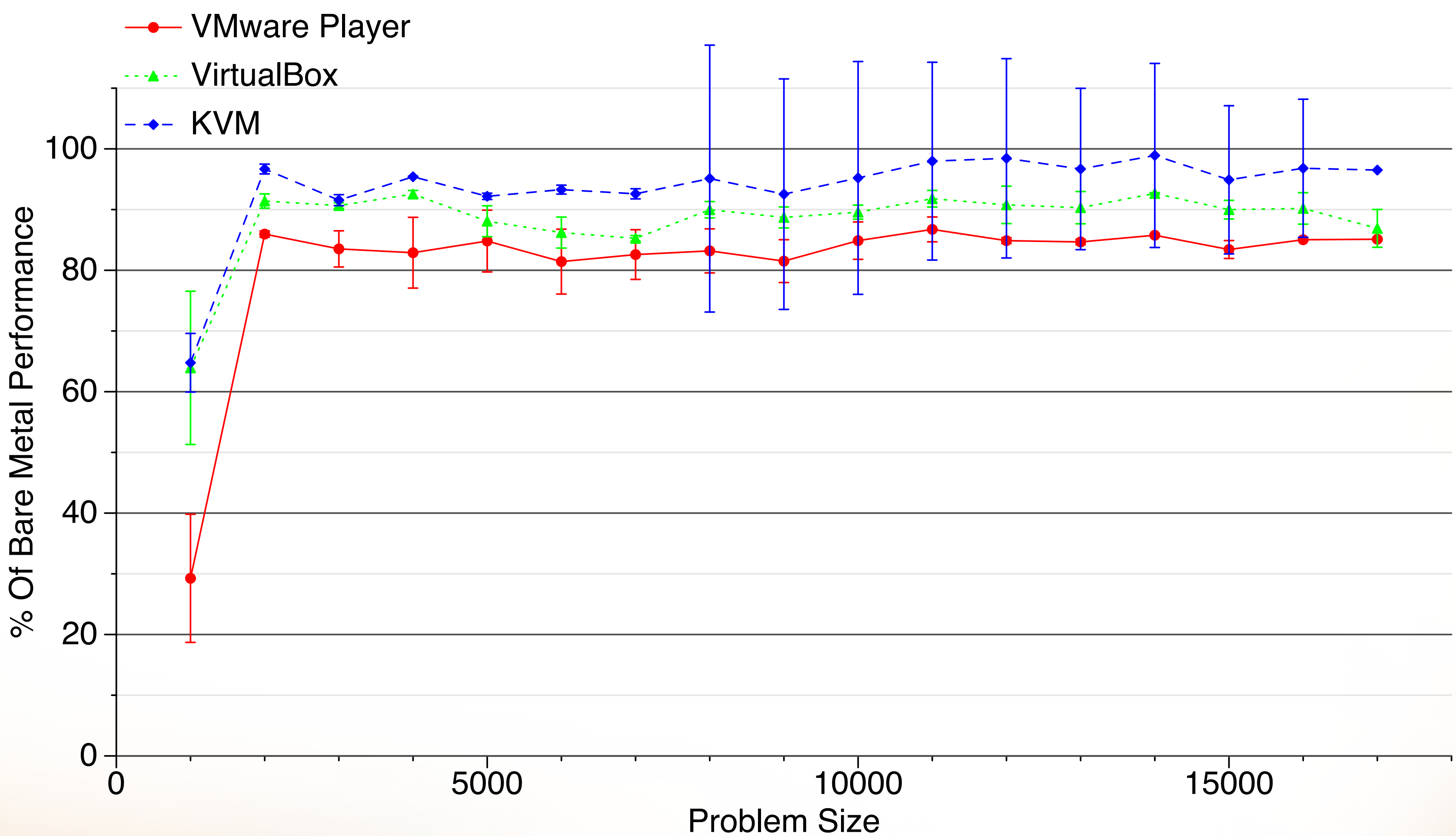
Variation in percentage difference between the measured CPU and wall clock times for MPIRandomAccess test of HPC Challenge. The vertical axis has been split to offer a better resolution for the majority of data points.

Accuracy Drift

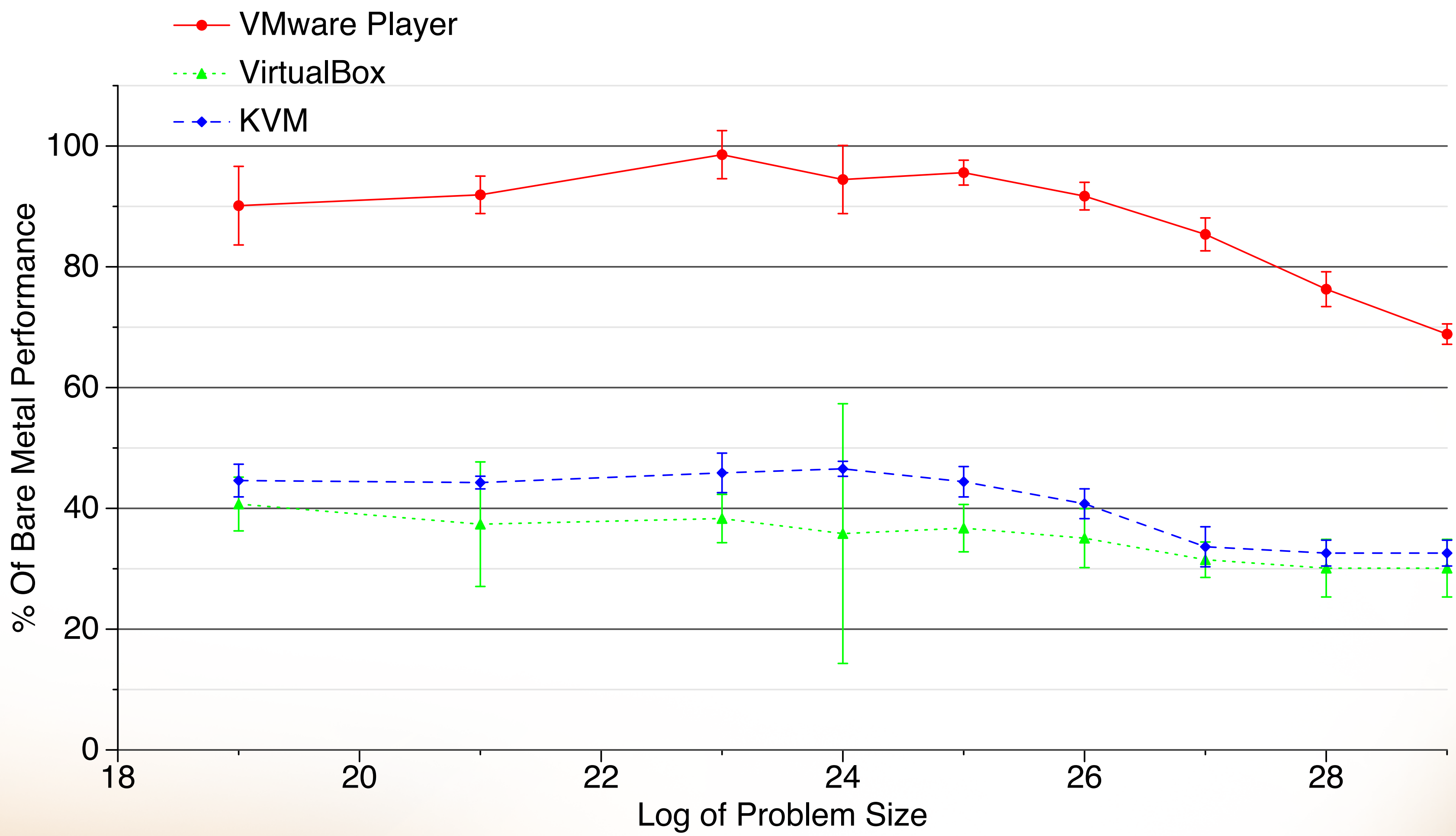


Variation in percentage difference between the measured wall clock times for HPL (a computationally intensive problem) for ascending and descending orders of problem sizes during execution.

HPL: Compute Intensive



MPIRandomAccess: Communication Intensive



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

Perfctr-Xen: a framework for performance counter virtualization.
Ruslan Nikolaev and Godmar Back.
In Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '11). ACM, New York, NY, USA, 15-26.
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`

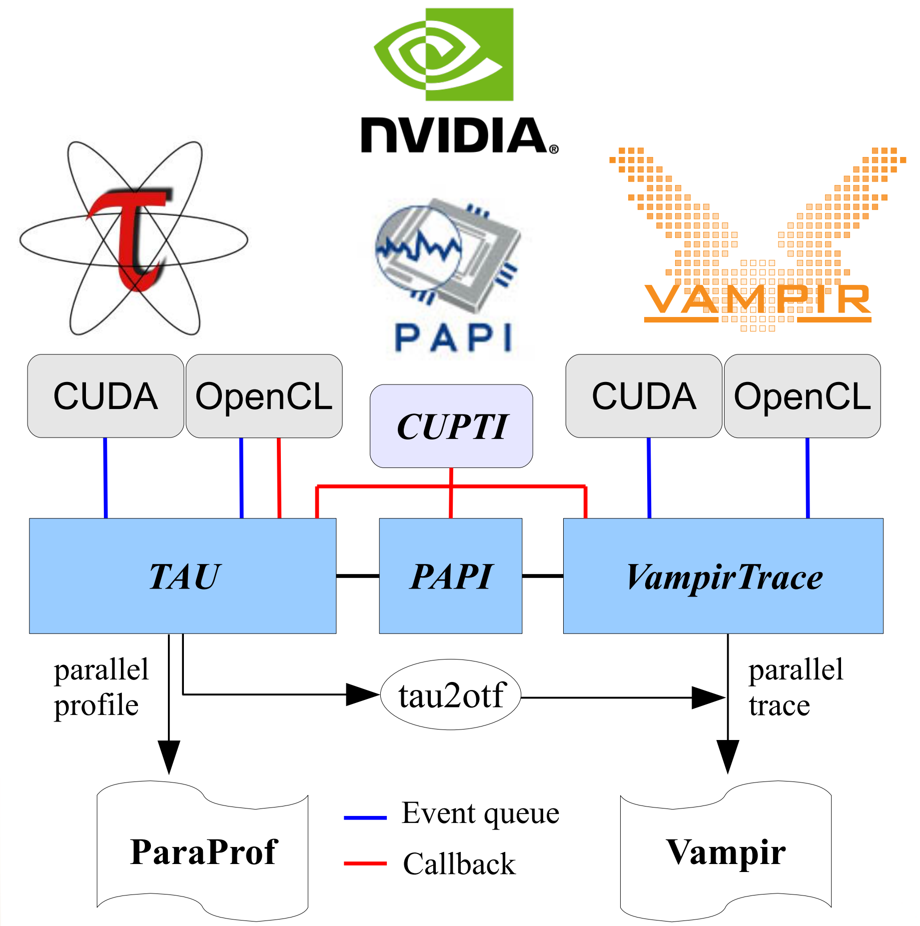
PAPI CUDA Component

Portion of CUDA events available on IG (GeForce GTX, Tesla C870)

…

Event Code	Symbol	Long Description
0x44000000	CUDA.GeForce_GTX_480.gpc0.local_load	# executed local load instructions per warp on a multiprocessor
0x44000001	CUDA.GeForce_GTX_480.gpc0.local_store	# executed local store instructions per warp on a multiprocessor
0x44000002	CUDA.GeForce_GTX_480.gpc0.gld_request	# executed global load instructions per warp on a multiprocessor
0x44000003	CUDA.GeForce_GTX_480.gpc0.gst_request	# executed global store instructions per warp on a multiprocessor
0x44000004	CUDA.GeForce_GTX_480.gpc0.shared_load	# executed shared load instructions per warp on a multiprocessor
0x44000005	CUDA.GeForce_GTX_480.gpc0.shared_store	# executed shared store instructions per warp on a multiprocessor
0x44000006	CUDA.GeForce_GTX_480.gpc0.branch	# branches taken by threads executing a kernel
0x44000007	CUDA.GeForce_GTX_480.gpc0.divergent_branch	# divergent branches within a warp
0x4400000b	CUDA.GeForce_GTX_480.gpc0.active_cycles	# cycles a multiprocessor has at least one active warp
0x4400000c	CUDA.GeForce_GTX_480.gpc0.sm_cta_launched	# thread blocks launched on a multiprocessor
0x4400000d	CUDA.GeForce_GTX_480.gpc0.l1_local_load_hit	# local load hits in L1 cache
0x4400000e	CUDA.GeForce_GTX_480.gpc0.l1_local_load_miss	# local load misses in L1 cache
0x44000011	CUDA.GeForce_GTX_480.gpc0.l1_global_load_hit	# global load hits in L1 cache
0x4400002e	CUDA.Tesla_C870.domain_a.tex_cache_hit	# texture cache misses
0x4400002f	CUDA.Tesla_C870.domain_a.tex_cache_miss	# texture cache hits
0x44000034	CUDA.Tesla_C870.domain_b.local_load	# local memory load transactions
0x44000037	CUDA.Tesla_C870.domain_b.branch	# branches taken by threads executing a kernel
0x44000038	CUDA.Tesla_C870.domain_b.divergent_branch	# divergent branches within a warp
0x44000039	CUDA.Tesla_C870.domain_b.instructions	# instructions executed

Tool interoperability

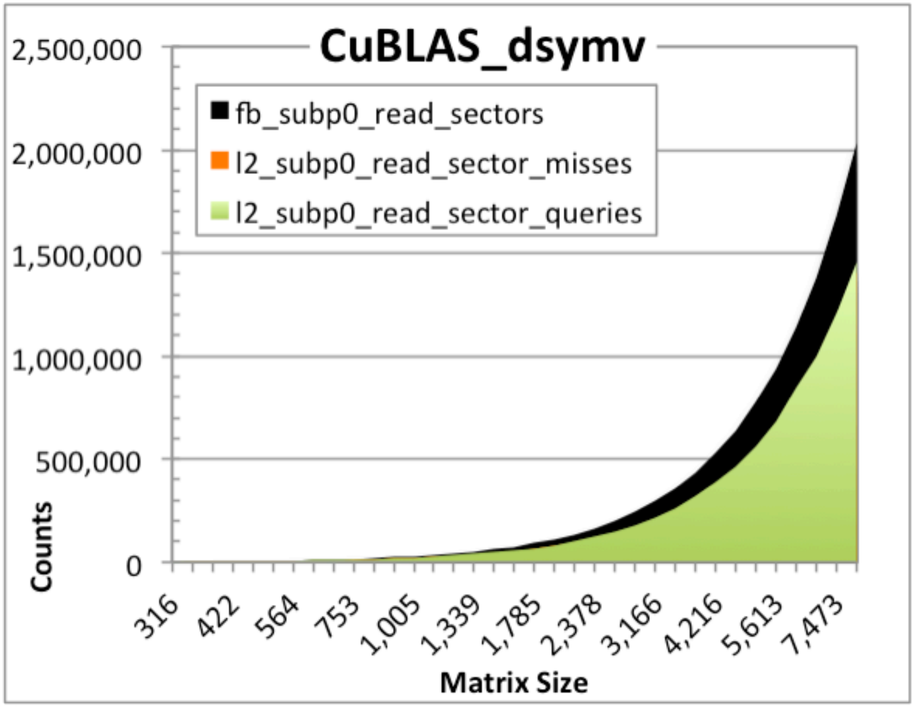


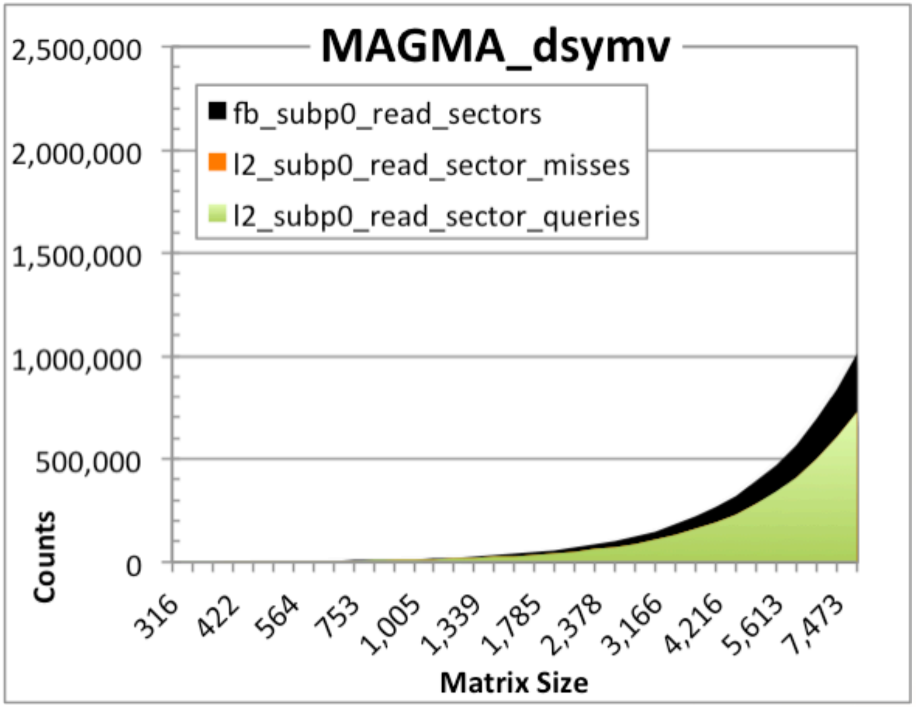
MAGMA versus CUBLAS: SYMV

- 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

ICL UT

CUDA performance counters for read behavior
(as measured by PAPI)

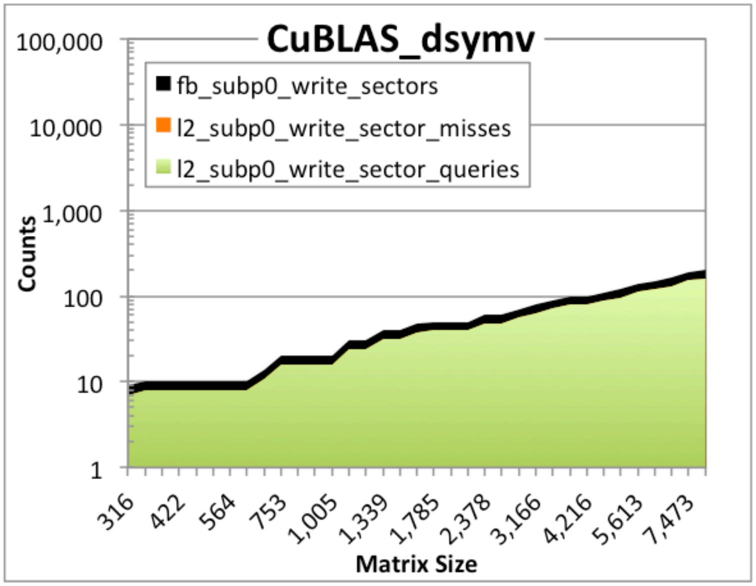


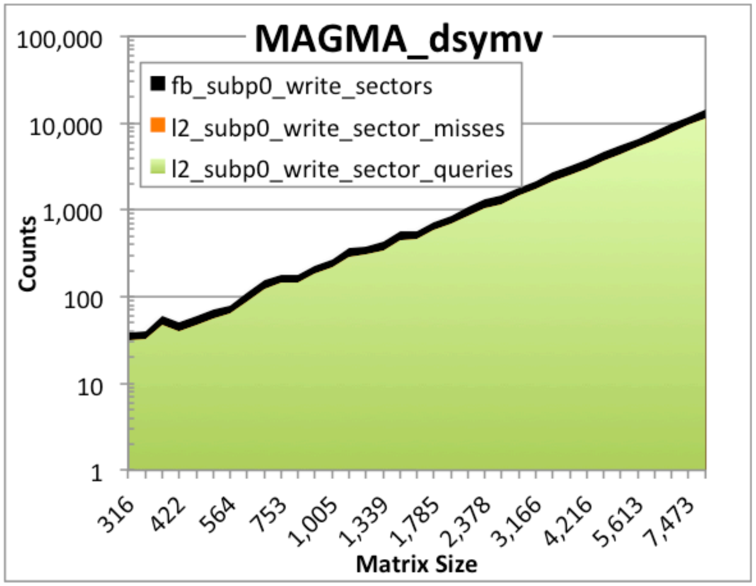


# of read requests from L1 to L2 (green), which is equal to # of read misses in L2 (orange); number of read requests from L2 to DRAM (black) for `CUBLAS_dsymv` (left) and `MAGMA_dsymv` (right)

CUDA performance counters for write behavior

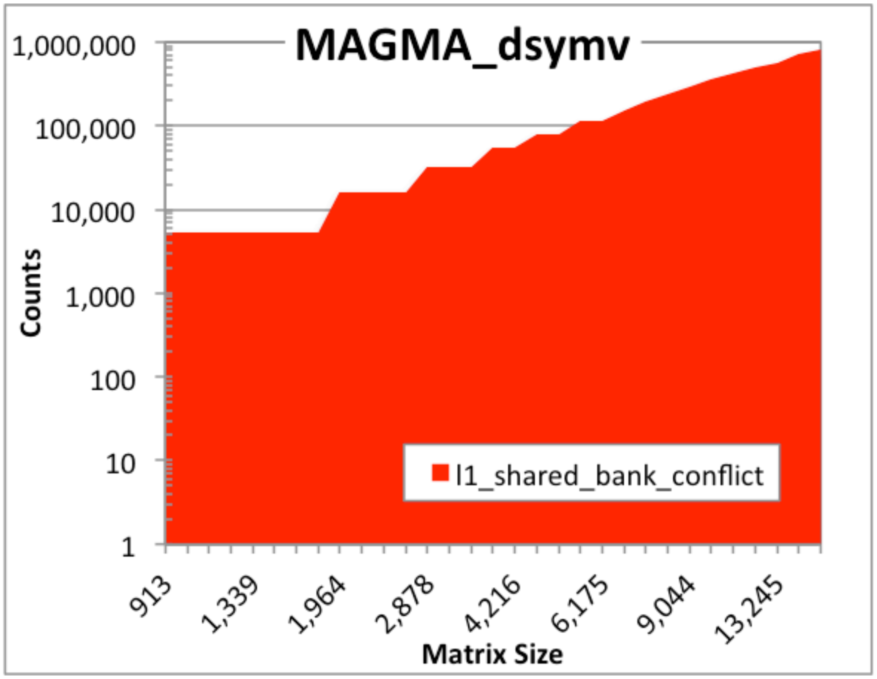
(as measured by PAPI)

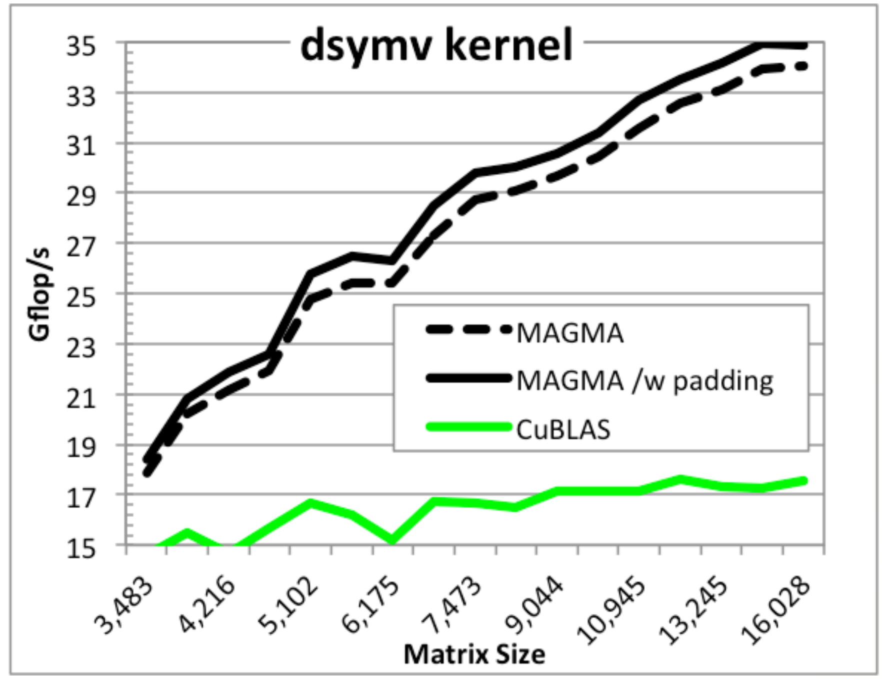




# of write requests from L1 to L2 (green), which is equal to # of write misses in L2 (orange); # of write requests from L2 to DRAM (black) for `CUBLAS_dsymv` (left) and `MAGMA_dsymv` (right)

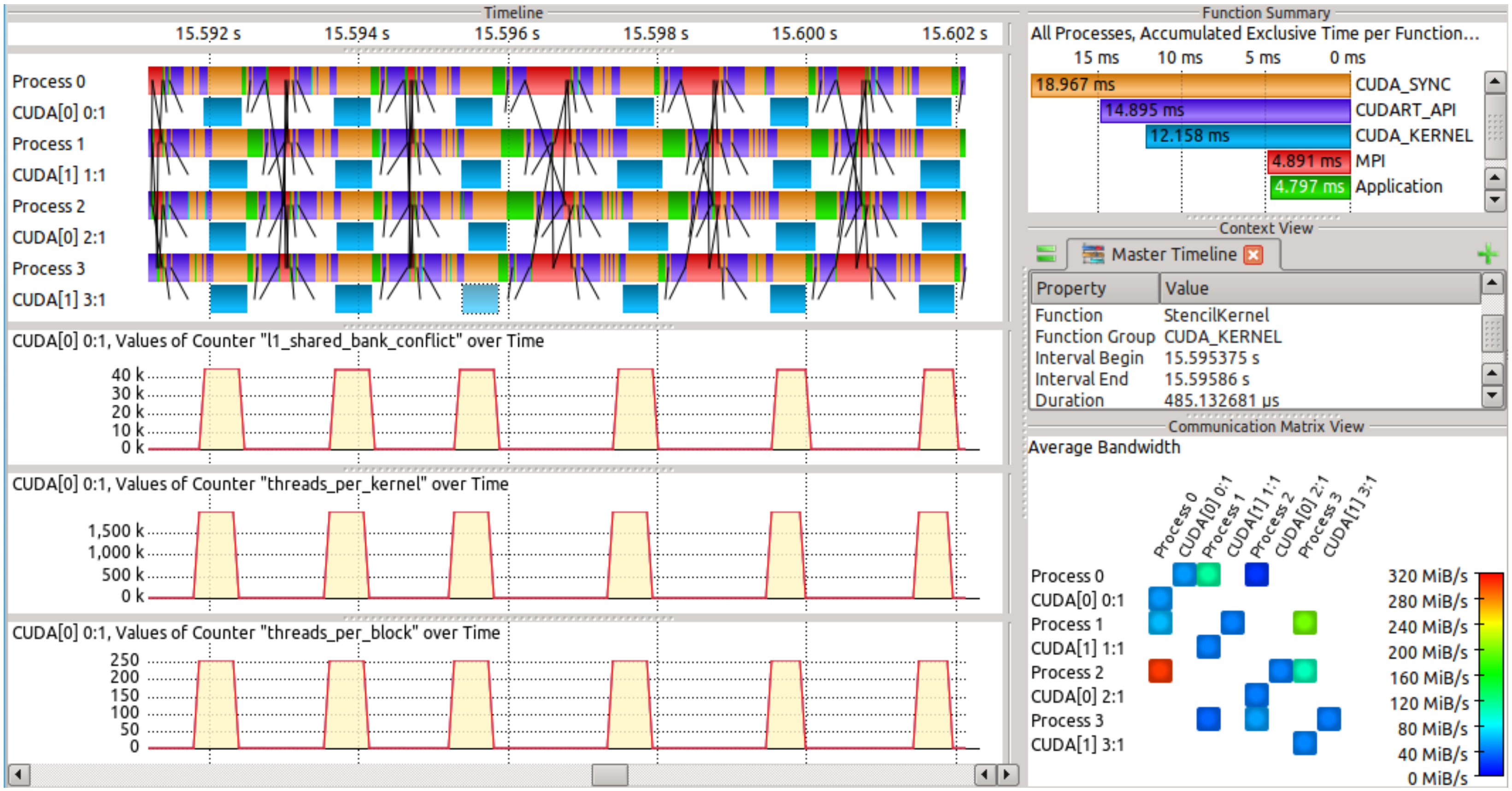
CUDA performance counter for L1 behavior
(as measured by PAPI)





# 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

Shirley Moore and James Ralph
ICCS 2011 Workshop on Tools for Program Development and Analysis in Computational Science June 1, 2011

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 | /

MB/s	STREAM output	Counters	% Delta
Copy	2227	2204	-1%
Scale	2332	2333	0%
Add	2471	2326	-6%
Triad	2473	2312	-6%

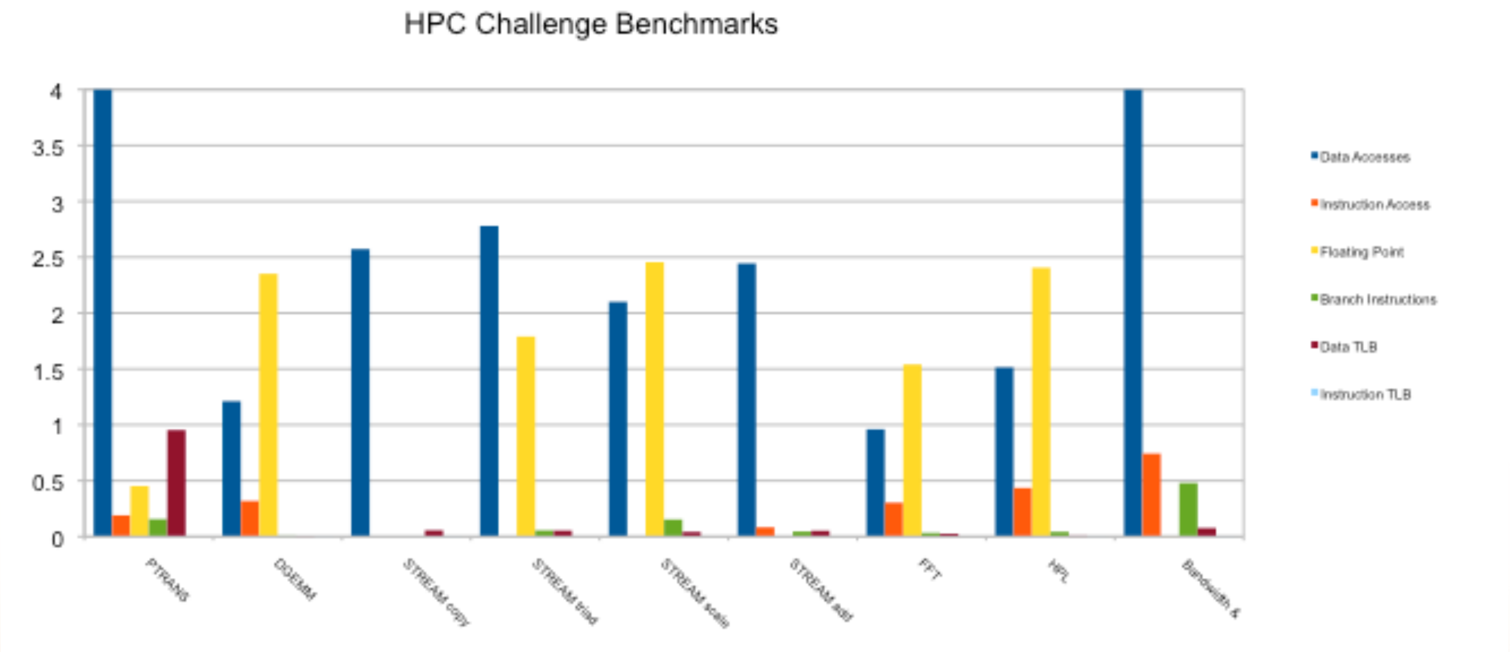
ICL UT

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
- Mulitplexing or multiple runs if necessary
- PAPI Utilities for enumerating / post-processing

PerfExpert LCPI

- *PerfExpert: An easy-to-use performance diagnosis tools for HPC applications*, in SC'10, New Orleans, 2010



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

PAPI

- Home
- Overview
- News
- Software
- Publications
- FAQ
- Links
- People
- Partners
- Documentation, etc
- Contact
- Software Archive
- Supported Platforms
- Tools
- User Forum
- Bug Reports
- Component Repository

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.

Latest PAPI News

2011-05-13
PAPI 4.1.3 Update

2011-05-12
PAPI 4.1.3 Now Available

2011-03-12
PAPI CUDA Component

2011-01-21
PAPI 4.1.2.1 Now Available

2011-01-20
PAPI 4.1.2 Now Available

Sponsored By:

Industry Support From: AMD, hp, IBM, intel, Microsoft

PAPI

PAPI Component Repository

Navigation

Main page
Help

Search

Go   Search

Toolbox

What links here
Related changes
Upload file
Special pages
Printable version
Permanent link

page   discussion   edit   history   delete   move   unprotect   refresh

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 [edit]

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.

Create or edit

Component Categories [edit]

Networks | GPU | File Systems | Power | System Health | Specialty CPU | Miscellaneous | Experimental

All Component Contributions [edit]

Please refresh the page to see the changes

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

Contents [hide]

CUDA	
Author(s)	PAPI team
Version	PAPI current
Last modified	2011/03/18
Author support	Yes
Component overview	Provides access to hardware counters inside NVIDIA GPUs through the CUDA / CUPTI interface
Source Code	http://icl.cs.utk.edu/viewcvs/viewcvs.cgi/PAPI/papi/src/components/cuda/

Description [edit]

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 [edit]

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 [edit]

Use as directed :)

Supported Platforms [edit]

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?