PAPI – Performance API

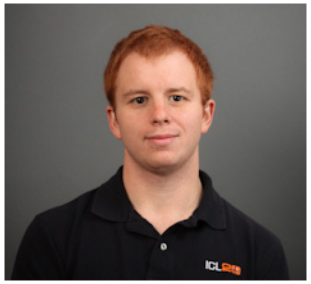
Shirley Moore
shirley@eecs.utk.edu

8th VI-HPS Tuning Workshop
5-9 September 2011

PAPI Team

Vince Weaver
Post Doc

Heike Jagode
PhD Candidate


James Ralph
Staff

Piotr Luszczek
Staff

Kiran Kasichayanula
Masters Student

Matt Johnson
Grad Student

John Nelson
Grad Student

Jack Dongarra
Distinguished Prof.

Shirley Moore
Research Prof.

Dan Terpstra
Staff

Phil Mucci
Consultant

Outline

Hardware Counters

Hardware performance counters available on most modern microprocessors can provide insight into:

1. Whole program timing
2. Instructions per cycle
3. Floating point efficiency
4. Cache behaviors
5. Branch behaviors
6. Memory and resource access patterns
7. Pipeline stalls

Hardware counter information can be obtained with:

1. Subroutine or basic block resolution
2. Process or thread attribution

What's PAPI?

- Middleware to provide a consistent programming interface for the performance counter hardware found in most major micro-processors
- Countable events are defined in two ways:
  - Platform-neutral preset events
  - Platform-dependent native events
- Presets can be derived from multiple native events.
- All events are referenced by name and collected in EventSets.
- Events can be multiplexed if physical counters are limited.
- Statistical sampling implemented by:
  - Hardware overflow if supported by the platform
  - Software overflow with timer driven sampling
- User can insert PAPI library calls into source code or use higher level tool (e.g., TAU, Scalasca, Vampirtrace).

Where's PAPI

- PAPI runs on most modern processors and operating systems of interest to HPC:
  - IBM POWER / AIX / Linux
  - Blue Gene / L / P / (Q)
  - Intel Pentium, Core2, Nehalem, SandyBridge / Linux
  - Intel Itanium / Linux
  - AMD Athlon, Opteron, (Bulldozer) / Linux
  - Cray XT(n) / CLE
  - Altix, Sparc, Niagara …
  - ARM Cortex
  - NVIDIA / CUDA 4

PAPI Utilities: *papi_cost*

```
$ utils/papi_cost -h
This is the PAPI cost program.
It computes min / max / mean / std. deviation for PAPI start/stop
   pairs; for PAPI reads, and for PAPI_accums.
   Usage:

    cost [options] [parameters]
    cost TESTS_QUIET

Options:

  -b BINS       set the number of bins for the graphical
                 distribution of costs. Default: 100
  -d            show a graphical distribution of costs
  -h            print this help message
  -s            show number of iterations above the first
                 10 std deviations
  -t THRESHOLD  set the threshold for the number of
                 iterations. Default: 100,000
```

PAPI Utilities: *papi_cost*

```
$ utils/papi_cost
Cost of execution for PAPI start/stop and PAPI read.
This test takes a while. Please be patient...
Performing start/stop test...

Total cost for PAPI_start/stop(2 counters) over 1000000 iterations
min cycles    : 63
max cycles    : 17991
mean cycles   : 69.000000
std deviation: 34.035263
 Performing start/stop test...

Performing read test...

Total cost for PAPI_read(2 counters) over 1000000 iterations
min cycles    : 288
max cycles    : 102429
mean cycles   : 301.000000
std deviation: 144.694053
 cost.c                                     PASSED
```

PAPI Utilities: *papi_cost*

```
Cost distribution profile

      63:**************************** 999969 counts ****************************
     153:
     243:
     [...]
    1863:
    1953:********************
    2043:
    2133:********************
    2223:
    2313:
    2403:********************
    2493:********************
    2583:****************************************
    2673:****************************************
    2763:****************************************************************************
    2853:****************************************
    2943:
    3033:********************
    3123:****************************************
    3213:****************************************
    3303:
    3393:
    3483:
    3573:
    3663:********************
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail -h
Usage: utils/papi_avail [options]
Options:

General command options:
    -a, --avail   Display only available preset events
    -d, --detail  Display detailed information about all preset events
    -e EVENTNAME  Display detail information about specified preset or native event
    -h, --help    Print this help message

Event filtering options:
        --br          Display branch related PAPI preset events
        --cache       Display cache related PAPI preset events
        --cnd         Display conditional PAPI preset events
        --fp          Display Floating Point related PAPI preset events
        --ins         Display instruction related PAPI preset events
        --idl         Display Stalled or Idle PAPI preset events
        --l1          Display level 1 cache related PAPI preset events
        --l2          Display level 2 cache related PAPI preset events
        --l3          Display level 3 cache related PAPI preset events
        --mem         Display memory related PAPI preset events
        --msc         Display miscellaneous PAPI preset events
        --tlb         Display Translation Lookaside Buffer PAPI preset events

This program provides information about PAPI preset and native events.
PAPI preset event filters can be combined in a logical OR.
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version              : 4.0.0.0
Vendor string and code    : GenuineIntel (1)
Model string and code     : Intel Core i7 (21)
CPU Revision              : 5.000000
CPUID Info                : Family: 6  Model: 26  Stepping: 5
CPU Megahertz             : 2926.000000
CPU Clock Megahertz       : 2926
Hdw Threads per core      : 1
Cores per Socket          : 4
NUMA Nodes                : 2
CPU's per Node            : 4
Total CPU's               : 8
Number Hardware Counters  : 7
Max Multiplex Counters    : 32
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

    [MORE...]
```

PAPI Utilities: *papi_avail*

```
      [CONTINUED…]

--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

    Name         Code     Avail Deriv Description (Note)
PAPI_L1_DCM   0x80000000   No     No    Level 1 data cache misses
PAPI_L1_ICM   0x80000001   Yes    No    Level 1 instruction cache misses
PAPI_L2_DCM   0x80000002   Yes    Yes   Level 2 data cache misses

[…]

PAPI_VEC_SP   0x80000069   Yes    No    Single precision vector/SIMD instructions
PAPI_VEC_DP   0x8000006a   Yes    No    Double precision vector/SIMD instructions
-------------------------------------------------------------------------
Of 107 possible events, 34 are available, of which 9 are derived.

avail.c                                          PASSED
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail -e PAPI_FP_OPS
[…]
---------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Event name:                    PAPI_FP_OPS
Event Code:                    0x80000066
Number of Native Events:       2
Short Description:            |FP operations|
Long Description:             |Floating point operations|
Developer's Notes:            ||
Derived Type:                 |DERIVED_ADD|
Postfix Processing String:    ||
 Native Code[0]: 0x4000801b |FP_COMP_OPS_EXE:SSE_SINGLE_PRECISION|
 Number of Register Values: 2
 Register[ 0]: 0x0000000f |Event Selector|
 Register[ 1]: 0x00004010 |Event Code|
 Native Event Description: |Floating point computational micro-ops, masks:SSE* FP single
    precision Uops|

 Native Code[1]: 0x4000081b |FP_COMP_OPS_EXE:SSE_DOUBLE_PRECISION|
 Number of Register Values: 2
 Register[ 0]: 0x0000000f |Event Selector|
 Register[ 1]: 0x00008010 |Event Code|
 Native Event Description: |Floating point computational micro-ops, masks:SSE* FP double
    precision Uops|
-------------------------------------------------------------------------
```

PAPI Utilities: *papi_native_avail*

```
UNIX> utils/papi_native_avail
Available native events and hardware information.
--------------------------------------------------------------------------------
[…]
Event Code   Symbol  | Long Description |
--------------------------------------------------------------------------------
0x40000010   BR_INST_EXEC  | Branch instructions executed                    |
  40000410   :ANY  | Branch instructions executed                            |
  40000810   :COND  | Conditional branch instructions executed               |
  40001010   :DIRECT  | Unconditional branches executed                      |
  40002010   :DIRECT_NEAR_CALL  | Unconditional call branches executed       |
  40004010   :INDIRECT_NEAR_CALL  | Indirect call branches executed          |
  40008010   :INDIRECT_NON_CALL  | Indirect non call branches executed       |
  40010010   :NEAR_CALLS  | Call branches executed                           |
  40020010   :NON_CALLS  | All non call branches executed                    |
  40040010   :RETURN_NEAR  | Indirect return branches executed               |
  40080010   :TAKEN  | Taken branches executed                               |
--------------------------------------------------------------------------------
0x40000011   BR_INST_RETIRED  | Retired branch instructions                  |
  40000411   :ALL_BRANCHES  | Retired branch instructions (Precise Event)    |
  40000811   :CONDITIONAL  | Retired conditional branch instructions (Precise |
           | Event)                                                          |
  40001011   :NEAR_CALL  | Retired near call instructions (Precise Event)    |
--------------------------------------------------------------------------------
[…]
```

PAPI Utilities: *papi_native_avail*

```
UNIX> utils/papi_native_avail -e DATA_CACHE_REFILLS
Available native events and hardware information.
--------------------------------------------------------------------------------
[...]
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Event name:                     DATA_CACHE_REFILLS
Event Code:                     0x4000000b
Number of Register Values:      2
Description:                   |Data Cache Refills from L2 or System|
 Register[ 0]:    0x0000000f   |Event Selector|
 Register[ 1]:    0x00000042   |Event Code|

Unit Masks:
 Mask Info:                    |:SYSTEM|Refill from System|
  Register[ 0]:   0x0000000f   |Event Selector|
  Register[ 1]:   0x00000142   |Event Code|
 Mask Info:                    |:L2_SHARED|Shared-state line from L2|
  Register[ 0]:   0x0000000f   |Event Selector|
  Register[ 1]:   0x00000242   |Event Code|
 Mask Info:                    |:L2_EXCLUSIVE|Exclusive-state line from L2|
  Register[ 0]:   0x0000000f   |Event Selector|
  Register[ 1]:   0x00000442   |Event Code|
```

PAPI Utilities: *papi_event_chooser*

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events that can be added with given events
--------------------------------------------------------------------
[...]
--------------------------------------------------------------------
    Name         Code     Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No    Level 1 data cache misses
PAPI_L1_ICM  0x80000001  No    Level 1 instruction cache misses
PAPI_L2_ICM  0x80000003  No    Level 2 instruction cache misses
[...]
PAPI_L1_DCA  0x80000040  No    Level 1 data cache accesses
PAPI_L2_DCR  0x80000044  No    Level 2 data cache reads
PAPI_L2_DCW  0x80000047  No    Level 2 data cache writes
PAPI_L1_ICA  0x8000004c  No    Level 1 instruction cache accesses
PAPI_L2_ICA  0x8000004d  No    Level 2 instruction cache accesses
PAPI_L2_TCA  0x80000059  No    Level 2 total cache accesses
PAPI_L2_TCW  0x8000005f  No    Level 2 total cache writes
PAPI_FML_INS 0x80000061  No    Floating point multiply instructions
PAPI_FDV_INS 0x80000063  No    Floating point divide instructions
--------------------------------------------------------------
Total events reported: 34
event_chooser.c                          PASSED
```

PAPI Utilities: *papi_event_chooser*

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM
Event Chooser: Available events that can be added with given events.
--------------------------------------------------------------------------------
[...]
---------------------------------------------------------------------------------

    Name        Code    Deriv Description (Note)
PAPI_TOT_INS 0x80000032  No   Instructions completed
PAPI_TOT_CYC 0x8000003b  No   Total cycles
--------------------------------------------------------------------------
Total events reported: 2
event_chooser.c                           PASSED
```

PAPI Utilities: *papi_command_line*

```
$ papi_command_line PAPI_FP_OPS
Successfully added: PAPI_FP_OPS

PAPI_FP_OPS :   100000000

----------------------------------
Verification: None.
 This utility lets you add events from the command line interface to see if they work.
command_line.c                           PASSED


$ papi_command_line PAPI_FP_OPS PAPI_L1_DCA
Successfully added: PAPI_FP_OPS
Successfully added: PAPI_L1_DCA

PAPI_FP_OPS :   100000000
PAPI_L1_DCA :   120034404

----------------------------------
Verification: None.
 This utility lets you add events from the command line interface to see if they work.
command_line.c                           PASSED
```

Example – Matrix Multiplication

```
#define ROWS 1000                // Number of rows in each matrix
#define COLUMNS 1000             // Number of columns in each matrix
```

```
void classic_matmul()
{
   // Multiply the two matrices
   int i, j, k;
   for (i = 0; i < ROWS; i++) {
   fofor (j = 0; j < COLUMNS; j++) {
   float sum = 0.0;
   for for (k = 0; k < COLUMNS; k++) {
   sum += sum +=
   matrix_   matrix_a[i][k] * matrix_b[k][j];
   }
   matrix_c[i][j] = sum;
   }
   }
}
```

```
void interchanged_matmul()
{
   // Multiply the two matrices
   int i, j, k;
   for (i = 0; i < ROWS; i++) {
     for (k = 0; k < COLUMNS; k++) {
       for (j = 0; j < COLUMNS; j++) {
          matrix_c[i][j] +=
             matrix_a[i][k] * matrix_b[k][j];
       }
     }
   }
}

// Note that the nesting of the innermost loops
// has been changed. The index variables j and k
// change the most frequently and the access
// pattern through the operand matrices is
// sequential using a small stride (one.) This
// change improves access to memory data through
// the data cache. Data translation lookaside
// buffer (DTLB) behavior is also improved.
```

IPC – instructions per cycle

```
Measurement                           Classic mat_mul          Reordered mat_mul
============================================================================

PAPI_IPC Test (PAPI_ipc)
Real time                                13.6093 sec             2.9796 sec
Processor time                           13.5359 sec             2.9556 sec
IPC                                       0.3697                 1.6936
Instructions                              9007035063             9009011383

High Level IPC Test (PAPI_{start,stop}_counters)
Real time                                13.6106 sec             2.9762 sec
IPC                                       0.3697                 1.6939
PAPI_TOT_CYC                             24362605525             5318626915
PAPI_TOT_INS                              9007034503             9009011245

Low  Level IPC Test (PAPI low level calls)
Real time                                13.6113 sec             2.9772 sec
IPC                                       0.3697                 1.6933
PAPI_TOT_CYC                             24362750167             5320395138
PAPI_TOT_INS                              9007034381             9009011130
```

- **All three PAPI methods consistent**
- **Roughly 460% improvement in reordered code**

Data Cache Access

Data Cache Misses can be considered in 3 categories:

- **Compulsory: Occurs on first reference to a data item.**
  - **Prefetching**
- **Capacity: Occurs when the working set exceeds the cache capacity.**
  - **Spatial locality**
  - **Smaller working set (blocking/tiling algorithms)**
- **Conflict: Occurs when a data item is referenced after the cache line containing the item was evicted earlier.**
  - **Temporal locality**
  - **Data layout; memory access patterns**

L1 Data Cache Access

```
Measurement                                  Classic mat_mul          Reordered mat_mul
============================================================================

DATA_CACHE_ACCESSES                              2002807841           3008528961
DATA_CACHE_REFILLS:L2_MODIFIED:L2_OWNED:L2_EXCLUSIVE:L2_SHARED
                                                  205968263             60716301
DATA_CACHE_REFILLS_FROM_SYSTEM:MODIFIED:OWNED:EXCLUSIVE:SHARED
                                                   61970925              1950282
----------------------
PAPI_L1_DCA                                      2002808034           3008528895
PAPI_L1_DCM                                       268010587             62680818

Data Cache Request Rate                          0.2224 req/inst      0.3339 req/inst
Data Cache Miss Rate                             0.0298 miss/inst     0.0070 miss/inst
Data Cache Miss Ratio                            0.1338 miss/req      0.0208 miss/req
```

- **Two techniques**
  - **First uses native events**
  - **Second uses PAPI presets only**
- **~50% more requests from reordered code**
- **1/4 as many misses per instruction**
- **1/6 as many misses per request**

Branching

```
Measurement                          Classic mat_mul         Reordered mat_mul
============================================================================

PAPI_BR_INS                              1001028240              1001006987
PAPI_BR_MSP                                 1028256                 1006984
PAPI_BR_TKN                              1000027233              1000005980

Branch Rate                              0.1111 br/inst          0.1111 br/inst
Branch Miss Rate                         0.0001 miss/inst        0.0001 miss/inst
Branch Miss Ratio                        0.0010 miss/br          0.0010 miss/br

Branch Taken Rate                        0.1110 tkn/inst         0.1110 tkn/inst
Branch Taken Ratio                       0.9990 tkn/br           0.9990 tkn/br
Instr / Branch                           8.9978 inst/br          8.9999 inst/br
```

- **Uses all PAPI Presets!**
- **Branch behavior nearly identical in both codes**
- **Roughly 1 branch every 9 instructions**
- **1 miss per 1000 branches (remember ROWS?)**
- **Branching and branch misses can be reduced with loop unrolling, loop fusion and function in-lining.**

Performance Measurement Categories

- Efficiency
  - Instructions per cycle (IPC)
  - Memory bandwidth
- Caches
  - Data cache misses and miss ratio
  - Instruction cache misses and miss ratio
  - Last level cache misses and miss ratio
- Translation lookaside buffers (TLB)
  - Data TLB misses and miss ratio
  - Instruction TLB misses and miss ratio
- Control transfers
  - Branch mispredictions
  - Near return mispredictions
- Instruction counts for different categories
- Miscellaneous
  - Unaligned data access
  - Floating point exceptions
  - Hardware interrupts

PAPI Preset Events

- Preset Events
  - Platform independent set of over 100 events for application performance tuning
  - No standardization of the exact definition
  - Mapped to either single or combinations of native events on each platform
  - Use papi_avail utility to see what preset events are available on a given platform

PAPI_L2_DCR:	Level 1 data cache reads
PAPI_L2_DCW:	Level 1 data cache writes
PAPI_L2_DCM:	Level 1 data cache misses
PAPI_L2_ICH:	Level 1 instruction cache hits
PAPI_L2_ICA:	Level 1 instruction cache accesses
PAPI_L2_ICR:	Level 1 instruction cache reads
PAPI_L2_ICW:	Level 1 instruction cache writes
PAPI_L2_ICM:	Level 1 instruction cache misses
PAPI_L2_TCH:	Level 1 total cache hits
PAPI_L2_TCA:	Level 1 total cache accesses
PAPI_L2_TCR:	Level 1 total cache reads
PAPI_L2_TCW:	Level 1 total cache writes
PAPI_L2_TCM:	Level 1 cache misses
PAPI_L2_LDM:	Level 1 load misses
PAPI_L2_STM:	Level 1 store misses

Level 3 Cache

PAPI_L3_DCH:	Level 1 data cache hits
PAPI_L3_DCA:	Level 1 data cache accesses
PAPI_L3_DCR:	Level 1 data cache reads
PAPI_L3_DCW:	Level 1 data cache writes
PAPI_L3_DCM:	Level 1 data cache misses
PAPI_L3_ICH:	Level 1 instruction cache hits
PAPI_L3_ICA:	Level 1 instruction cache accesses
PAPI_L3_ICR:	Level 1 instruction cache reads
PAPI_L3_ICW:	Level 1 instruction cache writes
PAPI_L3_ICM:	Level 1 instruction cache misses
PAPI_L3_TCH:	Level 1 total cache hits
PAPI_L3_TCA:	Level 1 total cache accesses
PAPI_L3_TCR:	Level 1 total cache reads
PAPI_L3_TCW:	Level 1 total cache writes
PAPI_L3_TCM:	Level 1 cache misses
PAPI_L3_LDM:	Level 1 load misses
PAPI_L3_STM:	Level 1 store misses

Cache Sharing

PAPI_CA_SNP:	Requests for a snoop
PAPI_CA_SHR:	Requests for exclusive access to shared cache line

PAPI Native Events

- Native Events
  - Any event countable by the CPU or component
  - Same interface as for preset events
  - Use *papi_native_avail* utility to see all available native events
- Use *papi_event_chooser* utility to select a compatible set of events

PAPI High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (less additional code) than low-level
- Supports 8 calls in C or Fortran:

```
PAPI_start_counters      PAPI_stop_counters
PAPI_read_counters       PAPI_accum_counters
PAPI_num_counters        PAPI_flips
PAPI_ipc                 PAPI_flops
```

- See high level C and Fortran examples in PAPI source distribution

PAPI High-level Example

```
#include "papi.h”
#define NUM_EVENTS 2
long_long values[NUM_EVENTS];
unsigned int
  Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};

 /* Start the counters */
 PAPI_start_counters((int*)Events,NUM_EVENTS);

 /* What we are monitoring… */
 do_work();

 /* Stop counters and store results in values */
 retval = PAPI_stop_counters(values,NUM_EVENTS);
```

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- Obtain information about the executable, the hardware, and the memory environment
- Multiplexing
- Callbacks on counter overflow
- Profiling
- About 60 functions
- See low level C and Fortran examples in PAPI distribution

PAPI Low-level Example

```
#include "papi.h"
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC};
int EventSet;
long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add Flops and total cycles to the eventset */
retval = PAPI_add_events(EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);

do_work();  /* What we want to monitor*/

/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

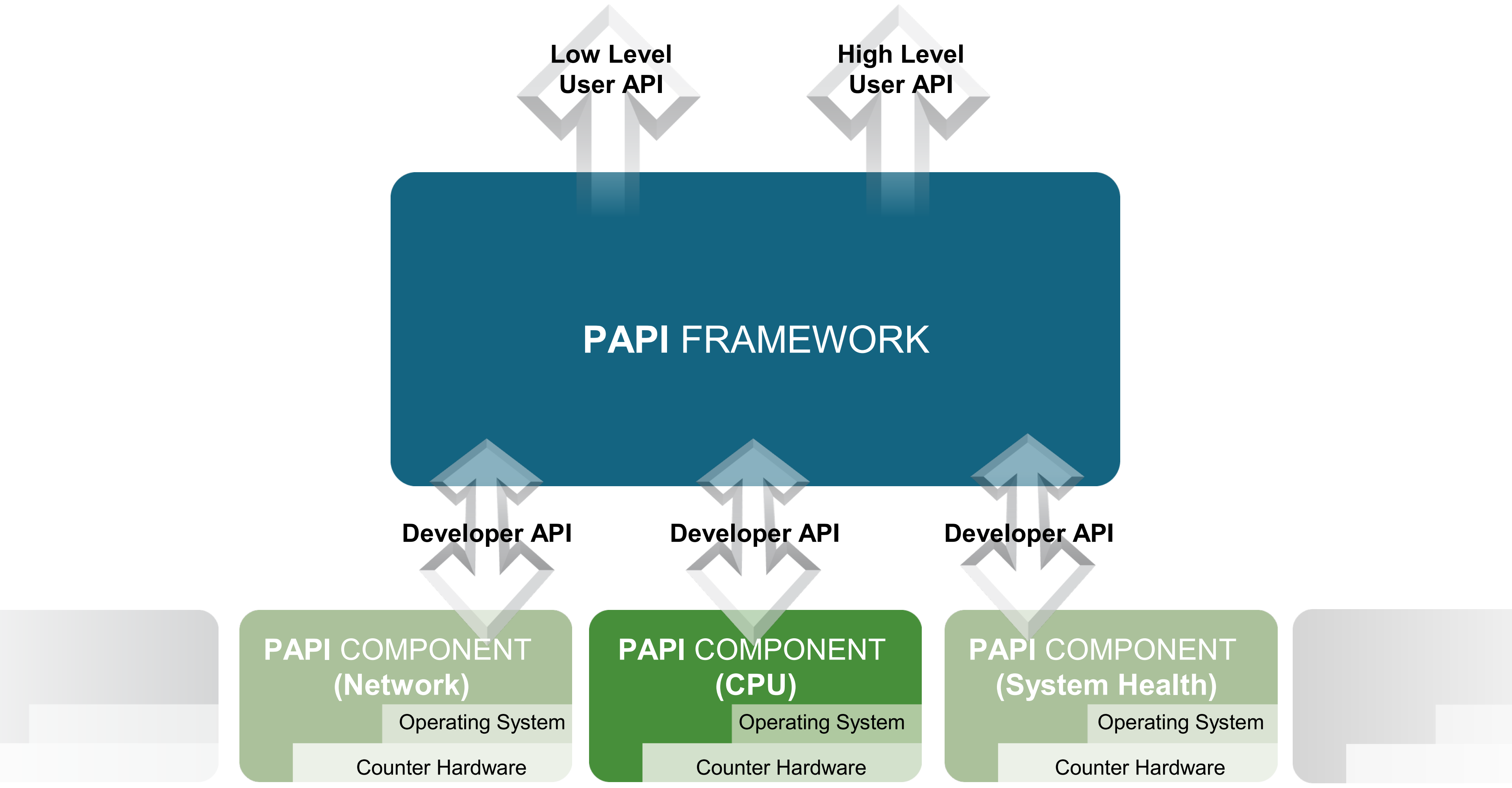
Extending PAPI beyond the CPU

- PAPI has historically targeted on on-processor performance counters.
- Several categories of off-processor counters exist
  - network interfaces: Myrinet, Infiniband, GigE, Cray SeaStar, Gemini
  - file systems: Lustre
  - thermal and power interfaces: ACPI, lm-sensors
  - GPUs
- CHALLENGE:
  - Extend the PAPI interface to address multiple counter domains
  - Preserve the PAPI calling semantics, ease of use, and platform independence for existing applications

Component PAPI Goals

- Support simultaneous access to on- and off-processor counters
- Isolate hardware dependent code in separable ‘component’ modules
- Extend platform independent code to support multiple simultaneous components
- Add or modify API calls to support access to any of several components
- Modify build environment for easy selection and configuration of multiple available components

Component **PAPI Architecture**



File System Components: Lustre

- Measures data collected in: `/proc/…/stats`
  and: `/proc/…/read_ahead_stats`

```
Hits631592284
misses                          9467662
readpage not consecutive        931757
miss inside window              81301
failed grab_cache_page          5621647
failed lock match               2135855
read but discarded              2089608
zero size window                6136494
read-ahead to EOF               160554
hit max r-a issue               25610
```

- Snippet of available native events for Lustre:

```
0x44000002  fastfs_llread        | bytes read on this lustre client
0x44000003  fastfs_llwrite       | bytes written on this lustre client
0x44000004  fastfs_wrong_readahead          | bytes read but discarded due to readahead
0x44000005  work_llread          | bytes read on this lustre client
0x44000006  work_llwrite         | bytes written on this lustre client
0x44000007  work_wrong_readahead | bytes read but discarded due to readahead
```

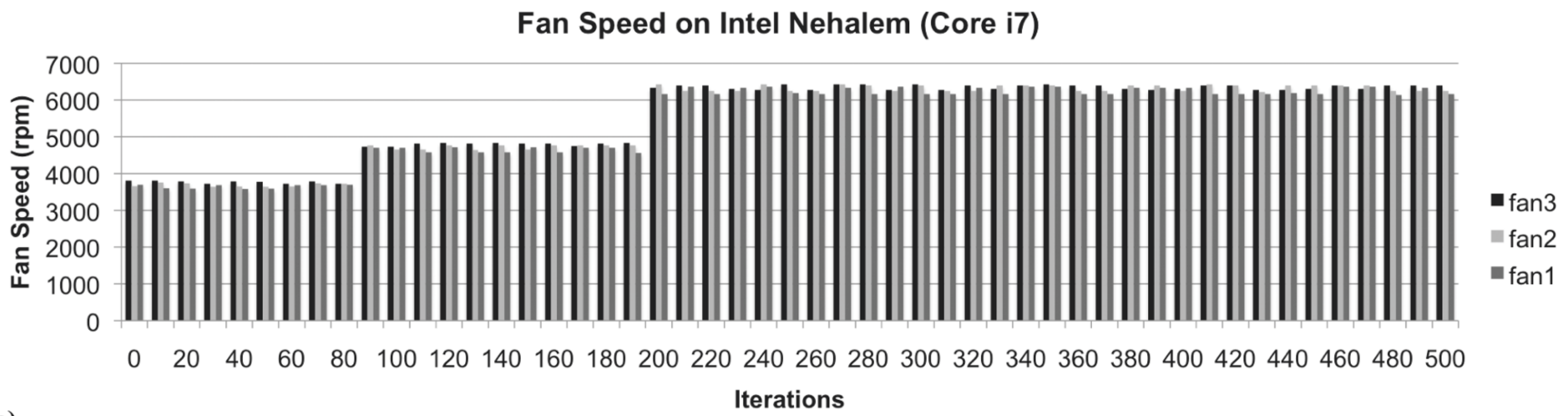
System Health Components: lm-sensors

- Access computer health monitoring sensors, exposed by lm_sensors library
- user is able to closely monitor the system's hardware health
  - observe feedback between performance and environmental conditions
- Available features and monitored events depend on hardware setup
- Snippet of available native events for lm-sensors:

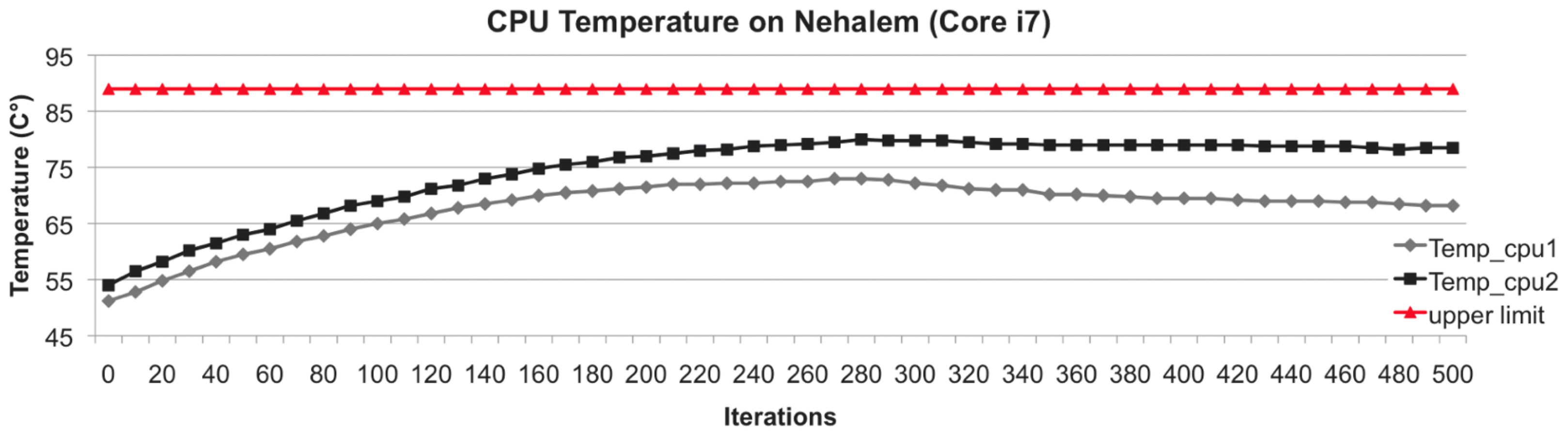
```
…
0x4c000000   LM_SENSORS.max1617-i2c-0-18.temp1.temp1_input
0x4c000001   LM_SENSORS.max1617-i2c-0-18.temp1.temp1_max
0x4c000002   LM_SENSORS.max1617-i2c-0-18.temp1.temp1_min
…
0x4c000049   LM_SENSORS.w83793-i2c-0-2f.fan1.fan1_input
0x4c00004a   LM_SENSORS.w83793-i2c-0-2f.fan1.fan1_min
0x4c00004b   LM_SENSORS.w83793-i2c-0-2f.fan1.fan1_alarm
…
```

lm-sensors Component Example

- libsensors version 3.1.1



(a)



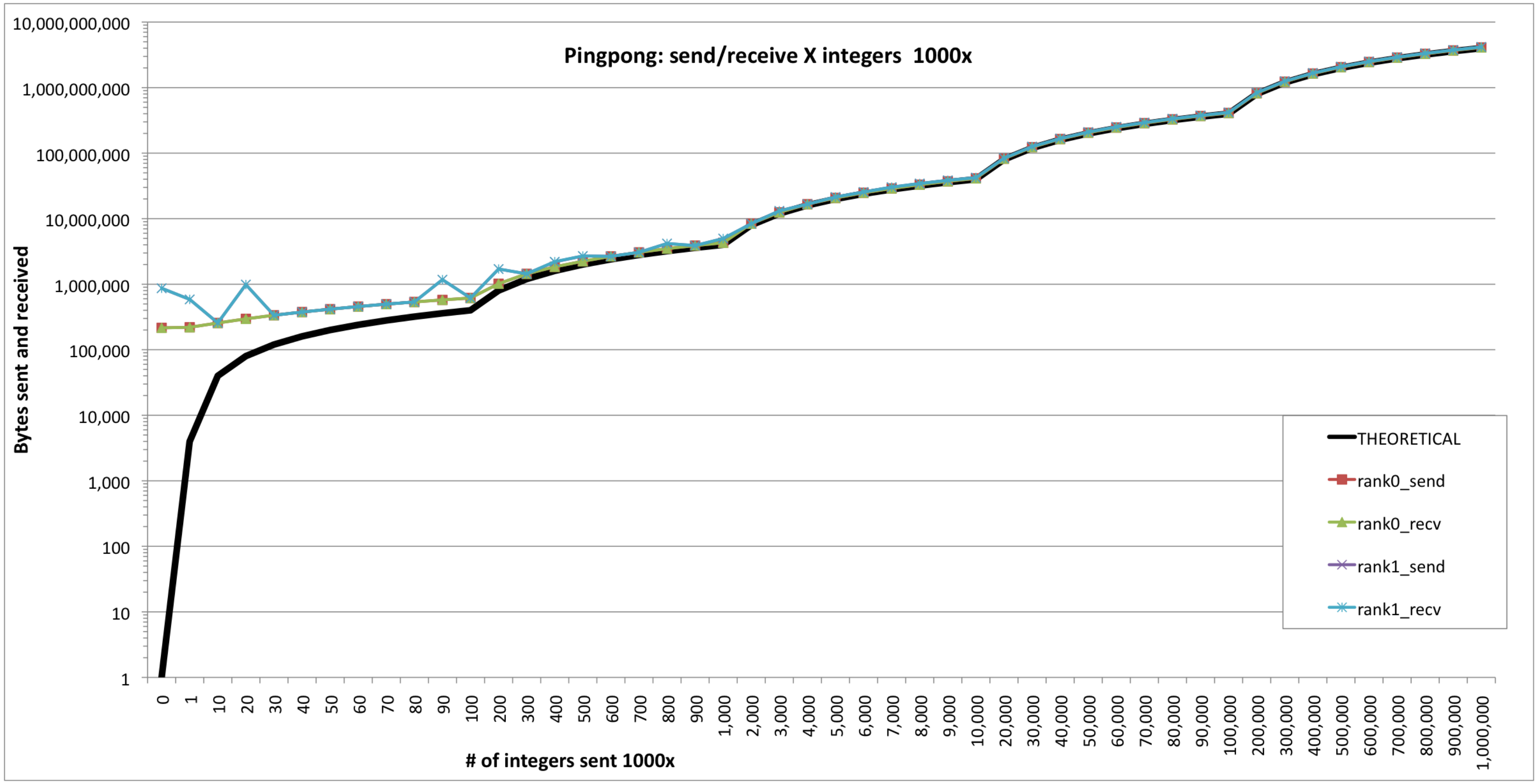
(b)

Network Components: InfiniBand

- Measures everything that is provided by the libibmad:
- Errors, Bytes, Packets, local IDs (LID), global IDs (GID), etc.
- ibmad library provides low-layer IB functions for use by the IB diagnostic and management programs, including MAD, SA, SMP, and other basic IB functions
- Snippet of available native events on a machine with 2 IB devices, mthca0 and mthca1:

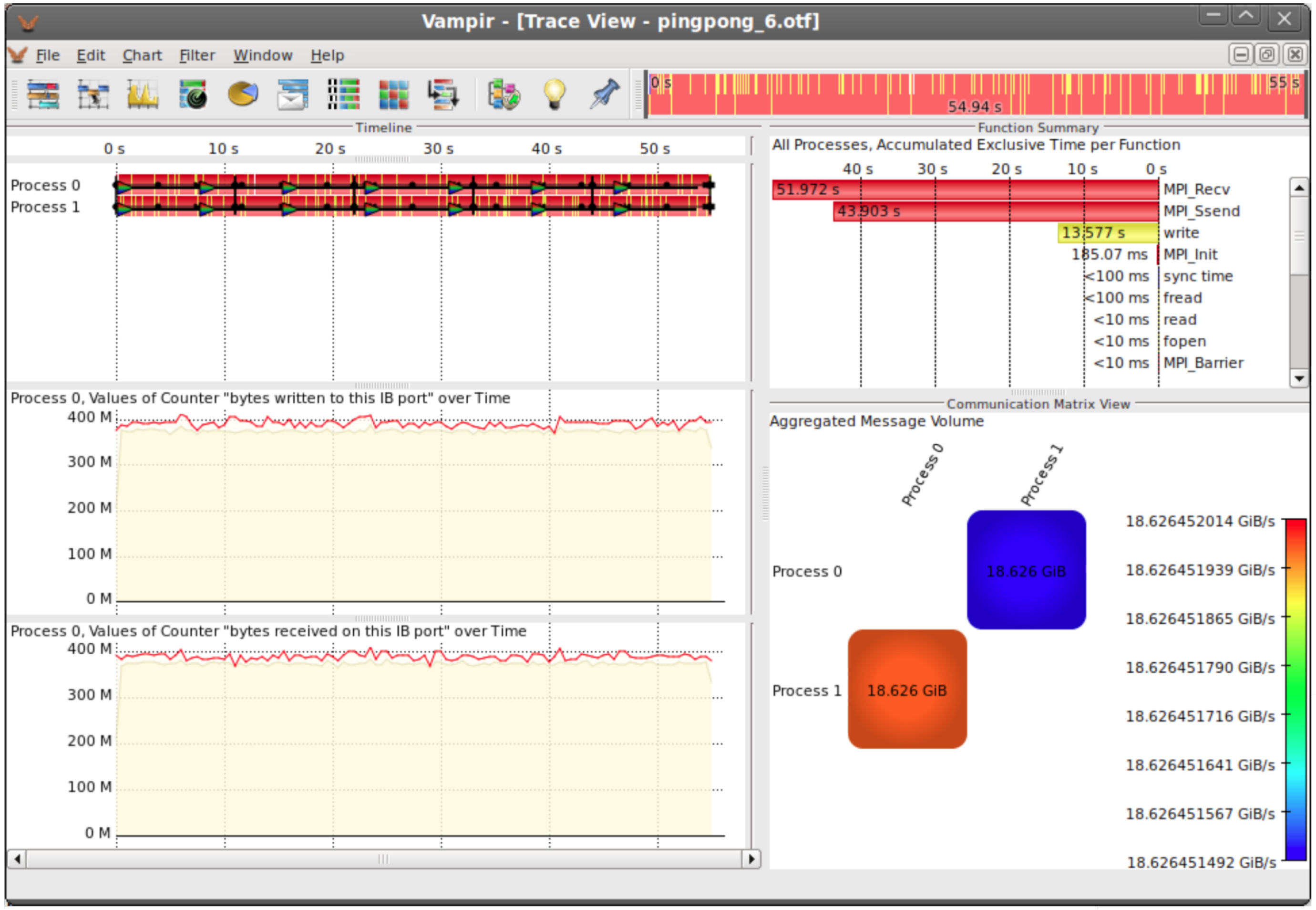
```
…
0x44000000   mthca0_1_recv  | bytes received on this IB port
0x44000001   mthca0_1_send  | bytes written to this IB port
0x44000002   mthca1_1_recv  | bytes received on this IB port
0x44000003   mthca1_1_send  | bytes written to this IB port
…
```

InfiniBand Component Results



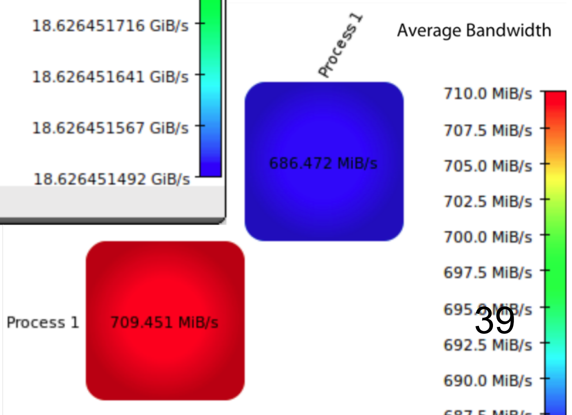
InfiniBand events measured over time (via Vampir linked with PAPI)

- Run Pingpong 5x: send 1,000,000 integers 1000x (theor: ~19GB)



IB Counter resolution in Vampir:

1 sec



PAPI CUDA Component

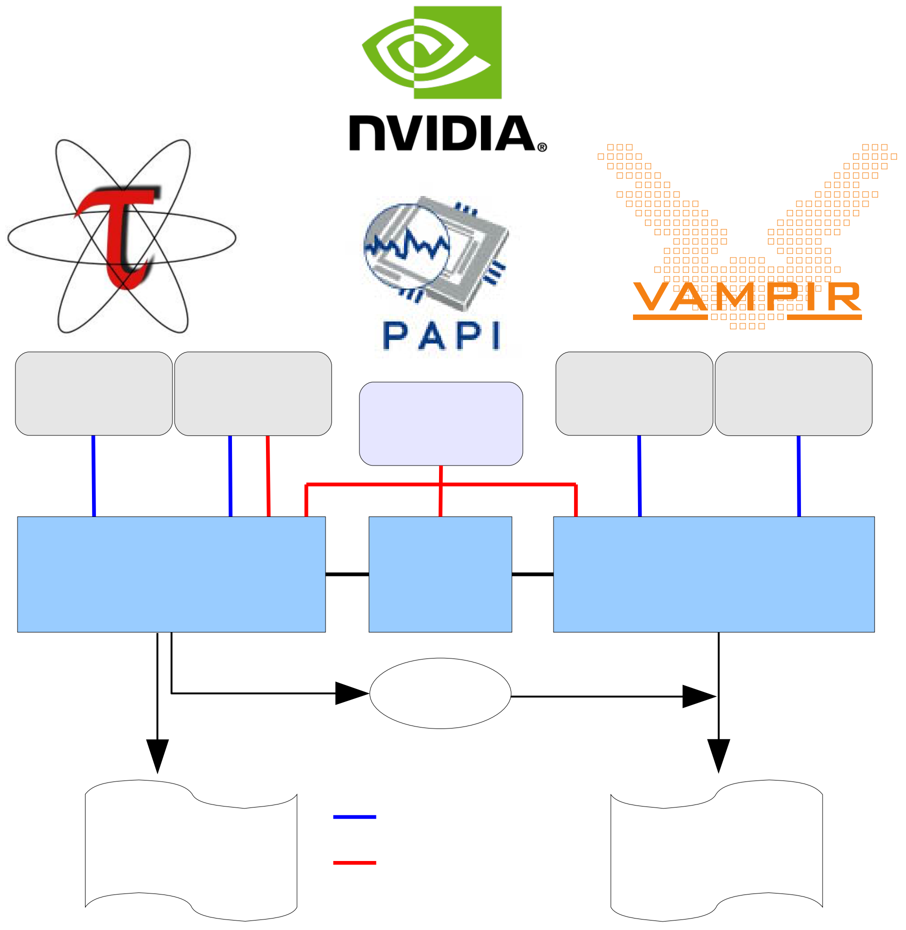
- HW performance counter measurement technology for NVIDIA CUDA platform
- Access to HW counters inside the GPUs
- Based on CUPTI (CUDA Performance Tool Interface) in CUDA 4.0
- In any environment with CUPTI, PAPI CUDA component can provide detailed performance counter info regarding execution of GPU 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

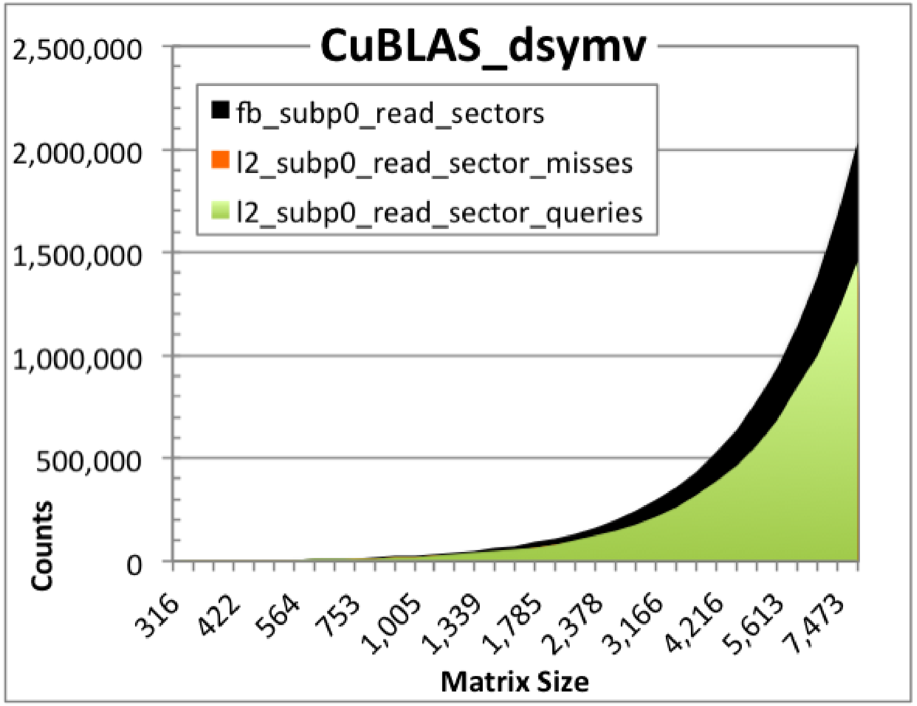
Tool interoperability

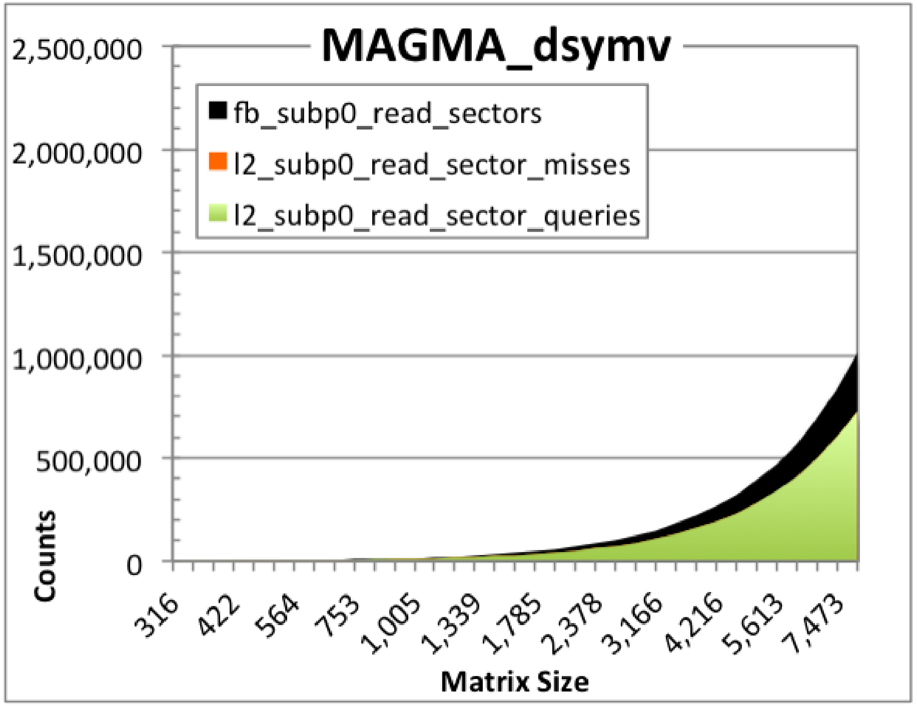


MAGMA versus CUBLAS: SYMV

- 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 → N2/2 element reads (vs. N2)
- 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

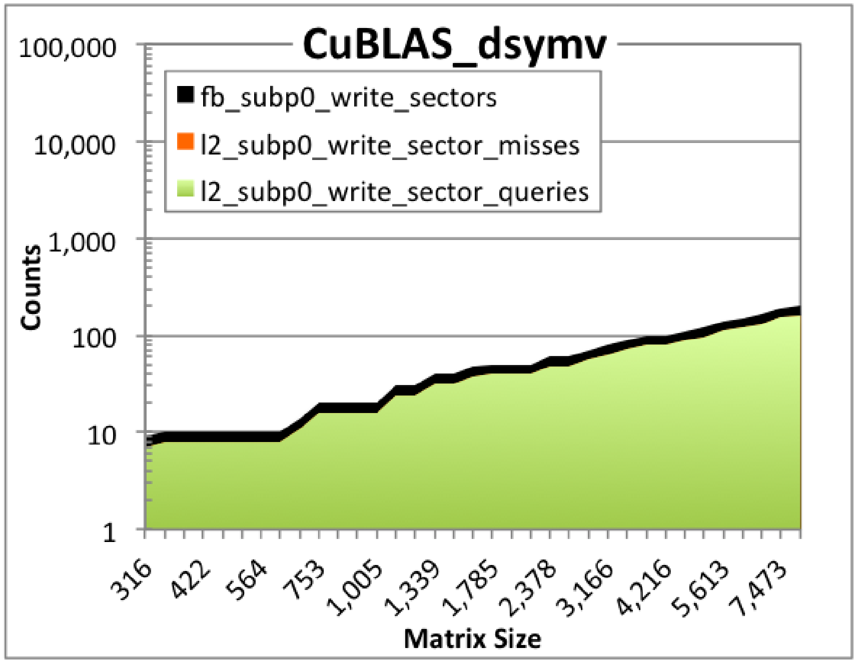
CUDA performance counters for read behavior

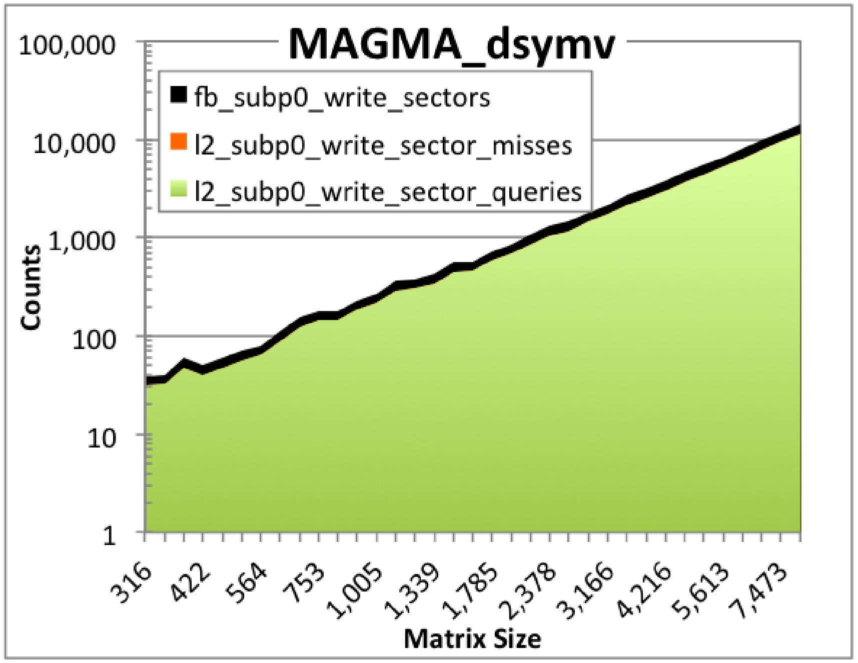




- # of read requests from L1 to L2 (green), which is equal to # of read misses in L2 (orange); number of read requests from L2 to DRAM (black) for CUBLAS_dsymv (left) and MAGMA_dsymv (right)

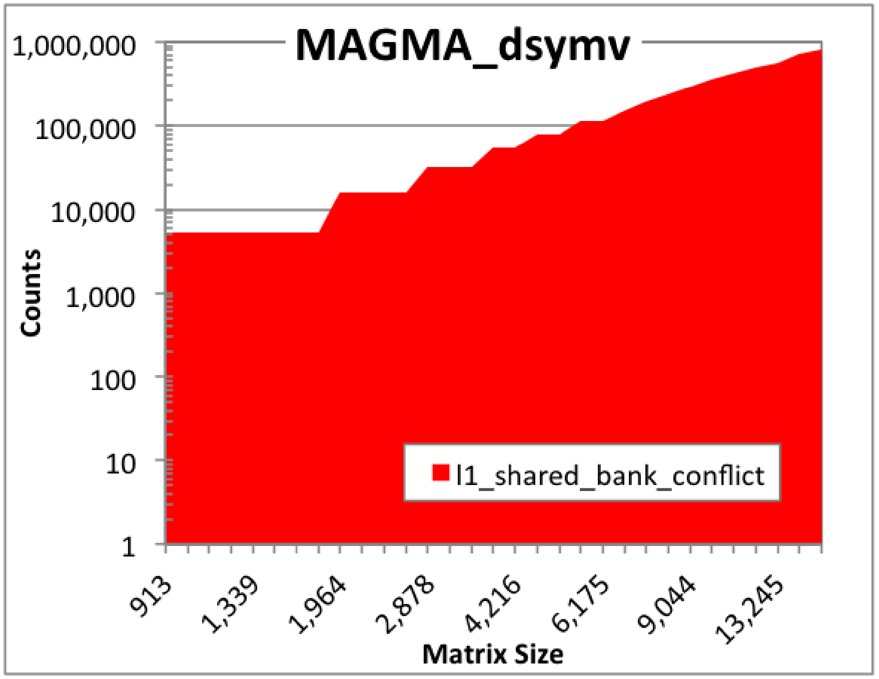
CUDA performance counters for write behavior

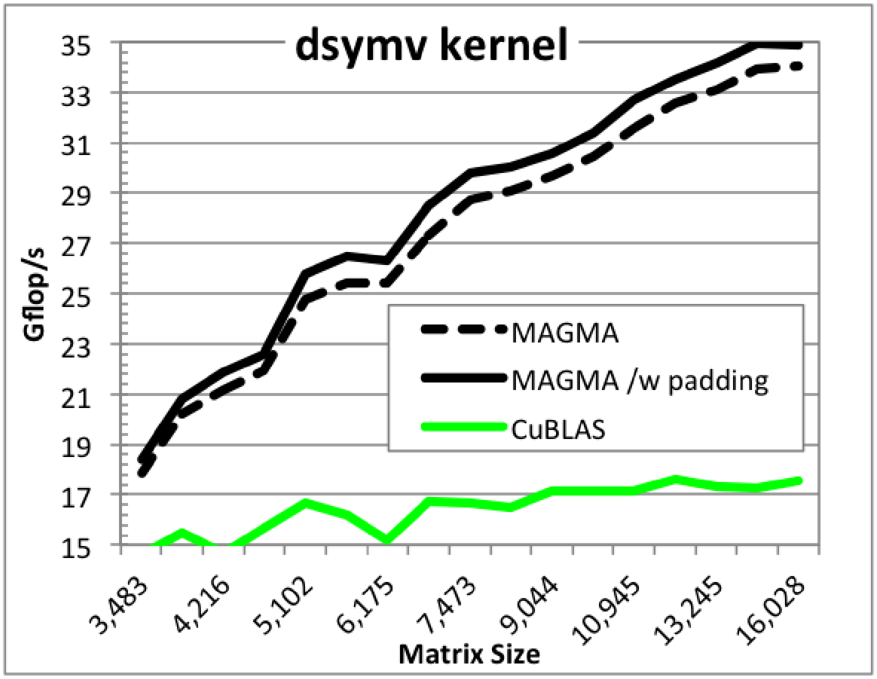




- # of write requests from L1 to L2 (green), which is equal to # of write misses in L2 (orange); # of write requests from L2 to DRAM (black) for CUBLAS_dsymv (left) and MAGMA_dsymv (right)

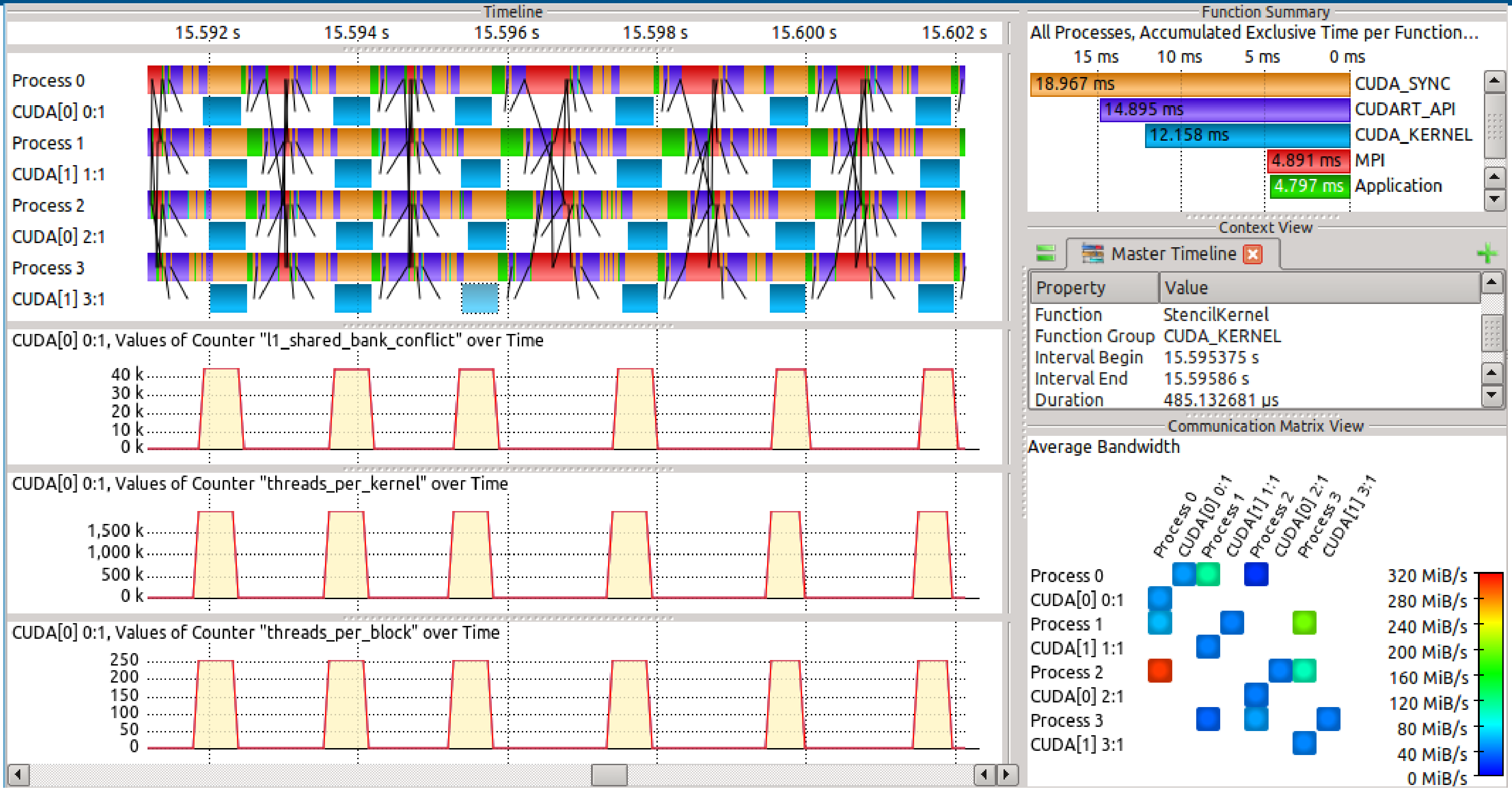
CUDA performance counter for L1 behavior





- # 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



All Processes, Accumulated Exclusive Time per Function...

18.967 ms	CUDA_SYNC
14.895 ms	CUDART_API
12.158 ms	CUDA_KERNEL
4.891 ms	MPI
4.797 ms	Application

Property	Value
Function	StencilKernel
Function Group	CUDA_KERNEL
Interval Begin	15.595375 s
Interval End	15.59586 s
Duration	485.132681 µs

- 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

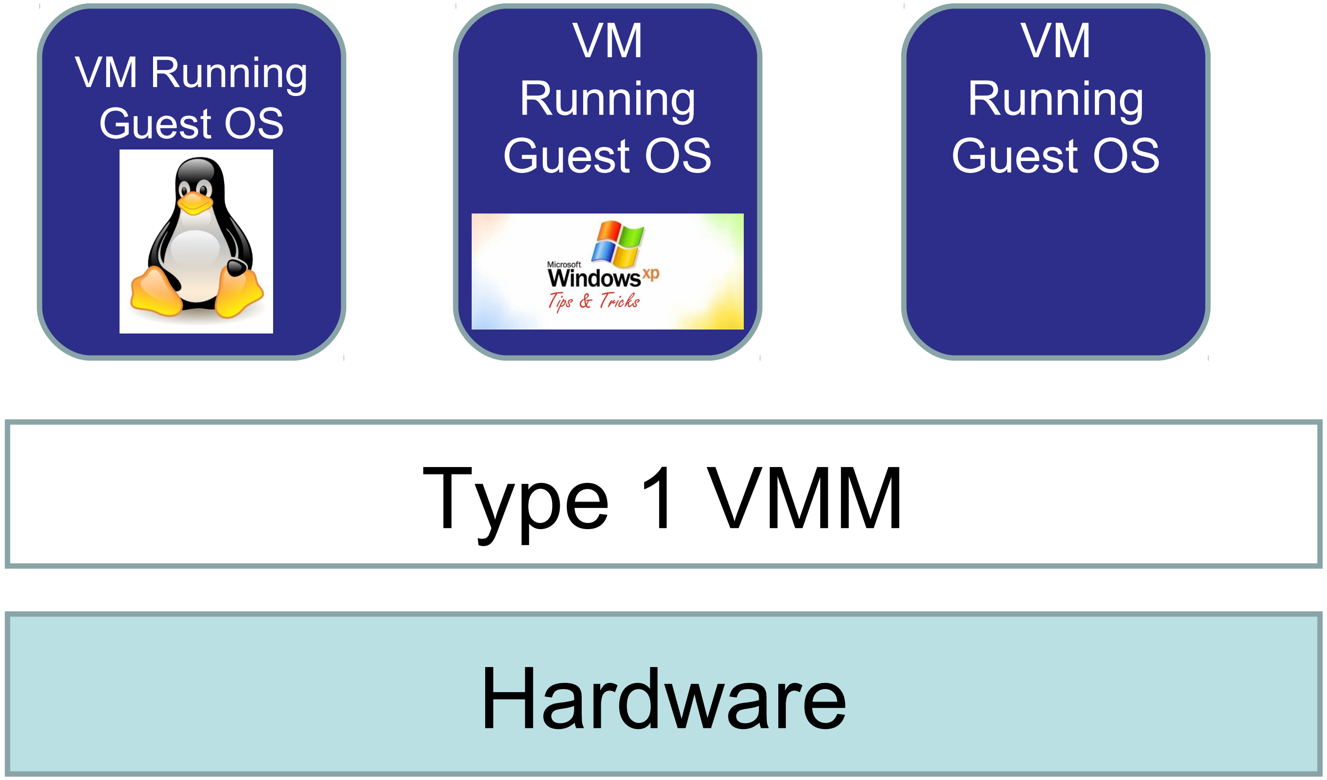
PAPI-G Future Goals

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

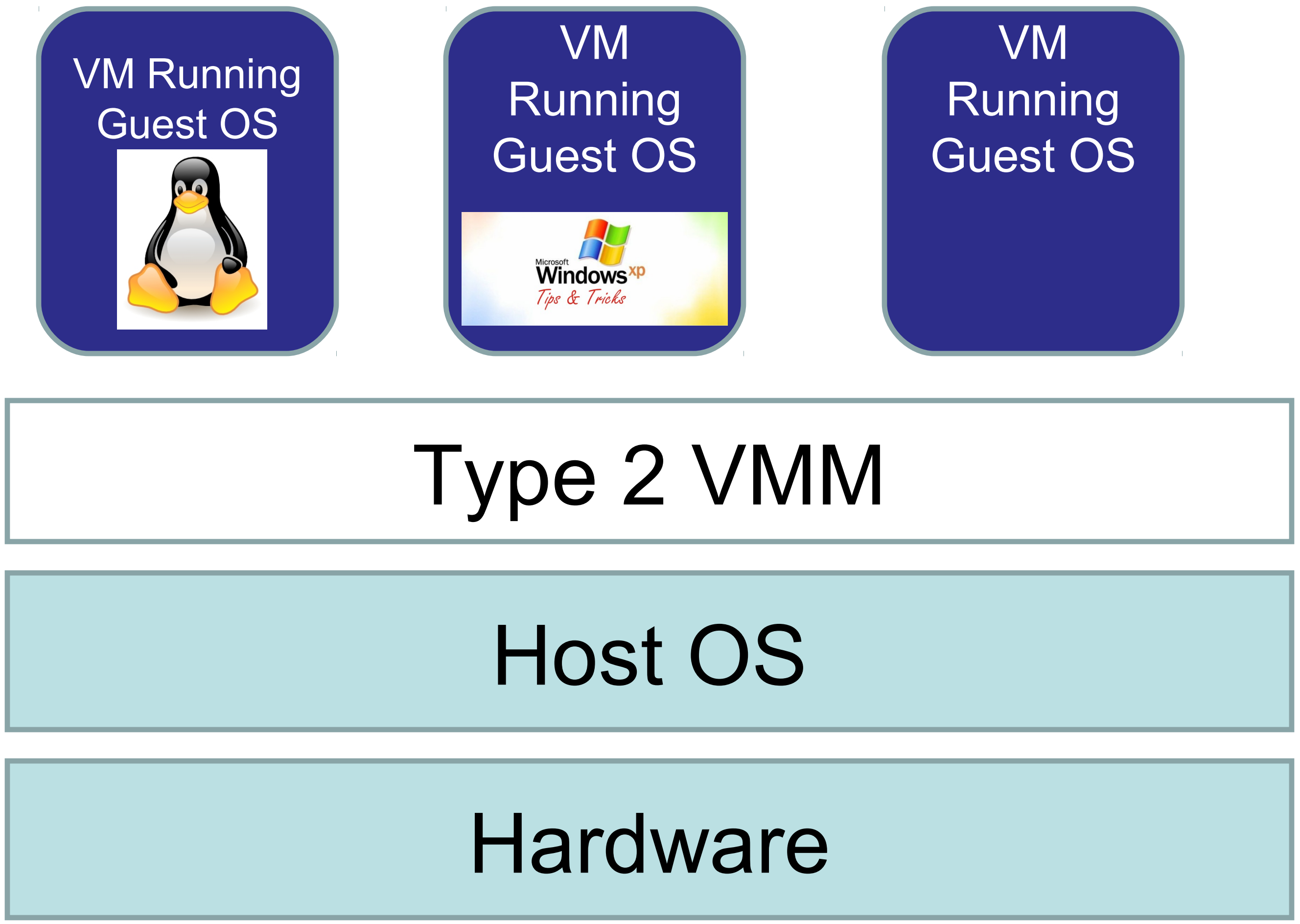
Virtualization

- Enabling technology for cloud computing
- Applications run inside a virtual machine
- Virtual Machine Monitor (VMM)
  - Exports set of virtual machines to guest operating systems
  - Manages access to underlying shared hardware resources

Bare Metal VMM



Hosted VMM


VM Running Guest OS
VM Running Guest OS
Microsoft Windows xp
Tips & Tricks
VM Running Guest OS
Type 2 VMM
Host OS
Hardware

PAPI-V(irtual)

- Hardware performance monitoring in virtualized environments (e.g., VMware, Xen, KVM, VirtualBox, Hyper-V, etc.)
- Funded by US National Science Foundation and VMware, Inc.
- Timer support
  - PAPI_get_real_usec
  - PAPI_get_virt_usec
  - Use timers exposed by VMM
  - Accurate timing for higher level tools
- PAPI events
  - Native counters
    - Potentially possible with some VMMs
    - Requires VMM to save and restore counters on CPU or domain switch
  - Virtualized counters
    - Requires VMM to emulate counter
    - trap+emulate → high overhead
- Components for virtualized hardware: network, I/O, GPUs

For more information

- PAPI Website: http://icl.eecs.utk.edu/papi/
  - Software
  - Release notes
  - Documentation
  - Links to tools that use PAPI
  - Mailing/discussion lists
- Questions?