PAPI

Dr. David Cronk
Research Director
Innovative Computing Lab
University of Tennessee, Knoxville

Outline

- Introduction
- PAPI Utilities
- An Example
- Some PAPI counters
- PAPI and Multi-core
- Component PAPI – The New Wave

Hardware Counters

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

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

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 for sampling
- Events can be multiplexed if counters are limited
- Statistical sampling implemented by:
  - Hardware overflow if supported by the platform
  - Software overflow with timer driven sampling

Where's PAPI

- PAPI runs on most modern processors and operating systems of interest to HPC:
  - IBM POWER / AIX / Linux
  - Blue Gene / L / P…
  - Intel Pentium, Core2, Core i7, Atom / Linux
  - Intel Itanium / Linux
  - AMD Athlon, Opteron / Linux
  - Cray XT(n) / CLE
  - Altix, Sparc, Niagara …
- Older Linux systems require a kernel patch
  - Perfctr
  - Perfmon
  - Latest Linux kernels include perf events

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|
--------------------------------------------------------------------------------
```

ICL UT

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|
```

Quad-core Opteron

- PAPI_FP_OPS and PAPI_FP_INS defined the same
  - Count only retired SSE instructions
  - Count sing and double precision add/subtract/multiply
  - Does not count x87 or MMX operations
  - Does not count SSE FP divide or sqrt operations
  - Some additional testing can reveal additional information:
    - 1 FLOPS for SSE scalar instructions
    - 2 FLOPS for SSE packed double instructions
    - 4 FLOPS for SSE packed single instructions

PAPI Utilities: *papi_event_chooser*

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events which 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 which 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_event_chooser*

```
$ utils/papi_event_chooser NATIVE RESOURCE_STALLS:LD_ST X87_OPS_RETIRED
    INSTRUCTIONS_RETIRED
[…]
-------------------------------------------------------------------------------
UNHALTED_CORE_CYCLES    0x40000000
 |count core clock cycles whenever the clock signal on the specific core is running
    (not halted). Alias to event CPU_CLK_UNHALTED:CORE_P|
 |Register Value[0]: 0x20003       Event Selector|
 |Register Value[1]: 0x3c          Event Code|
-------------------------------------------------------------------------------
UNHALTED_REFERENCE_CYCLES       0x40000002
 |Unhalted reference cycles. Alias to event CPU_CLK_UNHALTED:REF|
 |Register Value[0]: 0x40000       Event Selector|
 |Register Value[1]: 0x13c         Event Code|
-------------------------------------------------------------------------------
CPU_CLK_UNHALTED        0x40000028
 |Core cycles when core is not halted|
 |Register Value[0]: 0x60000       Event Selector|
 |Register Value[1]: 0x3c          Event Code|
    0x40001028  :CORE_P  |Core cycles when core is not halted|
    0x40008028  :NO_OTHER  |Bus cycles when core is active and the other is halted|
-------------------------------------------------------------------------------
Total events reported: 3
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
```

The Code

```
#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++) {
      for (j = 0; j < COLUMNS; j++) {
         float sum = 0.0;
         for (k = 0; k < COLUMNS; k++) {
            sum +=
               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

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

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**

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
- Lower 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

The Multicore Dilemma

- Multicore is the (near term) future of Petascale computing
- Minimizing Resource contention is key
  - Memory bandwidth
  - Cache sharing & collisions
  - Bus and other resource contention
- Current tools don’t support first-person counting of shared events
- Current architectures don’t encourage first-person counting of shared events

Current “State of the Art”

- Counter support for shared resources is broken
  - Every vendor has a different approach
  - Often 3rd person, not 1st person
  - Counts often polluted by other cores
  - No exclusive reservation of shared counter resources
  - No migration of events with tasks
- PAPI research is underway to address this

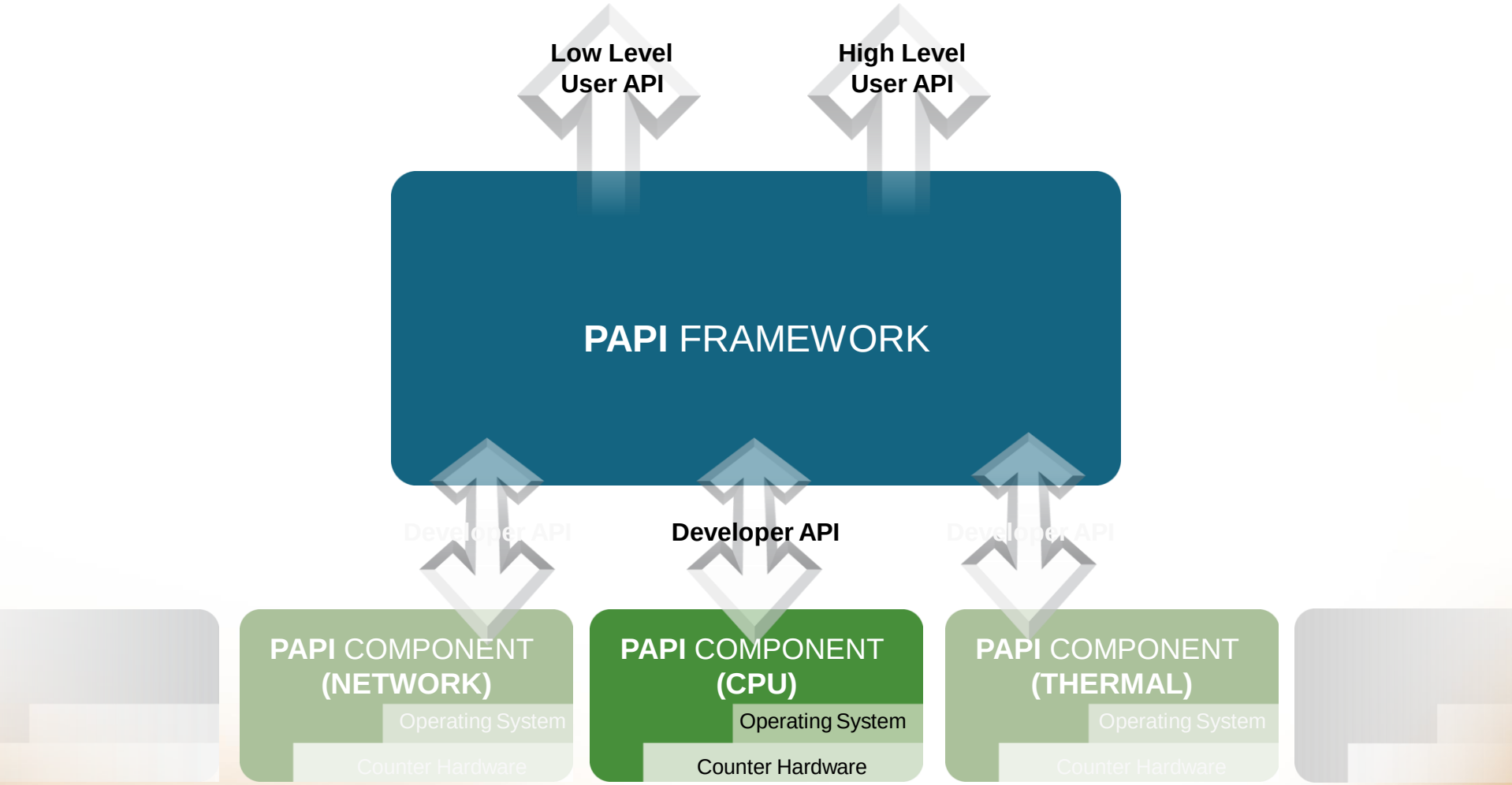
Extending PAPI beyond the CPU

- PAPI has historically targeted on on-processor performance counters
- Several categories of off-processor counters exist
  - network interfaces: Myrinet, Infiniband, GigE
  - memory interfaces: Cray SeaStar, Gemini
  - thermal and power interfaces: ACPI, lm-sensors
  - accelerators?
- 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

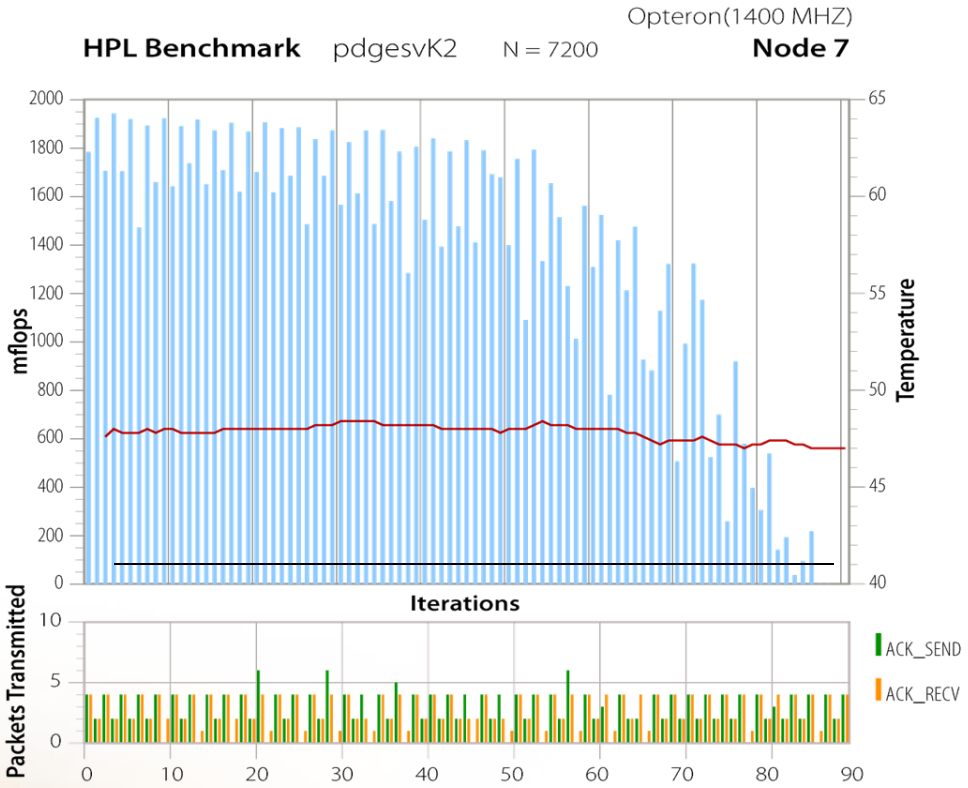
- Support simultaneous access to on- and off-processor counters
- Isolate hardware dependent code in separable ‘component’ modules
- Extend platform independent code to support multiple simultaneous components
- Add or modify API calls to support access to any of several components
- Modify build environment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

Component **PAPI**



Multi Component Measurements

- **HPCC HPL benchmark on Opteron with 3 performance metrics:**
  - **FLOPS; Temperature; Network Sends/Receives**
    - **Temperature is from an on-chip thermal diode**



For more information

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