COSC 594 Spring 2013
Scientific Computing for Engineers

Hardware Performance Counters

Dan Terpstra

terpstra@icl.utk.edu

Table of Contents

- PAPI & Performance Counters
- Useful Utilities
- Performance Measurement Categories*
- The Code*
- Performance Measurement Examples

* With thanks and apologies to Paul Drongowski of AMD:
http://developer.amd.com/wordpress/media/2012/10/Basic_Performance_Measurements.pdf
http://developer.amd.com/wordpress/media/2012/10/Introduction_to_CodeAnalyst.pdf

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 series / AIX
  - POWER4,5, PowerPC / Linux
  - Blue Gene/L/P
  - Intel Pentium series, Core2 / Linux
  - Intel Itanium 1, 2, Montecito
  - AMD Athlon, Opteron, Barcelona / Linux
  - Cray X1, X2, XT3/4 / Catamount / CNL
  - Altix, Sparc, SiCortex …

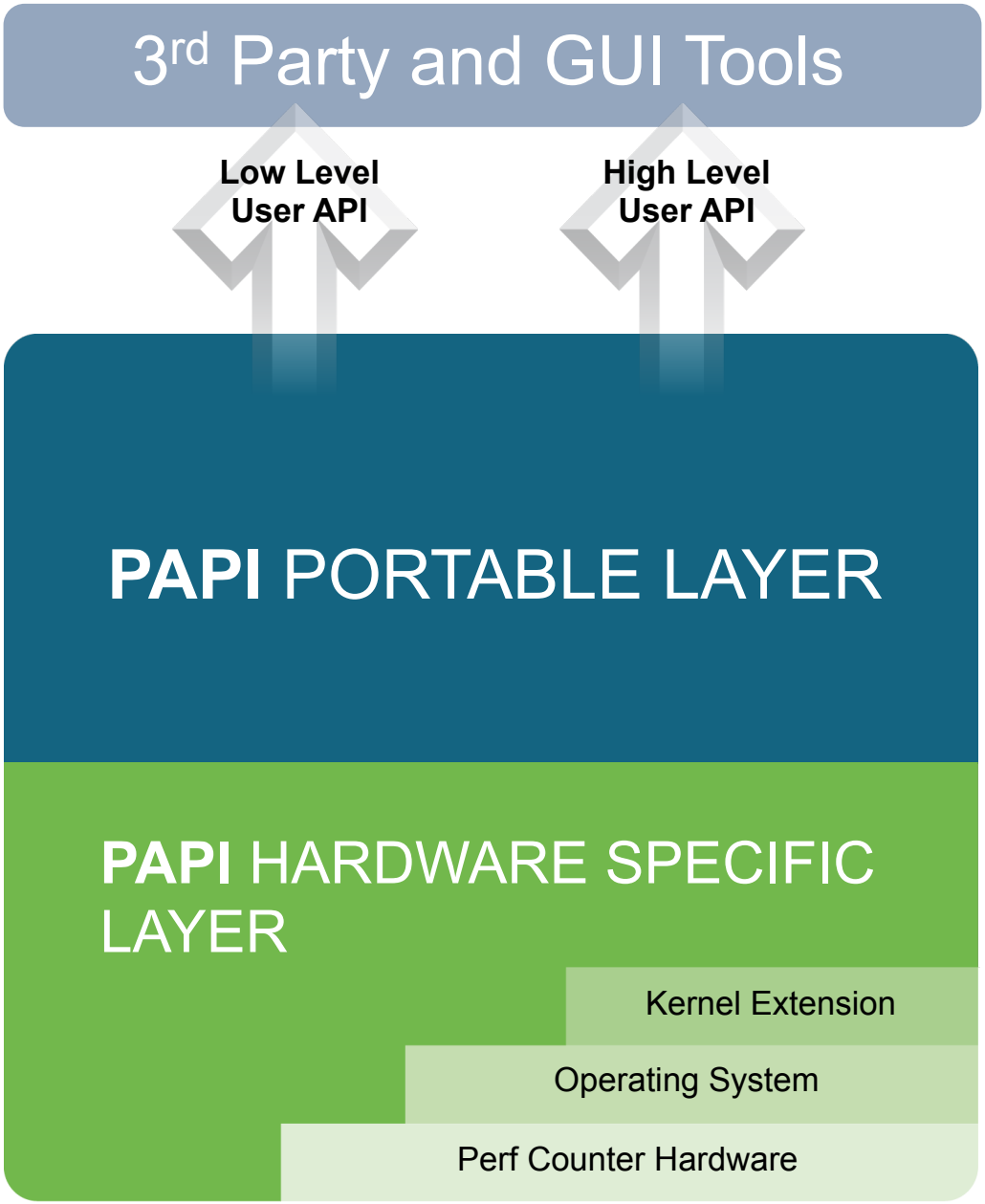
Some tools that use PAPI

- TAU (U Oregon)
  - http://www.cs.uoregon.edu/research/tau/
- PerfSuite (NCSA)
  - http://perfsuite.ncsa.uiuc.edu/
- Scalasca (UTK, FZ Juelich)
  - http://www.fz-juelich.de/jsc/scalasca/
- Vampir (TUDresden)
  - http://www.vampir.eu/
- HPCToolkit (Rice Univ.)
  - http://hpctoolkit.org/
- Open|Speedshop (SGI)
  - http://oss.sgi.com/projects/openspeedshop/

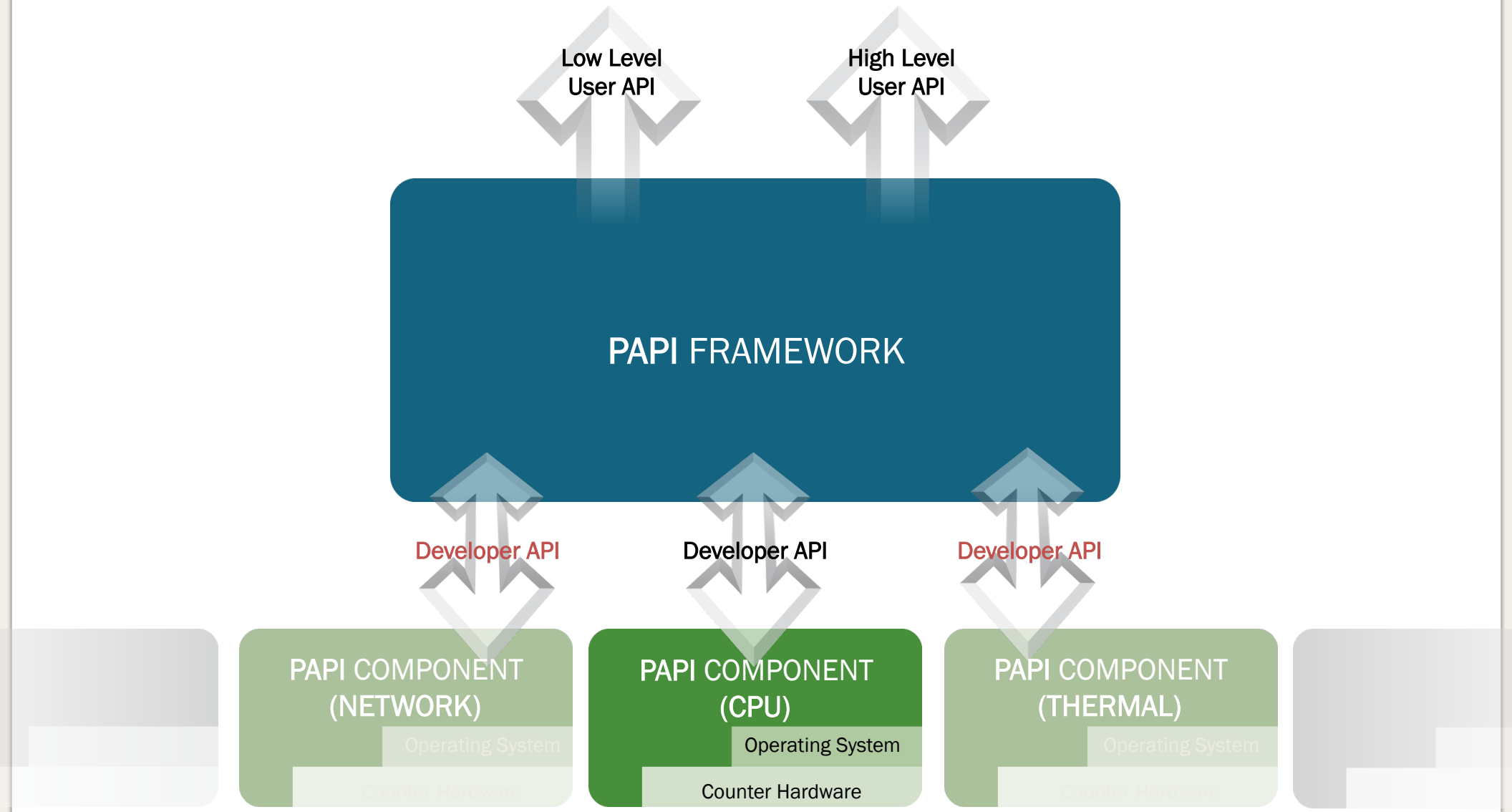
PAPI Counter Interfaces

PAPI provides 3 interfaces to the underlying counter hardware:

1. A Low Level API manages hardware events in user defined groups called *EventSets*, and provides access to advanced features.
2. A High Level API provides the ability to start, stop and read the counters for a specified list of events.
3. Graphical and end-user tools provide facile data collection and visualization.



Component **PAPI**



PAPI High Level Calls

1. PAPI_num_counters()
   - get the number of hardware counters available on the system
2. PAPI_flips(float *rtime, float *ptime, long long *flpins, float *mflips)
   - simplified call to get Mflips/s (floating point instruction rate), real and processor time
3. PAPI_flops (float *rtime, float *ptime, long long *flpops, float *mflops)
   - simplified call to get Mflops/s (floating point operation rate), real and processor time
4. PAPI_ipc (float *rtime, float *ptime, long long *ins, float *ipc)
   - gets instructions per cycle, real and processor time
5. PAPI_accum_counters (long long *values, int array_len)
   - add current counts to array and reset counters
6. PAPI_read_counters (long long *values, int array_len)
   - copy current counts to array and reset counters
7. PAPI_start_counters (int *events, int array_len)
   - start counting hardware events
8. PAPI_stop_counters (long long *values, int array_len)
   - stop counters and return current counts

PAPI Low Level Example

```
#include "papi.h"
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_OPS,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);
```

PAPI Preset Events

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

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_CA_CLN: Requests for exclusive access to clean cache line
PAPI_CA_INV: Requests for cache line invalidation
PAPI_CA_ITV: Requests for cache line intervention

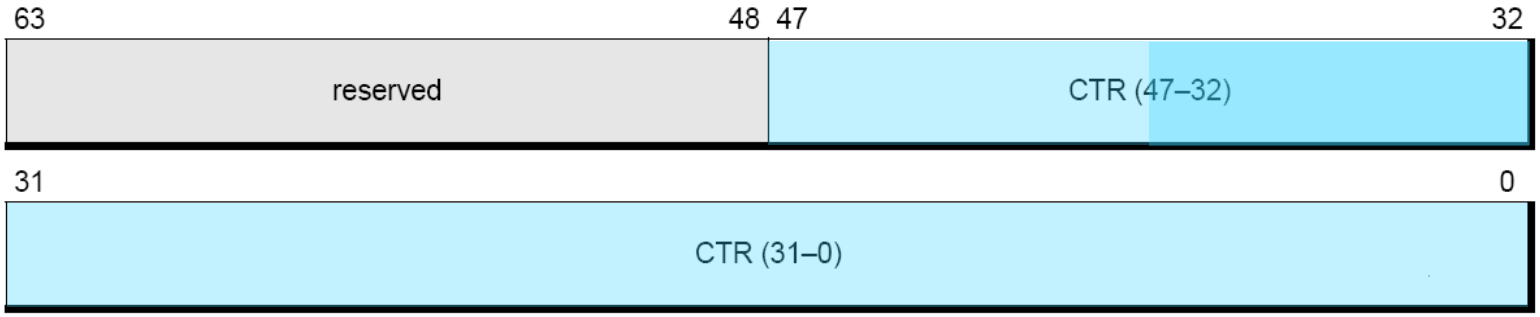
PAPI Native Events

- Native Events
  - Any event countable by the CPU
  - 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

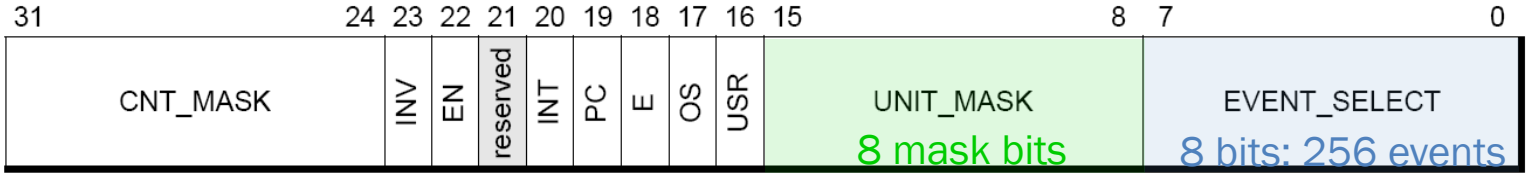
```
PRESET,
  PAPI_L1_DCH,
  DERIVED_SUB,
  DATA_CACHE_ACCESSES,
  DATA_CACHE_MISSES
```

```
/* 19 */{.pme_name =
"DATA_CACHE_REFILLS_FROM_SYSTEM",
    .pme_code  = 0x43,
    .pme_desc  = "Data Cache Refills from
       the northbridge",
    .pme_flags = PFMLIB_AMD64_UMASK_COMBO,
    .pme_numasks = 6,
    .pme_umasks  = {
       { .pme_uname = "INVALID",
         .pme_udesc = "Invalid",
         .pme_ucode = 0x01,
       },
       { .pme_uname = "SHARED",
         .pme_udesc = "Shared",
         .pme_ucode = 0x02,
       },
       { .pme_uname = "EXCLUSIVE",
         .pme_udesc = "Exclusive",
         .pme_ucode = 0x04,
       },
       { .pme_uname = "OWNED",
         .pme_udesc = "Owned",
         .pme_ucode = 0x08,
       },
       { .pme_uname = "MODIFIED",
         .pme_udesc = "Modified",
         .pme_ucode = 0x10,
       },
       { .pme_uname = "ALL",
         .pme_udesc = "All sub-events",
         .pme_ucode = 0x1F,
       },
    },
},
```

What's a Native Event?

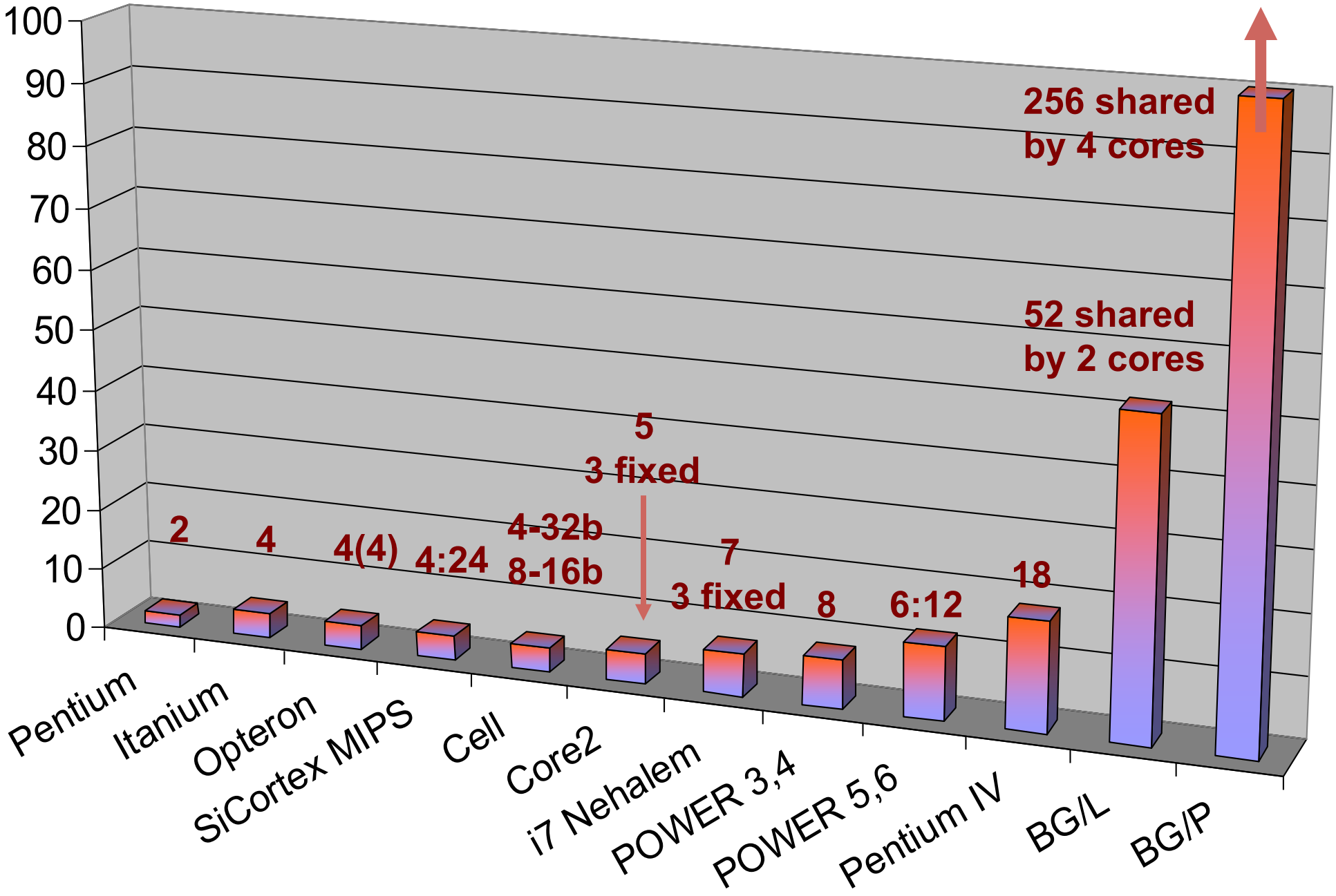


PMD: AMD, Intel



PMC: Intel Pentium II, III, M, Core, i7; AMD Athlon, Opteron

How many counters does it take?



Ivy Bridge Counters

- 3 Fixed Function Counters
  - Unhalted Core Cycles
  - Unhalted Reference Cycles
  - Instructions Retired
- 8 Programmable Counters
  - unless you're Hyperthreading (4 per thread)
  - or using an NMI watchdog timer (3 per thread)
- 4 Uncore Counters
  - chip wide; not core specific
  - unified cache measurement (L3)
  - shared resources

Useful PAPI Utilities

- *papi_cost*
- *papi_avail*
- *papi_native_avail*
- *papi_event_chooser*
- *papi_command_line*

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 and for PAPI reads.  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:

     [...]
    1683:
    1773:
    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             : 5.1.0.2
Vendor string and code   : AuthenticAMD (2)
Model string and code    : Six-Core AMD Opteron(tm) Processor 8439 SE (8)
CPU Revision             : 0.000000
CPUID Info               : Family: 16  Model: 8  Stepping: 0
CPU Max Megahertz        : 2812
CPU Min Megahertz        : 2812
Hdw Threads per core     : 1
Cores per Socket         : 6
NUMA Nodes               : 8
CPUs per Node            : 6
Total CPUs               : 48
Running in a VM          : no
Number Hardware Counters : 4
Max Multiplex Counters   : 64
--------------------------------------------------------------------------------

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

[...]
PAPI_FP_OPS  0x80000066  Yes   No   Floating point operations (Counts speculative adds and
   multiplies. Variable and higher than theoretical.)

[...]
PAPI_REF_CYC 0x8000006b  No    No   Reference clock cycles
--------------------------------------------------------------------------------
Of 108 possible events, 40 are available, of which 8 are derived.

avail.c                                  PASSED
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail -a
Available events and hardware information.
-------------------------------------------------------------------------
  ---
[…]
-------------------------------------------------------------------------
  ---
The following correspond to fields in the PAPI_event_info_t structure.

    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_DCM  0x80000002  No   Level 2 data cache misses
PAPI_L2_ICM  0x80000003  No   Level 2 instruction cache misses
[…]
PAPI_DP_OPS  0x80000068  No   Floating point operations; optimized to
  count scaled double precision vector operations
-------------------------------------------------------------------------
  -
Of 40 available events, 8 are derived.

avail.c                                  PASSED
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail -e PAPI_FP_OPS
Available events and hardware information.
-------------------------------------------------------------------------
  ---
[…]
-------------------------------------------------------------------------
  ---

Event name:                        PAPI_FP_OPS
Event Code:                        0x80000066
Number of Native Events:           1
Short Description:                |FP operations|
Long Description:                 |Floating point operations|
Developer's Notes:                ||
Derived Type:                     |NOT_DERIVED|
Postfix Processing String:        ||
 Native Code[0]: 0x4000001d |
  RETIRED_SSE_OPERATIONS:SINGLE_ADD_SUB_OPS:SINGLE_MUL_OPS:DOUBLE_ADD_SU
  B_OPS:DOUBLE_MUL_OPS:OP_TYPE|
 Number of Register Values: 0
 Native Event Description: |Retired SSE Operations, masks:Single
  precision add/subtract ops,Single precision multiply ops,Double
  precision add/subtract ops,Double precision multiply ops,Op type:
  0=uops. 1=FLOPS|

-------------------------------------------------------------------------
  -
avail.c                                          PASSED
```

PAPI Utilities: *papi_native_avail*

```
UNIX> utils/papi_native_avail
Available native events and hardware information.
--------------------------------------------------------------------------------

[...]
================================================================================
 Native Events in Component: perf_events
================================================================================
| perf::PERF_COUNT_HW_CPU_CYCLES                                               |
|            PERF_COUNT_HW_CPU_CYCLES                                          |
--------------------------------------------------------------------------------
| perf::CYCLES                                                                 |
|            PERF_COUNT_HW_CPU_CYCLES                                          |
--------------------------------------------------------------------------------
[…]
--------------------------------------------------------------------------------
| perf::PERF_COUNT_SW_PAGE_FAULTS                                              |
|            PERF_COUNT_SW_PAGE_FAULTS                                         |
--------------------------------------------------------------------------------
| perf::PERF_COUNT_SW_CONTEXT_SWITCHES                                         |
|            PERF_COUNT_SW_CONTEXT_SWITCHES                                    |
--------------------------------------------------------------------------------

[...]
```

PAPI Utilities: *papi_native_avail*

```
--------------------------------------------------------------------------------
| DISPATCHED_FPU                                                               |
|            Dispatched FPU Operations                                         |
|     :OPS_ADD                                                                 |
|            Add pipe ops excluding load ops and SSE move ops                  |
|     :OPS_MULTIPLY                                                            |
|            Multiply pipe ops excluding load ops and SSE move ops             |
|     :OPS_STORE                                                               |
|            Store pipe ops excluding load ops and SSE move ops                |
|     :OPS_ADD_PIPE_LOAD_OPS                                                   |
|            Add pipe load ops and SSE move ops                                |
|     :OPS_MULTIPLY_PIPE_LOAD_OPS                                              |
|            Multiply pipe load ops and SSE move ops                           |
|     :OPS_STORE_PIPE_LOAD_OPS                                                 |
|            Store pipe load ops and SSE move ops                              |
|     :ALL   All sub-events selected                                           |
|     :e=0   edge level                                                        |
|     :i=0   invert                                                            |
|     :c=0   counter-mask in range [0-255]                                     |
|     :g=0   measure in guest                                                  |
|     :u=0   monitor at user level                                             |
|     :k=0   monitor at kernel level                                           |
|     :h=0   monitor at hypervisor level                                       |
--------------------------------------------------------------------------------
```

PAPI Utilities: *papi_native_avail*

```
UNIX> utils/papi_native_avail -e DATA_CACHE_REFILLS
Available native events and hardware information.
-------------------------------------------------------------------------
[…]
-------------------------------------------------------------------------
Event name:                      DATA_CACHE_REFILLS
Description:                    |Data Cache Refills from L2 or Northbridge|

Unit Masks:
 Mask Info:                     |:SYSTEM|Refill from the Northbridge|
 Mask Info:                     |:L2_SHARED|Shared-state line from L2|
 Mask Info:                     |:L2_EXCLUSIVE|Exclusive-state line from L2|
 Mask Info:                     |:L2_OWNED|Owned-state line from L2|
 Mask Info:                     |:L2_MODIFIED|Modified-state line from L2|
 Mask Info:                     |:ALL|All sub-events selected|
 Mask Info:                     |:e=0|edge level|
 Mask Info:                     |:i=0|invert|
 Mask Info:                     |:c=0|counter-mask in range [0-255]|
 Mask Info:                     |:g=0|measure in guest|
 Mask Info:                     |:u=0|monitor at user level|
 Mask Info:                     |:k=0|monitor at kernel level|
 Mask Info:                     |:h=0|monitor at hypervisor level|
```

PAPI Utilities: *papi_event_chooser*

```
$ utils/papi_event_chooser
Usage: eventChooser NATIVE|PRESET evt1 evt2 ...
```

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
```

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
- Special cases
  - Unaligned data access
  - Floating point operations
  - Floating point exceptions

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++) {

         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.
```

Performance Data

IPC – instructions per cycle

- Measure instruction level parallelism
- An indicator of code efficiency

PAPI_ipc

```
retval = PAPI_ipc(&realtime, &processtime, &start_ins, &ipc);
classic_matmul();
retval = PAPI_ipc(&realtime, &processtime, &end_ins, &ipc);
retval = PAPI_stop_counters(NULL, 0));
```

PAPI High Level

```
int events[] = {PAPI_TOT_CYC, PAPI_TOT_INS};

realtime[0] = PAPI_get_real_usec();
retval = PAPI_start_counters(events, 2);
classic_matmul();
retval = PAPI_stop_counters(cvalues, 2);
realtime[1] = PAPI_get_real_usec();
```

PAPI Low Level

```
int events[] = {PAPI_TOT_CYC, PAPI_TOT_INS};

retval = PAPI_library_init (PAPI_VER_CURRENT);
retval = PAPI_create_eventset(&EventSet);
retval = PAPI_add_events(EventSet, events, 2);
realtime[0] = PAPI_get_real_usec();
retval = PAPI_start(EventSet);
classic_matmul();
retval = PAPI_stop(EventSet, cvalues);
realtime[1] = PAPI_get_real_usec();
```

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

Memory Bandwidth

Measurement	Classic mat_mul	Reordered mat_mul
SYSTEM_READ_RESPONSES		
:EXCLUSIVE:MODIFIED:SHARED	62081741	61831162
QUADWORDS_WRITTEN_TO_SYSTEM:ALL	1857263	1036769
DRAM_ACCESSES_PAGE:HIT:MISS:CONFLICT	45785165	33645477
Read Bandwidth	293.3849 MB/sec	1338.5084 MB/sec
Write Bandwidth	8.7770 MB/sec	22.4438 MB/sec
DRAM Bandwidth	216.3708 MB/sec	728.3504 MB/sec

- System Clock: 1.8 GHz
- Memory BlockSize assumed to be 64 bytes
- Similar number of reads from memory in both codes
- Dramatically improved Read Bandwidth
- Significantly improved Write Bandwidth

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

L1 Instruction Cache Access

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

PAPI_L1_ICR                           3014322225   3014205662
INSTRUCTION_CACHE_REFILLS_FROM_L2              22            3
INSTRUCTION_CACHE_REFILLS_FROM_SYSTEM          73           36
-----------------------------
PAPI_L1_ICR                           3014322033   3014205070
PAPI_L1_ICM                                   60           44

Instr Cache Request Rate              0.3347 req/inst   0.3346 req/inst
Instr Cache Miss Rate                 0.0000 miss/inst       0.0000 miss/inst
Instr Cache Miss Ratio                0.0000 miss/req   0.0000 miss/req
```

- Two techniques, like Data Cache case
  - First uses native events
  - Second uses PAPI presets only
- Small subroutines fit completely in cache
- Virtually no misses; pretty boring

L2 Cache Access

Measurement	Classic mat_mul	Reordered mat_mul
Direct L2 Cache Test		
REQUESTS_TO_L2:INSTRUCTIONS:DATA:TLB_WALK	1057556622	70996294
L2_CACHE_MISS:INSTRUCTIONS:DATA:TLB_WALK	62120093	4167947
L2_FILL_WRITEBACK:ALL	268201418	122740586
L2 Cache Request Rate	0.1472 req/inst	0.0215 req/inst
L2 Cache Miss Rate	0.0069 miss/inst	0.0005 miss/inst
L2 Cache Miss Ratio	0.0469 miss/req	0.0215 miss/req
Indirect L2 Cache Test		
INSTRUCTION_CACHE_REFILLS_FROM_L2	4	0
INSTRUCTION_CACHE_REFILLS_FROM_SYSTEM	30	9
L2_CACHE_MISS:TLB_WALK	260	5438
REQUESTS_TO_L2:TLB_WALK	787632271	803242
DATA_CACHE_REFILLS:L2_SHARED:L2_EXCLUSIVE:L2_OWNED:L2_MODIFIED	205977083	60715886
DATA_CACHE_REFILLS_FROM_SYSTEM:SHARED:EXCLUSIVE:OWNED:MODIFIED	61973057	1950318
L2 Cache Request Rate	0.1172 req/inst	0.0070 req/inst
L2 Cache Miss Rate	0.0069 miss/inst	0.0002 miss/inst
L2 Cache Miss Ratio	0.0587 miss/req	0.0308 miss/req

L2 Cache Access

Measurement	Classic mat_mul	Reordered mat_mul
L2 Cache Request Rate	0.1172 req/inst	0.0070 req/inst
L2 Cache Miss Rate	0.0069 miss/inst	0.0002 miss/inst
L2 Cache Miss Ratio	0.0587 miss/req	0.0308 miss/req
L2 Instr Fraction	0.0000	0.0000
L2 Data Fraction	0.2538	0.9873
L2 TLB Fraction	0.7462	0.0127

- L2 cache is unified on Opteron
- Two techniques:
  - First is coarser grained
  - Second provides more detail but requires 7 events (two passes)
  - No major differences for this code
- L2 requests and misses down dramatically in reordered code
  - Recall, memory accesses are up by 50%
- Almost all (98+%) L2 access are for data in reordered code

DTLB Access

Measurement	Classic mat_mul	Reordered mat_mul
PAPI_L1_DCA	2002809207	3008530341
L1_DTLB_MISS_AND_L2_DTLB_HIT:ALL	296943120	350824
L1_DTLB_AND_L2_DTLB_MISS:ALL	783208861	785470
L1 DTLB Request Rate	0.2224 req/inst	0.3339 req/inst
L1 DTLB Miss Rate	0.1199 miss/inst	0.0001 miss/inst
L1 DTLB Miss Ratio	0.5393 miss/req	0.0004 miss/req
L2 DTLB Request Rate	0.1199 req/inst	0.0001 req/inst
L2 DTLB Miss Rate	0.0870 miss/inst	0.0001 miss/inst
L2 DTLB Miss Ratio	0.7251 miss/req	0.6913 miss/req

- Goto and van de Geijn claim TLB misses can limit fast matrix multiply ("On Reducing TLB Misses in Matrix Multiplication")
- L1 Data Cache Access == DTLB Access
- More L1 requests in improved code
- Dramatically fewer misses

ITLB Access

Measurement	Classic mat_mul	Reordered mat_mul
PAPI_L1_ICR	3014320811	3014204576
L1_ITLB_MISS_AND_L2_ITLB_HIT	4	1
L1_ITLB_MISS_AND_L2_ITLB_MISS:ALL	9	6
L1 ITLB Request Rate	0.3347 req/inst	0.3346 req/inst
L1 ITLB Miss Rate	0.0000 miss/inst	0.0000 miss/inst
L1 ITLB Miss Ratio	0.0000 miss/req	0.0000 miss/req
L2 ITLB Request Rate	0.0000 req/inst	0.0000 req/inst
L2 ITLB Miss Rate	0.0000 miss/inst	0.0000 miss/inst
L2 ITLB Miss Ratio	0.6923 miss/req	0.8571 miss/req

- See DTLB…
- L1 Instruction Cache Reads == ITLB Access
- Boring…but useful in identifying code layout problems

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.

Homework

- You Pick Two
  - different architectures
  - find machines with PAPI or install your own
- Repeat IPC and L1 Dcache
  - use presets and native events
  - native events may be different on different architectures
- Explain what you see
- HINT: papi_command_line
- See resources on next slide and slide 2

Resources

- Intel Developers Guide, Vol 3B (Chapters 18, 19)
  - http://download.intel.com/products/processor/manual/253669.pdf
- AMD BIOS and Kernel Developer Guides
  - http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/
- Performance Analysis Examples
  - http://developer.amd.com/wordpress/media/2012/10/Basic_Performance_Measurements.pdf
  - http://developer.amd.com/wordpress/media/2012/10/Introduction_to_CodeAnalyst.pdf
- **Performance Tuning of Scientific Applications**
  - http://www.amazon.com/Performance-Scientific-Applications-Chapman-Computational/dp/1439815690
  - http://booksgreatchoice.com/getbook/p236074/
    (sign up for a 1 day account for $3.90)