CS 594: SCIENTIFIC COMPUTING FOR ENGINEERS

PAPI
Performance Application Programming Interface

Heike McCraw
mccraw@icl.utk.edu
1. Motivation
   - What is Performance?
   - Why being annoyed with Performance Analysis?

2. Concepts and Definitions
   - The performance analysis cycle
   - Measurement: profiling vs. tracing
   - Analysis: manual vs. automated

3. Performance Analysis Tools
   - PAPI: Access to hardware performance counters
   - Vampir Suite: Instrumentation and Trace visualization
   - KOJAK / Scalasca: automatic performance analysis tool
   - TAU: Toolset for profiling and tracing of MPI/OpenMP/Java/Python applications
WHY PERFORMANCE ANALYSIS?

Performance Analysis is important:

- **Large investments in HPC systems**
  - Procurement costs: ~$40 Mio
  - Operational costs: ~$5 Mio per year
  - Electricity costs: 1 MW / year ~$1 Mio

- **Efficient usage** is important because of expensive and limited resources
- **Scalability** is important to achieve next bigger simulation

- Performance analysis: *Get highest performance for a given cost*
- „Performance Analyst“: Anyone who is associated with computer systems,
  - i.e. system engineers, computer scientists, application developers and of course users
Performance Optimization cycle:

Measure & Analyze:
- Have an optimization phase
- just like testing & debugging phase
- Do profiling and tracing
- Use tools!
- avoid do-it-yourself with `printf` solutions
- … seriously!
WHAT ARE HARDWARE PERFORMANCE COUNTERS?

For many years, hardware engineers have designed in specialized registers to measure the performance of various aspects of a microprocessor.

HW performance counters provide application developers with valuable information about code sections that can be improved.

Hardware performance counters can provide insight into:

- Whole program timing
- Cache behaviors
- Branch behaviors
- Memory and resource contention and access patterns
- Pipeline stalls
- Floating point efficiency
- Instructions per cycle
- Subroutine resolution
- Process or thread attribution
PAPI

- Middleware that provides a consistent interface and methodology for the performance counter hardware found in most major microprocessors
- PAPI enables software engineers to see, in near real time, the relation between software performance and hardware events

**SUPPORTED ARCHITECTURES:**

- AMD
- ARM Cortex A8, A9, A15 (coming Soon: ARM64)
- CRAY
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, CNK, (coming soon: EMON2 power)
- IBM Power Series
- Intel Nehalem, Westmere, Sandy Bridge, Ivy Bridge, Haswell, Knights Corner
- NVidia Tesla, Kepler, NVML
- Infiniband
- Intel RAPL (power/energy)
- Intel MIC power/energy

**COMPONENT PAPI:**

- provides access to a collection of components that expose performance measurement opportunities across the system as a whole, including network, the I/O system, the Compute Node Kernel, power/energy
PAPI HARDWARE EVENTS

- Countable events are defined in two ways:
  - Platform-neutral **Preset Events** (e.g., PAPI_TOT_INS)
  - Platform-dependent **Native Events** (e.g., L3_CACHE_MISS)

- Preset Events can be **derived** from multiple Native Events
  (e.g. PAPI_L1_TCM might be the sum of L1 Data Misses and L1 Instruction Misses on a given platform)
PAPI HARDWARE 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 \texttt{papi_avail} to see what preset events are available on a given platform

Native Events

- Any event countable by the CPU
- Same interface as for preset events
- Use \texttt{papi_native_avail} utility to see all available native events

Use \texttt{papi_event_chooser} utility to select a compatible set of events
PAPI provides 3 interfaces to the underlying counter hardware:

1. A Low Level API manages hardware events (preset and native) in user defined groups called EventSets. Meant for experienced application programmers wanting fine-grained measurements.

2. A High Level API provides the ability to start, stop and read the counters for a specified list of events (preset only). Meant for programmers wanting simple event measurements.

3. Graphical and end-user tools provide facile data collection and visualization.
PAPI HIGH LEVEL CALLS

- **PAPI_num_counters()**
  - get the number of hardware counters available on the system

- **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

- **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

- **PAPI_ipc (float *rtime, float *ptime, long long *ins, float *ipc)**
  - gets instructions per cycle, real and processor time

- **PAPI_accum_counters (long long *values, int array_len)**
  - add current counts to array and reset counters

- **PAPI_read_counters (long long *values, int array_len)**
  - copy current counts to array and reset counters

- **PAPI_start_counters (int *events, int array_len)**
  - start counting hardware events

- **PAPI_stop_counters (long long *values, int array_len)**
  - stop counters and return current counts
EXAMPLE: LOW LEVEL API

#include "papi.h"
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={ PAPI_FP_OPS, PAPI_TOT_CYC };
int EventSet = PAPI_NULL;
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);
krakenpf7: cs594> 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
krakenpf7: cs594>**papi_avail** -h
Usage: 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

This program provides information about PAPI preset and native events.
krakenpf7: cs594> `aprun -n1 papi_avail -a`

Available events and hardware information.

```
PAPI Version             : 3.6.2.2
Vendor string and code   : AuthenticAMD (2)
Model string and code    : 6-Core AMD Opteron(tm) Processor 23 (D0) (16)
CPU Revision             : 0.000000
CPU Megahertz            : 2600.000000
CPU Clock Megahertz      : 2600
CPU's in this Node       : 12
Nodes in this System     : 1
Total CPU's              : 12
Number Hardware Counters : 4
Max Multiplex Counters   : 512
```

The following correspond to fields in the PAPI_event_info_t structure.

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Deriv</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>0x80000002</td>
<td>No</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>0x80000003</td>
<td>No</td>
<td>Level 2 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L1_TCM</td>
<td>0x80000006</td>
<td>Yes</td>
<td>Level 1 cache misses</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>0x80000066</td>
<td>No</td>
<td>Floating point operations</td>
</tr>
</tbody>
</table>

Of 41 available events, 9 are derived.
PAPI UTILITIES: PAPI_AVAIL

<table>
<thead>
<tr>
<th>krakenpf7: cs594&gt; aprun -n1 papi_avail -e PAPI_L1_TCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event name: PAPI_L1_TCM</td>
</tr>
<tr>
<td>Event Code: 0x80000006</td>
</tr>
<tr>
<td>Number of Native Events: 2</td>
</tr>
<tr>
<td>Short Description: L1 cache misses</td>
</tr>
<tr>
<td>Long Description: Level 1 cache misses</td>
</tr>
<tr>
<td>Developer's Notes:</td>
</tr>
<tr>
<td>Derived Type: DERIVED_ADD</td>
</tr>
<tr>
<td>Postfix Processing String:</td>
</tr>
<tr>
<td>Native Code[0]: 0x40000029 INSTRUCTION_CACHE_MISSES</td>
</tr>
<tr>
<td>Number of Register Values: 4</td>
</tr>
<tr>
<td>Register[ 0]: 0x00000081 Event Code</td>
</tr>
<tr>
<td>Register[ 1]: 0x00000081 Event Code</td>
</tr>
<tr>
<td>Register[ 2]: 0x00000081 Event Code</td>
</tr>
<tr>
<td>Register[ 3]: 0x00000081 Event Code</td>
</tr>
<tr>
<td>Native Event Description: Instruction Cache Misses</td>
</tr>
<tr>
<td>Native Code[1]: 0x40000011 DATA_CACHE_MISSES</td>
</tr>
<tr>
<td>Number of Register Values: 4</td>
</tr>
<tr>
<td>Register[ 0]: 0x00000041 Event Code</td>
</tr>
<tr>
<td>Register[ 1]: 0x00000041 Event Code</td>
</tr>
<tr>
<td>Register[ 2]: 0x00000041 Event Code</td>
</tr>
<tr>
<td>Register[ 3]: 0x00000041 Event Code</td>
</tr>
<tr>
<td>Native Event Description: Data Cache Misses</td>
</tr>
</tbody>
</table>

2/25/15
krakenpf7: cs594> `aprun -n1 papi_native_avail`

Available native events and hardware information.

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000003</td>
<td>RETIRED_SSE_OPERATIONS</td>
<td>Retired SSE Operations</td>
</tr>
<tr>
<td>40001003</td>
<td>:SINGLE_ADD_SUB_OPS</td>
<td>Single precision add/subtract ops</td>
</tr>
<tr>
<td>40002003</td>
<td>:SINGLE_MUL_OPS</td>
<td>Single precision multiply ops</td>
</tr>
<tr>
<td>40004003</td>
<td>:SINGLE_DIV_OPS</td>
<td>Single precision divide/square root ops</td>
</tr>
<tr>
<td>40008003</td>
<td>:DOUBLE_ADD_SUB_OPS</td>
<td>Double precision add/subtract ops</td>
</tr>
<tr>
<td>40010003</td>
<td>:DOUBLE_MUL_OPS</td>
<td>Double precision multiply ops</td>
</tr>
<tr>
<td>40020003</td>
<td>:DOUBLE_DIV_OPS</td>
<td>Double precision divide/square root ops</td>
</tr>
<tr>
<td>40040003</td>
<td>:OP_TYPE</td>
<td>Op type: 0=uops. 1=FLOPS</td>
</tr>
<tr>
<td>40080003</td>
<td>:ALL</td>
<td>All sub-events selected</td>
</tr>
</tbody>
</table>

Total events reported: 114
PAPI UTILITIES: PAPI_EVENT_CHOOSER

krakenpf7: cs594> aprun -n1 papi_event_chooser

Usage:
papi_event_chooser NATIVE|PRESET evt1 evt2 ...
PAPI UTILITIES: \textit{PAPI\_EVENT\_CHOOSER}

```
krakenpf7: cs594> aprun -n1 papi_event_chooser PRESET PAPI_L1_TCM

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Deriv</th>
<th>Description (Note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>0x80000002</td>
<td>No</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>0x80000003</td>
<td>No</td>
<td>Level 2 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_TCM</td>
<td>0x80000007</td>
<td>No</td>
<td>Level 2 cache misses</td>
</tr>
<tr>
<td>PAPI_L3_TCM</td>
<td>0x80000008</td>
<td>No</td>
<td>Level 3 cache misses</td>
</tr>
<tr>
<td>PAPI_FPU_IDL</td>
<td>0x80000012</td>
<td>No</td>
<td>Cycles floating point units are idle</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0x80000014</td>
<td>No</td>
<td>Data translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_TLB_IM</td>
<td>0x80000015</td>
<td>No</td>
<td>Instruction translation lookaside buffer miss</td>
</tr>
<tr>
<td>PAPI_TLB_TL</td>
<td>0x80000016</td>
<td>Yes</td>
<td>Total translation lookaside buffer misses</td>
</tr>
</tbody>
</table>

[...]

PAPI_FP_OPS  | 0x80000066  | No    | Floating point operations                    |

-------------------------------------------------------------------------
Total events reported: 39
```
PAPI UTILITIES: *PAPI_EVENT_CHOOSER*

```bash
krakenpf7: cs594> aprun -n1 papi_event_chooser
    PRESET PAPI_L1_TCM PAPI_TLB_TL

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Deriv</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0x80000014</td>
<td>No</td>
<td>Data translation lookaside buffer</td>
</tr>
<tr>
<td>PAPI_TLB_IM</td>
<td>0x80000015</td>
<td>No</td>
<td>Instruction translation lookaside</td>
</tr>
</tbody>
</table>

Total events reported: 4
krakenpf7: cs594> aprun -n1 papi_command_line PAPI_FP_OPS
Successfully added: PAPI_FP_OPS

PAPI_FP_OPS :  40000000

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

krakenpf7: cs594> aprun -n1 papi_command_line PAPI_FP_OPS PAPI_L1_TCM
Successfully added: PAPI_FP_OPS
Successfully added: PAPI_L1_TCM

PAPI_FP_OPS :  40000000
PAPI_L1_TCM :  40
PERFORMANCE MEASUREMENT CATEGORIES

• **Efficiency**
  o Instructions per cycle (IPC)
  o Memory bandwidth

• **Caches**
  o Data cache misses and miss ratio
  o Instruction cache misses and miss ratio

• **L2 cache misses and miss ratio**

• **Translation lookaside buffers (TLB)**
  o Data TLB misses and miss ratio
  o Instruction TLB misses and miss ratio

• **Control transfers**
  o Branch mispredictions
  o Near return mispredictions
#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.
• Measure instruction level parallelism
• An indicator of code efficiency

```c
int events[] = {PAPI_TOT_CYC, PAPI_TOT_INS};
realtime[0] = PAPI_get_real_usecs();
retval = PAPI_start_counters(events, 2);
classic_matmula();
retval = PAPI_stop_counters(cvalues, 2);
realtime[1] = PAPI_get_real_usecs();
```

```
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_usecs();
retval = PAPI_start(EventSet);
classic_matmula();
retval = PAPI_stop(EventSet, cvalues);
realtime[1] = PAPI_get_real_usecs();
```
### IPC – INSTRUCTIONS PER CYCLE

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Classic mat_mul</th>
<th>Reordered mat_mul</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High Level IPC Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(PAPI_{start,stop}_counters)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real time</td>
<td>13.6106 sec</td>
<td>2.9762 sec</td>
</tr>
<tr>
<td>IPC</td>
<td>0.3697</td>
<td>1.6939</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>24362605525</td>
<td>5318626915</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>9007034503</td>
<td>9009011245</td>
</tr>
<tr>
<td>Low Level IPC Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(PAPI low level calls)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real time</td>
<td>13.6113 sec</td>
<td>2.9772 sec</td>
</tr>
<tr>
<td>IPC</td>
<td>0.3697</td>
<td>1.6933</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>24362750167</td>
<td>5320395138</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>9007034381</td>
<td>9009011130</td>
</tr>
</tbody>
</table>

- Both PAPI methods are consistent
- Roughly 460% improvement in reordered code
DATA CACHE ACCESS

Cache miss: a failed attempt to read or write a piece of data in the cache
→ Results in main memory access with much longer latency
→ Important to keep data as close as possible to CPU

Data Cache Misses can be considered in 3 categories:

- **Compulsory misses:** Occurs on first reference to a data item
  - Prefetching can help

- **Capacity misses:** Occurs when the working set exceeds the cache capacity
  - **Spatial locality:** use all the data that is loaded into the cache
  - Smaller working set (blocking/tiling algorithms)

- **Conflict misses:** Occurs when a data item is referenced after the cache line containing the item was evicted earlier.
  - **Temporal locality:** reuse a word as long as possible
  - Data layout; memory access patterns
# L1 DATA CACHE ACCESS

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Classic mat_mul</th>
<th>Reordered mat_mul</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PAPI NATIVE EVENTS:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_CACHE_ACCESSES</td>
<td>2,002,807,841</td>
<td>3,008,528,961</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS:L2_MODIFIED:L2_OWNED:L2_EXCLUSIVE:L2_SHARED</td>
<td>205,968,263</td>
<td>60,716,301</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_SYSTEM:MODIFIED:OWNED:EXCLUSIVE:SHARED</td>
<td>61,970,925</td>
<td>1,950,282</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PAPI PRESET EVENTS:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>2,002,808,034</td>
<td>3,008,528,895</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>268,010,587</td>
<td>62,680,818</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Cache Request Rate</td>
<td>0.2224 req/inst</td>
<td>0.3339 req/inst</td>
</tr>
<tr>
<td>Data Cache Miss Rate</td>
<td>0.0298 miss/inst</td>
<td>0.0070 miss/inst</td>
</tr>
<tr>
<td>Data Cache Miss Ratio</td>
<td>0.1338 miss/req</td>
<td>0.0208 miss/req</td>
</tr>
</tbody>
</table>

- **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
3rd Party Tools Applying PAPI

- PaRSEC (UTK) [http://icl.cs.utk.edu/parsec/]
- TAU (U Oregon) [http://www.cs.uoregon.edu/research/tau/]
- PerfSuite (NCSA) [http://perfsuite.ncsa.uiuc.edu/]
- HPCToolkit (Rice University) [http://hpctoolkit.org/]
- KOJAK and SCALASCA (FZ Juelich, UTK) [http://icl.cs.utk.edu/kojak/]
- VampirTrace and Vampir (TU Dresden) [http://www.vamir.eu]
- Open|Speedshop (SGI) [http://oss.sgi.com/projects/openspeedshop/]
- SvPablo (UNC Renaissance Computing Institute) [http://www.renci.org/research/pablo/]
- ompP (UTK) [http://www.ompp-tool.com]