COSC 594: SCIENTIFIC COMPUTING FOR ENGINEERS

PAPI
Performance Application Programming Interface

Heike Jagode, Anthony Danalis

jagode@icl.utk.edu
adanalis@icl.utk.edu

2/27/18
OUTLINE

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?

• Large investments in HPC systems
  o Procurement costs: ~$40 Mio
  o Operational costs: ~$5 Mio per year
  o 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

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system: i.e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events across the entire compute system**
PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system:
  i.e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.

- PAPI enables software engineers to see, in near real time, the relation between
  **SW performance** and **HW events across the entire compute system**

**SUPPORTED ARCHITECTURES:**
- AMD
- IBM Blue Gene Series
- IBM Power Series
- Intel Westmere, Sandy|Ivy Bridge, Haswell, **Broadwell, Skylake**, KNC, **Knights Landing**
PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system:
  - i.e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events across the entire compute system**

**SUPPORTED ARCHITECTURES:**

- AMD
- ARM Cortex A8, A9, A15, ARM64
- CRAY: Gemini and Aries interconnects, power
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series
- Intel Westmere, Sandy/Ivy Bridge, Haswell, **Broadwell, Skylake**, KNC, **Knights Landing**
- Intel KNC, **Knights Landing power/energy**
- Intel RAPL (power/energy); **power capping**
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler: CUDA support for multiple GPUs; PC Sampling
- NVIDIA NVML
- Virtual Environments: VMware, KVM
PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system: i.e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events across the entire compute system**

**SUPPORTED ARCHITECTURES:**
- AMD
- ARM Cortex A8, A9, A15, ARM64
- CRAY: Gemini and Aries interconnects, power
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series
- Intel Westmere, SandyIvy Bridge, Haswell, **Broadwell, Skylake**, KNC, **Knights Landing**
- Intel KNC, **Knights Landing power/energy**
- Intel RAPL (power/energy); **power capping**
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler: CUDA support for multiple GPUs; PC Sampling
- NVIDIA NVML
- Virtual Environments: VMware, KVM
PAPI HARDWARE EVENTS

• Countable events are defined in two ways:
  o Platform-neutral **Preset Events** (e.g., PAPI_TOT_INS)
  o 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 `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 `papi_native_avail` utility to see all available native events

Use `papi_event_chooser` utility to select a compatible set of events
PAPI COUNTER INTERFACES

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.

1. 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

```c
#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);
```
PAPI UTILITIES: PAPI_COST

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
PAPI UTILITIES: PAPI_AVAIL

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.
**PAPI UTILITIES: PAPI_AVAIL**

```
krakenpf7: cs594> aprun -n1 papi_avail
```

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>Avail</th>
<th>Deriv</th>
<th>Description (Note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>0x80000002</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>0x80000003</td>
<td>Yes</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>Yes</td>
<td>Level 1 cache misses</td>
</tr>
</tbody>
</table>

[...]

Of 103 possible events, 41 are available, of which 9 are derived.
krakenpf7: cs594> aprun -nl 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 (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_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**

```
krakenpf7: cs594> aprun -n1 papi_avail -e PAPI_L1_TCM
[...]
Event name: PAPI_L1_TCM
Event Code: 0x80000006
Number of Native Events: 2
Short Description: L1 cache misses
Long Description: Level 1 cache misses
Developer's Notes: |
Derived Type: DERIVED_ADD
Postfix Processing String: |
Native Code[0]: 0x40000029 |INSTRUCTION_CACHE_MISSES|
Number of Register Values: 4
Register[ 0]: 0x00000081 |Event Code|
Register[ 1]: 0x00000081 |Event Code|
Register[ 2]: 0x00000081 |Event Code|
Register[ 3]: 0x00000081 |Event Code|
Native Event Description: Instruction Cache Misses
Native Code[1]: 0x40000011 |DATA_CACHE_MISSES|
Number of Register Values: 4
Register[ 0]: 0x00000041 |Event Code|
Register[ 1]: 0x00000041 |Event Code|
Register[ 2]: 0x00000041 |Event Code|
Register[ 3]: 0x00000041 |Event Code|
Native Event Description: Data Cache Misses
```
## PAPI UTILITIES: \textit{PAPI\_NATIVE\_AVAIL}

### Command

```
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>

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000010</td>
<td>DATA_CACHE_ACCESSES</td>
<td>Data Cache Accesses</td>
</tr>
<tr>
<td>0x40000011</td>
<td>DATA_CACHE_MISSES</td>
<td>Data Cache Misses</td>
</tr>
</tbody>
</table>

Total events reported: 114
PAPI UTILITIES: \texttt{PAPI\_EVENT\_CHOOSER}

```
krakenpf7: cs594> \texttt{aprun -n1 papi\_event\_chooser}
```

Usage:
```
papi\_event\_chooser \texttt{NATIVE|PRESET} \texttt{evt1 evt2 ...}
```

PAPI UTILITIES: **PAPI_EVENT_CHOOSER**

```bash
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</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

### 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 (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_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>
</tbody>
</table>

---

Total events reported: 4
PAPI UTILITIES: PAPI_COMMAND_LINE

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**
  - Instructions per cycle (IPC)
  - Memory bandwidth

• **Caches**
  - Data cache misses and miss ratio
  - Instruction 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 CODE

```c
#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

- Measure instruction level parallelism
- An indicator of code efficiency

```c
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 High Level

PAPI Low Level
IPC – INSTRUCTIONS PER CYCLE

- Measure instruction level parallelism
- An indicator of code efficiency

```c
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();
```

```c
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

<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 (PAPI_{\text{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 (PAPI \text{ 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 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
  o Prefetching can help

• **Capacity misses:** Occurs when the working set exceeds the cache capacity
  o **Spatial locality:** use all the data that is loaded into the cache
  o 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.
  o **Temporal locality:** reuse a word as long as possible
  o 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>PAPI NATIVE EVENTS:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_CACHE_ACCESSSES</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>PAPI PRESET EVENTS:</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>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
Counter Inspection Toolkit (CIT)
Key concepts

• **Goal:**

Create a set of micro-benchmarks for illustrating details in hardware events and how they relate to the behavior of the micro-architecture

• **Target audience:**

• Performance conscious application developers
• PAPI developers working on new architectures (think preset events)
• Developers interested in validating hardware event counters
Why bother?

Problem:
Hardware has become some complex that developers do not understand what it does and how it does it.

As a result it is not obvious:
• What to optimize
• How to optimize it
• How to measure the problem
Example: Loop unswitching

Compiler optimization that moves a conditional, which is inside a loop, outside of the loop
Example: Loop unswitching

Compiler optimization that moves a conditional, which is inside a loop, outside of the loop

Original

```c
for(i = 0; i < 1000; i++){
    x[i] += y[i];
    if(cond)
        y[i] = 0;
}
```
Example: Loop unswitching

Compiler optimization that moves a conditional, which is inside a loop, outside of the loop

Original

```c
for(i = 0; i < 1000; i++){
    x[i] += y[i];
    if(cond)
        y[i] = 0;
}
```

Unswitched

```c
if(cond){
    for(i = 0; i < 1000; i++){
        x[i] += y[i];
        y[i] = 0;
    }
}
else{
    for(i = 0; i < 1000; i++){
        x[i] += y[i];
    }
}
```
Why is loop unswitching beneficial?

• Common (miss)conception:
  • Because branches inside loops hurt performance

• Reality:
  • Because branches in loops inhibit compiler optimization
goto or if in loop? choose your poison

```java
while( list.has_elements() ){
    elem = list.pop();
    while( elem.has_members() ){
        rslt = process( elem.next_member() );
        if( rslt ){
            goto after_loop;
        }
    }
}

after_loop:
```
Micro-benchmark: branch in loop

bench1:
Pointer chaining

```c
for(i=0; i<size; i++){
    ptr = *ptr;
}
```

bench2:
Pointer chaining + branch

```c
for(i=0; i<size; i++){
    ptr = *ptr;
    if( 0 == (ptr&0xFF) ){
        RNG( val )
        sum += val;
    }
}
```

Question:
How does the branch affect performance?
Example: Branch instruction count

```c
temp = 0;
do{
    temp++;
    if( (temp % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size);
```

```c
temp = 0;
do{
    temp++;
    random_number( result );
    if ( (result % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size );
```

Question:
How many branch instructions will these codes execute per iteration?
Example: Branch instr. count disasm

temp = 0;
do{
    temp++;
    if ( (temp % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size );

temp = 0;
do{
    temp++;
    random_number( result );
    if ( (result % 2) == 0 ){
        global_var += 2;
    }
}<310>: mov eax,DWORD PTR [rip+0x2012b1],eax
<316>: add eax,0x1
<319>: mov DWORD PTR [rip+0x2012b1],eax
<325>: mov eax,DWORD PTR [rip+0x2012ab]
<331>: and eax,0x1
<334>: test eax,eax
<336>: jne 0x400e77 <353>
<351>: mov eax,DWORD PTR [rip+0x2012e3]
<358>: add eax,0x2
<361>: mov DWORD PTR [rip+0x2012e6],eax
<363>: cmp eax,DWORD PTR [rbp-0x24]
<365>: jl 0x400e4c <310>
<368>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
<365>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<362>: jl 0x400e4c <310>
Example: Branch instr. count disasm

```
Example:
Branch
temp = 0;
doinstr.
    temp++;
    if( (temp % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size);
```

```
Expectation:
2
temp = 0;
doinstr.
    temp++;
    random_number( result );
    if ( (result % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size )
```

```
Expectation: 2
```

```
<310>: mov eax,DWORD PTR [rip+0x2012ba]
<316>: add eax,0x1
<319>: mov DWORD PTR [rip+0x2012ba],eax
<325>: mov eax,DWORD PTR [rip+0x2012ab]
<331>: and eax,0x1
<336>: test eax,eax
<337>: jne <353>   <338>: mov eax,DWORD PTR [rip+0x20127e]
<344>: add eax,0x2
<347>: mov DWORD PTR [rip+0x201275],eax
<353>: mov eax,DWORD PTR [rip+0x20128f]
<359>: cmp eax,DWORD PTR [rbp-0x24]
<360>: jl <310>
```

```
<54>: pseudo_random_generator();
<585>: mov DWORD PTR [rip+0x20119f],eax
<591>: mov DWORD PTR [rip+0x201199],eax
<610>: add eax,0x2
<613>: mov DWORD PTR [rip+0x201174],eax
<620>: add eax,0x2
<623>: mov DWORD PTR [rip+0x201185],eax
<630>: cmp eax,DWORD PTR [rbp-0x24]
<638>: jl <310>
```
Example: Branch instr. count disasm

Measured: 2 2.5
Example: Branch instr. count disasm

```c
temp = 0;
do{
    temp++;
    if( (temp % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size);
```

```c
// Example with random branch count

```
Example: Branch instr. count disasm

```
temp = 0;
do{
    temp++;
    if( (temp % 2) == 0 ){
        global_var += 2;
    }
}while( temp < size );
```

```
temp = 0;
do{
    temp++;
    random_number( result );
    if( (result % 2) == 0 ){
        global_var += 2;
    }
    random_number( result );
}while( temp < size );
```

Measured:

2  2.5
Example: Branch instr. count disasm

temp = 0;
do{
    temp++;
    if( (temp / 2) == 0 ){
        EXECUTED: 2
        RETIRED: 2
        MISSPREDICTED: 0
    }
}while( temp < size);

temp = 0;
do{
    temp++;
    random_number( result );
    if( (result % 2) == 0 ){
        EXECUTED: 2.5
        RETIRED: 2
        MISSPREDICTED: 0.5
        global_var += 2;
    }
} while( temp < size );

temp = 0;
do{
    random_number( result );
    if( (result % 2) == 0 ){
        EXECUTED: 2
        RETIRED: 2
        MISSPREDICTED: 0.5
        global_var += 2;
    }
} while( temp < size );
Example: Branch instr. count disasm

Executed: 1
Retired: 1
Taken: 0.5
Misspredicted: 0.5
Example: Branch instr. count disasm

```
Example:
Branch instr. count disasm

```
Take away message

- “Executed” does NOT mean what many software developers think it does
- “Retired” means “Really Executed”
- The effect of missprediction depends on the code that follows the branch.
Event signature

```
do{
    if ( temp < (size/2) ){
        global_var2 += 2;
    }
    random_number( result );
    temp++;
}while( temp < size );
(a) C=2, T=1.5
```

```
do{
    global_var2 += 2;
    if ( temp < global_var2 ){
        global_var1 += 2;
    }
    random_number( result );
    temp++;
}while( temp < size );
(b) C=2, T=1
```

```
do{
    global_var2 += 2;
    if ( temp > global_var2 ){
        global_var1 += 2;
    }
    random_number( result );
    temp++;
}while( temp < size );
(c) C=2, T=2
```

```
do{
    random_number( result );
    global_var2 += 2;
    if ( (result % 2) == 0 ){
        global_var1 += 2;
    }
    random_number( result );
    temp++;
}while( temp < size );
(d) C=2, T=1.5, M=0.5
```

```
do{
    random_number( result );
    global_var2 += 2;
    if ( (result % 2) == 0 ){
        global_var1 += 2;
    }
    temp++;
}while( temp < size );
(e) C:E=2.5, C:R=2, T=1.5, M=0.5
```

```
do{
    global_var2 += 2;
    if ( temp < global_var2 ){
        global_var1 += 2;
        goto zz;
    }
    random_number( result );
    zz: temp++;
    random_number( result );
}while( temp < size );
(f) C=2, T=1, D=1
```

```
do{
    global_var2 += 2;
    temp++;
}while( temp < size );
(g) C=1, T=1
```
Loop without conditional branch

All these kernels were inside a “do{   }while(C);”

Can I create a loop (code with iterations) without using a conditional jump?
Loop without conditional branch (kernel)

count(ptr, step) {
    start:
    i += step;
    trigger = ptr[i];

    random_number(result);
    if (0 == (result % 2)) {
        global_var++;
    } else {
        global_var += 2;
    }
    goto start;
}
Loop without conditional branch (driver)

```c
act.sa_handler = handler;
sigaction(SIGSEGV, &act, NULL);

ptr = mmap(NULL, (pcnt+1)*psz, PROT_READ,...);
mprotect(ptr+pcnt*psz, 1, PROT_NONE);

step = (pcnt*psz)/iter_cnt;
count(ptr, step);
```
branch in loop revisited

bench1:
Pointer chaining

```c
for(i=0; i<size; i++){
    ptr = *ptr;
}
```

bench2:
Pointer chaining + branch

```c
for(i=0; i<size; i++){
    ptr = *ptr;
    if( 0 == (ptr&0xFF) ){
        RNG( val )
        sum += val;
    }
}
```
branch in loop revisited

bench1:
Pointer chaining

```c
for(i=0; i<size; i++){
    ptr = *ptr;
}
```

bench2:
Pointer chaining + branch

```c
for(i=0; i<size; i++){
    ptr = *ptr;
    if( 0 == (ptr&0xFF) ){
        RNG( val )
        sum += val;
    }
}
```

These two micro-benchmarks have same performance!
Characterization of Native Events

![Graph showing the average count per 100 accesses for L1 HIT, L2 HIT, L3 HIT, L1 MISS, L2 MISS, and L3 MISS with buffer size in KB on the x-axis and average count per 100 accesses on the y-axis.]
Discovering cache behavior (prefetc)
LRU replacement policy

Fits in L1
No
Eviction

offset=0
Cascading
Eviction

offset=1
Minimal
Eviction

32KB

36KB

36KB
LRU replacement policy

- Fits in L1: No Eviction
- Offset = 0: Cascading Eviction
- Offset = 1: Minimal Eviction
LRU replacement policy

- Fits in L1:
  - No Eviction
  - 32KB

- offset=0:
  - Cascading Eviction
  - 36KB

- offset=1:
  - Minimal Eviction
  - 36KB
Optimizing for cache replacement

Average Access Time in ns vs. Buffer size in KB for different offsets.
Conclusions

• Hardware is complex
• It gets more complex with every generation
• Experts can use help to reduce the pain
• Developers need help to understand the details
• PAPI, through CIT, is aiming to provide some help in
  • Categorizing, Validating, Understanding
3rd Party Tools applying PAPI

- PaRSEC (UTK) [http://icl.cs.utk.edu/parsec/](http://icl.cs.utk.edu/parsec/)
- Caliper (LLNL) [github.com/LLNL/caliper-compiler](https://github.com/LLNL/caliper-compiler)
- Kokkos (SNL) [https://github.com/kokkos](https://github.com/kokkos)
- HPCToolkit (Rice University) [http://hpctoolkit.org/](http://hpctoolkit.org/)
- TAU (U Oregon) [http://www.cs.uoregon.edu/research/tau/](http://www.cs.uoregon.edu/research/tau/)
- Scalasca (FZ Juelich, TU Darmstadt) [http://scalasca.org/](http://scalasca.org/)
- VampirTrace and Vampir (TU Dresden) [http://www.vamir.eu](http://www.vamir.eu)
- PerfSuite (NCSA) [http://perfsuite.ncsa.uiuc.edu/](http://perfsuite.ncsa.uiuc.edu/)
- Open|Speedshop [https://openspeedshop.org/](https://openspeedshop.org/)
- SvPablo (RENCI at UNC) [http://www.renci.org/research/pablo/](http://www.renci.org/research/pablo/)