OUTLINE

1. Motivation
   • What is Performance?
   • Why bother with Performance Analysis?

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

3. Performance Analysis
   • PAPI: Access to hardware performance counters
   • Performance Measurement Categories: MMM example
   • Counter Analysis Toolkit (CAT)
WHY PERFORMANCE ANALYSIS?

- 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

- 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
- ARM Cortex A8, A9, A15, ARM64
- IBM Blue Gene Series
- IBM Power Series, **PCP for P9-uncore**
- Intel Sandy|Ivy Bridge, Haswell, **Broadwell, Skylake()-x, Kaby Lake**, KNC, KNL, Knights Mill
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler, Maxwell, Pascal, Volta: support for multiple GPUs
- NVIDIA NVML (power/energy); power capping
- 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, **GPU Vega**
• ARM Cortex A8, A9, A15, ARM64
• CRAY: Gemini and Aries interconnects
• IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
• IBM Power Series, **PCP for P9-uncore**
• Intel Sandy|Ivy Bridge, Haswell, **Broadwell, Skylake()-x, Kaby Lake**, KNC, KNL, Knights Mill

• InfiniBand
• Lustre FS
• NVIDIA Tesla, Kepler, Maxwell, Pascal, **Volta**: support for multiple GPUs
• NVIDIA NVML
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, **GPU Vega**
- ARM Cortex A8, A9, A15, ARM64
- CRAY: Gemini and Aries interconnects, power/energy
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series, **PCP for P9-uncore**
- Intel Sandy|Ivy Bridge, Haswell, **Broadwell, Skylake()-x, Kaby Lake**, KNC, KNL, Knights Mill
- Intel KNC, KNL, Knights Mill power/energy
- Intel RAPL (power/energy), **power capping**
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler, Maxwell, Pascal, **Volta**: support for multiple GPUs
- NVIDIA NVML (power/energy); **power capping**
- Virtual Environments: VMware, KVM
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 `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 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_hl_region_begin (const char *region)**
  - Read events at the beginning of a region (also start counting the events).

- **PAPI_hl_region_end (const char *region)**
  - Read events at the end of a region and store the difference from the beginning.

- **PAPI_hl_read (const char *region)**
  - Read events inside a region and store the difference from the beginning.

- **PAPI_hl_stop ()**
  - Stop a running high-level event set (optional).

export PAPI_EVENTS="PAPI_TOT_INS,PAPI_TOT_CYC"

https://bitbucket.org/icl/papi/wiki/PAPI-HL.md
PAPI RATE CALLS

- **PAPI_flops_rate**
  - Get Mflops/s (floating point operation rate), real and processor time.
- **PAPI_flips_rate**
  - Get Mflips/s (floating point instruction rate), real and processor time.
- **PAPI_ipc**
  - Get instructions per cycle, real and processor time.
- **PAPI_epc**
  - Get arbitrary events per cycle, real and processor time.
- **PAPI_rate_stop**
  - Stop a running event set of a rate function.

https://bitbucket.org/icl/papi/wiki/PAPI-Rates.md
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);
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
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.

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_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes  Level 1 cache misses

Of 103 possible events, 41 are available, of which 9 are derived.
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 (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>
</tbody>
</table>

Of 41 available events, 9 are derived.
PAPI UTILITIES: \textit{PAPI\_AVAIL}

\begin{verbatim}
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
\end{verbatim}
### PAPI UTILITIES: `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>30001003</td>
<td>:SINGLE_ADD_SUB_OPS</td>
<td>Single precision add/subtract ops</td>
</tr>
<tr>
<td>30002003</td>
<td>:SINGLE_MUL_OPS</td>
<td>Single precision multiply ops</td>
</tr>
<tr>
<td>30004003</td>
<td>:SINGLE_DIV_OPS</td>
<td>Single precision divide/square root ops</td>
</tr>
<tr>
<td>30008003</td>
<td>:DOUBLE_ADD_SUB_OPS</td>
<td>Double precision add/subtract ops</td>
</tr>
<tr>
<td>30010003</td>
<td>:DOUBLE_MUL_OPS</td>
<td>Double precision multiply ops</td>
</tr>
<tr>
<td>30020003</td>
<td>:DOUBLE_DIV_OPS</td>
<td>Double precision divide/square root ops</td>
</tr>
<tr>
<td>30040003</td>
<td>:OP_TYPE</td>
<td>Op type: 0=uops. 1=FLOPS</td>
</tr>
<tr>
<td>30080003</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: `PAPI_EVENT_CHOOSER`**

```bash
krakenpf7: cs594> aprun -n1 papi_eventChooser 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
```
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
#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

```bash
$> export PAPI_EVENTS="PAPI_TOT_CYC, PAPI_TOT_INS"

realtime[0] = PAPI_get_real_usec();
PAPI.hl_region_begin("MATMUL");
classic_matmul();
PAPI.hl_region_end("MATMUL");
realtime[1] = PAPI_get_real_usec();
```

PAPI High Level

PAPI Low Level
• Measure instruction level parallelism
• An indicator of code efficiency

```c
$> \text{export } \text{PAPI\_EVENTS="PAPI\_TOT\_CYC, PAPI\_TOT\_INS"}

\text{realtime[0] = PAPI\_get\_real\_usec();}
\text{PAPI\_hl\_region\_begin("MATMUL");}
\text{classic\_matmul();}
\text{PAPI\_hl\_region\_end("MATMUL");}
\text{realtime[1] = PAPI\_get\_real\_usec();}
```

```c
\text{int events[]} = \{\text{PAPI\_TOT\_CYC, PAPI\_TOT\_INS}\};
\text{retval = PAPI\_library\_init (PAPI\_VER\_CURRENT);} \quad \text{PAPI\_High\_Level}
\text{retval = PAPI\_create\_eventset(&EventSet);} \quad \text{PAPI\_Low\_Level}
\text{retval = PAPI\_add\_events(EventSet, events, 2);} \quad \text{PAPI\_High\_Level}
\text{realtime[0] = PAPI\_get\_real\_usec();}
\text{retval = PAPI\_start(EventSet);} \quad \text{PAPI\_Low\_Level}
\text{classic\_matmul();}
\text{retval = PAPI\_stop(EventSet, cvalues);} \quad \text{PAPI\_High\_Level}
\text{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><strong>High Level IPC Test</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAPI_hl_region_{begin/end}</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><strong>Low Level IPC Test</strong></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 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><strong>Data Cache Accesses</strong></td>
<td>2,002,807,841</td>
<td>3,008,528,961</td>
</tr>
<tr>
<td><strong>Data Cache Refills</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 Modified: L2 Owned: L2 Exclusive: L2 Shared</td>
<td>205,968,263</td>
<td>60,716,301</td>
</tr>
<tr>
<td>From System: Modified: Owned: Exclusive: Shared</td>
<td>61,970,925</td>
<td>1,950,282</td>
</tr>
</tbody>
</table>

**PAPI Native Events:**

- **PAPI_L1_DCA**
  - Classic mat_mul: 2,002,808,034
  - Reordered mat_mul: 3,008,528,895
- **PAPI_L1_DCM**
  - Classic mat_mul: 268,010,587
  - Reordered mat_mul: 62,680,818

**Data Cache Request Rate**

- Classic mat_mul: 0.2224 req/inst
- Reordered mat_mul: 0.3339 req/inst

**Data Cache Miss Rate**

- Classic mat_mul: 0.0298 miss/inst
- Reordered mat_mul: 0.0070 miss/inst

**Data Cache Miss Ratio**

- Classic mat_mul: 0.1338 miss/req
- Reordered mat_mul: 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
Counter Analysis Toolkit (CAT)
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
Cache Line Discovery

Buffer Size >> Cache Size

16 B
Cache Line Discovery

**Buffer Size >> Cache Size**

- 16 B
- 20 B
Cache Line Discovery

Buffer Size >> Cache Size

16 B

20 B

24 B
Timing results indicate line size

![Graph showing latency results vs size](image-url)
**Pointer Chasing**

```c
SETUP() {
    p = (uintptr_t **) &array[0];
    for (i = random() ) {
        next = &array[i];
        *p = next;
        p = (uintptr_t **) next;
    }
}

MEASURE() {
    start_measurement();
    for (...) {
        p = (uintptr_t **) *p;
    }
    stop_measurement();
}
```

**Min Buffer Size < Cache Size**

**Max Buffer Size > Cache Size**
Timing results indicate Cache sizes
Stress testing the Data Caches

SETUP( ) {
    p = (uintptr_t **) &array[0];
    for (i = random( ) ) {
        next = &array[i];
        *p = next;
        p = (uintptr_t **) next;
    }
}

MEASURE( ) {
    start_measurement();
    for (...) {
        p = (uintptr_t **) *p;
    }
    stop_measurement();
}

Block Size
Access Pattern
Stride
Min Buffer Size < Cache Size
Max Buffer Size > Cache Size
Measured Curves (L2)

L2 cache, Skylake (PAPI 5.7.1, CPU: Xeon(R) Gold 6140 CPU @ 2.30GHz, 6/85/4)
Measured Curves (L3)

L3 cache, Skylake (PAPI 5.7.1, CPU: Xeon(R) Gold 6140 CPU @ 2.30GHz, 6/85/4)

Normalized response (e.g. hits/access)

Measurement index
ML for automatic classification: NMSE

\[
\frac{1}{N} \sum_i \frac{(P_i - M_i)^2}{\overline{P} \cdot \overline{M}}, \quad \overline{P} = \frac{1}{N} \sum_i P_i, \quad \overline{M} = \frac{1}{N} \sum_i M_i
\]
Conclusions

- Modern CPUs have > 1K native events.
- Name & Description does not always match reality.
- Benchmarks produce useful responses per event.
- Matching responses to signatures is an ML problem.
- More advanced ML is needed to solve the problem.
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)
- Open|Speedshop  [https://openspeedshop.org/](https://openspeedshop.org/)