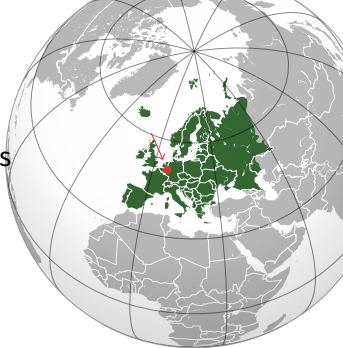


Performance Counters and Tools

OpenPOWER Tutorial, SC16, Salt Lake City

Forschungszentrum Jülich

- One of Europe's largest research facilities
- Energy, environmental sciences, health, information technology
- Home to
 - ≈ 6000 employees
 - Two Top 500 supercomputers (#13, #57)
 - POWER Acceleration and Design Centre



Goals of this session

- Get to know Performance Counters
- Use counters on POWER8
- Use counters on NVIDIA GPUs

→ Hands-on

Motivation

Performance Counters

Introduction

General Description

Counters on POWER8

Measuring Counters

perf

PAPI

Score-P

Performance Counters on GPUs

NVIDIA GPU Introduction

GPU Counters

Measuring

Conclusion

*[...] premature optimization is
the root of all evil.*

– Donald Knuth

- **Problem:** *Suspect application is running slow!*
- Many reasons for slowness:
 - Problem size
 - Inherently algorithmic
 - Limiting hardware resources (CPU, memory, ...)
 - Non-optimal implementation
 - Omitting hardware features

- **Problem:** *Suspect application is running slow!*
Solution: Analyze it, optimize it!
- Many reasons for slowness:
 - ✗ Problem size
 - ✗ Inherently algorithmic
 - ? Limiting hardware resources (CPU, memory, ...)
 - ? Non-optimal implementation
 - ? Omitting hardware features

- **Problem:** *Suspect application is running slow!*
Solution: Analyze it, optimize it!
- Many reasons for slowness:
 - ✗ Problem size
 - ✗ Inherently algorithmic
 - ? Limiting hardware resources (CPU, memory, ...)
 - ? Non-optimal implementation
 - ? **Omitting hardware features**

- Optimization **after** programming
 - Purpose dictates usability
 - Two distinct steps:
 - 1 Functionality
 - 2 Efficiency
- More Knuth: *Programmers waste enormous amounts of time [...] worrying about [...] the speed of noncritical parts of their programs [...]. [...] premature optimization is the root of all evil.*

- Optimization **after** programming
 - Purpose dictates usability
 - Two distinct steps:
 - 1 Functionality
 - 2 Efficiency
- More Knuth: *Programmers waste enormous amounts of time [...] worrying about [...] the speed of noncritical parts of their programs [...]. [...] premature optimization is the root of all evil. Yet we should not pass up our [optimization] opportunities [...].*

- Optimization **after** programming
 - Purpose dictates usability
 - Two distinct steps:
 - 1 Functionality
 - 2 Efficiency
- More Knuth: *Programmers waste enormous amounts of time [...] worrying about [...] the speed of noncritical parts of their programs [...]. [...] premature optimization is the root of all evil. **Yet we should not pass up our [optimization] opportunities [...].***
- Don't trust your gut; profile!
Optimize the right parts!

Investigation objectives

- Where does code spend time?
- What are performance limiters?
 - Computing-limited
 - Bandwidth-limited
- Architecture features exploited?
 - L1\$, L2\$
 - (S)MT
 - SIMD
 - ...
- How does performance change with versions of code?
- Impact of compiler options?

Motivation

Stage 3: Investigation

Investigation objectives

- Where does code spend time?
- What are performance limiters?
 - Computing-limited
 - Bandwidth-limited
- Architecture features exploited?
 - L1\$, L2\$ *Caching*
 - (S)MT *Threading, Hyperthreading*
 - SIMD *Vectorization*
 - ...
- How does performance change with versions of code?
- Impact of compiler options?

Investigation objectives

- Where does code spend time?
- What are performance limiters?
 - Computing-limited
 - Bandwidth-limited
- Architecture features exploited?
 - L1\$, L2\$ *Caching*
 - (S)MT *Threading, Hyperthreading*
 - SIMD *Vectorization*
 - ...
- How does performance change with versions of code?
- Impact of compiler options? *Next session*

Motivation

Stage 3: Investigation

Investigation objectives

- Where does code spend time?
- What are performance limiters?
 - Computing-limited
 - Bandwidth-limited
- **Architecture features exploited?**
 - L1\$, L2\$ *Caching*
 - (S)MT *Threading, Hyperthreading*
 - SIMD *Vectorization*
 - ...
- How does performance change with versions of code?
- Impact of compiler options? *Next session*

Two ways to evaluate a program's performance

1 The coarse way™

`clock()` or `gettimeofday()` or the likes

- Might make sense for first look
- No insight to inner workings

2 The informative way™

→ **Performance counters!**

Performance Counters

- Part of processor periphery, but dedicated registers
- History
 - First occurrence: Intel Pentium, reverse-engineered 1994 (RDPMC) [5]
 - Originally for chip developers
 - Later embraced for software developers and tuners
- Operation: Certain events counted at logic level, then aggregated to registers

- Part of processor periphery, but dedicated registers
- History
 - First occurrence: Intel Pentium, reverse-engineered 1994 (RDPMC) [5]
 - Originally for chip developers
 - Later embraced for software developers and tuners
- Operation: Certain events counted at logic level, then aggregated to registers

Pros

- Low overhead
- Very specific requests possible; detailed information
- Information about CPU core, *nest*, cache, memory

Cons

- Processor-specific
 - No standard interface
 - Some might become unavailable...
- Limited amount of counter registers
- Compressed information content

- Mind the clock rates!
 - Modern processors have dynamic clock rates (CPUs, GPUs)
 - Might skew results
 - Some counters might not run at nominal clock rate
- Limited counter registers
POWER8: 6 slots for hardware counters
- Cores, Threads (OpenMP)
 - Absolutely possible
 - Complicates things slightly
 - Pinning necessary
 - `OMP_PROC_BIND`, `OMP_PLACES`; `PAPI_thread_init()`
- Nodes (MPI): Counters independent of MPI, but aggregation tool useful (Score-P, ...)

Performance Counters Examples

What's possible

Number of ...

- Instructions →
- Cycles → CPI, IPC
- Floating point operations
- Stalled cycles
- Cache misses, cache hits
- Prefetches
- Flushs
- Branches
- CPU migrations

Performance Counters Examples

What's possible

Number of ...

- Instructions →
- Cycles → CPI, IPC
- Floating point operations
- Stalled cycles
- Cache misses, cache hits
- Prefetches
- Flushes
- Branches
- CPU migrations

- *Native*
- *Derived*
- *Software*

Performance Counters on POWER8

POWER8 Compartments

Sources of PMU events

POWER8

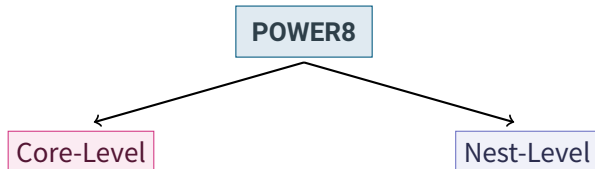
POWER8 Compartments

Sources of PMU events

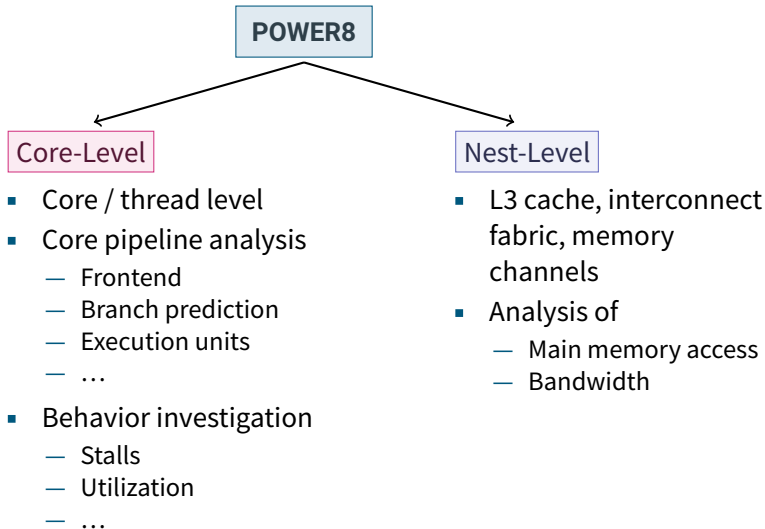


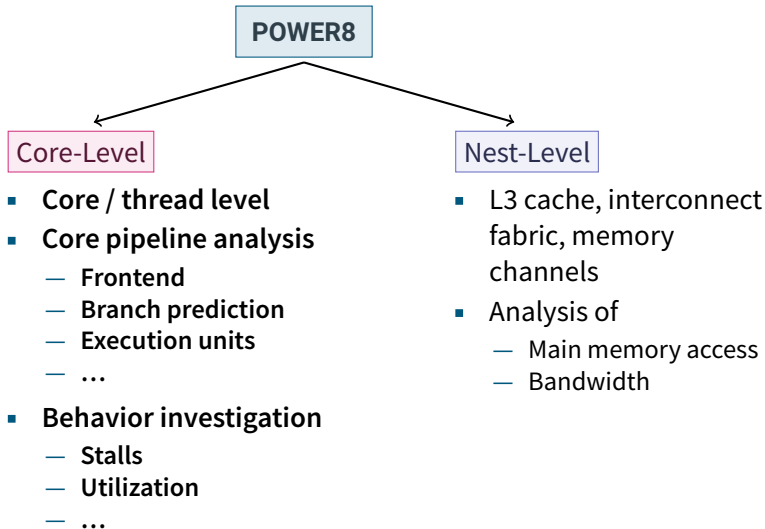
POWER8 Compartments

Sources of PMU events



- Core / thread level
- Core pipeline analysis
 - Frontend
 - Branch prediction
 - Execution units
 - ...
- Behavior investigation
 - Stalls
 - Utilization
 - ...





POWER8 Performance Counters

Instructions, Stalls

NAME Some description

POWER8 Performance Counters

Instructions, Stalls

NAME	Some description
PM_INST_CMPL	Instructions completed <i>Also: PM_RUN_INST_CMPL</i>
PM_RUN_CYC	Total cycles run <i>Processor cycles gated by the run latch</i>

POWER8 Performance Counters

Instructions, Stalls

NAME	Some description
PM_INST_CMPL	Instructions completed <i>Also: PM_RUN_INST_CMPL</i>
PM_RUN_CYC	Total cycles run <i>Processor cycles gated by the run latch</i>
PM_CMPLU_STALL	Completion stall <i>Cycles in which a thread did not complete any groups, but there were entries</i>
PM_CMPLU_STALL_THRD	Completion stall due to thread conflict <i>After stall, thread unable to complete because other thread uses completion port</i>
PM_GCT_NOSLOT_CYC	Pipeline empty <i>Global Completion Table has no slots from thread</i>

POWER8 Performance Counters

Instructions, Stalls

NAME Some description
PM_INST_CMPL Instructions completed

Also: PM_RUN_INST_CMPL

PM_RUN_CYC Total cycles run

Processor cycles gated by the run latch

PM_CMPLU_STALL Completion stall

Cycles in which a thread did not complete any groups, but there were entries

PM_CMPLU_STALL_THRD Completion stall due to thread conflict

After stall, thread unable to complete because other thread uses completion port

PM_GCT_NOSLOT_CYC Pipeline empty

Global Completion Table has no slots from thread

POWER8 Performance Counters

Instructions, Stalls

NAME Some description
PM_INST_CMPL Instructions completed

Also: PM_RUN_INST_CMPL

PM_RUN_CYC Total cycles run

Processor cycles gated by the run latch

PM_CMPLU_STALL Completion stall

Cycles in which a thread did not complete any groups, but there were entries

PM_CMPLU_STALL_THRD Completion stall due to thread conflict

After stall, thread unable to complete because other thread uses completion port

PM_GCT_NOSLOT_CYC Pipeline empty

Global Completion Table has no slots from thread

PM_CMPLU_STALL_BRU_CRU Stall due to IFU

IFU: Instruction Fetching Unit

PM_CMPLU_STALL_VSU Stall due to VSU

VSU: Vector Scalar Unit

NAME Some description
PM_INST_CMPL Instructions completed

Also: PM_RUN_INST_CMPL

PM_RUN_CYC Total cycles run

Processor cycles gated by the run latch

PM_CMPLU_STALL Completion stall

Cycles in which a thread did not complete any groups, but there were entries

PM_CMPLU_STALL_THRD Completion stall due to thread conflict

After stall, thread unable to complete because other thread uses completion port

PM_GCT_NOSLOT_CYC Pipeline empty

Global Completion Table has no slots from thread

PM_CMPLU_STALL_BRU_CRU Stall due to IFU

IFU: Instruction Fetching Unit

PM_CMPLU_STALL_VSU Stall due to VSU

VSU: Vector Scalar Unit

NAME Some description
PM_INST_CMPL Instructions completed

Also: PM_RUN_INST_CMPL

PM_RUN_CYC Total cycles run

Processor cycles gated by the run latch

PM_CMPLU_STALL Completion stall

Cycles in which a thread did not complete any groups, but there were entries

PM_CMPLU_STALL_THRD Completion stall due to thread conflict

After stall, thread unable to complete because other thread uses completion port

PM_GCT_NOSLOT_CYC Pipeline empty

Global Completion Table has no slots from thread

PM_CMPLU_STALL_BRU_CRU Stall due to IFU

IFU: Instruction Fetching Unit

PM_CMPLU_STALL_BRU Stall due to BRU

BRU: Branch Unit

PM_CMPLU_STALL_VSU Stall due to VSU

VSU: Vector Scalar Unit

PM_CMPLU_STALL_SCALAR_LONG Stall due to long scalar instruction

Floating point divide or square root instructions

NAME Some description
PM_INST_CMPL Instructions completed

Also: *PM_RUN_INST_CMPL*

PM_RUN_CYC Total cycles run

Processor cycles gated by the run latch

PM_CMPLU_STALL Completion stall

Cycles in which a thread did not complete any groups, but there were entries

PM_CMPLU_STALL_THRD Completion stall due to thread conflict

After stall, thread unable to complete because other thread uses completion port

PM_GCT_NOSLOT_CYC Pipeline empty

Global Completion Table has no slots from thread

PM_CMPLU_STALL_BRU_CRU Stall due to IFU

IFU: Instruction Fetching Unit

PM_CMPLU_STALL_BRU Stall due to BRU

BRU: Branch Unit

PM_CMPLU_STALL_VSU Stall due to VSU

VSU: Vector Scalar Unit

PM_CMPLU_STALL_SCALAR_LONG Stall due to long scalar instruction

Floating point divide or square root instructions

POWER8 Performance Counters

Instructions, Stalls



NAME Some description
PM_INST_CMPL Instructions completed

Also: *PM_RUN_INST_CMPL*

PM_RUN_CYC Total cycles run

Processor cycles gated by the run latch

PM_CMPLU_STALL Completion stall

Cycles in which a thread did not complete any groups, but there were entries

PM_CMPLU_STALL_THRD Completion stall due to thread conflict

After stall, thread unable to complete because other thread uses completion port

PM_GCT_NOSLOT_CYC Pipeline empty

Global Completion Table has no slots from thread

PM_CMPLU_STALL_BRU_CRU Stall due to IFU

IFU: Instruction Fetching Unit

PM_CMPLU_STALL_BRU Stall due to BRU

BRU: Branch Unit

PM_CMPLU_STALL_VSU Stall due to VSU

VSU: Vector Scalar Unit

PM_CMPLU_STALL_SCALAR_LONG Stall due to long scalar instruction

Floating point divide or square root instructions

NAME	Some description
PM_INST_CMPL	Instructions completed

Number of counters for POWER8:

1144

*See appendix for more on counters
(CPI stack; resources)*

PM_CMPLU_STALL_VSU	Stall due to VSU
	VSU: Vector Scalar Unit

PM_CMPLU_STALL_SCALAR_LONG	Stall due to long scalar instruction
	Floating point divide or square root instructions

Measuring Counters

perf Linux' tool

PAPI C/C++ API

Score-P Measurement environment

Likwid Set of command line utilities for detailed analysis

- Part of Linux kernel since 2009 ([v. 2.6.31](#))
- Command line utility as *prefix* for targeted application

- Part of Linux kernel since 2009 ([v. 2.6.31](#))
- Command line utility as *prefix* for targeted application
- Main program `perf`; all functionality by sub-commands
 - `perf list` List available counters
 - `perf stat` Run program; report performance data
 - `perf record` Run program; **sample** and save performance data
 - `perf report` Analyzed saved performance data ([appendix](#))
 - `perf top` Like top, live-view of counters

Usage: `perf stat ./app`

```

herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test — ssh * ssh juppext
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ perf stat ./matmul.bin

Performance counter stats for './matmul.bin':

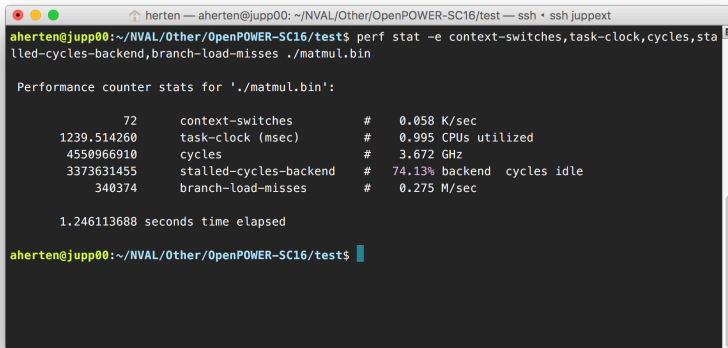
      1240.751188      task-clock (msec)      #    0.995 CPUs utilized
           73         context-switches      #    0.059 K/sec
           4          cpu-migrations        #    0.003 K/sec
          161         page-faults          #    0.130 K/sec
    4556578255        cycles                 #    3.672 GHz              (66.49%)
      1593849        stalled-cycles-frontend #    0.03% frontend cycles idle (49.87%)
    3352091957        stalled-cycles-backend #   73.57% backend cycles idle (50.44%)
    4615646896        instructions          #    1.01 insns per cycle
                                           #    0.73 stalled cycles per insn (67.07%)
      280175883        branches              # 225.811 M/sec             (50.13%)
       311201         branch-misses         #    0.11% of all branches  (49.81%)

1.247284426 seconds time elapsed

aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$

```

Example: `perf stat -e context-switches,[...] ./app`



```
herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test — ssh • ssh juppext
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ perf stat -e context-switches,task-clock,cycles,sta
lled-cycles-backend,branch-load-misses ./matmul.bin

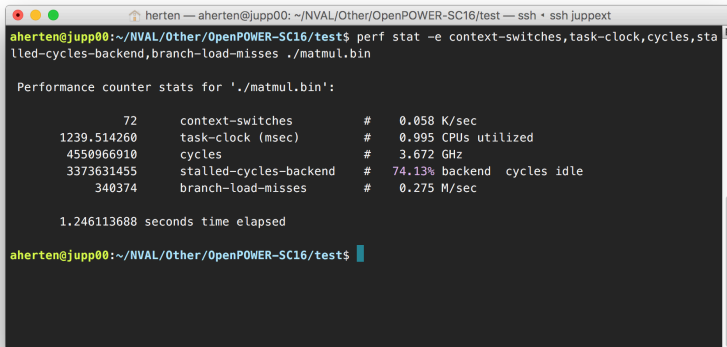
Performance counter stats for './matmul.bin':

      72      context-switches      #    0.058 K/sec
1239.514260    task-clock (msec)      #    0.995 CPUs utilized
4550966910    cycles                  #    3.672 GHz
3373631455    stalled-cycles-backend      #   74.13% backend  cycles idle
   340374    branch-load-misses      #    0.275 M/sec

1.246113688 seconds time elapsed

aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$
```

Example: `perf stat -e context-switches,[...] ./app`



```
herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test — ssh • ssh juppext
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ perf stat -e context-switches,task-clock,cycles,stalled-cycles-backend,branch-load-misses ./matmul.bin

Performance counter stats for './matmul.bin':

      72      context-switches      #    0.058 K/sec
1239.514260 task-clock (msec)      #    0.995 CPUs utilized
4550966910 cycles                  #    3.672 GHz
3373631455 stalled-cycles-backend #   74.13% backend  cycles idle
   340374 branch-load-misses      #    0.275 M/sec

1.246113688 seconds time elapsed

aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$
```

perf

Tipps, Tricks

- Option `--repeat` for statistical measurements

```
1.239 seconds time elapsed ( +- 0.16% )
```

- Measure unaliased, raw counters with `-e rABC`

```
perf stat -e r4d018 ./matmul.bin
551681          r4d018
```

- Restrict counters to certain user-level modes by `-e counter:m`, with `m = u` (user), `= k` (kernel), `= h` (hypervisor)
- perf modes: Per-thread (default), per-process (`-p PID`), per-CPU (`-a`)
- Other options
 - d More details
 - d -d More more details
 - B Add thousands' delimiters
 - x Print machine-readable output

→ <https://perf.wiki.kernel.org/>

- Performance Application Programming Interface
- API for C/C++, Fortran

- Performance Application Programming Interface
- API for C/C++, Fortran
- Goal: Create common (and easy) interface to performance counters
- Two API layers
 - High-Level API: Most-commonly needed information capsuled by convenient functions
 - Low-Level API: Access all the counters! *See [appendix](#)!*

- Performance Application Programming Interface
- API for C/C++, Fortran
- Goal: Create common (and easy) interface to performance counters
- Two API layers
 - High-Level API: Most-commonly needed information capsuled by convenient functions
 - Low-Level API: Access all the counters! *See [appendix!](#)*
- Command line utilities

`papi_avail` List aliased, common counters

Use `papi_avail -e EVENT` to get description
and options for EVENT

`papi_native_avail` List all possible counters, with details

- Extendable by Component PAPI (GPU!)

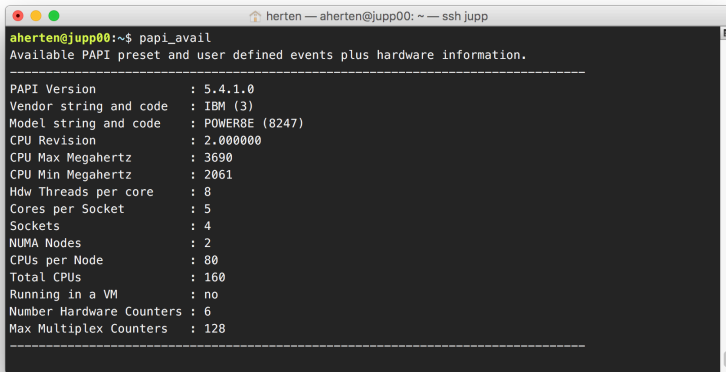
- Performance Application Programming Interface
- API for C/C++, Fortran
- Goal: Create common (and easy) interface to performance counters
- Two API layers
 - High-Level API: Most-commonly needed information capsuled by convenient functions
 - Low-Level API: Access all the counters! *See [appendix!](#)*
- Command line utilities

`papi_avail` List aliased, common counters

Use `papi_avail -e EVENT` to get description and options for EVENT

`papi_native_avail` List all possible counters, with details

- Extendable by Component PAPI (GPU!)
- Comparison to perf: Instrument specific parts of code, with different counters

Usage: `papi_avail`

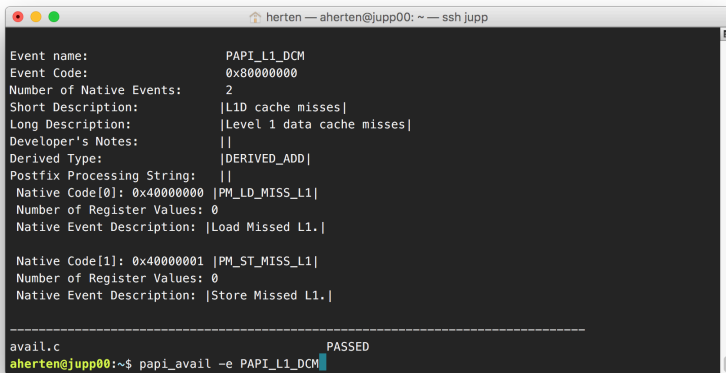
```
aherten@jupp00:~$ papi_avail
Available PAPI preset and user defined events plus hardware information.
-----
PAPI Version           : 5.4.1.0
Vendor string and code : IBM (3)
Model string and code  : POWER8E (8247)
CPU Revision           : 2.000000
CPU Max Megahertz      : 3690
CPU Min Megahertz      : 2061
Hdw Threads per core   : 8
Cores per Socket       : 5
Sockets                : 4
NUMA Nodes             : 2
CPUs per Node          : 80
Total CPUs             : 160
Running in a VM        : no
Number Hardware Counters : 6
Max Multiplex Counters : 128
-----
```

Usage: papi_avail

```

=====
PAPI Preset Events
=====
  Name      Code    Avail Deriv Description (Note)
PAPI_L1_DCM 0x80000000 Yes  Yes  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 No   No   Level 2 instruction cache misses
PAPI_L3_DCM 0x80000004 No   No   Level 3 data cache misses
PAPI_L3_ICM 0x80000005 Yes  No   Level 3 instruction cache misses
PAPI_L1_TCM 0x80000006 No   No   Level 1 cache misses
PAPI_L2_TCM 0x80000007 No   No   Level 2 cache misses
PAPI_L3_TCM 0x80000008 No   No   Level 3 cache misses
PAPI_CA_SNP 0x80000009 No   No   Requests for a snoop
PAPI_CA_SHR 0x8000000a No   No   Requests for exclusive access to shared cache line
PAPI_CA_CLN 0x8000000b No   No   Requests for exclusive access to clean cache line
PAPI_CA_INV 0x8000000c No   No   Requests for cache line invalidation
PAPI_CA_ITV 0x8000000d No   No   Requests for cache line intervention
PAPI_L3_LDM 0x8000000e No   No   Level 3 load misses
PAPI_L3_STM 0x8000000f No   No   Level 3 store misses
  
```

Example (with details): `papi_avail -e PAPI_L1_DCM`

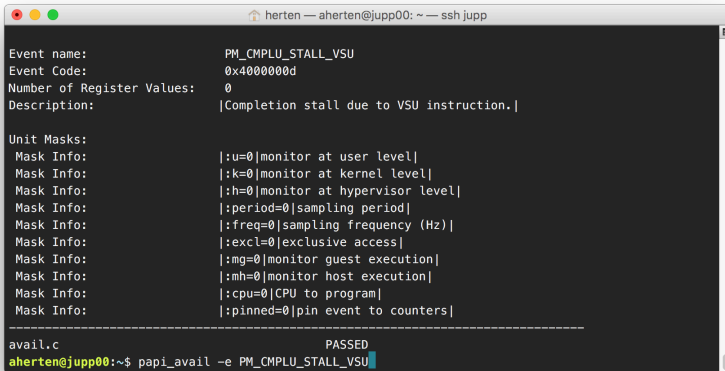


```
herten — aherten@jupp00: ~ — ssh jupp
Event name:          PAPI_L1_DCM
Event Code:          0x80000000
Number of Native Events: 2
Short Description:    |L1D cache misses|
Long Description:    |Level 1 data cache misses|
Developer's Notes:   ||
Derived Type:        |DERIVED_ADD|
Postfix Processing String: ||
Native Code[0]: 0x40000000 |PM_LD_MISS_L1|
Number of Register Values: 0
Native Event Description: |Load Missed L1.|

Native Code[1]: 0x40000001 |PM_ST_MISS_L1|
Number of Register Values: 0
Native Event Description: |Store Missed L1.|

-----
avail.c                                     PASSED
aherten@jupp00:~$ papi_avail -e PAPI_L1_DCM
```

Example (native): `papi_avail -e PM_CMPLU_STALL_VSU`



```
herten — aherten@jupp00: ~ — ssh jupp
Event name:          PM_CMPLU_STALL_VSU
Event Code:          0x4000000d
Number of Register Values: 0
Description:          |Completion stall due to VSU instruction.|

Unit Masks:
Mask Info:           |:u=0|monitor at user level|
Mask Info:           |:k=0|monitor at kernel level|
Mask Info:           |:h=0|monitor at hypervisor level|
Mask Info:           |:period=0|sampling period|
Mask Info:           |:freq=0|sampling frequency (Hz)|
Mask Info:           |:excl=0|exclusive access|
Mask Info:           |:mg=0|monitor guest execution|
Mask Info:           |:mh=0|monitor host execution|
Mask Info:           |:cpu=0|CPU to program|
Mask Info:           |:pinned=0|pin event to counters|

-----
avail.c                PASSED
aherten@jupp00:~$ papi_avail -e PM_CMPLU_STALL_VSU
```

PAPI: High Level API

Usage: Source Code

```
// Setup
```

```
float realTime, procTime, mflops, ipc;  
long long flpins, ins;
```

```
// Initial call
```

```
PAPI_flops(&realTime, &procTime, &flpins, &mflops);  
PAPI_ipc(&realTime, &procTime, &ins, &ipc);
```

```
// Compute
```

```
mult(m, n, p, A, B, C);
```

```
// Finalize call
```

```
PAPI_flops(&realTime, &procTime, &flpins, &mflops);  
PAPI_ipc(&realTime, &procTime, &ins, &ipc);
```

See [appendix for Low Level API snippet!](#)

- Important functions in **High Level API**

`PAPI_num_counters()` # available counter registers

`PAPI_flops()` Get real time, processor time, # floating point operations, and MFLOPs/s

`PAPI_ipc()` # instructions and IPC (+rttime/ptime)

`PAPI_epc()` # counts of arbitrary event (+rttime/ptime)

- Important functions in **High Level API**

- `PAPI_num_counters()` # available counter registers

- `PAPI_flops()` Get real time, processor time, # floating point operations, and MFLOPs/s

- `PAPI_ipc()` # instructions and IPC (+rttime/ptime)

- `PAPI_epc()` # counts of arbitrary event (+rttime/ptime)

- Important functions in **Low Level API**

- `PAPI_add_event()` Add aliased event to event set

- `PAPI_add_named_event()` Add any event to event set

- `PAPI_thread_init()` Initialize thread support in PAPI

- Documentation [online](#) and in man pages (`man papi_add_event`)

- Important functions in **High Level API**

- `PAPI_num_counters()` # available counter registers

- `PAPI_flops()` Get real time, processor time, # floating point operations, and MFLOPs/s

- `PAPI_ipc()` # instructions and IPC (+rttime/ptime)

- `PAPI_epc()` # counts of arbitrary event (+rttime/ptime)

- Important functions in **Low Level API**

- `PAPI_add_event()` Add aliased event to event set

- `PAPI_add_named_event()` Add any event to event set

- `PAPI_thread_init()` Initialize thread support in PAPI

- Documentation [online](#) and in man pages (`man papi_add_event`)
- All PAPI calls return status code; check for it! *By macro* ([appendix](#))
- Convert names of performance counters with `libpfm4` ([appendix](#))

- Important functions in **High Level API**

- `PAPI_num_counters()` # available counter registers

- `PAPI_flops()` Get real time, processor time, # floating point operations, and MFLOPs/s

- `PAPI_ipc()` # instructions and IPC (+rttime/ptime)

- `PAPI_epc()` # counts of arbitrary event (+rttime/ptime)

- Important functions in **Low Level API**

- `PAPI_add_event()` Add aliased event to event set

- `PAPI_add_named_event()` Add any event to event set

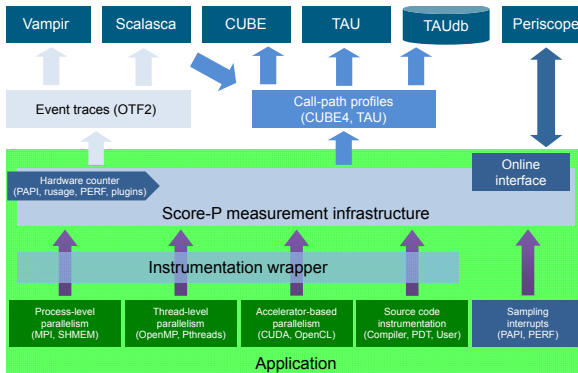
- `PAPI_thread_init()` Initialize thread support in PAPI

- Documentation [online](#) and in man pages (`man papi_add_event`)
- All PAPI calls return status code; check for it! *By macro* ([appendix](#))
- Convert names of performance counters with `libpfm4` ([appendix](#))

→ <http://icl.cs.utk.edu/papi/>

- Measurement infrastructure for profiling, event tracing, online analysis
- Output format input for many analysis tools (Cube, Vampir, Periscope, Scalasca, Tau)

- Measurement infrastructure for profiling, event tracing, online analysis
- Output format input for many analysis tools (Cube, Vampir, Periscope, Scalasca, Tau)



- Prefix compiler executable by scorep

```
~\>$ scorep clang++ -o app code.cpp
```

→ Adds instrumentation calls to binary

- Profiling output is stored to file after run of binary
- Steer with environment variables at **run time**

```
~\>$ export SCOREP_METRIC_PAPI=PAPI_FP_OPS,PM_CMPLU_STALL_VSU
```

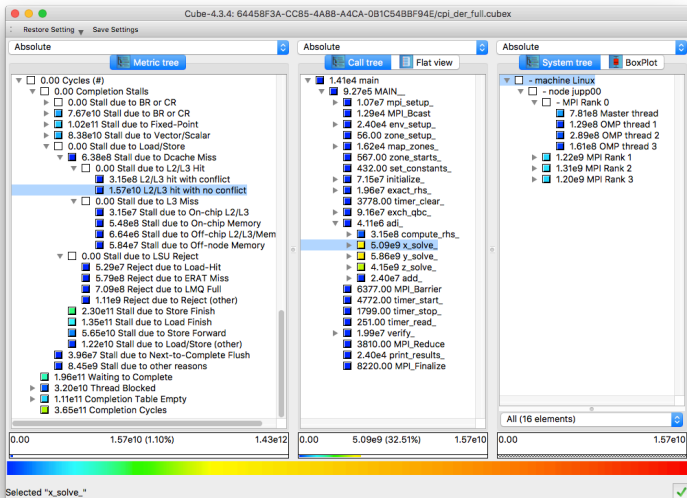
```
~\>$ ./app
```

⇒ Use different PAPI counters per run!

- Quick visualization with [Cube](#); scoring with `scorep-score`
appendix

Score-P

Analysis with Cube

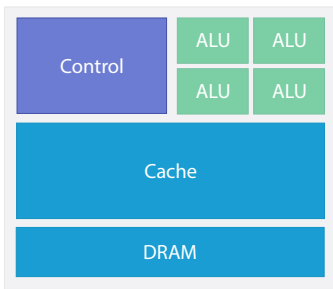


Performance Counters on GPUs

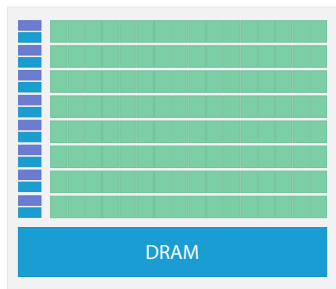
NVIDIA GPU Introduction

GPU 101

- Dedicated devices, connected with CPU by bus (PCIe, NVLink)
- Computations in Streaming Multiprocessor (*SM*)
- Single Instruction, Multiple Threads (*SIMT*)
- CUDA cores for single precision, for double precision, for ...
- Hierarchical, specialized memory



CPU



GPU

- Counters built right in
- Grouped into *domains* by topic
- Access backend: CUPTI (CUDA Profiling Tools Interface)
- Tools for instrumenting already-compiled code, for augmenting source code, for tracing
- NVIDIA differentiates between
 - Event** Countable activity or occurrence on GPU device
 - Metric** Characteristic calculated from one or more events

Example Events and Metrics

NAME NVIDIA Description (quoted)

Example Events and Metrics

NAME	NVIDIA Description (quoted)
<code>threads_launched</code>	Number of threads launched on a multiprocessor.
<code>shared_store</code>	Number of executed store instructions where state space is specified as shared, increments per warp on a multiprocessor.

Example Events and Metrics

NAME	NVIDIA Description (quoted)
<code>threads_launched</code>	Number of threads launched on a multiprocessor.
<code>shared_store</code>	Number of executed store instructions where state space is specified as shared, increments per warp on a multiprocessor.
<code>ipc</code>	Instructions executed per cycle
<code>achieved_occupancy</code>	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
<code>l1_cache_local_hit_rate</code>	Hit rate in L1 cache for local loads and stores
<code>gld_efficiency</code>	Ratio of requested global memory load throughput to required global memory load throughput.
<code>flop_count_dp</code>	Number of double-precision floating-point operations executed non-predicated threads (add, multiply, multiply-accumulate and special)
<code>stall_pipe_busy</code>	Percentage of stalls occurring because a compute operation cannot be performed because the compute pipeline is busy

CUPTI C/C++-API through `cupti.h`

- Activity API: Race CPU/GPU activity
- Callback API: Hooks for own functions
- Event / Metric API: Read counters and metrics

→ Targets developers of profiling tools

Measuring GPU counters

Tools

CUPTI C/C++-API through `cupti.h`

- Activity API: Race CPU/GPU activity
- Callback API: Hooks for own functions
- Event / Metric API: Read counters and metrics

→ Targets developers of profiling tools

PAPI All PAPI instrumentation through PAPI-C, e.g.
`cuda::device:0:threads_launched`

Score-P Mature CUDA support

- Prefix `nvcc` compilation with `scorep`
- Set environment variable `SCOREP_CUDA_ENABLE=yes`
- Run, analyze

Measuring GPU counters

Tools

CUPTI C/C++-API through `cupti.h`

- Activity API: Race CPU/GPU activity
- Callback API: Hooks for own functions
- Event / Metric API: Read counters and metrics

→ Targets developers of profiling tools

PAPI All PAPI instrumentation through PAPI-C, e.g.
`cuda::device:0:threads_launched`

Score-P Mature CUDA support

- Prefix `nvcc` compilation with `scorep`
- Set environment variable `SCOREP_CUDA_ENABLE=yes`
- Run, analyze

nvprof, Visual Profiler NVIDIA's solutions

Measuring GPU counters

Tools

CUPTI C/C++-API through `cupti.h`

- Activity API: Race CPU/GPU activity
- Callback API: Hooks for own functions
- Event / Metric API: Read counters and metrics

→ Targets developers of profiling tools

PAPI All PAPI instrumentation through PAPI-C, e.g.
`cuda::device:0:threads_launched`

Score-P Mature CUDA support

- Prefix `nvcc` compilation with `scorep`
- Set environment variable `SCOREP_CUDA_ENABLE=yes`
- Run, analyze

nvprof, Visual Profiler **NVIDIA's solutions**

Usage: `nvprof --events AB --metrics C,D ./app`

```

herten — aherten@JUHYDRA: ~/cudaSamples/NVIDIA_CUDA-7.5_Samples/bin/x86_64/linux/release — ..linux/release — ssh juhydra
ixMulCUDA<int=32>(float*, float*, float*, int, int)" (0 of 3)==18158== Replaying kernel "void matrixMulCUDA<int=32>(float*, float*, flo
at*, int, int)" (0 of 3)==18158== Replaying kernel "void matrixMulCUDA<int=32>(float*, float*, float*, int, int)" (done)
==18158== Replaying kernel "void matrixMulCUDA<int=32>(float*, float*, float*, int, int)" (0 of 3)==18158== Replaying kernel "void matr
ixMulCUDA<int=32>(float*, float*, float*, int, int)" (0 of 3)==18158== Replaying kernel "void matrixMulCUDA<int=32>(float*, float*, flo
at*, int, int)" (0 of 3)==18158== Replaying kernel "void matrixMulCUDA<int=32>(float*, float*, float*, int, int)" (done)
Performance= 1.69 GFlop/s, Time= 77.513 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
==18158== Profiling application: ./matrixMul
==18158== Profiling result:
==18158== Event result:
Invocations
Device "Tesla K40m (0)"
Kernel: void matrixMulCUDA<int=32>(float*, float*, float*, int, int)
301          threads_launched           204800          204800          204800

==18158== Metric result:
Invocations
Device "Tesla K40m (0)"
Kernel: void matrixMulCUDA<int=32>(float*, float*, float*, int, int)
301          flop_count_sp  Floating Point Operations(Single Precisi
301          ipc            Executed IPC
301          achieved_occupancy          Achieved Occupancy
301          131072000      131072000      131072000
301          1.472345      1.486837      1.480249
301          0.960357      0.989658      0.975385

# aherten @ JUHYDRA in ~/cudaSamples/NVIDIA_CUDA-7.5_Samples/bin/x86_64/linux/release [21:47:45]
$ nvprof --events threads_launched --metrics flop_count_sp,ipc,achieved_occupancy ./matrixMul

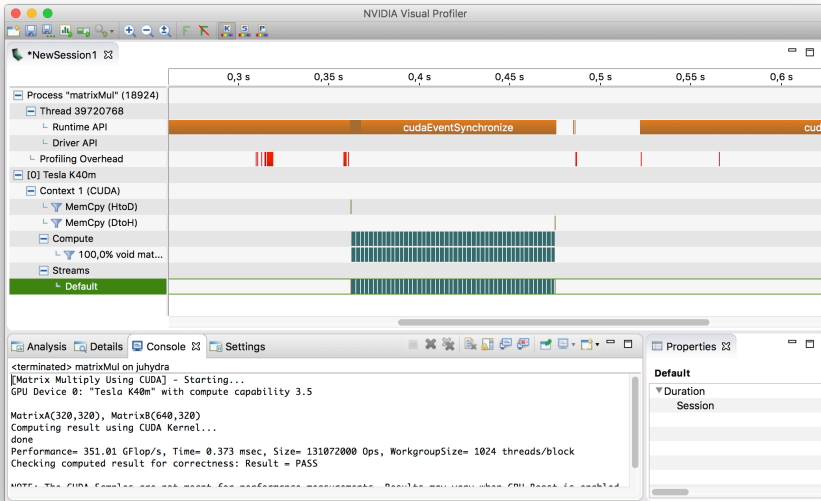
```

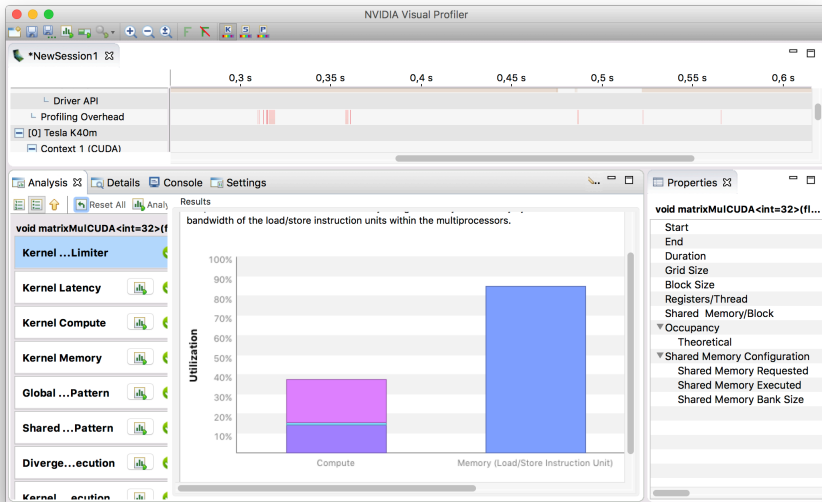
Useful parameters to nvprof

- `--query-metrics` List all metrics
- `--query-events` List all events
- `--kernels name` Limit scope to kernel
- `--print-gpu-trace` Print timeline of invocations
- `--aggregate-mode off` No aggregation over all multiprocessors (average)
 - `--csv` Output a CSV
- `--export-profile` Store profiling information, e.g. for Visual Profiler

Visual Profiler

An annotated time line view





Conclusions

What we've learned

- Large set of performance counters on POWER8 and NVIDIA processors
- Right next to (*inside*) core(s)
- Provide detailed insight for performance analysis on many levels
- Different measurement strategies and tools
 - perf
 - PAPI
 - Score-P
 - nvprof

Conclusions

What we've learned

- Large set of performance counters on POWER8 and NVIDIA processors
- Right next to (*inside*) core(s)
- Provide detailed insight for performance analysis on many levels
- Different measurement strategies and tools
 - perf
 - PAPI
 - Score-P
 - nvprof

**Thank you
for your attention!**
a.herten@fz-juelich.de

Appendix

Knuth on Optimization

References

POWER8 Performance Counters

perf Supplementary

PAPI Supplementary

Score-P Supplementary

Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered.

We should forget about small efficiencies, say about 97 % of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3 %.

– Donald Knuth in “Structured Programming with Go to Statements” [6]

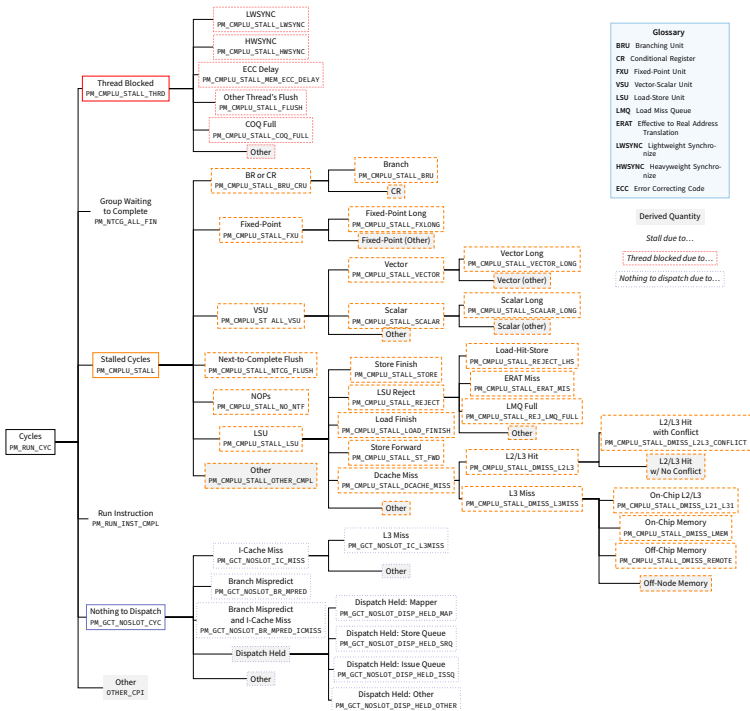
- [5] Terje Mathisen. *Pentium Secrets*. URL: http://www.gamedev.net/page/resources/_/technical/general-programming/pentium-secrets-r213 (pages 17, 18).
- [6] Donald E. Knuth. “Structured Programming with Go to Statements”. In: *ACM Comput. Surv.* 6.4 (Dec. 1974), pp. 261–301. ISSN: 0360-0300. DOI: 10.1145/356635.356640. URL: <http://doi.acm.org/10.1145/356635.356640> (page 80).

- [1] Forschungszentrum Jülich. *Hightech made in 1960: A view into the control room of DIDO*. URL: http://historie.fz-juelich.de/60jahre/DE/Geschichte/1956-1960/Dekade/_node.html (page 2).
- [2] Forschungszentrum Jülich. *Forschungszentrum Bird's Eye*. (Page 2).
- [3] Forschungszentrum Jülich. *JUQUEEN Supercomputer*. URL: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html (page 2).
- [4] Rob984 via Wikimedia Commons. *Europe orthographic Caucasus Urals boundary (with borders)*. URL: [https://commons.wikimedia.org/wiki/File:Europe_orthographic_Caucasus_Urals_boundary_\(with_borders\).svg](https://commons.wikimedia.org/wiki/File:Europe_orthographic_Caucasus_Urals_boundary_(with_borders).svg) (page 2).

POWER8 Performance Counters

- Further information on counters at IBM website
 - CPI events and metrics for POWER8
 - Events and groups supported on POWER8 architecture
 - Derived metrics defined for POWER8 architecture
 - Table 11-18 and Table D-1 of POWER8 Processor User's Manual for the Single-Chip Module
 - OProfile: [ppc64 POWER8 events](#)
- List available counters on system
 - With PAPI: `papi_native_avail`
 - With `showevtinfo` from [libpfm](#)'s `/examples/` directory


```
./showevtinfo | \
  grep -e "Name" -e "Desc" | sed "s/^.\+: //g" | paste -d'\t' - -
```
- See [next slide](#) for CPI stack visualization
- Most important counters for **OpenMP**:
`DMISS_PM_CMPLU_STALL_DMISS_L3MISS,`
`PM_CMPLU_STALL_DMISS_REMOTE, PM_CMPLU_STALL_DMISS_DISTANT`



perf Supplementary

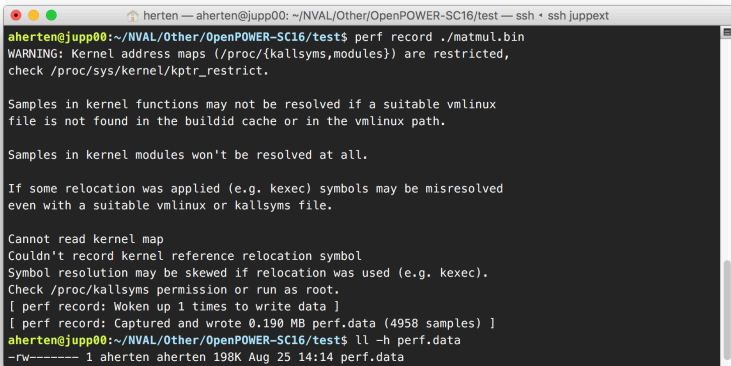
More resources for the Linux tool

- web.eece.maine.edu/~vweaver/projects/perf_events/

Deeper Analysis with perf

perf report

Usage: `perf record ./app`



```
herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test — ssh • ssh juppext
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ perf record ./matmul.bin
WARNING: Kernel address maps (/proc/{kallsyms,modules}) are restricted,
check /proc/sys/kernel/kptr_restrict.

Samples in kernel functions may not be resolved if a suitable vmlinux
file is not found in the buildid cache or in the vmlinux path.

Samples in kernel modules won't be resolved at all.

If some relocation was applied (e.g. kexec) symbols may be misresolved
even with a suitable vmlinux or kallsyms file.

Cannot read kernel map
Couldn't record kernel reference relocation symbol
Symbol resolution may be skewed if relocation was used (e.g. kexec).
Check /proc/kallsyms permission or run as root.
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.190 MB perf.data (4958 samples) ]
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ ll -h perf.data
-rw----- 1 aherten aherten 198K Aug 25 14:14 perf.data
```

Deeper Analysis with perf

perf report

Usage: perf report

```
2. aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test (ssh)
Samples: 7K of event 'cycles', Event count (approx.): 7079214805
Overhead Command Shared Object Symbol
 98.37% matmul.bin matmul.bin [.] _Z4multiiiPKf50_Pf
  1.42% matmul.bin libc-2.21.so [.] __random
  0.07% matmul.bin matmul.bin [.] main
  0.05% matmul.bin libc-2.21.so [.] __random_r
  0.04% matmul.bin ld-2.21.so [.] _dl_map_object_from_fd
  0.02% matmul.bin libc-2.21.so [.] rand
  0.01% matmul.bin [unknown] [k] 0xc000000000009958
  0.01% matmul.bin [unknown] [k] 0xc00000000000508d0
  0.01% matmul.bin [unknown] [k] 0xc0000000000026bc64
  0.00% matmul.bin [unknown] [k] 0xc000000000002012b4
  0.00% matmul.bin [unknown] [k] 0xc000000000001b01a0
  0.00% matmul.bin [unknown] [k] 0xc0000000000009954
  0.00% matmul.bin [unknown] [k] 0xc0000000000013c16c
  0.00% matmul.bin [unknown] [k] 0xc0000000000013c38c
  0.00% matmul.bin [unknown] [k] 0xc00000000000bb8a0
  0.00% matmul.bin [unknown] [k] 0xc00000000000a9280
  0.00% matmul.bin [unknown] [k] 0xc00000000000a9140

For a higher level overview, try: perf report --sort comm,dso
```

Deeper Analysis with perf

perf report

Usage: perf report

Use `c++filt`
to demangle!

```

2. aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test (ssh)
Samples: 7K of event 'cycles', Event count (approx.): 7079214805
Overhead Command Shared Object Symbol
 98.37% matmul.bin matmul.bin [.] _Z4multiiiiPKfS0_Pf
  1.42% matmul.bin libc-2.21.so [.] __random
  0.07% matmul.bin matmul.bin [.] main
  0.05% matmul.bin libc-2.21.so [.] __random_r
  0.04% matmul.bin ld-2.21.so [.] _dl_map_object_from_fd
  0.02% matmul.bin libc-2.21.so [.] rand
  0.01% matmul.bin [unknown] [k] 0xc000000000009958
  0.01% matmul.bin [unknown] [k] 0xc00000000000508d0
  0.01% matmul.bin [unknown] [k] 0xc0000000000026bc64
  0.00% matmul.bin [unknown] [k] 0xc000000000002012b4
  0.00% matmul.bin [unknown] [k] 0xc000000000001b01a0
  0.00% matmul.bin [unknown] [k] 0xc0000000000009954
  0.00% matmul.bin [unknown] [k] 0xc0000000000013c16c
  0.00% matmul.bin [unknown] [k] 0xc0000000000013c38c
  0.00% matmul.bin [unknown] [k] 0xc00000000000bb8a0
  0.00% matmul.bin [unknown] [k] 0xc00000000000a9280
  0.00% matmul.bin [unknown] [k] 0xc00000000000a9140

For a higher level overview, try: perf report --sort comm,dso
  
```

Deeper Analysis with perf

perf report

Usage: perf report

```

2. aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test (ssh)
Z4multiPKfS0_Pf /home/aherten/NVAL/Other/OpenPOWER-SC16/test/matmul.bin
2.51      ld      r7,-40(r1)
0.04      rldicr  r6,r6,2,61
0.07      lxsspx  vs1,r7,r6
0.07      xsmuls  vs0,vs0,vs1
0.07      add     r3,r3,r5
33.03     extsw   r6,r3
41.20     ld      r7,-48(r1)
41.20     rldicr  r6,r6,2,61
         add     r6,r7,r6
0.07      lfs     f1,0(r6)
         xsadds  vs0,vs1,vs0
7.34      stxssp  vs0,0,r6
         // C = n x p
         void mult(const int m, const int n, const int p, const float * __restrict__ A, con
         // C = A * B
         for (int row = 0; row < n; row++) {
           for (int col = 0; col < p; col++) {
             for (int elem = 0; elem < m; elem++) {
1.80      lwz     r3,-60(r1)
         addi    r3,r3,1
Press 'h' for help on key bindings
  
```

PAPI: Low Level API

Usage: Source Code

```
int EventSet = PAPI_NULL;
long long values[2];

// PAPI: Setup
PAPI_library_init(PAPI_VER_CURRENT);
PAPI_create_eventset(&EventSet);
// PAPI: Test availability of counters
PAPI_query_named_event("PM_CMPLU_STALL_VSU");
PAPI_query_named_event("PM_CMPLU_STALL_SCALAR");
// PAPI: Add counters
PAPI_add_named_event(EventSet, "PM_CMPLU_STALL_VSU");
PAPI_add_named_event(EventSet, "PM_CMPLU_STALL_SCALAR");
// PAPI: Start collection
PAPI_start(EventSet);
// Compute
mult(m, n, p, A, B, C);
// PAPI: End collection
PAPI_CALL( PAPI_stop(EventSet, values), PAPI_OK );
```

PAPI: Low Level API

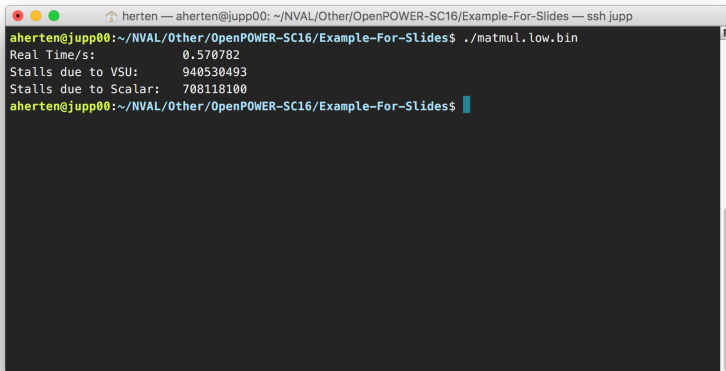
Usage: Source Code

```
int EventSet = PAPI_NULL;
long long values[2];

// PAPI: Setup
PAPI_library_init(PAPI_VER_CURRENT);
PAPI_create_eventset(&EventSet);
// PAPI: Test availability of counters
PAPI_query_named_event("PM_CMPLU_STALL_VSU");
PAPI_query_named_event("PM_CMPLU_STALL_SCALAR");
// PAPI: Add counters
PAPI_add_named_event(EventSet, "PM_CMPLU_STALL_VSU");
PAPI_add_named_event(EventSet, "PM_CMPLU_STALL_SCALAR");
// PAPI: Start collection
PAPI_start(EventSet);
// Compute
mult(m, n, p, A, B, C);
// PAPI: End collection
PAPI_CALL( PAPI_stop(EventSet, values), PAPI_OK );
```

Macro for
checking results!

Example: `./matmul.low.bin` (Matrix Multiplication)



```
herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/Example-For-Slides — ssh jupp
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/Example-For-Slides$ ./matmul.low.bin
Real Time/s:      0.570782
Stalls due to VSU: 940530493
Stalls due to Scalar: 708118100
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/Example-For-Slides$
```

PAPI Error Macro

For easier status code checking

```
#define PAPI_CALL( call, success )      \
{                                       \
    int err = call;                     \
    if ( success != err)                \
        std::cerr << "PAPI error for " << #call << " in L" << \
    ↪ __LINE__ << " of " << __FILE__ << ": " << \
    ↪ PAPI_strerror(err) << std::endl; \
}                                       \
// Second argument is code for GOOD, \
// e.g. PAPI_OK or PAPI_VER_CURRENT or ... \
// ... \
// Call like: \
PAPI_CALL( PAPI_start(EventSet), PAPI_OK );
```

- Helper library for setting up counters interfacing with perf kernel environment
- Used by PAPI to resolve counters
- Handy as translation: Named counters → raw counters
- Use command line utility `perf_examples/evt2raw` to get raw counter for perf

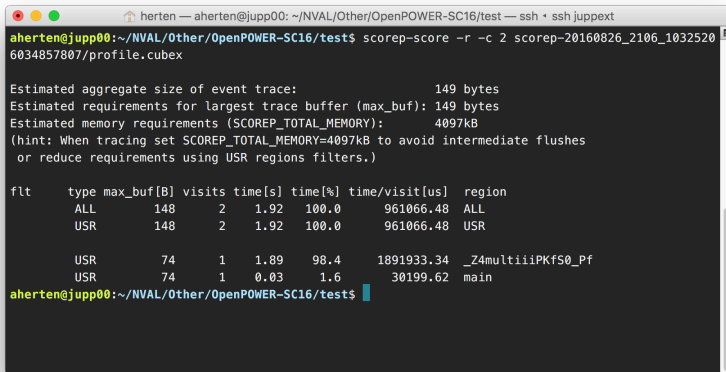
```
~\$. ./evt2raw PM_CMPLU_STALL_VSU  
r2d012
```

→ http://perfmon2.sourceforge.net/docs_v4.html

Score-P

Principle analysis with *scorep-score*

Usage: `scorep-score -r FILE`



```
herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test — ssh • ssh juppext
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ scorep-score -r -c 2 scorep-20160826_2106_1032520
6034857807/profile.cubex

Estimated aggregate size of event trace:                149 bytes
Estimated requirements for largest trace buffer (max_buf): 149 bytes
Estimated memory requirements (SCOREP_TOTAL_MEMORY):    4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
or reduce requirements using USR regions filters.)

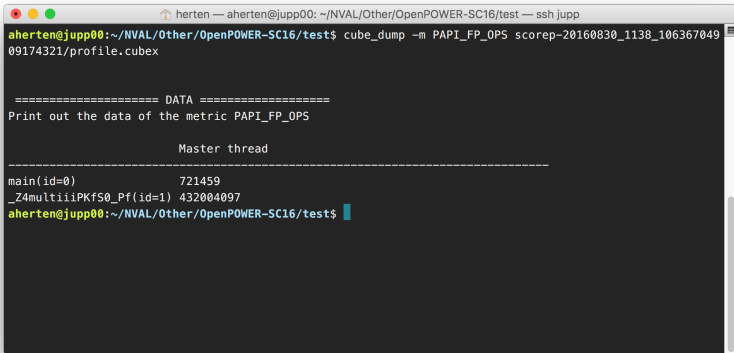
flt      type max_buf[B] visits time[s] time[%] time/visit[us] region
ALL      148      2    1.92   100.0    961066.48  ALL
USR      148      2    1.92   100.0    961066.48  USR

      USR      74      1    1.89    98.4    1891933.34  _Z4multiPKf50_Pf
      USR      74      1    0.03    1.6     30199.62   main
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$
```

Score-P

Performance counter analysis with *cube_dump*

Usage: `cube-dump -m METRIC FILE`



```
herten — aherten@jupp00: ~/NVAL/Other/OpenPOWER-SC16/test — ssh jupp
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$ cube_dump -m PAPI_FP_OPS scorep-20160830_1138_106367049
09174321/profile.cubex

===== DATA =====
Print out the data of the metric PAPI_FP_OPS

Master thread
-----
main(id=0)          721459
_Z4multiPKfS0_Pf(id=1) 432004097
aherten@jupp00:~/NVAL/Other/OpenPOWER-SC16/test$
```

The End

Thanks for reading until here