INNOVATIVE COMPUTING LABORATORY

SI2-SSI: PAPI-EX Performance Application Programming Interface for Extreme-Scale Environments

Jack Dongarra
Heike Jagode
Anthony Danalis
Daniel Barry
UNIVERSITY OF TENNESSEE

Vince Weaver
UNIVERSITY OF MAINE

PAPI

- PAPI provides a consistent interface (and methodology) for hardware performance counters found across a compute system: i. e., CPUs, GPUs, on- and off-chip memory, interconnects, I/O system, file system, energy/power, etc.
- PAPI enables software engineers to see, in near real time, the relationship between software performance and hardware events across the entire compute system.

AMD CPU: up to Fam17 Zeppelin Zen GPU: ROCm, ROCm-smi	arm Cortex, Cavium ThunderX, ARM64	CRAY Gemini and Aries interconnect, power	IBM Blue Gene Series, Q: 5-D Torus, I/O System, EMON power, energy	IBM Power 5,6,7,8,9 Power monitoring support
IBM Power9 NEST event support via Performance Co-Pilot (PCP) PAPI component	intel Westmore, Sandy/Ivy Bridge, Haswell, Broadwell, Skylake(-X), Kaby Lake, Cascadelake	intel KNC, KNL, Knights Mill including power/energy	intel RAPL (power/energy), powercap	INFINIBAND
lustre	NVIDIA Tesla, Kepler, Maxwell, Pascal, Volta	NVIDIA Power monitoring and capping support (NVML), NVLink	KVM Virtual Environment	vmware Virtual Environment

PART 1 PAPI for Arithmetic Intensity

Anthony Danalis, Heike Jagode, Daniel Barry

The goal of this work is to create a set of PAPI presets (predefined events) for effortless computation of the Arithmetic Intensity (a.k.a. Computational Intensity), measured as ratio of computation to traffic (flops / bytes).

Floating-point Operations: **ddot, dgemm**

FLOPS involve multiple events for capturing operations of different vector length.

IBM Power9:

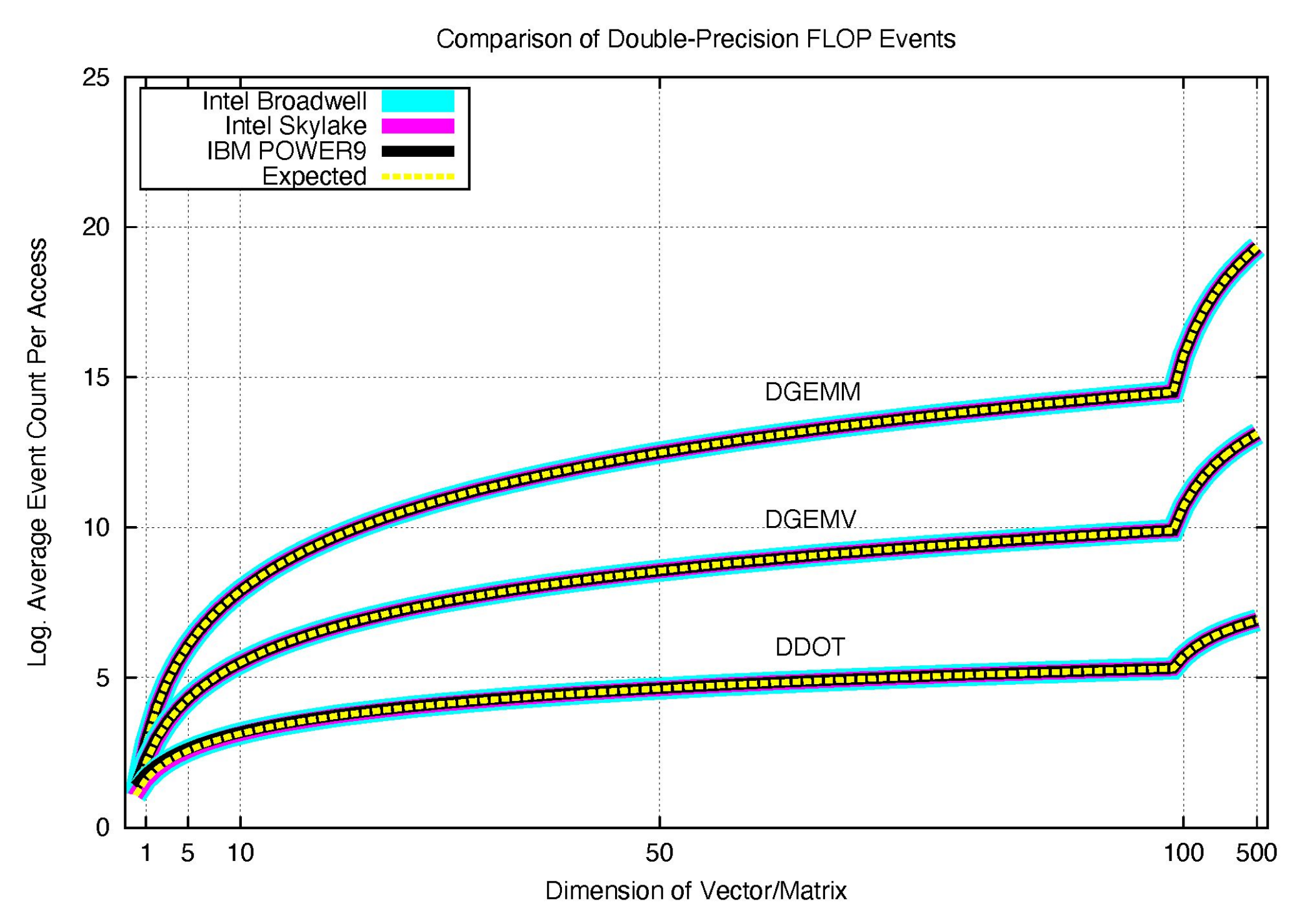
DOUBLE-precision FLOPs = PM_DP_QP_FLOP_CMPL

SINGLE-precision FLOPs = PM_SP_FLOP_CMPL

Intel Skylake:

DOUBLE-precision FLOPs = 1 *FP_ARITH_INST_RETIRED.SCALAR_DOUBLE* +
2 FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE +
4 FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE +
8 FP_ARITH_INST_RETIRED.512B_PACKED_DOUBLE

SINGLE-precision FLOPs = 1 *FP_ARITH_INST_RETIRED.PACKED_SINGLE* +
4 FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE +
8 *FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE +*
16 FP_ARITH_INST_RETIRED.512B_PACKED_SINGLE



Memory Traffic: **ddot, dgemm**

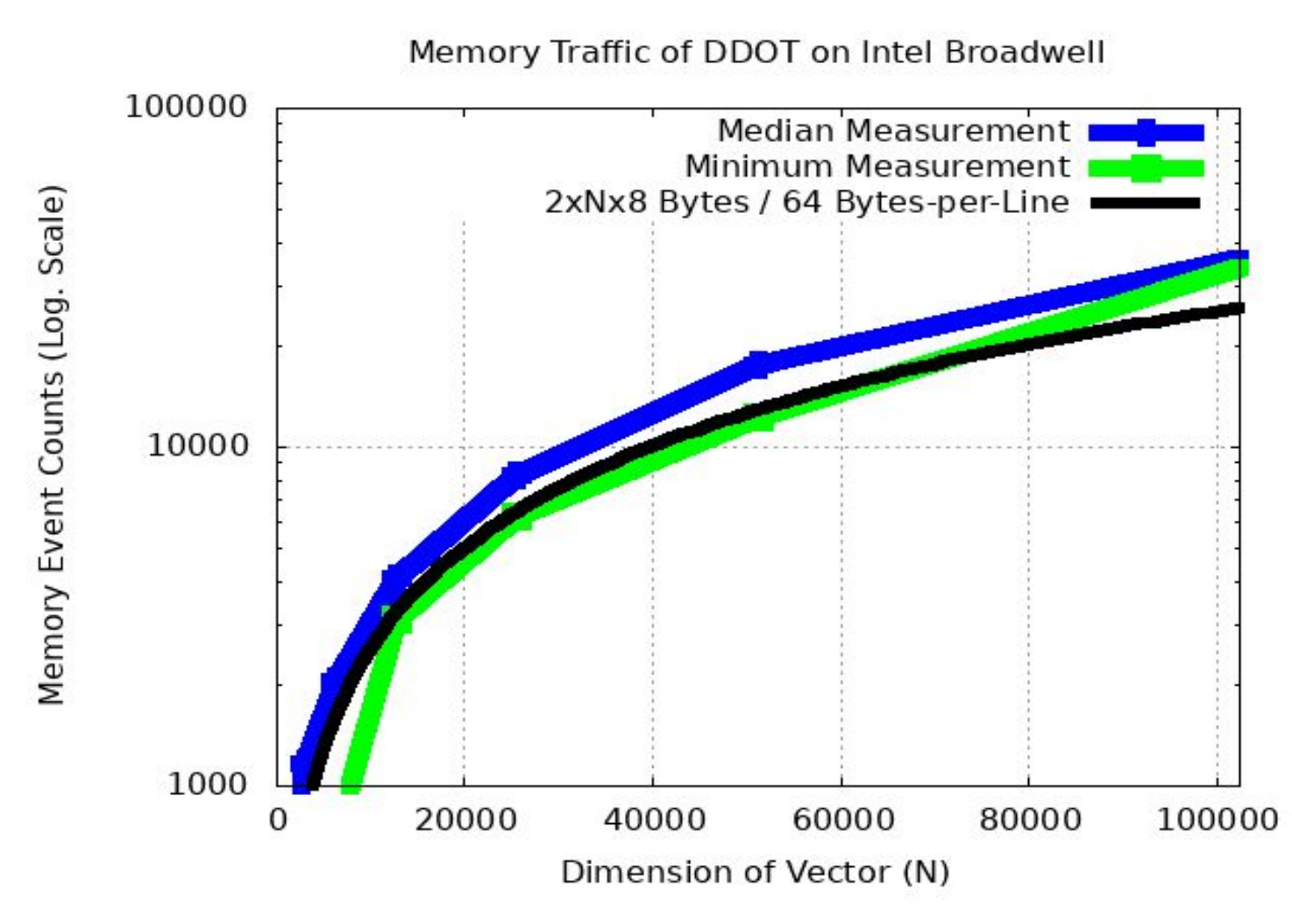
Traffic to DRAM involves multiple non-trivial uncore (Intel)/northbridge (AMD)/nest (IBM) events.

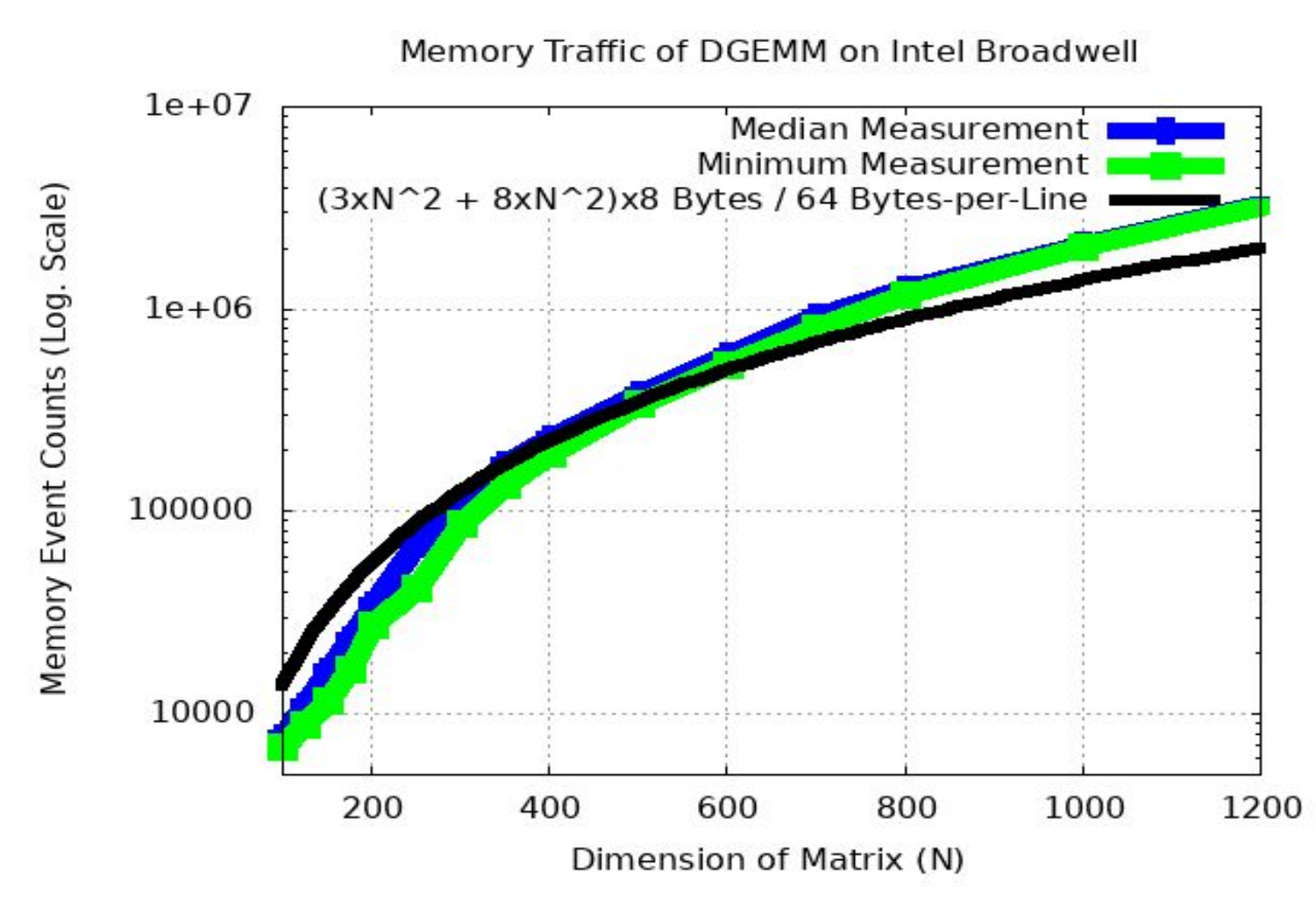
IBM Power9:

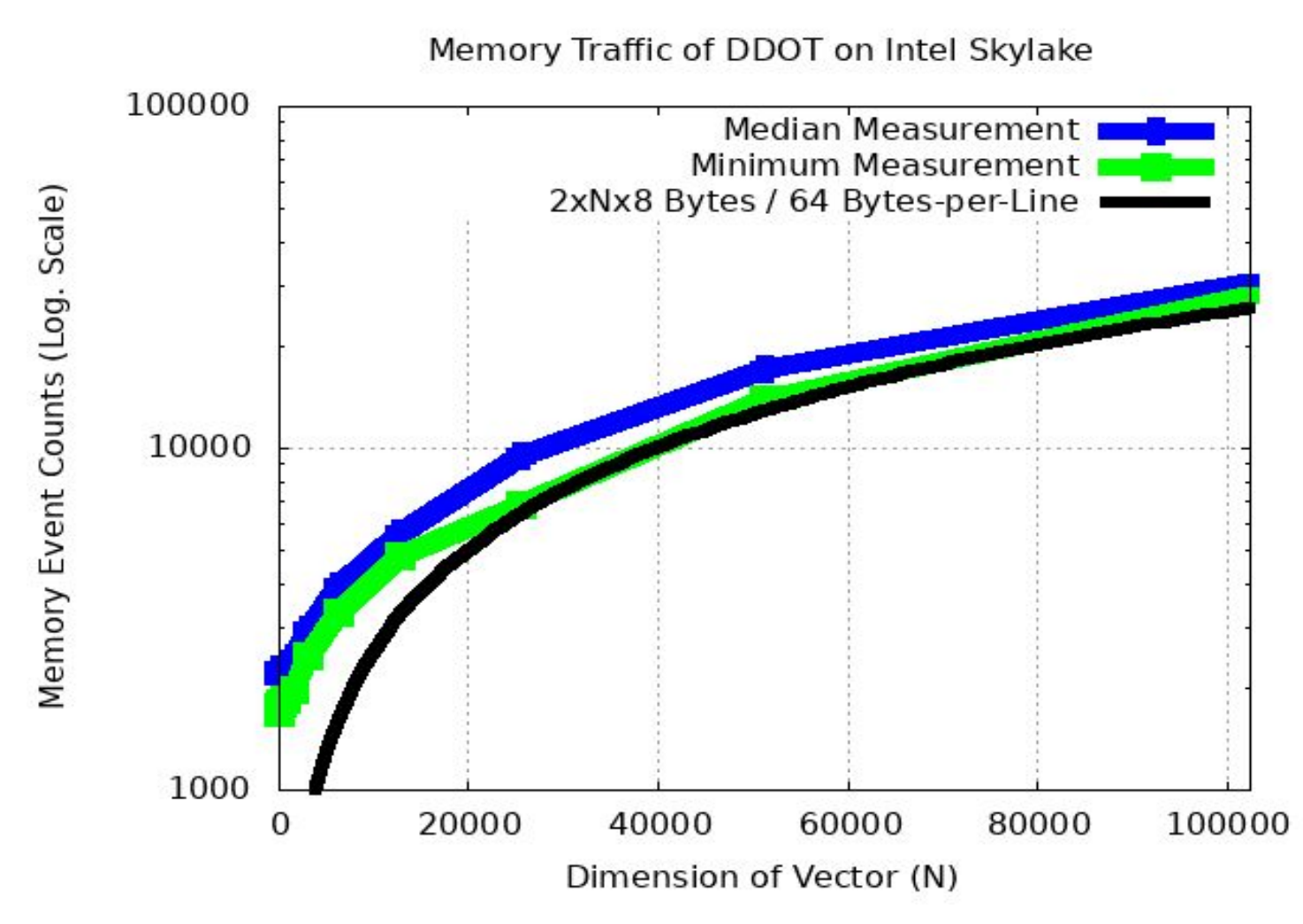
```
pcp:::perfevent.hwcounters.nest_mcs01_imc.PM_MCS01_128B_RD_DISP_PORT01.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs01_imc.PM_MCS01_128B_RD_DISP_PORT23.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs01_imc.PM_MCS01_128B_WR_DISP_PORT01.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs01_imc.PM_MCS01_128B_WR_DISP_PORT23.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs23_imc.PM_MCS23_128B_RD_DISP_PORT01.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs23_imc.PM_MCS23_128B_RD_DISP_PORT23.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs23_imc.PM_MCS23_128B_WR_DISP_PORT01.value:cpu84
pcp:::perfevent.hwcounters.nest_mcs23_imc.PM_MCS23_128B_WR_DISP_PORT23.value:cpu84
```

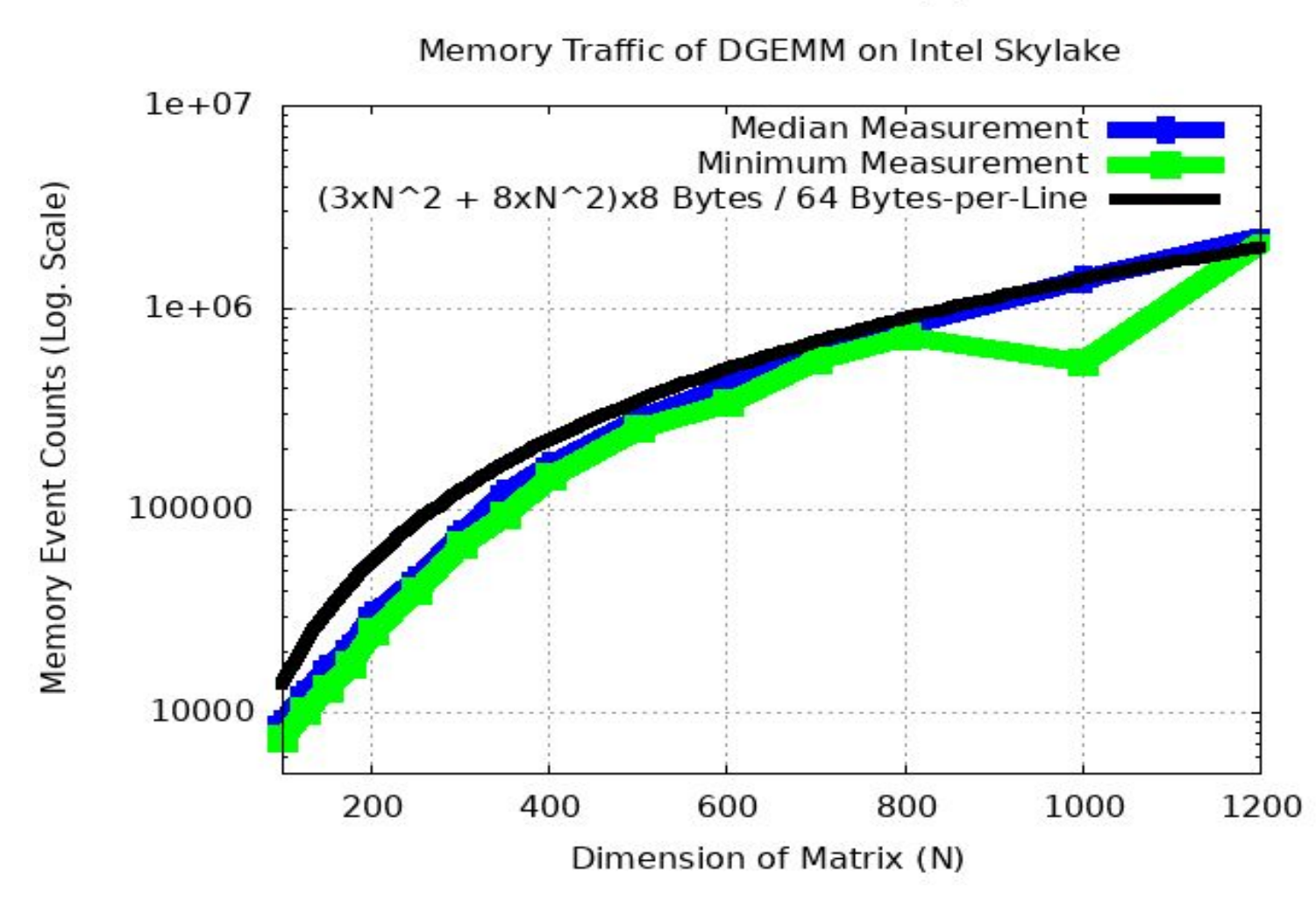
Intel Skylake 2 Sockets:

```
skx_unc_imc0::UNC_M_CAS_COUNT:WR:cpu=0   skx_unc_imc0::UNC_M_CAS_COUNT:WR:cpu=18
skx_unc_imc1::UNC_M_CAS_COUNT:WR:cpu=0   skx_unc_imc1::UNC_M_CAS_COUNT:WR:cpu=18
skx_unc_imc2::UNC_M_CAS_COUNT:WR:cpu=0   skx_unc_imc2::UNC_M_CAS_COUNT:WR:cpu=18
skx_unc_imc3::UNC_M_CAS_COUNT:WR:cpu=0   skx_unc_imc3::UNC_M_CAS_COUNT:WR:cpu=18
skx_unc_imc4::UNC_M_CAS_COUNT:WR:cpu=0   skx_unc_imc4::UNC_M_CAS_COUNT:WR:cpu=18
skx_unc_imc5::UNC_M_CAS_COUNT:WR:cpu=0   skx_unc_imc5::UNC_M_CAS_COUNT:WR:cpu=18
skx_unc_imc0::UNC_M_CAS_COUNT:RD:cpu=0   skx_unc_imc0::UNC_M_CAS_COUNT:RD:cpu=18
skx_unc_imc1::UNC_M_CAS_COUNT:RD:cpu=0   skx_unc_imc1::UNC_M_CAS_COUNT:RD:cpu=18
skx_unc_imc2::UNC_M_CAS_COUNT:RD:cpu=0   skx_unc_imc2::UNC_M_CAS_COUNT:RD:cpu=18
skx_unc_imc3::UNC_M_CAS_COUNT:RD:cpu=0   skx_unc_imc3::UNC_M_CAS_COUNT:RD:cpu=18
skx_unc_imc4::UNC_M_CAS_COUNT:RD:cpu=0   skx_unc_imc4::UNC_M_CAS_COUNT:RD:cpu=18
skx_unc_imc5::UNC_M_CAS_COUNT:RD:cpu=0   skx_unc_imc5::UNC_M_CAS_COUNT:RD:cpu=18
```









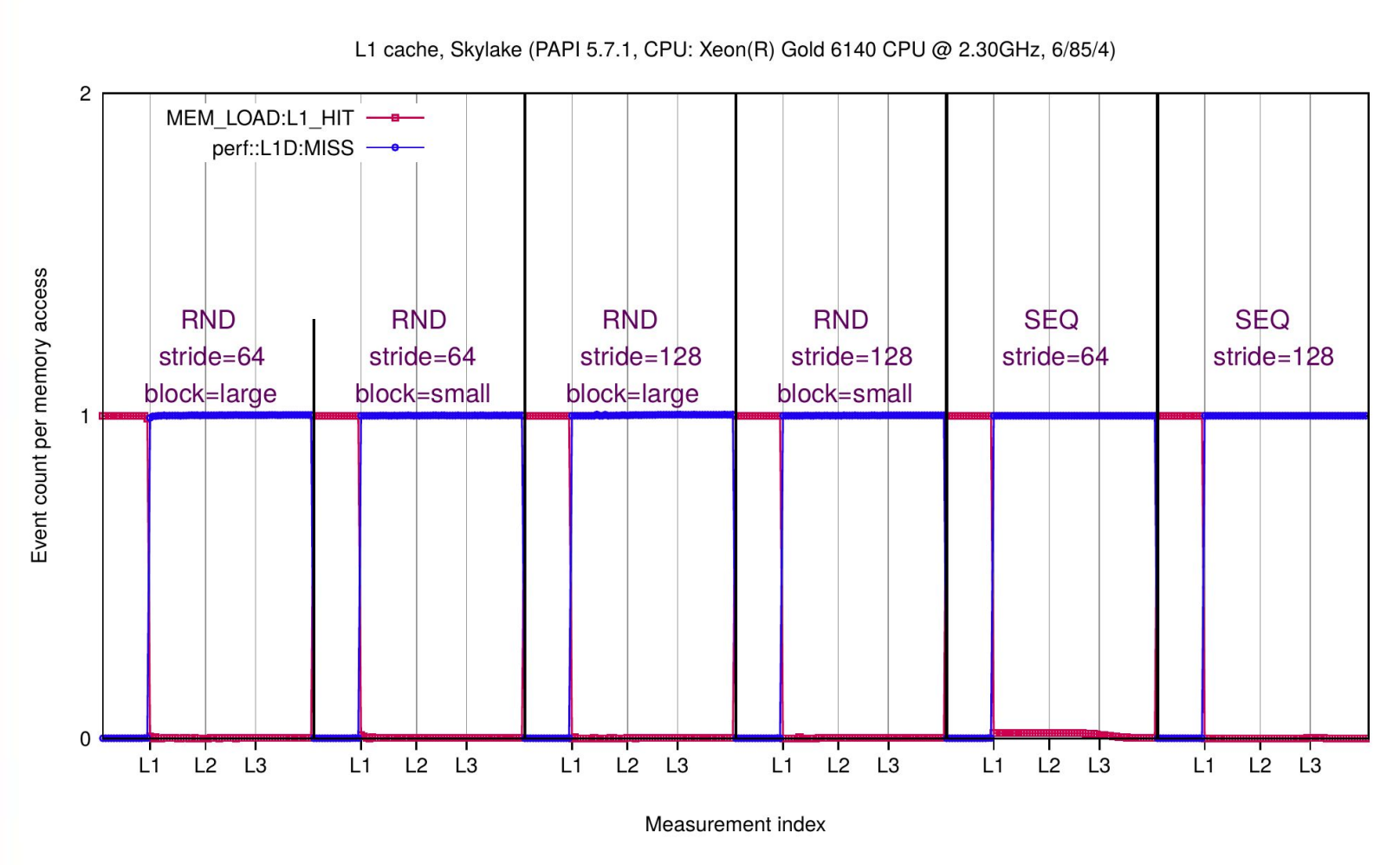
PART 2 PAPI's Counter Analysis Toolkit

Anthony Danalis, Heike Jagode, Daniel Barry

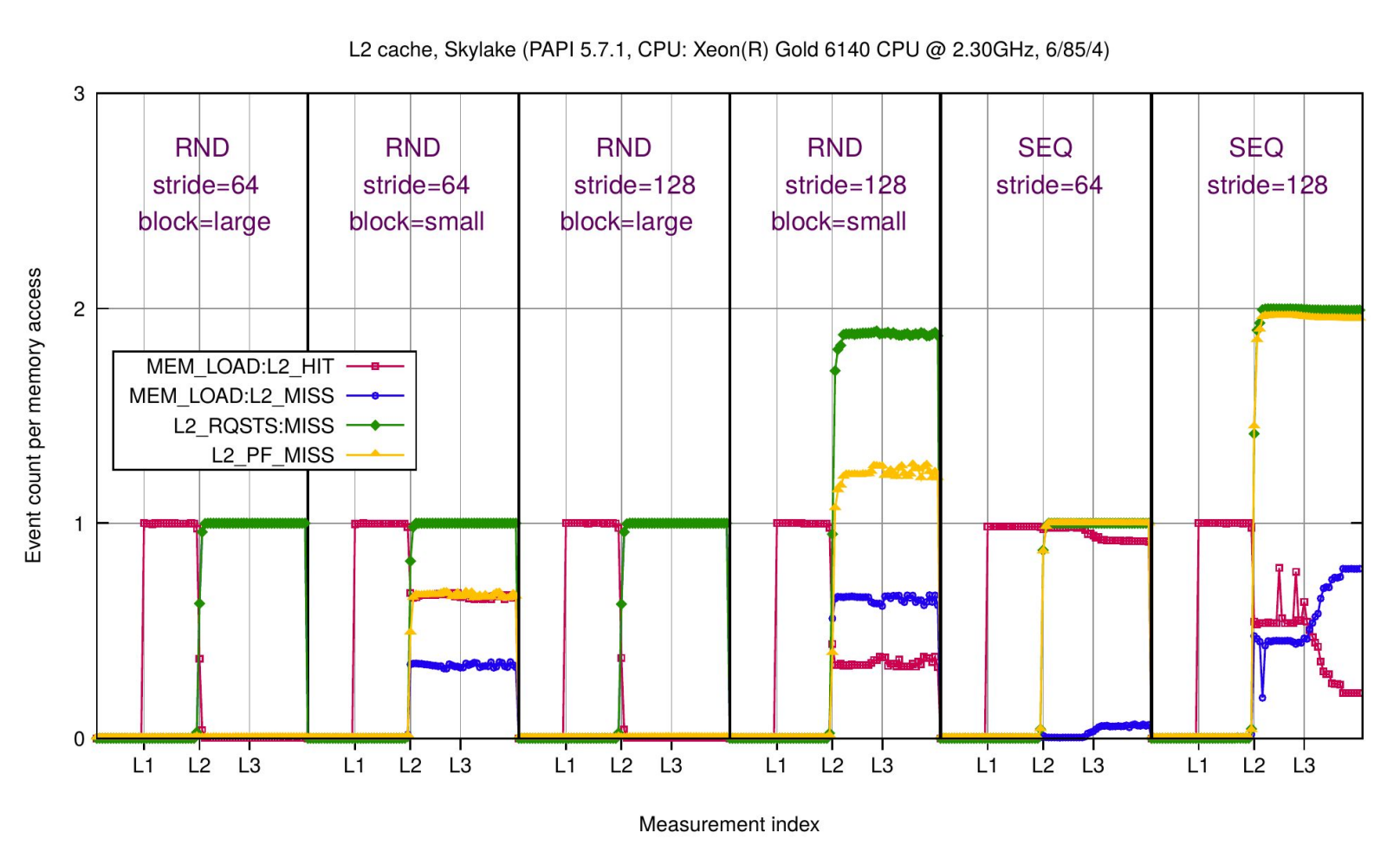
The goal of this work is to create a set of microbenchmarks for illustrating details in hardware events and how they relate to the behavior of the microarchitecture.

Target Audience

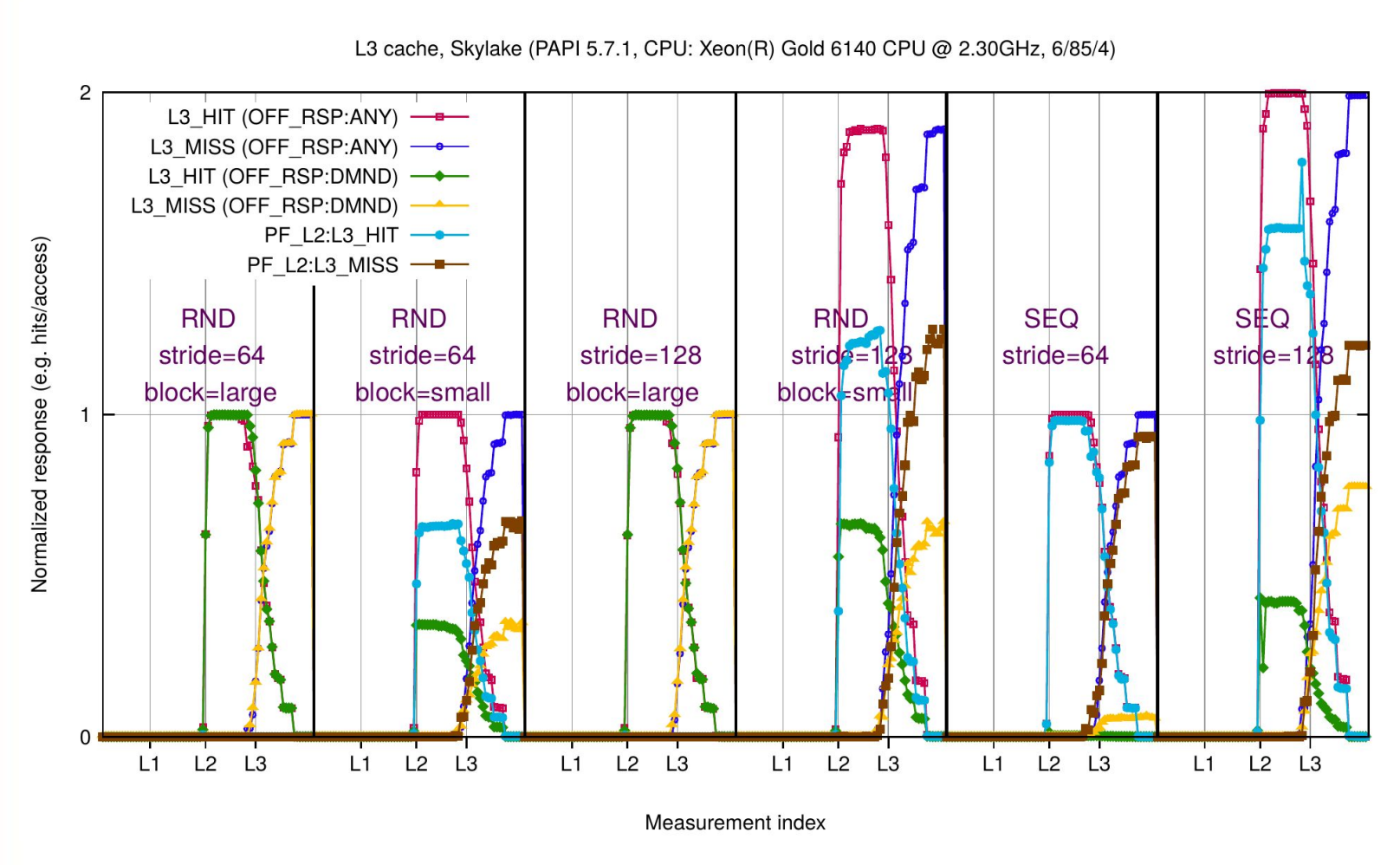
- Performance-conscious application developers
- PAPI developers working on new architectures (think preset events)
- Developers interested in validating hardware event counters



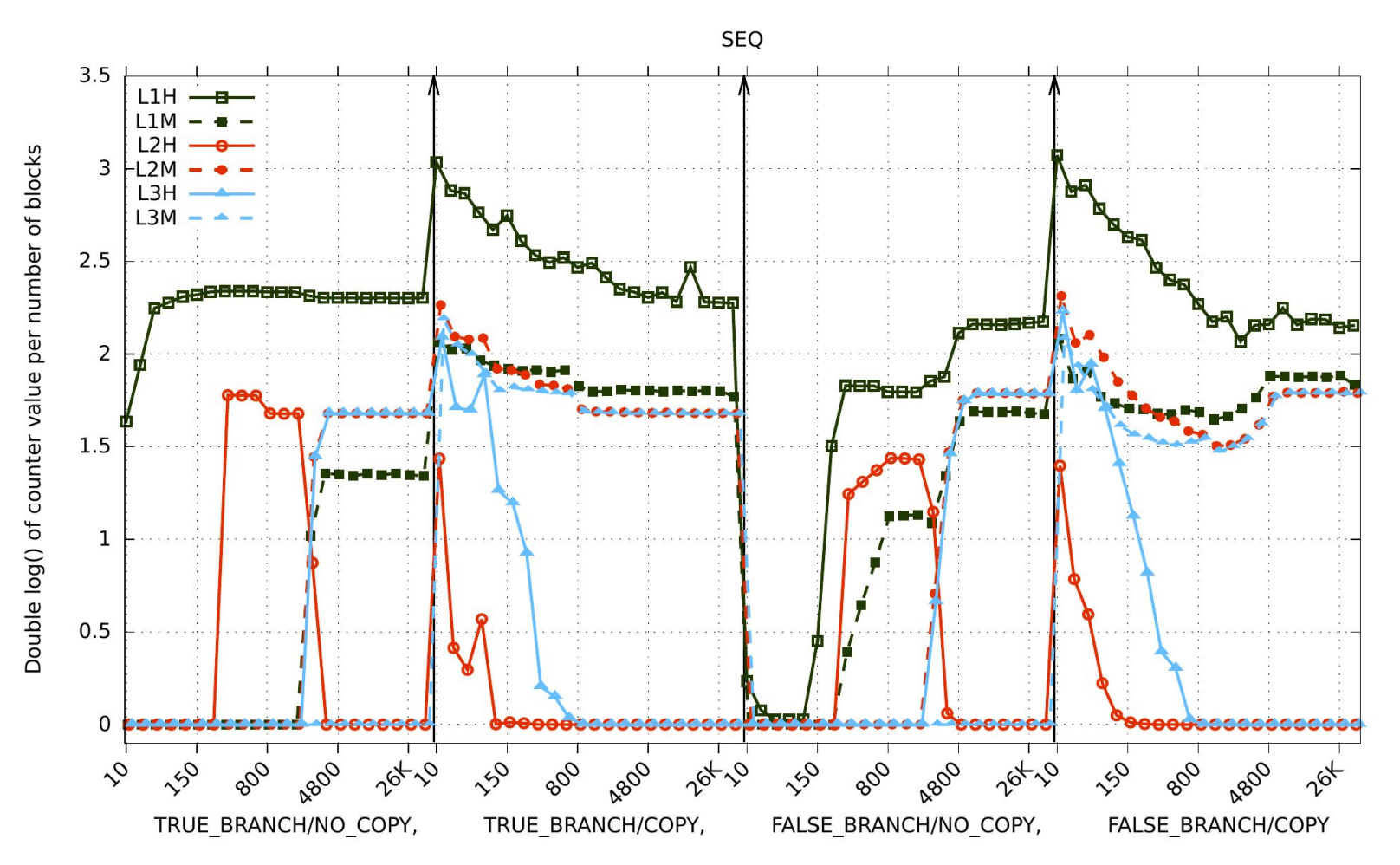
Events that count the hits and misses on the L1 D-Cache follow very sharp step functions that perfectly match the expected signatures .



Events that pertain to the L2 D-Cache have more complex signatures due to the effects of prefetching.



Events that pertain to the L3 D-Cache have very complex signatures without sharp boundaries. However, they still roughly follow the expected shapes for the different regions of interest.



Events that pertain to the Instruction cache have the most complex signatures and are challenging to match automatically. However, the curves of the different events are distinctly different from each other.

PART 3 Modernizing PAPI Infrastructure

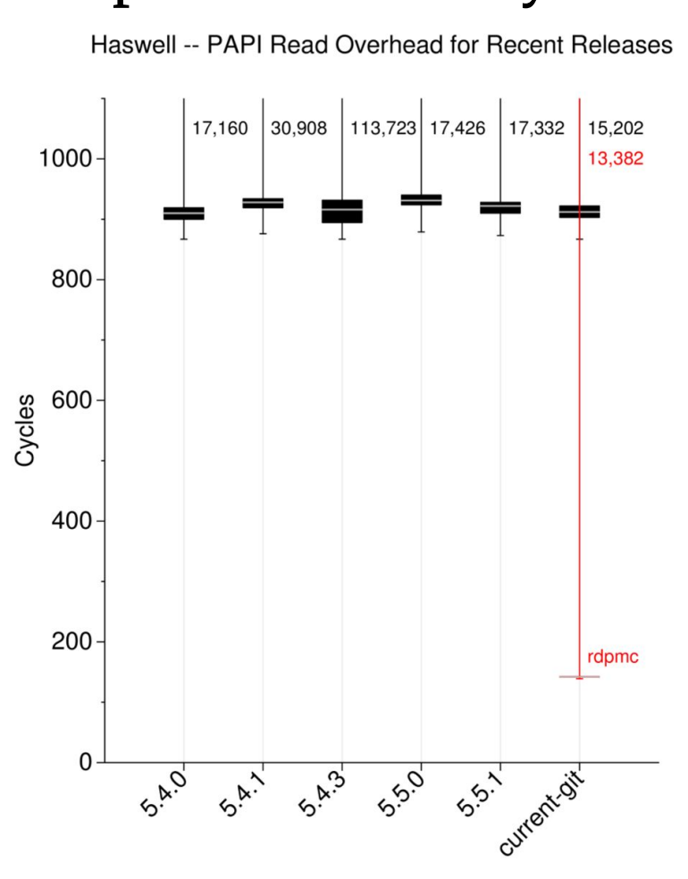
Vince Weaver and Yan Liu

Improved PAPI Test Infrastructure

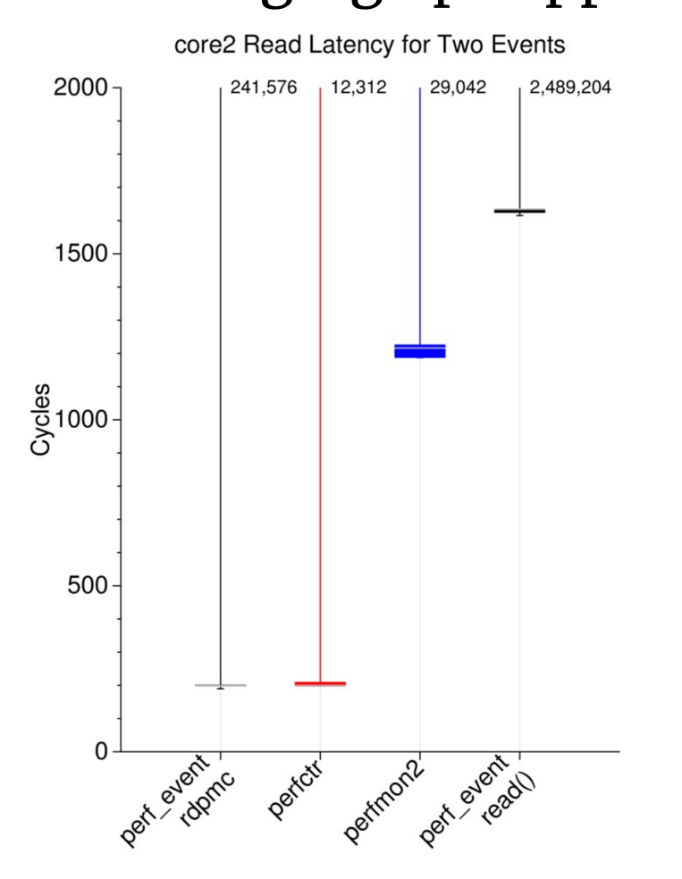
- The existing PAPI test suite is used to test the correctness of PAPI before release.
- The hardware and operating systems used by PAPI are always changing, and some of the existing tests were outdated or gave false negatives.
- Existing tests were checked to ensure accurate results on modern hardware.
- New counter validation tests were created, which should provide a sanity check when bringing up support for a new processor architecture.

Low-Overhead PAPI_read() Support

- Traditionally, PAPI_read() counter reads went through the standard Linux read() system call, which can be slow (around 1,000 cycles).
- x86 hardware supports a userspace rdpmc() instruction that bypasses the kernel and requires 200 cycles (a 5× speedup).
- Various bugs in the Linux kernel around this interface were found and fixed so that rdpmc() can be enabled by default.



Boxplot showing read latency for various versions of PAPI and the large improvement by using rdpmc.



Comparison of historical performance counter interfaces (perfmon2, perfctr) showing that perf_event rdpmc matches even the best historical interface.

Enhanced Sampling Interface

- PAPI currently has a limited counter-sampling interface that only allows gathering the instruction pointer at regular intervals.
- Modern processors support much richer sampling information, including the cause of cache misses, where in the cache hierarchy the miss happened, and the cycles taken.
- We extended the PAPI sampling interface to provide this additional sampling information.

ICL INNOVATIVE COMPUTING LABORATORY · THE UNIVERSITY OF TENNESSEE KNOXVILLE · 1865 THE UNIVERSITY OF MAINE · NSF

ACKNOWLEDGMENTS
This material is based upon work supported in part by the National Science Foundation NSF under awards No. 1450429. A portion of this research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.