Shirley Moore
Daniel Terpstra
Vincent Weaver
Heike Jagode
James Ralph
Philip Mucci
Kiran Kasichayanula
Eric Meek
and Jack Dongarra

INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

SUPPORT FROM

SPONSORED BY

CUDA COMPONENT

- HW performance counter measurement technology for NVIDIA CUDA platform
- Access to HW counters inside the GPUs
- Based on CUPTI (CUDA Performance Tool Interface) (CUDA 4.0)
- In any environment with CUPTI, PAPI CUDA component can provide detailed performance counter info regarding execution of GPU kernel
- Initialization, device management and context management is enabled by CUDA driver API
- Domain and event management is enabled by CUPTI
- Name of events is established by the following hierarchy: `Component.Device.Domain.Event`

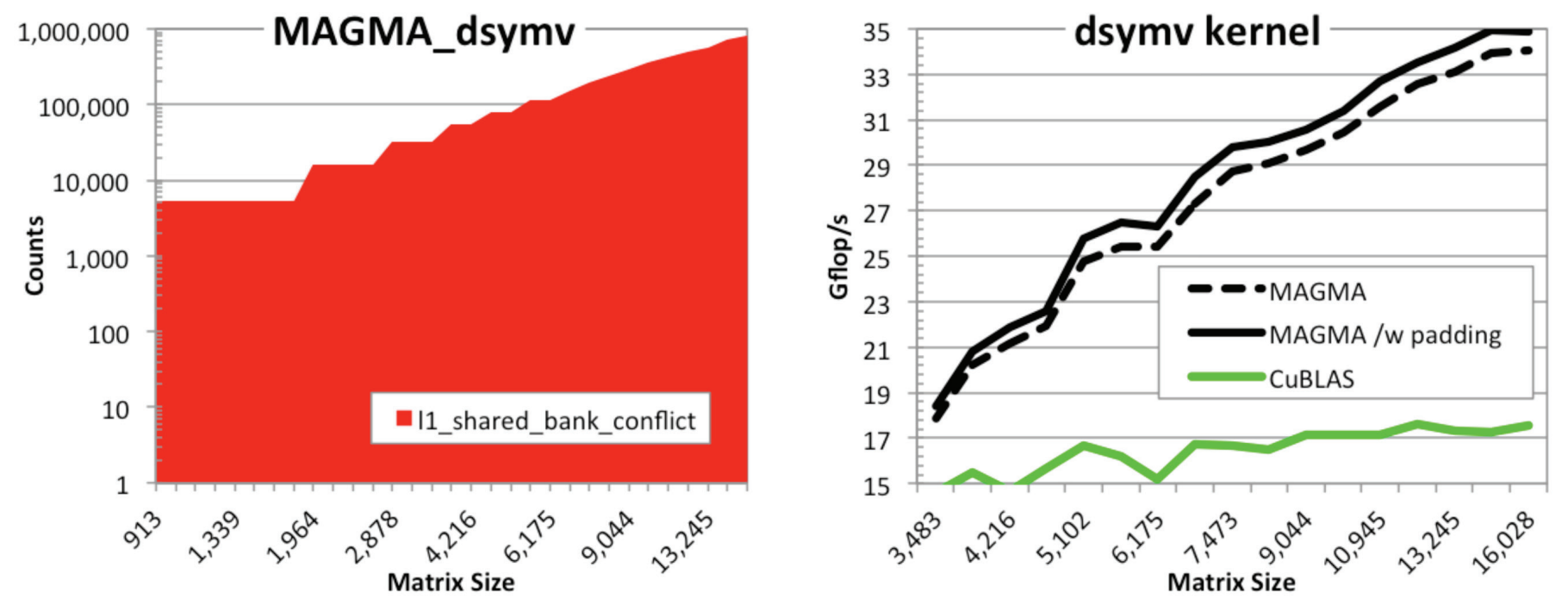
EXPERIMENT **MAGMA vs. CUBLAS LIBRARY**

- We ran experiments using `CUBLAS_dsymv` (general, means NO symmetry exploitation) and `MAGMA_dsymv` (exploits symmetry) to observe the effects of cache behavior on Tesla S2050 (Fermi) GPU
- As one example, from the PAPI measurements we were able to detect shared cache bank conflicts in the MAGMA kernel
- Those are due to addresses for two or more shared memory requests that fall in the same memory bank
- To address those conflicts, we applied array padding which causes cache lines to be placed in different cache locations

As a Result:

- We were able to completely eliminate the shared cache bank conflicts
- This minor change to the kernel code also gives us a performance improvement of 1Gflop/s for larger matrix sizes

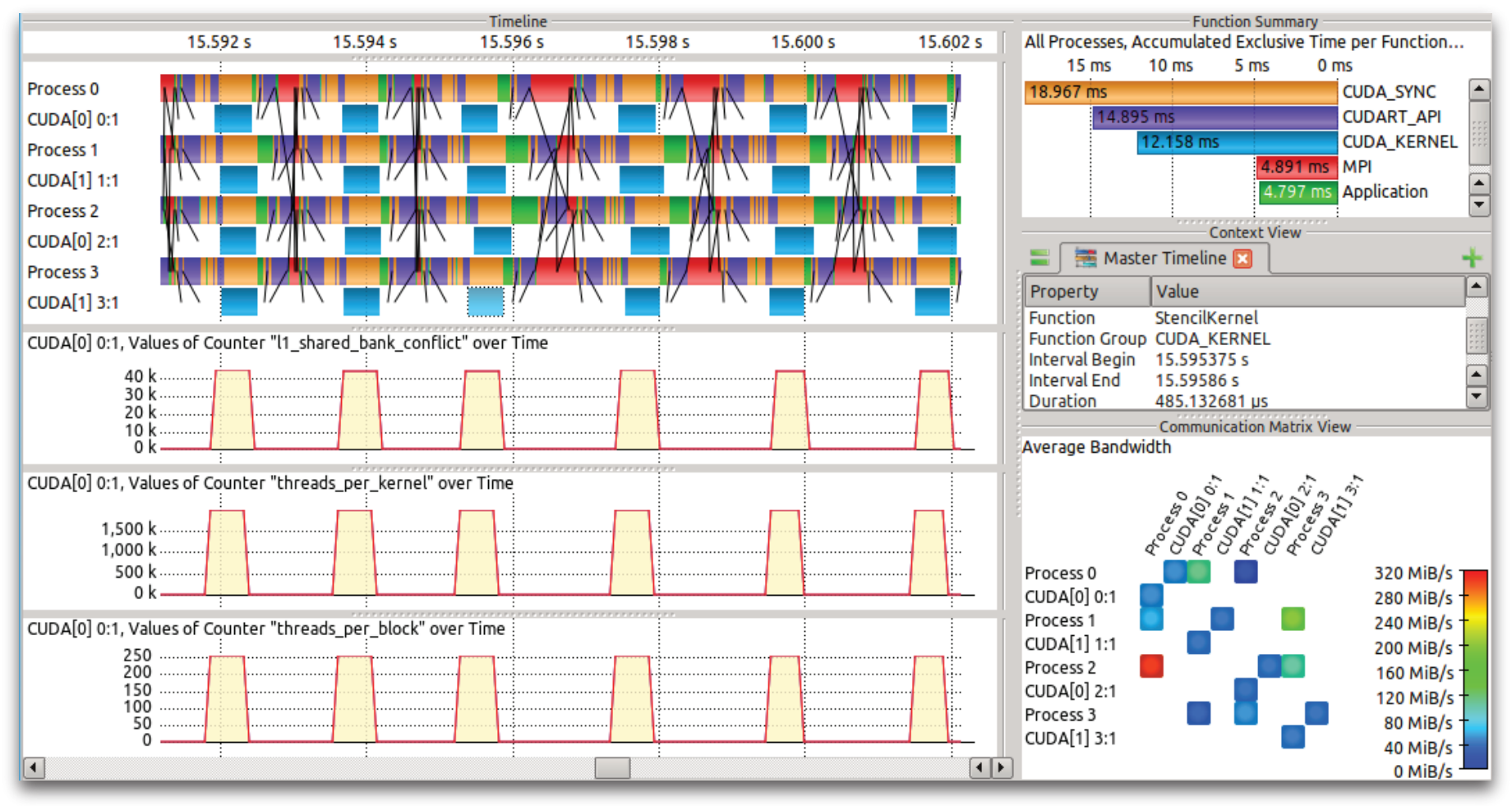
NOTE: Poor performance of `CUBLAS_dsymv` is due to lack of symmetry exploitation



LEFT: # of L1 shared bank conflicts in the `MAGMA_dsymv` kernel for medium to large matrix sizes

RIGHT: Performance of `MAGMA_dsymv` kernel with and without shared bank conflicts

EXPERIMENT **SHOC BENCHMARKS – STENCIL2D**


Function Summary
All Processes, Accumulated Exclusive Time per Function...

18.967 ms	CUDA_SYNC
14.895 ms	CUDART_API
12.158 ms	CUDA_KERNEL
4.891 ms	MPI
4.797 ms	Application

Context View — Master Timeline

Property	Value
Function	StencilKernel
Function Group	CUDA_KERNEL
Interval Begin	15.595375 s
Interval End	15.59586 s
Duration	485.132681 µs

CUDA[0] 0:1, Values of Counter "l1_shared_bank_conflict" over Time

CUDA[0] 0:1, Values of Counter "threads_per_kernel" over Time

CUDA[0] 0:1, Values of Counter "threads_per_block" over Time

- The VAMPIR display shows the timeline of a portion of a Stencil2D execution on 4 MPI processes with 4 GPUs, with CPU-GPU memory transfers and CPU-CPU communication
- The CPU and GPU counters were accessed via PAPI at each event and recorded here
- So here we show 3 different CUDA events for Process 0, CUDA[0]

PAPI-V

PAPI AND THE BOLD CLOUD COMPUTING FUTURE

- Much work is being done to investigate the practicality of moving High Performance Computing to the "cloud"
- Before such a move is made, the tradeoffs of moving to a cloud environment must be investigated
- PAPI is the ideal tool for making such measurements, but it will need enhancements before it works in a virtualized cloud environment

OBSTACLES WITH PAPI AND VIRTUALIZATION

- Virtualization makes time measurements difficult; virtualized time can run faster or slower than wall-clock time in unpredictable ways
- Hardware performance counter readings require the co-operation of both the operating system and hypervisor. Support for this is still under development.
- Virtualized hardware (such as network cards and disk) may require new PAPI components to be written

PAPI-V FUTURE PLANS

- Support for enhanced timing support, including access to real wall-clock time (if available)
- Provide components for collecting performance of virtualized hardware, such as virtual network, infiniband, GPU, and disk devices
- Provide transparent access to virtualized hardware performance counters

NEW FEATURES OF THE PAPI HARDWARE COUNTER LIBRARY

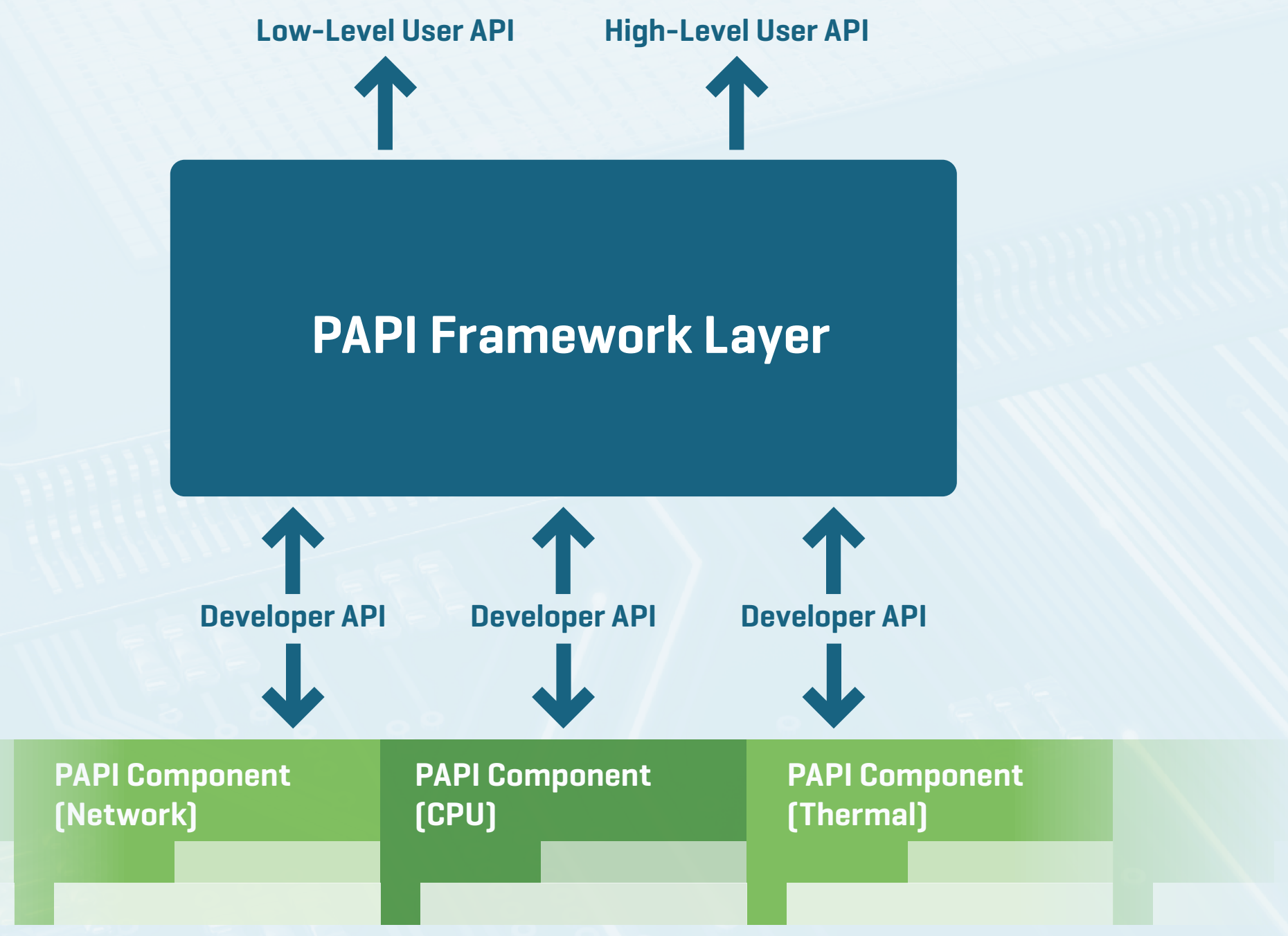
The PAPI specification and library have evolved from a cross-platform interface for accessing processor hardware performance counters to a component-based library for simultaneously accessing hardware monitoring information from various components of a computer system, including processors, memory controllers, network switches and interface cards, I/O subsystem, temperature sensors and power meters, and GPU counters. An illustration of the PAPI CUDA component used with NVIDIA hardware is shown to the left. A list of currently available PAPI components and a 3rd party component repository is shown below.

A new feature called user-defined events adds a layer of abstraction above native and preset events that allows users to define new metrics consisting of a combination of previously defined events and machine constants and to share those metrics with other users. This is illustrated on the right.

One current effort is the development of a PAPI interface for virtual machines, called PAPI-V, that will allow users to access processor and component hardware performance information from applications running within virtual machines. PAPI-V is discussed in further detail above.

PAPI continues to be widely used by application developers and by higher level performance analysis tools such as TAU, PerfSuite, Scalasca, IPM, HPCtoolkit, Vampir, and CrayPat.

PAPI Software Architecture



PAPI 4.2

NEW RELEASE

PAPI 4.2.0 is now available for download from the software page of the PAPI website. This release uses the libpfm4 and perf_events counter interface by default on linux systems. Documentation has been unified, reviewed, and updated with doxygen-driven man pages. Several components, particularly the CUDA component, have been updated, and a test environment for component tests has been implemented. Two new utilities have been added: papi_error_codes and papi_component_avail. A host of bug fixes and code clean-ups have also been implemented.

NEW PLATFORMS SUPPORTED

- AMD Family14h (Bobcat) and Family15h (Bulldozer)
- Intel Sandybridge, Westmere
- ARM Cortex A8, Cortex A9
- AIX Power7

OTHER COMPONENTS

Other PAPI components are available from the PAPI Component Repository (http://icl.eecs.utk.edu/projects/papi/repository/) including:

ACPI
Advanced Configuration and Power Interface Component

CoreTemp
Access hardware sensors through the coretemp sysfs interface

Infiniband
Infiniband Network Component

Lm-sensors
Component interface for lm-sensors system health measurement

Lustre
Measure performance data on a Lustre filesystem

USER EVENTS

MOTIVATION FOR USER-DEFINED EVENTS

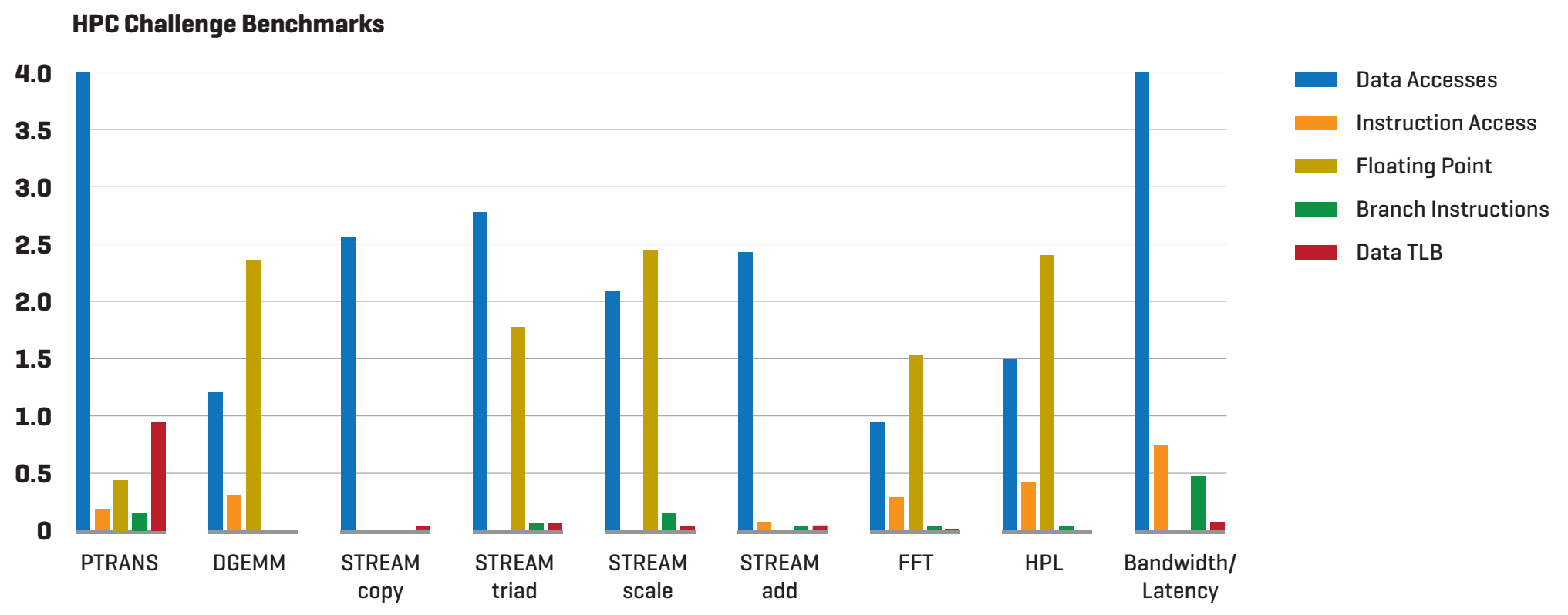
- Derived metrics are needed for performance analysis and modeling
  - Sums and ratios of native and preset events
  - Combinations of events with system constants
- Higher level tools (e.g., TAU and Scalasca) leave it to the user to select relevant events.
- Desirable to be able to define new metrics at run-time, rather than at PAPI installation time
- Enable performance modelers to publish metric definitions in a well-defined way

PAPI USER-DEFINED EVENT MECHANISM

- Allows users to define their own metrics
  - User can combine events and constants in an expression to define and name a new metric, in a postfix notation
  - Maps the new metric to events available on a platform without the need to re-install PAPI
- User-defined event names can be used in PAPI library calls the same way as preset and native events.
- User-defined events can be used with end-user performance tools such as TAU and Scalasca without modifying those tools.

EXAMPLE 1 **PERFEXPERT METRICS**

- M. Burtscher et. al., **PerfExpert: An easy-to-use performance diagnosis tool for HPC applications**, in SC10, New Orleans, 2010
- Methodology combines hardware counter measurements with architectural parameters to compute upper bounds on local cycle-per-instruction (LCPI) contributions of various instruction categories:
  - branches
  - data memory access
  - instruction memory access
  - data TLB access
  - instruction TLB access
  - floating-point operations



EXAMPLE 2 **MEMORY BANDWIDTH**

- Memory bandwidth actually achieved can be measured by performance counters on most platforms.

For example, on AMD Opteron:

```
DRAM_ACCESSES | 64 | * | core_frequency | * | PAPI_TOT_CYC | /
```

Note: DRAM_ACCESSES may be a combination of native events with masks.

On Intel Core2:

```
BUS_TRANS:SELF | 64 | * | core_frequency | * | PAPI_TOT_CYC | /
```

- Currently validating counter results using variations of the STREAM benchmark

Reasonable agreement with STREAM output on Intel Core2:

MB/s	STREAM output	Counters	% Delta
Copy	2227	2204	-1%
Scale	2332	2333	0%
Add	2471	2326	-6%
Triad	2473	2312	-6%

and on AMD Opteron 8358:

MB/s	STREAM output	Counters	% Delta
Copy	3155	3126	-1%
Scale	3001	2979	-1%
Add	3150	3026	-4%
Triad	3031	2965	-2%

FIND OUT MORE AT **http://icl.eecs.utk.edu/papi/**