Introduction to
PAPI
the
Performance Application Programming Interface

Dan Terpstra, Heike Jagode

terpstra | jagode@eecs.utk.edu

February 16th 2009

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:

1. Whole program timing
2. Cache behaviors
3. Branch behaviors
4. Memory and resource contention and access patterns
5. Pipeline stalls
6. Floating point efficiency
7. Instructions per cycle
8. Subroutine resolution
9. Process or thread attribution

What's PAPI?

- Middleware that provides a consistent and efficient programming interface for the performance counter hardware found in most major microprocessors.

- Started as a Parallel Tools Consortium project in 1998
  - Goal was to produce a specification for a **portable interface** to the hardware performance counters.

- 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

Where's PAPI ?

- PAPI runs on most modern processors and Operating Systems of interest to HPC:
  - IBM POWER{4, 5, 5+, 6} / AIX or Linux
  - PowerPC{-32, -64, 970} / Linux
  - Cell
  - Blue Gene / {L, P}
  - Intel Pentium II, III, 4, M, Core, etc. / Linux
  - Intel Itanium{1, 2, Montecito, Montvale}
  - AMD Athlon, Opteron / Linux
  - Cray XT{3, 4} Catamount, CNL
  - Altix, Sparc, SiCortex…
  - …and even Windows {XP, 2003 Server; PIII, Athlon, Opteron}!
  - …but not Mac ☹

- At the VI-HPS workshop PAPI is available on:
  - POWER6 Cluster in Juelich
  - BlueGene/P in Juelich
  - SGI Altix in Dresden
  - Sun Niagara2 cluster (RWTH) in Aachen

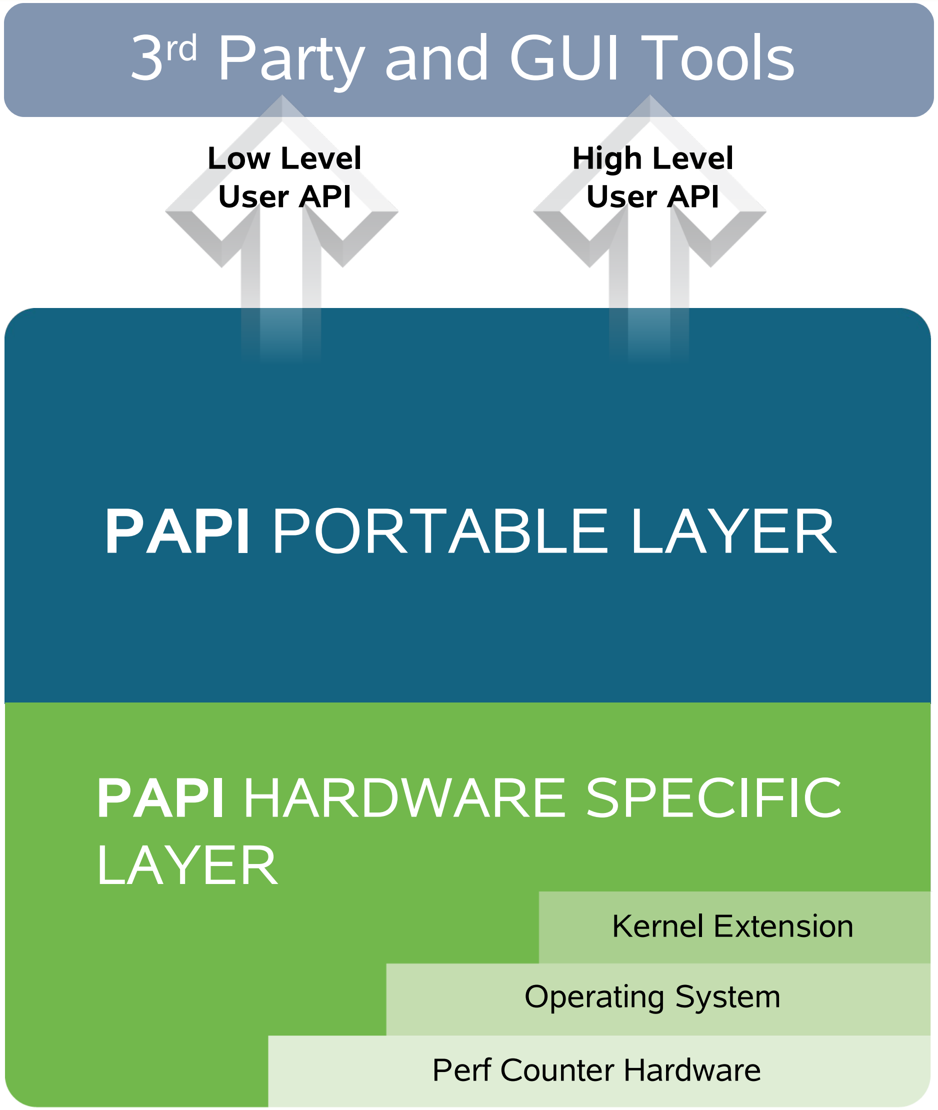
PAPI Counter Interfaces

PAPI provides 3 interfaces to the underlying counter hardware:

- 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.

- A High Level API provides the ability to start, stop and read the counters for a specified list of events (preset only). Meant for for programmers wanting simple event measurements.

1. Graphical and end-user tools provide facile data collection and visualization.



PAPI High Level Calls

1. PAPI_num_counters()
   - get the number of hardware counters available on the system
2. 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
3. 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
4. PAPI_ipc **(float** **rtime*, **float** **ptime*, **long long** **ins*, **float** **ipc*)
   - gets instructions per cycle, real and processor time
5. PAPI_accum_counters **(long long** **values*, **int** *array_len*)
   - add current counts to array and reset counters
6. PAPI_read_counters **(long long** **values*, **int** *array_len*)
   - copy current counts to array and reset counters
7. PAPI_start_counters **(int** **events*, **int** *array_len*)
   - start counting hardware events
8. PAPI_stop_counters **(long long** **values*, **int** *array_len*)
   - stop counters and return current counts

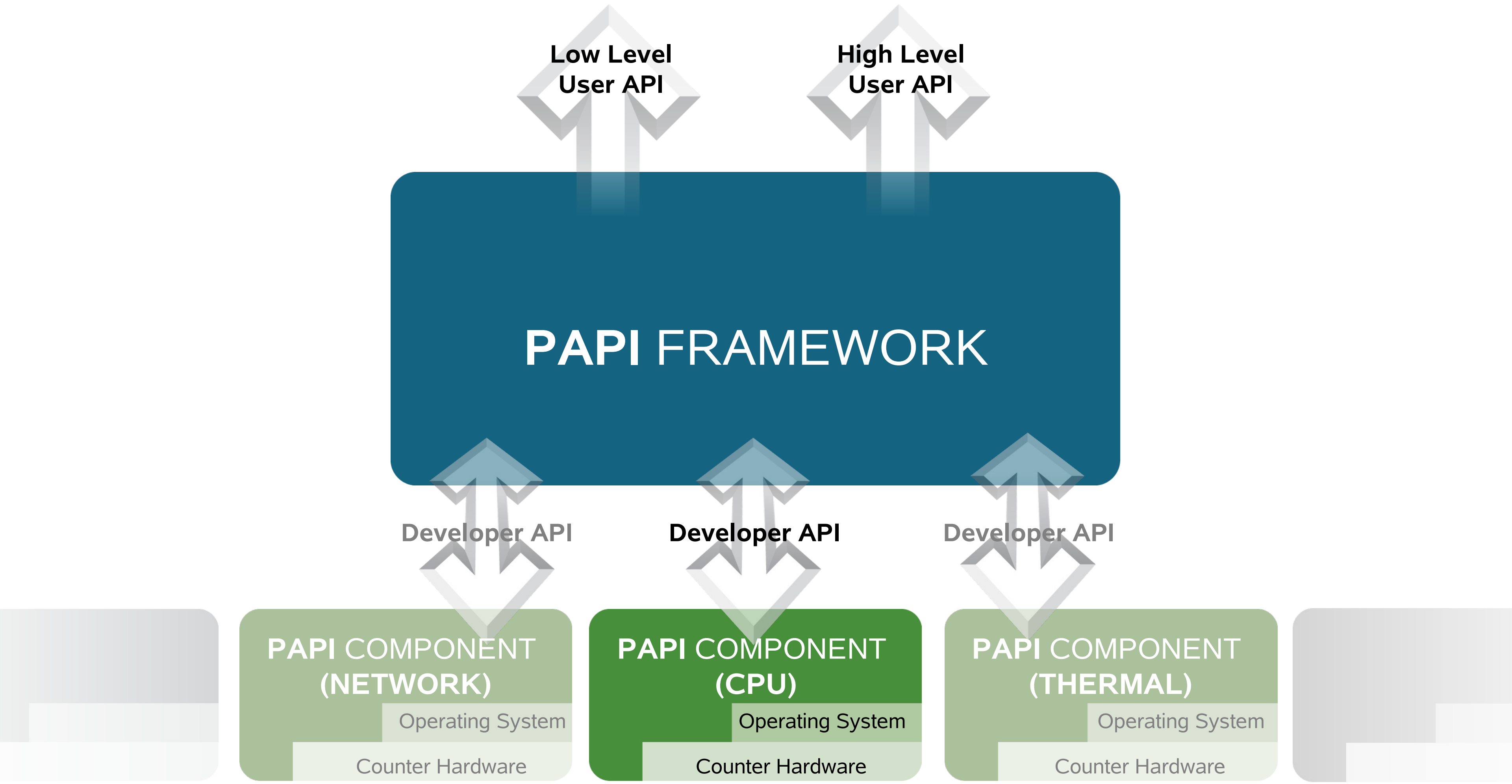
Example: Low Level API

```
#include "papi.h”
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_OPS,PAPI_TOT_CYC},
int EventSet;
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);
```

Component PAPI (PAPI-C)

- **Motivation:**
  - Hardware counters aren't just for cpus anymore
    - Network counters; thermal & power measurement…
  - Often insightful to measure multiple counter domains at once
- **Goals:**
  - Support simultaneous access to on- and off-processor counters
  - Isolate hardware dependent code in a separable component module
  - Extend platform independent code to support multiple simultaneous components
  - Add or modify API calls to support access to any of several components
  - Modify build environment for easy selection and configuration of multiple available components

Component PAPI



Some Tools that use PAPI

- TAU (U Oregon) http://www.cs.uoregon.edu/research/tau/
- PerfSuite (NCSA) http://perfsuite.ncsa.uiuc.edu/
- HPCToolkit (Rice Univ) http://hipersoft.cs.rice.edu/hpctoolkit/
- KOJAK and SCALASCA (FZ Juelich, UTK) http://icl.cs.utk.edu/kojak/
- VampirTrace and Vampir (TU Dresden) http://www.vamir.eu
- Open|Speedshop (SGI) http://oss.sgi.com/projects/openspeedshop/
- SvPablo (UNC Renaissance Computing Institute) http://www.renci.unc.edu/Software/Pablo/pablo.htm
- ompP (UTK) http://www.ompp-tool.com

Introduction to
PAPI
the
Performance Application Programming Interface

Dan Terpstra, Heike Jagode
terpstra | jagode@eecs.utk.edu

February 16th 2009

THE UNIVERSITY of TENNESSEE