Collecting Performance Data with PAPI-C

Dan Terpstra
Jack Dongarra
Shirley Moore
Haihang You
Heike Jagode

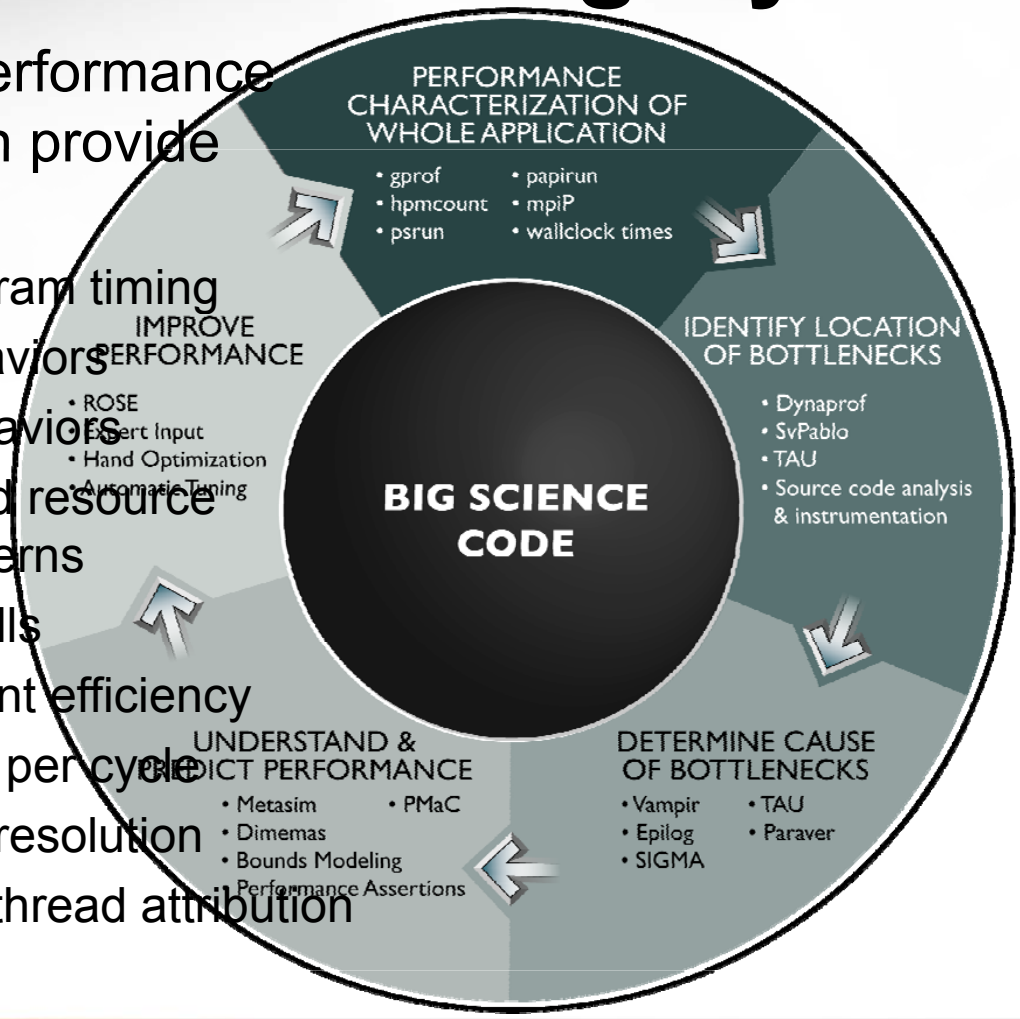
3rd Parallel Tools Workshop
Dresden, September 14-15

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

The Tuning Cycle

Hardware performance counters can provide insight into:

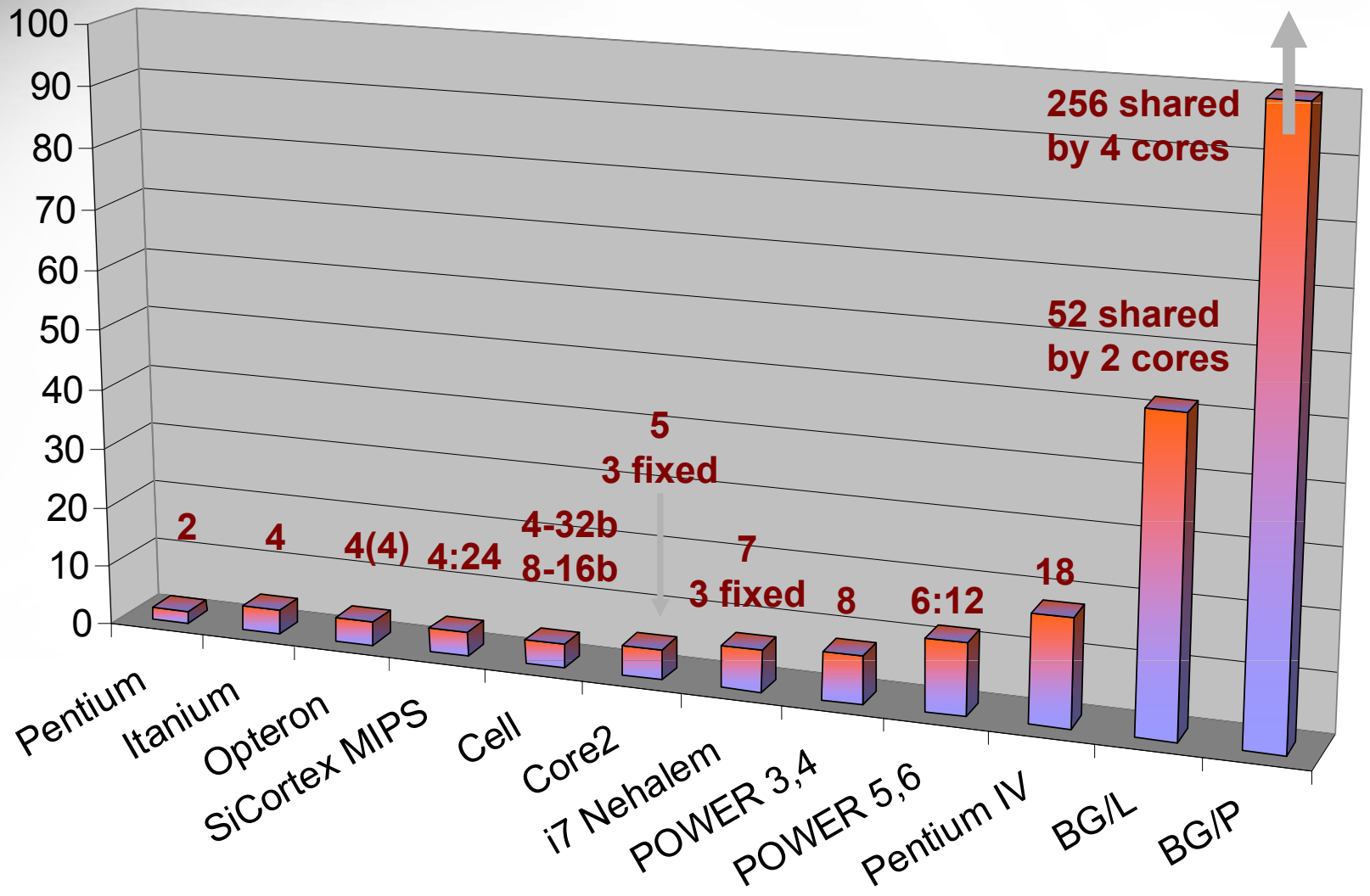
1. Whole program timing
2. Cache behaviors
3. Branch behaviors
4. Memory and resource access patterns
5. Pipeline stalls
6. Floating point efficiency
7. Instructions per cycle
8. Subroutine resolution
9. Process or thread attribution



What's PAPI?

- A platform independent interface to hardware performance counters.
- PAPI counts events:
  - ***Native*** events are platform specific
  - ***Preset*** events are platform neutral
    - Linear combinations of native events
- Events are:
  - Referenced by name
  - Abstracted into EventSets and counted together
- Events can be:
  - Multiplexed when counters are limited
  - Counted discretely over specific code regions
  - Sampled via interrupt on overflow for statistical profiling

How many counters does it take?



PAPI Preset 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 utility to see what preset events are available on a given platform

PAPI_L2_DCW: Level 1 data cache writes
PAPI_L2_DCM: Level 1 data cache misses

PAPI_L2_ICH: Level 1 instruction cache hits
PAPI_L2_ICA: Level 1 instruction cache accesses
PAPI_L2_ICR: Level 1 instruction cache reads
PAPI_L2_ICW: Level 1 instruction cache writes
PAPI_L2_ICM: Level 1 instruction cache misses

PAPI_L2_TCH: Level 1 total cache hits
PAPI_L2_TCA: Level 1 total cache accesses
PAPI_L2_TCR: Level 1 total cache reads
PAPI_L2_TCW: Level 1 total cache writes
PAPI_L2_TCM: Level 1 cache misses

PAPI_L2_LDM: Level 1 load misses
PAPI_L2_STM: Level 1 store misses

Level 3 Cache

PAPI_L3_DCH: Level 1 data cache hits
PAPI_L3_DCA: Level 1 data cache accesses
PAPI_L3_DCR: Level 1 data cache reads
PAPI_L3_DCW: Level 1 data cache writes
PAPI_L3_DCM: Level 1 data cache misses

PAPI_L3_ICH: Level 1 instruction cache hits
PAPI_L3_ICA: Level 1 instruction cache accesses
PAPI_L3_ICR: Level 1 instruction cache reads
PAPI_L3_ICW: Level 1 instruction cache writes
PAPI_L3_ICM: Level 1 instruction cache misses

PAPI_L3_TCH: Level 1 total cache hits
PAPI_L3_TCA: Level 1 total cache accesses
PAPI_L3_TCR: Level 1 total cache reads
PAPI_L3_TCW: Level 1 total cache writes
PAPI_L3_TCM: Level 1 cache misses

PAPI_L3_LDM: Level 1 load misses
PAPI_L3_STM: Level 1 store misses

Cache Sharing

PAPI_CA_SNP: Requests for a snoop
PAPI_CA_SHR: Requests for exclusive access to shared cache line

PAPI Native Events

- 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

PRESET,
PAPI_L2_DCA,
DERIVED_ADD,
L2_LD:SELF:ANY:MESI,
L2_ST:SELF:MESI

```
{ .pme_name = "L2_ST",
  .pme_code = 0x2a,
  .pme_flags = PFMLIB_CORE_CSPEC,
  .pme_desc =  "L2 store requests",
  .pme_umasks = {
    { .pme_uname = "MESI",
      .pme_udesc = "Any cacheline access",
      .pme_ucode = 0xf
    },
    { .pme_uname = "I_STATE",
      .pme_udesc = "Invalid cacheline",
      .pme_ucode = 0x1
    },
    { .pme_uname = "S_STATE",
      .pme_udesc = "Shared cacheline",
      .pme_ucode = 0x2
    },
    { .pme_uname = "E_STATE",
      .pme_udesc = "Exclusive cacheline",
      .pme_ucode = 0x4
    },
    { .pme_uname = "M_STATE",
      .pme_udesc = "Modified cacheline",
      .pme_ucode = 0x8
    }
    { .pme_uname = "SELF",
      .pme_udesc = "This core",
      .pme_ucode = 0x40
    },
    { .pme_uname = "BOTH_CORES",
      .pme_udesc = "Both cores",
      .          = 0xc0
    }
   },
   .pme_numasks = 7
},
```

Where's PAPI

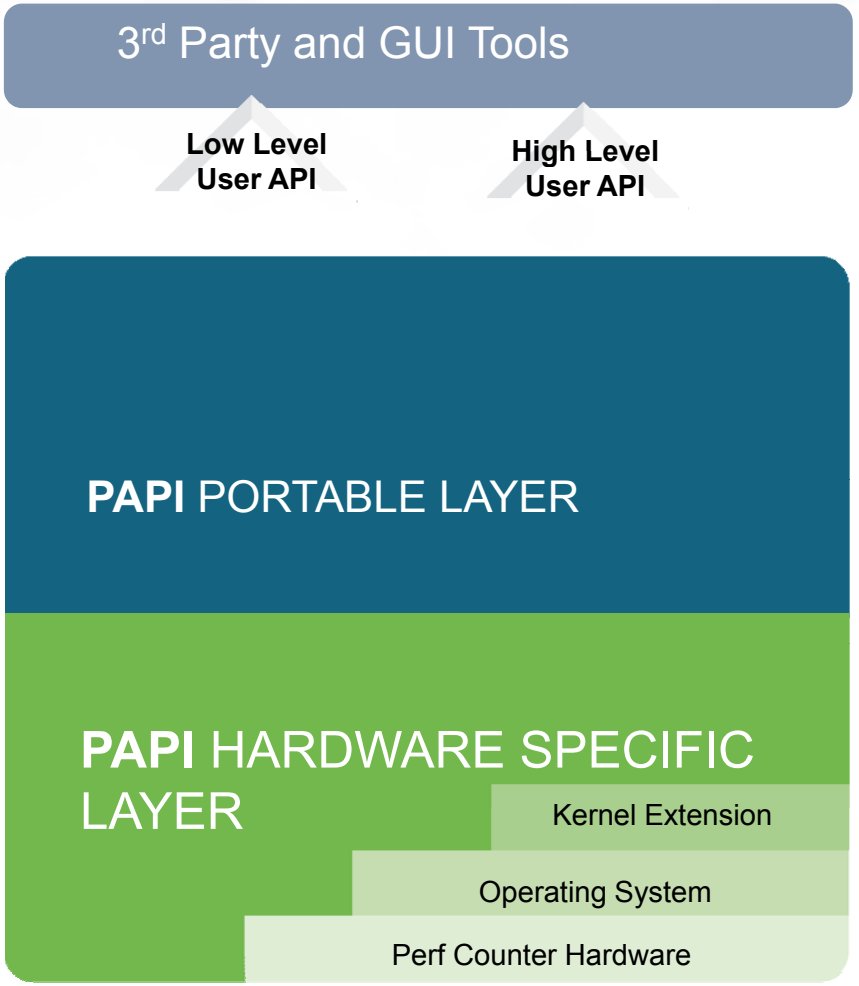
- PAPI runs on most modern processors and operating systems of interest to HPC:
  - IBM POWER / AIX / Linux
  - Blue Gene / L / P
  - Intel Pentium, Core2, Core i7, Atom / Linux
  - Intel Itanium / Linux
  - AMD Athlon, Opteron / Linux
  - Cray XT(n) / CLE
  - Sun Niagara2
  - Altix, Sparc, SiCortex

Some tools that use PAPI

- TAU (U Oregon)
  - http://www.cs.uoregon.edu/research/tau/
- PerfSuite (NCSA)
  - http://perfsuite.ncsa.uiuc.edu/
- Scalasca (UTK, FZ Juelich)
  - http://www.fz-juelich.de/jsc/scalasca/
- Vampir (TUDresden)
  - http://www.vampir.eu/
- HPCToolkit (Rice Univ.)
  - http://hpctoolkit.org/
- Open|Speedshop (SGI)
  - http://oss.sgi.com/projects/openspeedshop/

PAPI Counter Interfaces

PAPI provides 3 interfaces to the underlying counter hardware:



1. **A Low Level API manages hardware events in user defined groups called EventSets, and provides access to advanced features.**
2. **A High Level API provides the ability to start, stop and read the counters for a specified list of events.**
3. **Graphical and end-user tools provide facile data collection and visualization.**

PAPI High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (less additional code) than low-level
- Supports 8 calls in C or Fortran:

```
PAPI_start_counters
PAPI_stop_counters
PAPI_read_counters
PAPI_accum_counters
PAPI_num_counters
PAPI_flips
PAPI_ipc
PAPI_flops
```

PAPI High-level Example

```
#include "papi.h"
#define NUM_EVENTS 2
long_long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};

 /* Start the counters */
 PAPI_start_counters((int*)Events,NUM_EVENTS);

 /* What we are monitoring… */
 do_work();

 /* Stop counters and store results in values */
 retval = PAPI_stop_counters(values,NUM_EVENTS);
```

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- Obtain information about the executable, the hardware, and the memory environment
- Multiplexing
- Callbacks on counter overflow
- Profiling
- About 60 functions

PAPI Low-level Example

```
#include "papi.h"
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,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);
```