PAPI

Performance Application Programming Interface
Philip J. Mucci, mucchi@cs.utk.edu
Felix Wolf, fwolf@cs.utk.edu
“The single most important impediment to good parallel performance is still poor single-node performance.”

- William Gropp
  Argonne National Lab
Optimizing Performance

Start

Is it too slow?

Yes

Find the Problem

Fix the Problem

No

Stop
Tools for Performance Evaluation

• Traditionally, timing and performance evaluation has been an art
  – Resolution of the clock
  – Issues about cache effects
  – Different systems
  – Can be cumbersome and inefficient with traditional tools

• Situation has changed
  – Today’s processors have internal counters
• Today most high performance processors include hardware performance counters.
• Some are easy to access, others not available to users.
• On most platforms the APIs, if they exist, are not appropriate for the end user or well documented.
• Existing performance counter APIs
  – Compaq Alpha EV 6 & 6/7
  – SGI MIPS R10000
  – IBM Power Series
  – CRAY T3E, X1
  – Sun Solaris
  – Pentium and AMD
  – IA-64
  – HP-PA RISC
  – Hitachi
  – Fujitsu
  – NEC
• Performance Application Programming Interface

• The purpose of PAPI is to implement a standardized portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.

• The goal of PAPI is to facilitate the optimization of parallel and serial code performance by encouraging the development of cross-platform optimization tools.
PAPI Counter Interfaces

- **PAPI provides 3 interfaces to the underlying counter hardware:**

1. The low level interface manages hardware events in user defined groups called EventSets, and provides access to advanced features.

2. The high level interface 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 Implementation

- Portable Layer:
  - PAPI Low Level
  - PAPI High Level

- Machine Specific Layer:
  - PAPI Machine Dependent Substrate
  - Kernel Extension
  - Operating System
  - Hardware Performance Counters
High Level Interface

• Meant for application programmers wanting coarse-grained measurements
• Not tuned for efficiency
• Calls the lower level API
• Only allows PAPI Presets
• Easier to use and less setup (less additional code) than low level
High Level Functions

- `PAPI_num_counters()`
- `PAPI_start_counters()`
- `PAPI_stop_counters()`
- `PAPI_read_counters()`
- `PAPI_accum_counters()`
- `PAPI_ipc`
- `PAPI_flips()`
- `PAPI_flops()`
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 the counters and store the 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 & the memory
- Thread-safe
- Fully programmable (native events)
- Multiplexing
- Callbacks on counter overflow
- Profiling
- 54 functions
Available Performance Data

- **Cycle count**
- **Instruction count**
  - All instructions
  - Floating point
  - Integer
  - Load/store
- **Branches**
  - Taken / not taken
  - Mispredictions
- **Pipeline stalls due to**
  - Memory subsystem
  - Resource conflicts
- **Cache**
  - I/D cache misses for different levels
  - Invalidations
- **TLB**
  - Misses
  - Invalidations
## Parallel Ocean Program Performance

**Run:** x1 Data Set, 2x2 Procs, 10 Steps

<table>
<thead>
<tr>
<th>Raw Data</th>
<th>Debug</th>
<th>Optimized</th>
<th>Metric</th>
<th>Debug</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_LD_INS</td>
<td>1.21E+011</td>
<td>2.104E+10</td>
<td>% Ld Ins</td>
<td>36.86</td>
<td>33.63</td>
</tr>
<tr>
<td>PAPI_SR_INS</td>
<td>2.02E+010</td>
<td>7.783E+09</td>
<td>% Sr Ins</td>
<td>6.17</td>
<td>12.44</td>
</tr>
<tr>
<td>PAPI_BR_INS</td>
<td>8.64E+009</td>
<td>5.043E+09</td>
<td>% Br Ins</td>
<td>2.63</td>
<td>8.06</td>
</tr>
<tr>
<td>PAPI_FP_INS</td>
<td>2.21E+010</td>
<td>2.251E+10</td>
<td>% FP Ins</td>
<td>6.75</td>
<td>35.98</td>
</tr>
<tr>
<td>PAPI_FMA_INS</td>
<td>1.04E+010</td>
<td>1.007E+10</td>
<td>% FMA Ins</td>
<td>3.16</td>
<td>16.09</td>
</tr>
<tr>
<td>PAPI_FPU_FDIV</td>
<td>2.551E+08</td>
<td></td>
<td>% FP Divide</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td>PAPI_FPU_FSQRT</td>
<td>1.317E+08</td>
<td></td>
<td>% FP SQRT</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>3.28E+011</td>
<td>6.257E+10</td>
<td>MFLIPS</td>
<td>12.19</td>
<td>72.31</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>3.63E+011</td>
<td>6.226E+10</td>
<td>% MFLIPS Pea</td>
<td>3.05</td>
<td>18.08</td>
</tr>
</tbody>
</table>

|                |        |            | IPC          | 0.90   | 1.00       |
|                |        |            | Mem Opts/FLIF| 6.38   | 1.28       |
| PAPI_L1_LDM    | 1.03E+009 | 1.011E+09  | % L1 Ld HR   | 99.15  | 95.19      |
| PAPI_L1_STM    | 3.54E+008 | 3.475E+08  | % L1 Sr HR   | 98.25  | 95.54      |
| PAPI_L2_LDM    | 6.94E+008 | 6.894E+08  | % L2 Ld HR   | 99.43  | 96.72      |
| PAPI_FPU_IDL   | 1.66E+011 | 1.411E+10  | % FPU Idle Cyc| 45.77 | 22.66      |
| PAPI_LSU_IDL   | 4.06E+010 | 1.483E+10  | % LSU Idle Cyc| 11.17 | 23.82      |
| PAPI_MEM_RC    | 1.03E+011 | 1.368E+10  | % Ld Stall Cyc| 28.28 | 21.97      |
| PAPI_MEM_SC    | 1.26E+011 | 2.413E+10  | % Sr Stall Cyc| 34.59 | 38.76      |
| PAPI_STL_CCY   | 2.01E+011 | 3.367E+10  | % No Ins. Cyc| 55.25  | 54.08      |
Example Tool: Perfometer

Machine info

Total Flops

Flop/s Rate

Flop/s Min/Max

Process & Real time
Supported Architectures

- AMD Athlon and Opteron
- Cray T3E and X1
- HP Alpha (caveats)
- IBM POWER3, POWER4
- Intel Pentium Pro, II, III + 4, Itanium 1 + 2
- MIPS R10K, R12K, R14K
- Sun UltraSparc I, II, III
• *Standardized Access to Performance Counters*

• *Standardized Performance Metrics*

• *Easy Access to Platform-Specific Metrics*

• *Multiplexed Event Measurement*

• *Dispatch on Overflow*

• *Event Profiling*

• *Bindings for C, Fortran, Matlab, and Java*
New in PAPI 3.0

• Lower Measurement Overheads
• Overflow and Profiling on Multiple Simultaneous Events
• Easy Access to Platform-Specific Metrics
• High level API is now thread safe
• Internal timer/signal/thread abstractions
Overhead example: PAPI 3.0 vs PAPI 2.3.4

(PAPI 2.3.4 overhead – PAPI 3.0 overhead)/PAPI 2.3.4 overhead

% Overhead Reduction

Start/Stop Pair

Read

Processor

Athlon

P3

R12K

Itanium

Ultra II

Power

-25

0

25

50

75

100
New in PAPI 3.0

• Lower Measurement Overheads
• Overflow and Profiling on Multiple Simultaneous Events
• Easy Access to Platform-Specific Metrics
• High level API is now thread safe
• Internal timer/signal/thread abstractions
Multiple Counter Profiling

Event Count

Program Text Addresses

L1DCM  L2DCM  DTLB
New in PAPI 3.0

- Lower Measurement Overheads
- Overflow and Profiling on Multiple Simultaneous Events
- Easy Access to Platform-Specific Metrics
- High level API is now thread safe
- Internal timer/signal/thread abstractions
Other Tools that use PAPI

- **TAU (Sameer Shende, U Oregon)**
  http://www.cs.uoregon.edu/research/paracomp/tau/

- **SvPablo (Celso Mendes, UIUC)**
  http://www-pablo.cs.uiuc.edu/Project/SVPablo/

- **HPCToolkit (John Mellor-Crummey, Rice Univ)**
  http://hipster.cs.rice.edu/hpctoolkit/

- **KOJAK (Bernd Mohr, FZ Juelich; U Tenn)**
  http://www.fz-juelich.de/zam/kojak/

- **psrun (Rick Kufrin, NCSA, UIUC)**
  http://www.ncsa.uiuc.edu/~rkufrin/perfsuite/psrun/

- **Titanium (Dan Bonachea, UC Berkeley)**
  http://www.cs.berkeley.edu/Research/Projects/titanium/

- **SCALEA (Thomas Fahringer, U Innsbruck)**
  http://www.par.univie.ac.at/project/scalea/
Questions?

- http://icl.cs.utk.edu/papi/ - PAPI Homepage