cse5441 - parallel computing

advanced

instrumentation

J. S. Jones

Performance API (PAPI)

- discussion standard for hardware-enabled performance measurement
- cross-platform support
- potential for portable solution
- incentive for multi-vendor support
- ease-of-use

https://developer.nvidia.com/sites/default/files/styles/main_image/public/akamai/cuda/images/papi_logo.png

PAPI interfaces

high-level interface

- start, read and stop event counters

Function name	Description
PAPI_num_counters( )	get the number of hardware counters available on the system
PAPI_flips ( )	Mflips/s (floating point instruction rate)
PAPI_flops( )	Mflops/s (floating point operation rate)
PAPI_ipc( )	gets instructions per cycle
PAPI_accum_counters( )	add current counts to array and reset counters
PAPI_read_counters( )	copy current counts to array and reset counters
PAPI_start_counters( )	start counting hardware events
PAPI_stop_counters( )	stop counters and return current counts

PAPI standard events

examples

PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICM	Level 2 instruction cache misses
PAPI_L3_DCM	Level 3 data cache misses
PAPI_L3_ICM	Level 3 instruction cache misses
PAPI_L1_TCM	Level 1 total cache misses
PAPI_L2_TCM	Level 2 total cache misses
PAPI_L3_TCM	Level 3 total cache misses
. . .	
PAPI_LD_INS	Load instructions executed
PAPI_SR_INS	Store instructions executed
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
. . .	
PAPI_TOT_CYC	Total cycles
PAPI_TOT_INS	Total instructions executed
PAPI_FP_INS	Floating point instructions executed
. . .	

PAPI cache metrics

L1 data cache hit rate:

1.0 – (PAPI_L1_DCM / (PAPI_LD_INS + PAPI_SR_INS))

L2 data cache hit rate:

1.0 – (PAPI_L2_DCM / PAPI_L1_DCM)

basic PAPI demo

Function name	Description
PAPI_start_counters( )	start counting hardware events
PAPI_stop_counters( )	stop counters and return current counts

```
/*****************************************************************************\
* Initialize the PAPI library and get the number of counters available *
\*****************************************************************************/
int            Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
int            num_hwcntrs = 0;
long_long      values[2];

if ((num_hwcntrs = PAPI_num_counters()) <= PAPI_OK)
{
    cerr << "PAPI init error -- exiting" << endl;
    exit(1);
}
cout << "This system has " << num_hwcntrs << " available counters.\n";

t = clock();
if (PAPI_start_counters(Events, 2) != PAPI_OK)
{
     cerr << "PAPI start error -- exiting" << endl;
    exit(1);
}
/**************************************************************\
* run a simple stencil computation on the grid                *
\**************************************************************/
Mesh.dissipate(Mesh.affect_rate, Mesh.epsilon);
```

```
/***************\
* print results    *
\***************/
if (PAPI_stop_counters(values, 2) != PAPI_OK)
{
    cerr << "PAPI count error -- exiting" << endl;
    exit(1);
}

t = clock() - t;
cout << "Covergence time reported by clock():" << setw(20) << t << endl;
cout << "                  PAPI total cycles:" << setw(20) << values[0] << endl;
cout << "            PAPI total instructions:" << setw(20) << values[1] << endl;
```

PAPI interfaces

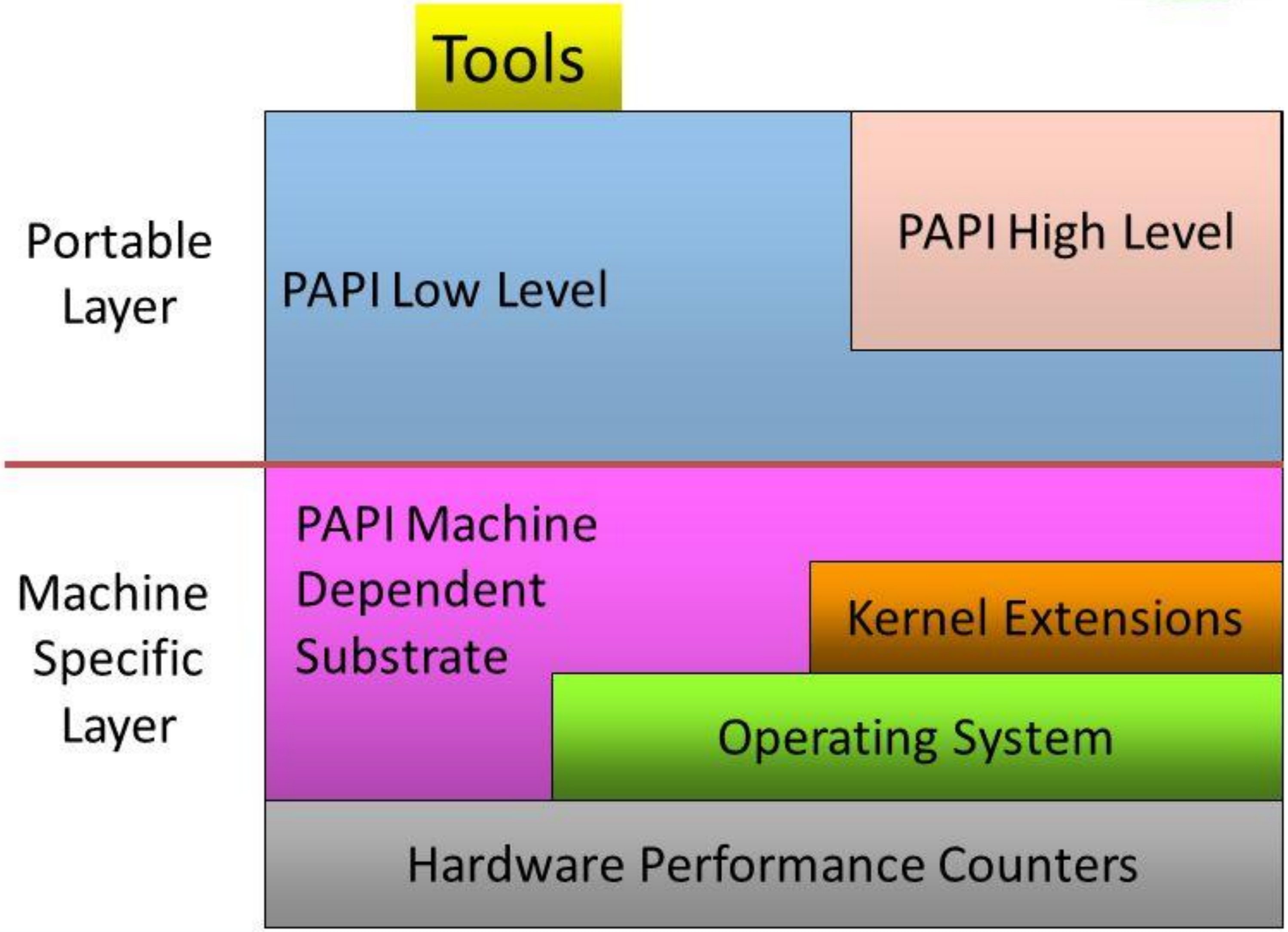
low-level interface

- customizable access to event counters

```
#include <papi.h>
#include <stdio.h>


main()
{
int retval, EventSet = PAPI_NULL;
long_long values[1];


/* Initialize the PAPI library */
retval = PAPI_library_init(PAPI_VER_CURRENT);


if (retval != PAPI_VER_CURRENT) {
  fprintf(stderr, "PAPI library init error!\n");
  exit(1);
}


/* Create the Event Set */
if (PAPI_create_eventset(&EventSet) != PAPI_OK)
    handle_error(1);


/* Add Total Instructions Executed to our EventSet */
if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
    handle_error(1);
```

```
/* Start counting */
if (PAPI_start(EventSet) != PAPI_OK)
    handle_error(1);

/* Do some computation here */

if (PAPI_read(EventSet, values) != PAPI_OK)
    handle_error(1);

/* Do some computation here */

if (PAPI_stop(EventSet, values) != PAPI_OK)
    handle_error(1);
}
```

PAPI Architecture



PAPI thread support

- must use kernel (or “bound”) threads for thread-level measurement
  - PTHREAD_SCOPE_SYSTEM for Pthreads
  - otherwise, all threads report process values

```
include <papi.h>
#include <pthread.h>

main()
{
unsigned long int tid;

if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT)     ← uses low-level interface
  exit(1);


if (PAPI_thread_init(pthread_self) != PAPI_OK)                   ← each thread must initialize
  exit(1);


if ((tid = PAPI_thread_id()) == (unsigned long int)-1)           ← can now request thread id
  exit(1);


printf("Initial thread id is: %lu\n",tid);                       ← will be  0  for master thread
}
```

PAPI thread support

PAPI_register_thread()	each thread must register
PAPI_unregister_thread()	release prior to reuse
PAPI_get_thr_specific("tag, ptr")	four defined memory locations for thread-specific data (why 4?)
PAPI_set_thr_specific("tag, ptr")	

PAPI portability

- goal is not to compare systems
- implementation of as many standard events as possible
  - while avoiding misleading or erroneous results
- vendor counter interfaces sometimes have bugs
  - counter overflow

PAPI on OSC

- version 5.4.1 available on Oakley and Ruby
  - module load gnu/4.8.5
  - module load papi
- g++ -lpapi

PAPI cautions

- not all functions in both C and Fortran
- documentation is inconsistent
- not all functions supported on all platforms
- some errors in hardware counters
- no commercial support

PAPI further reading

Carmen:

• papi-journal-final_2000	original journal paper
• papi-conference-2001	similar, more brief
• papi-dongarra-2003	update

On-line documentation:

- http://icl.cs.utk.edu/projects/papi/wiki/Main_Page

cse5441 - parallel computing

advanced

instrumentation