C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
(Mode-1:Multi-Core)

Lecture Topic:

Multi-Core Processors : Tuning & Perf : PAPI tool

PAPI (Performance Application Program Interface)

Venue : CMSD, UoHYD ; Date : October 15-18, 2013

Performance Application Program Interface

Lecture Outline

Following Topics will be discussed

- What is PAPI ?
- PAPI Architecture
- Using PAPI
  - Events
  - PAPI library Interface
  - Error Handling
- Advanced PAPI Features
- Example Programs using PAPI

Source : http://icl.cs.utk.edu/papi/index.html

What is PAPI ?

- PAPI is an acronym for Performance Application Programming interface.

- PAPI is a specification of a cross-platform interface to hardware performance counters on modern microprocessors. These counters exist as a small set of registers that count events, which are occurrences of specific signals related to a processor's function.

Source : http://icl.cs.utk.edu/papi/index.html

Why PAPI ?

- The purpose of the PAPI is to design, standardize and implement a portable API to access the hardware performance monitor counters found on most modern microprocessors.

- PAPI can
  - Provide a solid foundation for cross platform performance analysis tools
  - Characterize application and system workload on the CPU
  - simulate the performance tool development
  - simulate research on more sophisticated feedback driven compilation techniques

Source : http://icl.cs.utk.edu/papi/index.html

Hardware Performance Counters ?

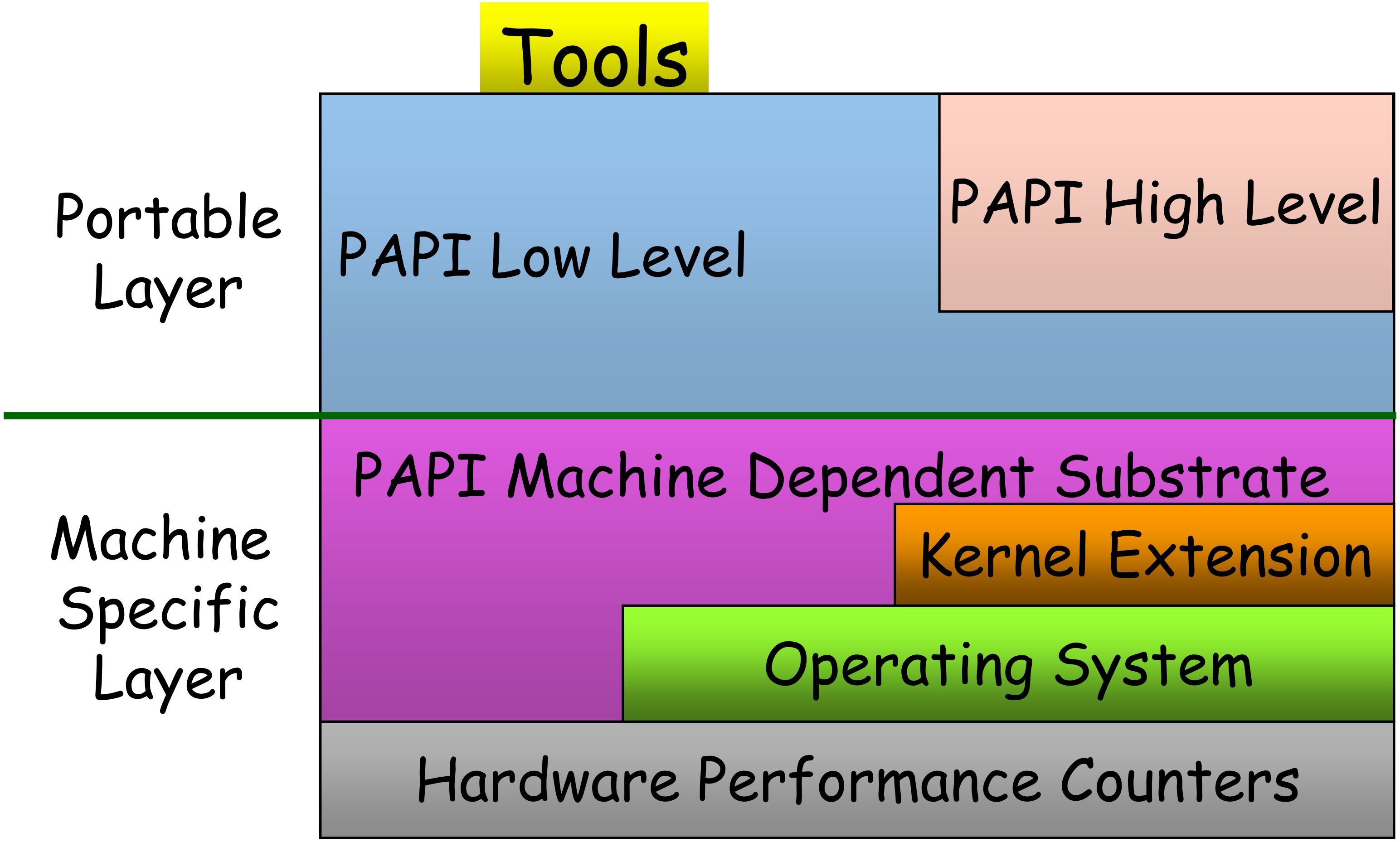
- ❖ Hardware performance counters, or Hardware counters are a set of special-purpose registers built in modern microprocessors to store the counts of hardware-related activities within computer systems.

- ❖ Compared to software profilers, hardware counters provide low-overhead access to a wealth of detailed performance information related to CPU's function units, caches and main memory etc.

Following table shows some examples of hardware counters

Processor	# HC
Intel Pentium	18
IA-64	4
Power 4	8
AMD-Athlon	4

PAPI Architecture



Tools

Portable Layer

PAPI Low Level

PAPI High Level

Machine Specific Layer

PAPI Machine Dependent Substrate

Kernel Extension

Operating System

Hardware Performance Counters

Using PAPI

- Installation of PAPI on Linux-x86 the kernel be patched and recompiled with the `PerfCtr` patch

- Include the header file “**`papi.h`**” for C programs and “**`fpapi.h`**” for Fortran programs

- Compiling with PAPI

  Use **`-L<PAPI PATH>/lib -lpapi`** with the compilation process of the application.

Using PAPI

- Using PAPI in an application typically requires a few steps :
  - Including the event definition
  - Initializing the PAPI lib
  - Setting up the performance counters
  - Linking with PAPI lib

Using PAPI

- Relevant hardware counter data:
  - Total cycles
  - Total instructions
  - Floating point operations
  - Load/store instructions
  - Cycles stalled
    - ✓ waiting for memory access
    - ✓ waiting for resource
  - Conditional branch instructions
    - ✓ executed
    - ✓ mispredicted

Using PAPI : How do I optimize my application ?

1. Optimize compiler switches
2. Integrate libraries
3. Profile
4. Optimize blocks of code that dominate execution time by using hardware counter data to determine why the bottlenecks exist
5. Always examine correctness at every stage!
6. Go To 3…

Using PAPI Utilities

Commands available in bin dir of PAPI Installation:

- **`papi_avail`**
  - It is a utility program that provides availability and detail information for PAPI preset events.
  - Available options are **`-a, -d, -t,`**
    `-e <event_name>`

- **`papi_cost`**
  - Computes execution time cost for basic PAPI operations
  - Computes min, max, mean std. Deviation of execution times for PAPI start/stop pairs and for PAPI reads.

- **`papi_mem_info`**
  - Utility program provides information on the memory architecture

Using PAPI : Events

- ❖ Events are occurrences of specific signals related to a processor's function.

  Ex: cache misses, number of floating point operations

- ❖ Preset events are mappings from symbolic names to machine specific definitions for a particular hardware resource.
  - ❖ Ex: **PAPI_TOT_CYC** (I.e Total Cycles), **PAPI_FLOPS**

- ❖ Native events comprise the set of all events that are countable by the CPU.

Using PAPI : PAPI library Interface

- PAPI provides two APIs to access the underlying counter hardware:
  - The low level interface manages hardware events in user defined groups called EventSets. (PAPI low level)
  - The high level interface simply provides the ability to start, stop and read the counters for a specified list of events. (PAPI high level)

Using PAPI : PAPI library Interface

- C and Fortran bindings
- Java
- Lisp
- Matlab wrappers (Windows only)

Using PAPI : PAPI High level API

- Meant for application programmers wanting coarse-grained measurements.
- Provides the ability to start, stop, and read the counters for a specified list of events
- PAPI High level API are
  - Not tuned for efficiency
  - No guarantee of thread safe
  - Only allows PAPI Preset events
  - Calls the lower level API
  - Can be mixed with low level API

Using PAPI : PAPI High level API

Ex :

- `PAPI_num_counters()`
  - Returns the number of available counters
- `PAPI_start_counters(int *cntrs, int alen)`
  - Start counters
- `PAPI_stop_counters(long_long *vals, int alen)`
  - Stop counters and put counter values in array
- `PAPI_read_counters(long_long *vals, int alen)`
  - Copy counter values into array and reset counters
- `PAPI_flops(float *rtime, float *ptime, long_long *flpins, float *mflops)`
  - Wall clock time, process time, FP ins since start, Mflop/s since last call

PAPI High level API : papi_flops

- **`int PAPI_flops(float *real_time, float *proc_time, long_long *flpins, float *mflops)`**

  - Only two calls needed, PAPI_flops before and after the code you want to monitor
  - real_time is the wall-clock time between the two calls
  - proc_time is the “virtual” time or time the process was actually executing between the two calls (not as fine grained as real_time but better for longer measurements)
  - flpins is the total floating point instructions executed between the two calls
  - mflops is the Mflop/s rate between the two calls

Using PAPI : PAPI Low level API

- It is meant for experienced application programmers and tool developers wanting fine-grained measurement and control of the PAPI interface.
- Unlike the high-level interface, it allows both PAPI preset and native events.
- PAPI library needs to be initialized prior to the first low-level PAPI call

Using PAPI : PAPI Low level API

- Initializing PAPI library

  The PAPI library must be initialized before it can be used. It can be initialized by calling the following low-level function:

  C language :

  ```
  PAPI_library_init(version)
  ```

  Fortran language :

  ```
  PAPIF_library_init(check)
  ```

Using PAPI : PAPI low level API

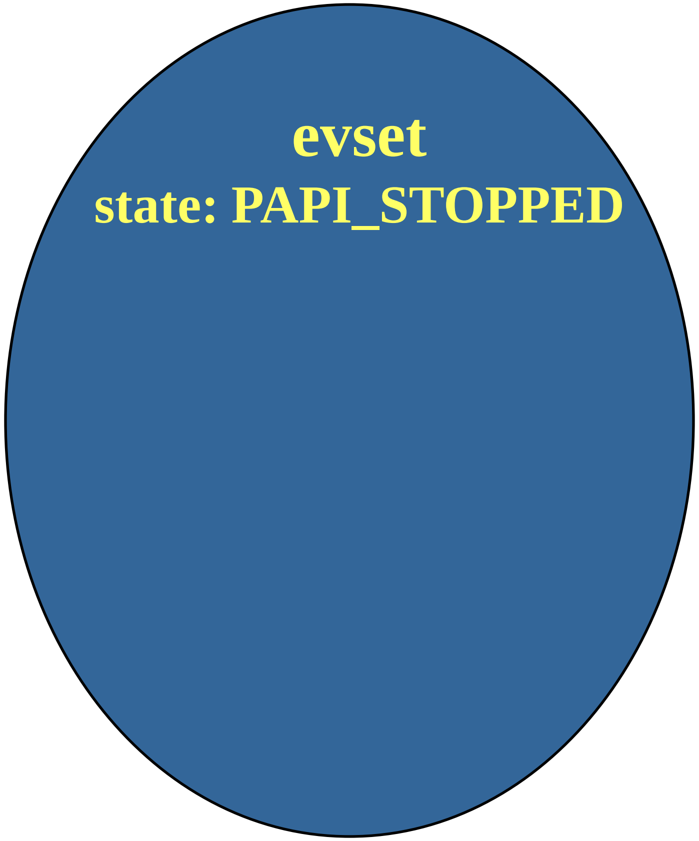
Ex :

- **PAPI_create_eventset(*Eventset)**
  - Creates an event set
- **PAPI_add_event(Eventset, Eventcode)**
  - Hardware events can be added to the eventset
- **PAPI_start(Eventset)**
  - Copy counter values into array
- **PAPI_read(Eventset, *values)**
  - Copy counter values into array and reset counters
- **PAPI_remove_event(Eventset, Eventcode,check)**
  - hardware events can be removed from an event set

PAPI Low level : Simple Example

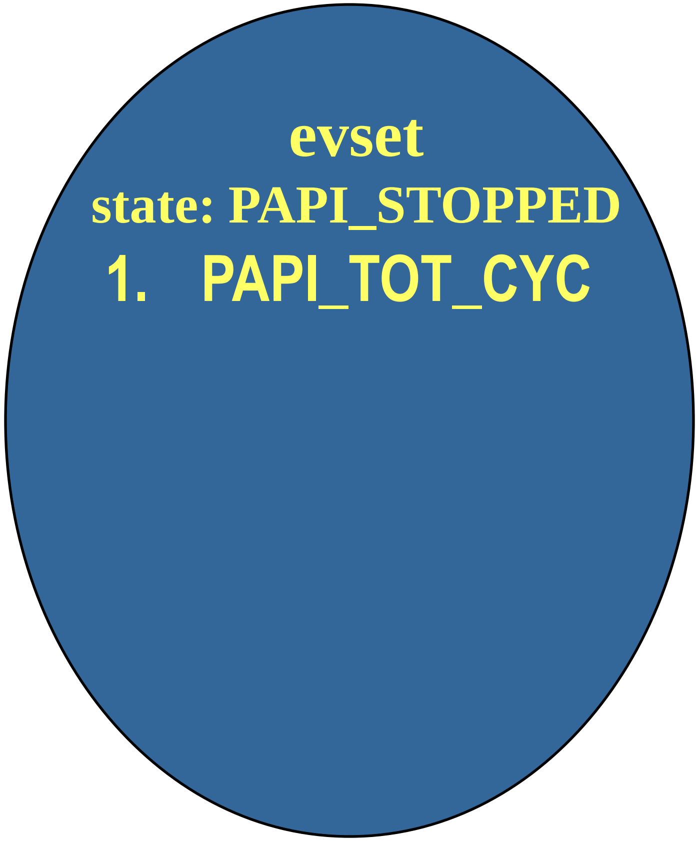
```
#include "papi.h”
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC}, EventSet;
   long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* 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);
```

PAPI Low level : Creating EventSet



```
integer evset, status
integer*8 values(2)
call papif_create_eventset(evset, status)
```

PAPI Low level : Adding Events



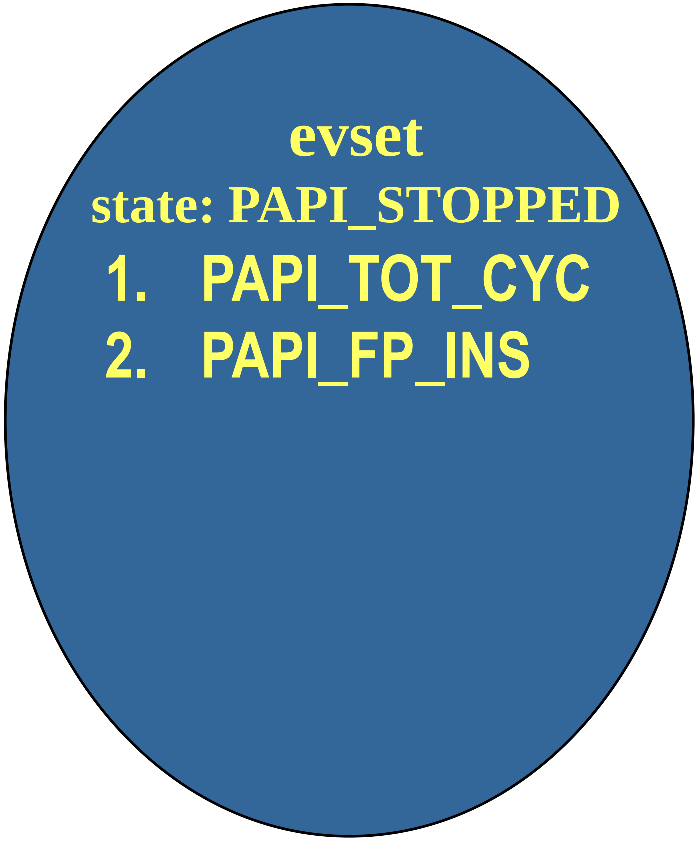
evset

state: PAPI_STOPPED

1. PAPI_TOT_CYC

```
integer evset , status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
```

PAPI Low level : Adding Events



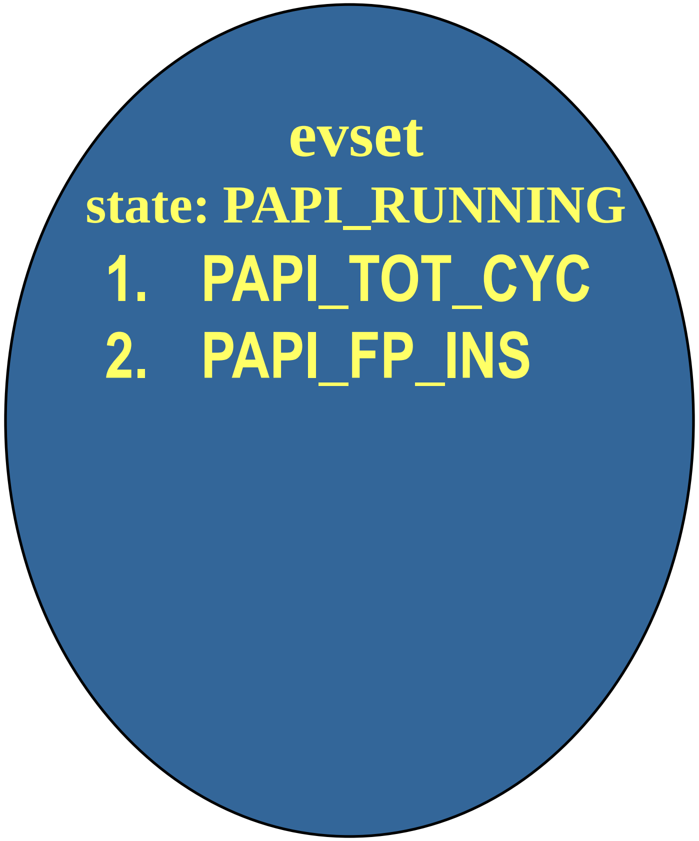
evset

state: PAPI_STOPPED

1. PAPI_TOT_CYC
2. PAPI_FP_INS

```
integer evset , status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
```

PAPI Low level : Starting EventSet



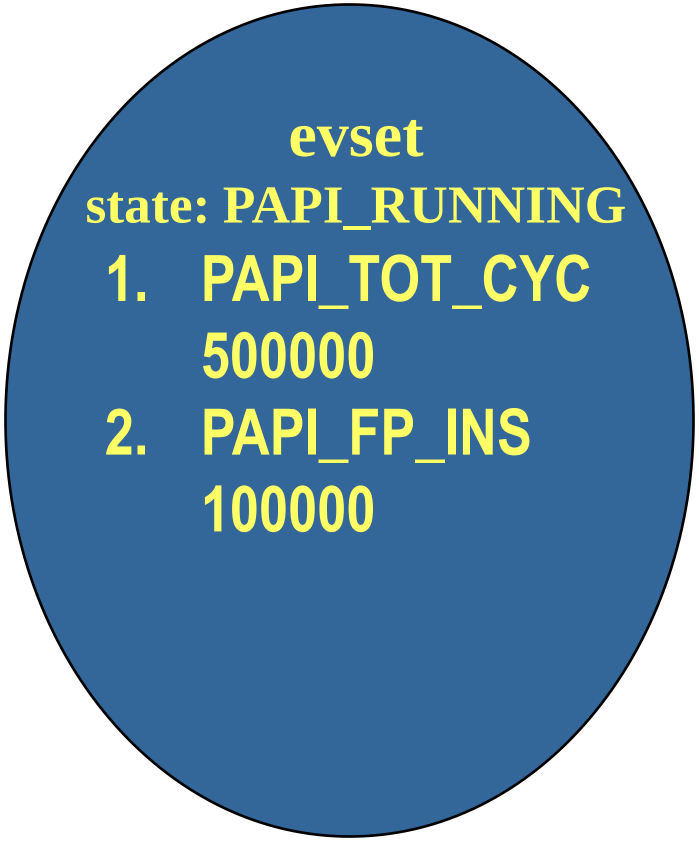
evset

state: PAPI_RUNNING

1. PAPI_TOT_CYC
2. PAPI_FP_INS

```
integer evset , status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
```

PAPI Low level : Reading an EventSet



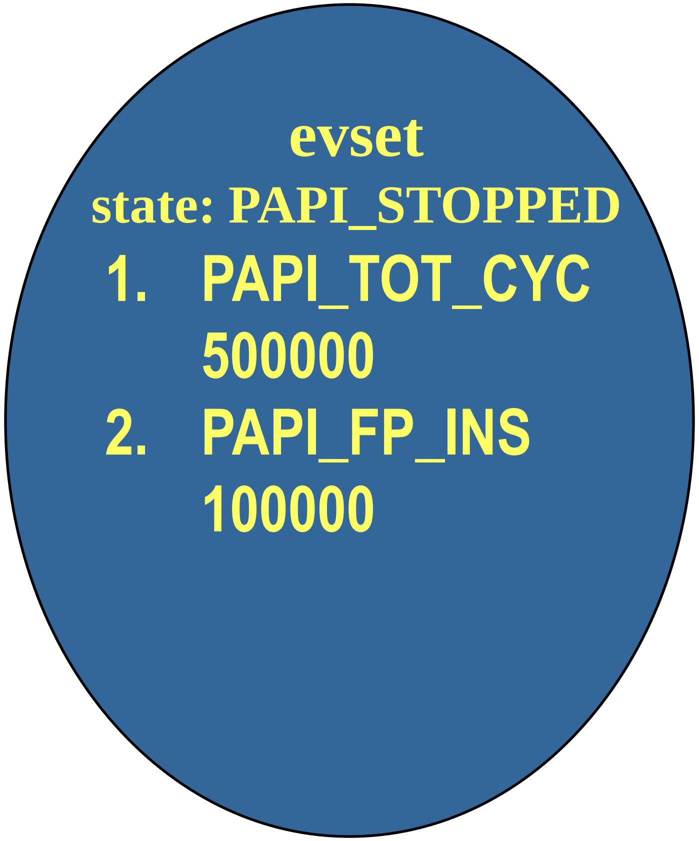
evset

state: PAPI_RUNNING

1. PAPI_TOT_CYC
   500000
2. PAPI_FP_INS
   100000

```
integer evset , status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C do 100000 flops in 500000 cycles
call papif_read(evset, values, status)
C values contains the metrics in order of addition
C values(1) = 500000
C values(2) = 100000
```

PAPI Low level : Stopping an EventSet



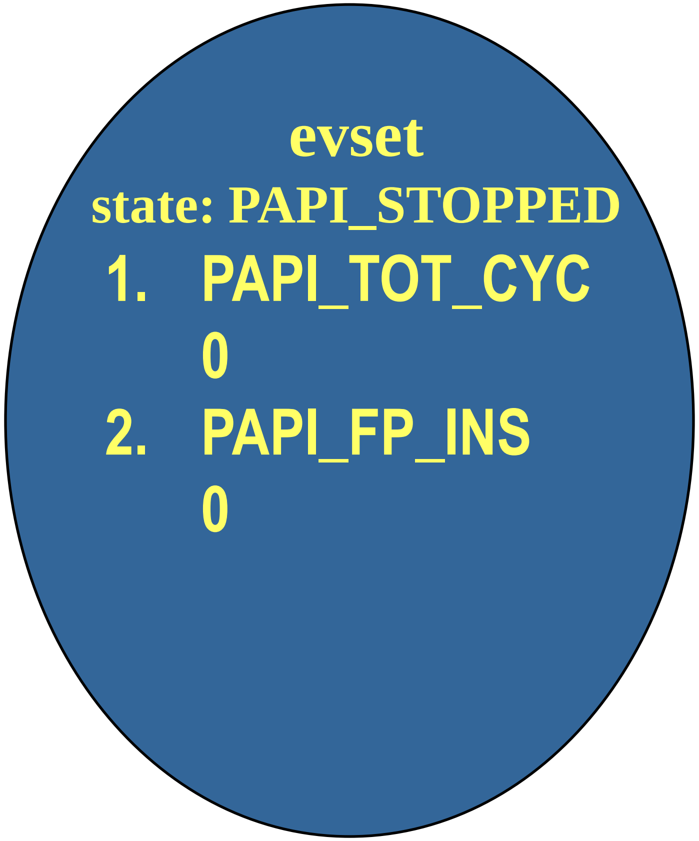
evset

state: PAPI_STOPPED

1. PAPI_TOT_CYC
   500000
2. PAPI_FP_INS
   100000

```
integer evset , status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C do 100000 flops in 500000 cycles
call papif_read(evset, values, status)
C values contains the metrics in order of addition
C values(1) = 500000
C values(2) = 100000
call papif_stop(evset, values, status)
```

PAPI Low level : Resetting an EventSet



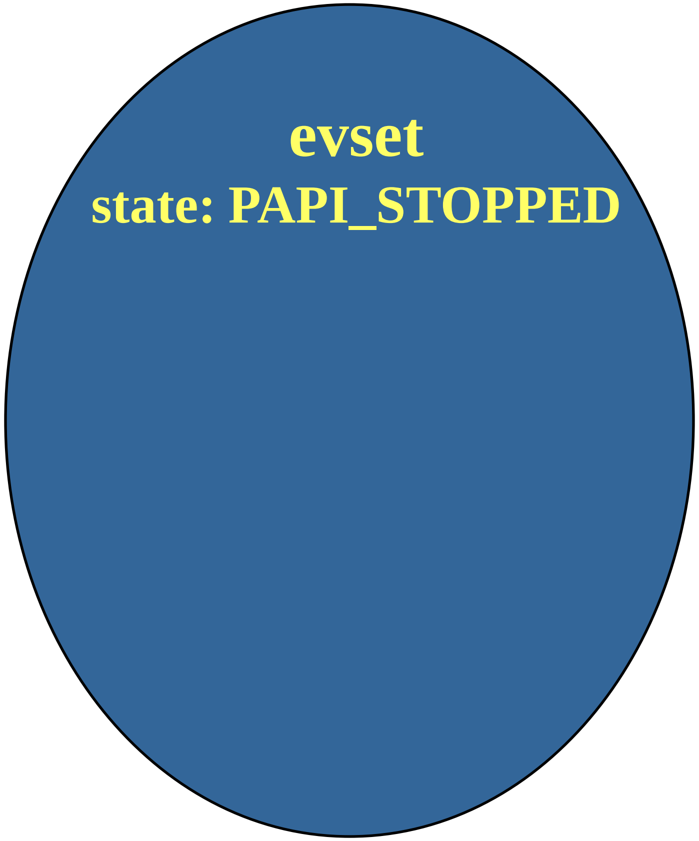
evset

state: PAPI_STOPPED

1. PAPI_TOT_CYC
   0
2. PAPI_FP_INS
   0

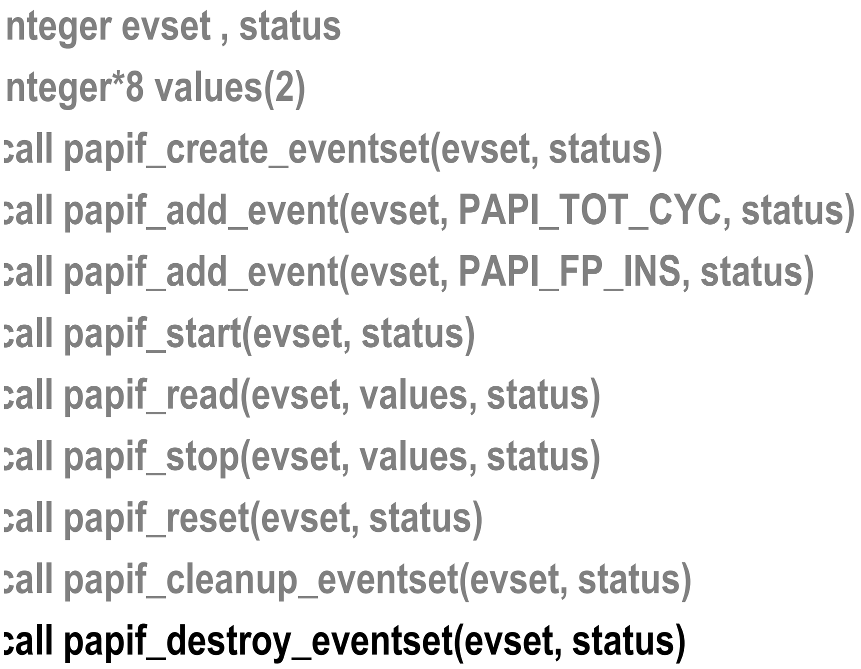
```
integer evset , status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C do 100000 flops in 500000 cycles
call papif_read(evset, values, status)
C values contains the metrics in order of addition
C values(1) = 500000
C values(2) = 100000
call papif_stop(evset, values, status)
C state can be either RUNNING or STOPPED
C to call reset
call papif_reset(evset, status)
```

PAPI Low level : Emptying an EventSet



```
integer evset , status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
call papif_read(evset, values, status)
call papif_stop(evset, values, status)
call papif_reset(evset, status)
call papif_cleanup_eventset(evset, status)
```

PAPI Low level : Freeing an EventSet



```
nteger evset , status
nteger*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
call papif_read(evset, values, status)
call papif_stop(evset, values, status)
call papif_reset(evset, status)
call papif_cleanup_eventset(evset, status)
call papif_destroy_eventset(evset, status)
```

Using PAPI : Error Handling

- All of the functions contained in the PAPI library return standardized error codes from 0 to 14.

  (Refer error codes on the provided web notes.)

- Error codes can be converted to error messages by calling the following low-level functions:

```
PAPI_perror(code, destination, length)
PAPI_strerror(code)
```

(Refer provided web notes for arguments info.)

Using PAPI : Error Handling

Name	**Description**
PAPI_OK	No error
PAPI_EINVAL	Invalid argument
PAPI_ENOMEM	Insufficient memory
PAPI_ESYS	A system/C library call failed. Check errno variable
PAPI_ESBSTR	Substrate returned an error. E.g. unimplemented feature
PAPI_ECLOST	Access to the counters was lost or interrupted
PAPI_EBUG	Internal error
PAPI_ENOEVNT	Hardware event does not exist
PAPI_ECNFLCT	Hardware event exists, but resources are exhausted
PAPI_ENOTRUN	Event or envent set is currently counting
PAPI_EISRUN	Events or event set is currently running
PAPI_ENOEVST	No event set available
PAPI_ENOTPRESET	Argument is not a preset
PAPI_ENOCNTR	Hardware does not support counters
PAPI_EMISC	Any other error occured

Advanced PAPI features : PAPI with Threads

- PAPI must be able to support both explicit (library calls) and implicit (compiler directives) threading models.
- PAPI only supports thread level measurements only if the threads have a scheduling entity known and handled by the operating system's kernel.
- Thread support in the PAPI library can be initialized by calling the function

```
PAPI_thread_init(handle)
```

***handle* --** Pointer to a routine that returns the current thread ID.

Advanced PAPI features : PAPI with Threads

API’s for Threads

- **PAPI_thread_init(handle)**
  - Thread support in PAPI is initialised
- **PAPI_thread_id()**
  - get the thread identifier of the current thread
- **PAPI_get_thr_specific(tag, ptr)**
  - retrieve the pointer from the array with index tag
- **PAPI_set_thr_specific(tag, ptr)**
  - save ptr into an array indexed by tag

Advanced PAPI features : Multiplexing

- ❖ Multiplexing allows more events to be counted than can be supported by the hardware. I.e Multiplexing allows simultaneous use of more counters than are supported by the hardware

- ❖ Multiplex support in the PAPI library can be enabled and initialized by calling the following low-level function

PAPI_muliplex_init()

Note : The above function should be used after calling

PAPI_library_init()

Advanced PAPI features : Multiplexing

- Multiplexing is accomplished through timesharing the counter hardware and extrapolating the results.
- A standard event set can be converted to a multiplexed event set by calling the following low-level function:

  PAPI_set_multiplex(EventSet)

  Note : The above function should be used after calling creating event set.

- Hardware multiplexing is not supported by all platforms.

References

1. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley

2. Butenhof, David R **(1997),** Programming with POSIX Threads , Boston, MA : Addison Wesley Professional

3. Culler, David E., Jaswinder Pal Singh **(1999),** Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann

4. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar **(2003),** Introduction to Parallel computing, Boston, MA : Addison-Wesley

5. Intel Corporation, **(2003),** Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com

6. Shameem Akhter, Jason Roberts **(April 2006),** Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,

7. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell **(1996),** Pthread Programming O'Reilly and Associates, Newton, MA 02164,

8. James Reinders, Intel Threading Building Blocks – (**2007**) , O'REILLY series

9. Laurence T Yang & Minyi Guo (Editors), (**2006**) *High Performance Computing - Paradigm and Infrastructure* Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor

10. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right **(March 2003),** Intel Corporation

References

11. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..

12. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California

13. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.

14. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork

15. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

16. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996),** Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,

17. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann

18. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995

19. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com

20. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995)

21. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999

References

22. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998

23. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991

24. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000)**. Web site URL : http://www.hoard.org/

25. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].

26. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*

27. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**

28. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)”, From the OpenMP web site, URL **: http://www.openmp.org/**

29. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

30. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**

31. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, “Hyperthreading, Technology Architecture and Microarchitecture”, Intel (**2000-01)**

References

- Refer http://icl.cs.utk.edu/papi/index.html for
  - Software download
  - Documentation
  - Third party tools
  - Mailing tools

Thank You

Any questions ?