PAPI

Introduction

dir="auto">Performance Application Programming Interface >(PAPI) is a portable interface to access hardware performance counters (such as instruction counts and cache misses) found in most modern architectures. With the new component framework, PAPI is not limited only to CPU counters, but offers also components for CUDA, network, Infiniband etc.

PAPI provides two levels of interface - a simpler, high level interface and more detailed low level interface.

PAPI can be used with parallel as well as serial programs.

Usage

To use PAPI, load module papi :

```
$ module load papi
```

This will load the default version. Execute module avail papi for a list of installed versions.

Utilites

The bin directory of PAPI (which is automatically added to $PATH upon loading the module) contains various utilites.

papi_avail

Prints which preset events are available on the current CPU. The third column indicated whether the preset event is available on the current CPU.

```
$ papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version : 5.3.2.0
Vendor string and code : GenuineIntel (1)
Model string and code : Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz (45)
CPU Revision : 7.000000
CPUID Info : Family: 6 Model: 45 Stepping: 7
CPU Max Megahertz : 2601
CPU Min Megahertz : 1200
Hdw Threads per core : 1
```

```
Cores per Socket : 8
Sockets : 2
NUMA Nodes : 2
CPUs per Node : 8
Total CPUs : 16
Running in a VM : no
Number Hardware Counters : 11
Max Multiplex Counters : 32
--------------------------------------------------------------------------------
Name Code Avail Deriv Description (Note)
PAPI_L1_DCM 0x80000000 Yes No Level 1 data cache misses
PAPI_L1_ICM 0x80000001 Yes No Level 1 instruction cache misses
PAPI_L2_DCM 0x80000002 Yes Yes Level 2 data cache misses
PAPI_L2_ICM 0x80000003 Yes No Level 2 instruction cache misses
PAPI_L3_DCM 0x80000004 No No Level 3 data cache misses
PAPI_L3_ICM 0x80000005 No No Level 3 instruction cache misses
PAPI_L1_TCM 0x80000006 Yes Yes Level 1 cache misses
PAPI_L2_TCM 0x80000007 Yes No Level 2 cache misses
PAPI_L3_TCM 0x80000008 Yes No Level 3 cache misses
....
```

papi_native_avail

Prints which native events are available on the current CPU.

class="s1">papi_cost

Measures the cost (in cycles) of basic PAPI operations.

papi_mem_info

Prints information about the memory architecture of the current CPU.

PAPI API

PAPI provides two kinds of events:

- **Preset events** is a set of predefined common CPU events, >standardized across platforms.
- **Native events** is a set of all events supported by the current hardware. This is a larger set of features than preset. For other components than CPU, only native events are usually available.

To use PAPI in your application, you need to link the appropriate include file.

- papi.h for C
- f77papi.h for Fortran 77
- f90papi.h for Fortran 90
- fpapi.h for Fortran with preprocessor

The include path is automatically added by papi module to $INCLUDE.

High level API

Please refer to http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:High_Level for a description of the High level API.

Low level API

Please refer to http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Low_Level for a description of the Low level API.

Timers

PAPI provides the most accurate timers the platform can support. See http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Timers

System information

PAPI can be used to query some system infromation, such as CPU name and MHz. See http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:System_Information

Example

The following example prints MFLOPS rate of a naive matrix-matrix multiplication :

```
#include <stdlib.h>
#include <stdio.h>
#include "papi.h"
#define SIZE 1000

int main(int argc, char **argv) {
 float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
 float real_time, proc_time, mflops;
 long long flpins;
 int retval;
```

```
int i,j,k;

/* Initialize the Matrix arrays */
for ( i=0; i<SIZE*SIZE; i++ ){
mresult[0][i] = 0.0;
matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1;
}

/* Setup PAPI library and begin collecting data from the counters */
if((retval=PAPI_flops( &real_time, &proc_time, &flpins, &mflops))<PAPI_OK)
printf("Error!");

/* A naive Matrix-Matrix multiplication */
for (i=0;i<SIZE;i++)
for(j=0;j<SIZE;j++)
for(k=0;k<SIZE;k++)
mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];

/* Collect the data into the variables passed in */
if((retval=PAPI_flops( &real_time, &proc_time, &flpins, &mflops))<PAPI_OK)
printf("Error!");

printf("Real_time:t%fnProc_time:t%fnTotal flpins:t%lldnMFLOPS:tt%fn", real_time, proc_time,
PAPI_shutdown();
return 0;
}
```

Now compile and run the example :

```
$ gcc matrix.c -o matrix -lpapi
$ ./matrix
Real_time: 8.852785
Proc_time: 8.850000
Total flpins: 6012390908
MFLOPS: 679.366211
```

Let's try with optimizations enabled :

```
$ gcc -O3 matrix.c -o matrix -lpapi
$ ./matrix
Real_time: 0.000020
Proc_time: 0.000000
Total flpins: 6
MFLOPS: inf
```

Now we see a seemingly strange result - the multiplication took no time and only 6 floating point instructions were issued. This is because the compiler optimizations have completely removed the multiplication loop, as the result is

actually not used anywhere in the program. We can fix this by adding some "dummy" code at the end of the Matrix-Matrix multiplication routine :

```
for (i=0; i<SIZE;i++)
 for (j=0; j<SIZE; j++)
   if (mresult[i][j] == -1.0) printf("x");
```

Now the compiler won't remove the multiplication loop. (However it is still not that smart to see that the result won't ever be negative). Now run the code again:

```
$ gcc -O3 matrix.c -o matrix -lpapi
$ ./matrix
Real_time: 8.795956
Proc_time: 8.790000
Total flpins: 18700983160
MFLOPS: 2127.529297
```

Intel Xeon Phi

PAPI currently supports only a subset of counters on the Intel Xeon Phi processor compared to Intel Xeon, for example the floating point operations counter is missing.

To use PAPI in Intel Xeon Phi native applications, you need to load module with " -mic" suffix, for example " papi/5.3.2-mic" :

```
$ module load papi/5.3.2-mic
```

Then, compile your application in the following way:

```
$ module load intel
$ icc -mmic -Wl,-rpath,/apps/intel/composer_xe_2013.5.192/compiler/lib/mic matrix-mic.c -o mat
```

To execute the application on MIC, you need to manually set LD_LIBRARY_PATH :

```
$ qsub -q qmic -A NONE-0-0 -I
$ ssh mic0
$ export LD_LIBRARY_PATH=/apps/tools/papi/5.4.0-mic/lib/
$ ./matrix-mic
```

Alternatively, you can link PAPI statically (-static flag), then LD_LIBRARY_PATH does not need to be set.

You can also execute the PAPI tools on MIC :

```
$ /apps/tools/papi/5.4.0-mic/bin/papi_native_avail
```

To use PAPI in offload mode, you need to provide both host and MIC versions of PAPI:

```
$ module load papi/5.4.0
$ icc matrix-offload.c -o matrix-offload -offload-option,mic,compiler,"-L$PAPI_HOME-mic/lib -l
```

References

1. http://icl.cs.utk.edu/papi/ Main project page
2. http://icl.cs.utk.edu/projects/papi/wiki/Main_Page Wiki
3. http://icl.cs.utk.edu/papi/docs/ API Documentation