

---

```

/*****
** Notes on Performance Application Programming Interface
**
** Intended audience: Those who would like to learn more about
** measuring program execution time in modern computer systems.
**
** Used: CPE 631 Advanced Computer Systems and Architectures
**       CPE 619 Modeling and Analysis of Computer and Communication Systems
**
** ver 0.1, Spring 2005 PAPI-2.5.x
** ver 1.0, Spring 2006 PAPI-3.0.7
** ver 2.0, Spring 2007 PAPI-3.5.0 (changed interface)
** ver 3.0, Spring 2011 PAPI-4.1.0
** ver 4.0, Spring 2012 PAPI-5.3.0
** ver 5.0, Spring 2014 PAPI-5.3.0
** ver 6.0, Spring 2016, PAPI-5.5.1
** ver 7.0, Spring 2017, PAPI-5.5.1
** ver 8.0, Spring 2018, PAPI-5.6.1
** ver 9.0, Spring 2021, PAPI-6.0.0.1
** ver 10.0, Spring 2022, PAPI-6.0.0.1 (tested on gauss)
** @Aleksandar Milenkovic, milenkovic@computer.org
*****/

```

# Performance Application Programming Interface

---

PAPI stands for Performance Application Programming Interface. It is a portable and efficient API (Application Programming Interface) to access hardware performance monitoring registers found on most modern microprocessors. The PAPI project is being developed by the ICL at the University of Tennessee.

1. To Learn More: <http://icl.cs.utk.edu/papi/> (ultimate source of information)
2. PAPI documentation and ultimate source of information:  
[http://icl.cs.utk.edu/projects/papi/wiki/Main\\_Page](http://icl.cs.utk.edu/projects/papi/wiki/Main_Page)  
 On the right-hand side you can choose PAPI Topics. Recommended reading includes: Getting Started with PAPI. In addition, PAPI Documentation Archive includes the PAPI Classic User's Guide.
3. To play with PAPI on Linux machine ([gauss.ece.uah.edu](http://gauss.ece.uah.edu)), follow the steps below.
  - a) Login to your account.

## b) Setup your account to have access to PAPI.

- PAPI system variables should be in your path. We will be using the papi 6.0.0.1 located in the /apps/arch/papi-6.0.0.1.
- Add path to papi utility programs and demo examples. They are located in the papi bin directory and share directory. To do that edit the `.bashrc` file in your home directory to include the following line.

```
export PATH=$PATH:/apps/arch/papi-6.0.0.1/bin:/apps/arch/papi-6.0.0.1/share/papi/ctests:/apps/arch/papi-6.0.0.1/share/papi/ftests
```

- Add path to PAPI libraries. To do that edit the `.bashrc` file in your home directory to include the following lines.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/apps/arch/papi-6.0.0.1/lib
export LIBRARY_PATH=$LIBRARY_PATH:/apps/arch/papi-6.0.0.1/lib
```

- Locate the PAPI include files in /apps/arch/papi-6.0.0.1/include.

## c) To see some predefined examples:

```
-bash-4.2$ cd /apps/arch/papi-6.0.0.1/share/papi/ctests
-bash-4.2$ cd /apps/arch/papi-6.0.0.1/share/papi/ftests
```

You can copy these directories and study examples that demonstrate the use of PAPI.

## d) You can run some papi utilities

- To get the current version of the papi:

```
<<~~~~~
(base) [milenska@gauss ~]$ papi_version
PAPI Version: 6.0.0.1
~~~~~>>
```

- To get available events:

```
<<~~~~~
(base) [milenska@gauss ~]$ papi_avail
Available PAPI preset and user defined events plus hardware information.
-----
PAPI version           : 6.0.0.1
Operating system       : Linux 4.18.0-348.7.1.el8_5.x86_64
Vendor string and code : GenuineIntel (1, 0x1)
Model string and code  : Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz (85, 0x55)
CPU revision           : 7.000000
CPUID                  : Family/Model/Stepping 6/85/7, 0x06/0x55/0x07
CPU Max MHz            : 4100
CPU Min MHz            : 1200
Total cores            : 40
SMT threads per core  : 1
```

```

Cores per socket      : 20
Sockets               : 2
Cores per NUMA region : 20
NUMA regions         : 2
Running in a VM      : no
Number Hardware Counters : 11
Max Multiplex Counters : 384
Fast counter read (rdpmc): yes

```

---



---

```

=====
PAPI Preset Events
=====

```

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_CA_SNP	0x80000009	Yes	No	Requests for a snoop
PAPI_CA_SHR	0x8000000a	Yes	No	Requests for exclusive access to shared cache line
PAPI_CA_CLN	0x8000000b	Yes	No	Requests for exclusive access to clean cache line
PAPI_CA_INV	0x8000000c	No	No	Requests for cache line invalidation
PAPI_CA_ITV	0x8000000d	Yes	No	Requests for cache line intervention
PAPI_L3_LDM	0x8000000e	Yes	No	Level 3 load misses
PAPI_L3_STM	0x8000000f	No	No	Level 3 store misses
PAPI_BRU_IDL	0x80000010	No	No	Cycles branch units are idle
PAPI_FXU_IDL	0x80000011	No	No	Cycles integer units are idle
PAPI_FPU_IDL	0x80000012	No	No	Cycles floating point units are idle
PAPI_LSU_IDL	0x80000013	No	No	Cycles load/store units are idle
PAPI_TLB_DM	0x80000014	Yes	Yes	Data translation lookaside buffer misses
PAPI_TLB_IM	0x80000015	Yes	No	Instruction translation lookaside buffer misses
PAPI_TLB_TL	0x80000016	No	No	Total translation lookaside buffer misses
PAPI_L1_LDM	0x80000017	Yes	No	Level 1 load misses
PAPI_L1_STM	0x80000018	Yes	No	Level 1 store misses
PAPI_L2_LDM	0x80000019	Yes	No	Level 2 load misses
PAPI_L2_STM	0x8000001a	Yes	No	Level 2 store misses
PAPI_BTAC_M	0x8000001b	No	No	Branch target address cache misses
PAPI_PRF_DM	0x8000001c	Yes	No	Data prefetch cache misses
PAPI_L3_DCH	0x8000001d	No	No	Level 3 data cache hits
PAPI_TLB_SD	0x8000001e	No	No	Translation lookaside buffer shutdowns
PAPI_CSR_FAL	0x8000001f	No	No	Failed store conditional instructions
PAPI_CSR_SUC	0x80000020	No	No	Successful store conditional instructions
PAPI_CSR_TOT	0x80000021	No	No	Total store conditional instructions
PAPI_MEM_SCY	0x80000022	No	No	Cycles Stalled Waiting for memory accesses
PAPI_MEM_RCY	0x80000023	No	No	Cycles Stalled Waiting for memory Reads
PAPI_MEM_WCY	0x80000024	Yes	No	Cycles Stalled Waiting for memory writes
PAPI_STL_ICY	0x80000025	Yes	No	Cycles with no instruction issue
PAPI_FUL_ICY	0x80000026	Yes	Yes	Cycles with maximum instruction issue
PAPI_STL_CCY	0x80000027	Yes	No	Cycles with no instructions completed
PAPI_FUL_CCY	0x80000028	Yes	No	Cycles with maximum instructions completed
PAPI_HW_INT	0x80000029	No	No	Hardware interrupts
PAPI_BR_UCN	0x8000002a	Yes	Yes	Unconditional branch instructions
PAPI_BR_CN	0x8000002b	Yes	No	Conditional branch instructions
PAPI_BR_TKN	0x8000002c	Yes	Yes	Conditional branch instructions taken
PAPI_BR_NTK	0x8000002d	Yes	No	Conditional branch instructions not taken
PAPI_BR_MSP	0x8000002e	Yes	No	Conditional branch instructions mispredicted
PAPI_BR_PRC	0x8000002f	Yes	Yes	Conditional branch instructions correctly predicted

PAPI_FMA_INS	0x80000030	No	No	FMA instructions completed
PAPI_TOT_IIS	0x80000031	No	No	Instructions issued
PAPI_TOT_INS	0x80000032	Yes	No	Instructions completed
PAPI_INT_INS	0x80000033	No	No	Integer instructions
PAPI_FP_INS	0x80000034	No	No	Floating point instructions
PAPI_LD_INS	0x80000035	Yes	No	Load instructions
PAPI_SR_INS	0x80000036	Yes	No	Store instructions
PAPI_BR_INS	0x80000037	Yes	No	Branch instructions
PAPI_VEC_INS	0x80000038	No	No	Vector/SIMD instructions (could include integer)
PAPI_RES_STL	0x80000039	Yes	No	Cycles stalled on any resource
PAPI_FP_STAL	0x8000003a	No	No	Cycles the FP unit(s) are stalled
PAPI_TOT_CYC	0x8000003b	Yes	No	Total cycles
PAPI_LST_INS	0x8000003c	Yes	Yes	Load/store instructions completed
PAPI_SYC_INS	0x8000003d	No	No	Synchronization instructions completed
PAPI_L1_DCH	0x8000003e	No	No	Level 1 data cache hits
PAPI_L2_DCH	0x8000003f	No	No	Level 2 data cache hits
PAPI_L1_DCA	0x80000040	No	No	Level 1 data cache accesses
PAPI_L2_DCA	0x80000041	Yes	No	Level 2 data cache accesses
PAPI_L3_DCA	0x80000042	Yes	Yes	Level 3 data cache accesses
PAPI_L1_DCR	0x80000043	No	No	Level 1 data cache reads
PAPI_L2_DCR	0x80000044	Yes	No	Level 2 data cache reads
PAPI_L3_DCR	0x80000045	Yes	No	Level 3 data cache reads
PAPI_L1_DCW	0x80000046	No	No	Level 1 data cache writes
PAPI_L2_DCW	0x80000047	Yes	Yes	Level 2 data cache writes
PAPI_L3_DCW	0x80000048	Yes	No	Level 3 data cache writes
PAPI_L1_ICH	0x80000049	No	No	Level 1 instruction cache hits
PAPI_L2_ICH	0x8000004a	Yes	No	Level 2 instruction cache hits
PAPI_L3_ICH	0x8000004b	No	No	Level 3 instruction cache hits
PAPI_L1_ICA	0x8000004c	No	No	Level 1 instruction cache accesses
PAPI_L2_ICA	0x8000004d	Yes	No	Level 2 instruction cache accesses
PAPI_L3_ICA	0x8000004e	Yes	No	Level 3 instruction cache accesses
PAPI_L1_ICR	0x8000004f	No	No	Level 1 instruction cache reads
PAPI_L2_ICR	0x80000050	Yes	No	Level 2 instruction cache reads
PAPI_L3_ICR	0x80000051	Yes	No	Level 3 instruction cache reads
PAPI_L1_ICW	0x80000052	No	No	Level 1 instruction cache writes
PAPI_L2_ICW	0x80000053	No	No	Level 2 instruction cache writes
PAPI_L3_ICW	0x80000054	No	No	Level 3 instruction cache writes
PAPI_L1_TCH	0x80000055	No	No	Level 1 total cache hits
PAPI_L2_TCH	0x80000056	No	No	Level 2 total cache hits
PAPI_L3_TCH	0x80000057	No	No	Level 3 total cache hits
PAPI_L1_TCA	0x80000058	No	No	Level 1 total cache accesses
PAPI_L2_TCA	0x80000059	Yes	Yes	Level 2 total cache accesses
PAPI_L3_TCA	0x8000005a	Yes	No	Level 3 total cache accesses
PAPI_L1_TCR	0x8000005b	No	No	Level 1 total cache reads
PAPI_L2_TCR	0x8000005c	Yes	Yes	Level 2 total cache reads
PAPI_L3_TCR	0x8000005d	Yes	Yes	Level 3 total cache reads
PAPI_L1_TCW	0x8000005e	No	No	Level 1 total cache writes
PAPI_L2_TCW	0x8000005f	Yes	Yes	Level 2 total cache writes
PAPI_L3_TCW	0x80000060	Yes	No	Level 3 total cache writes
PAPI_FML_INS	0x80000061	No	No	Floating point multiply instructions
PAPI_FAD_INS	0x80000062	No	No	Floating point add instructions
PAPI_FDV_INS	0x80000063	No	No	Floating point divide instructions
PAPI_FSQ_INS	0x80000064	No	No	Floating point square root instructions
PAPI_FNV_INS	0x80000065	No	No	Floating point inverse instructions
PAPI_FP_OPS	0x80000066	No	No	Floating point operations
PAPI_SP_OPS	0x80000067	Yes	Yes	Floating point operations; optimized to count scaled single precision vector operations
PAPI_DP_OPS	0x80000068	Yes	Yes	Floating point operations; optimized to count scaled double precision vector operations
PAPI_VEC_SP	0x80000069	Yes	Yes	Single precision vector/SIMD instructions
PAPI_VEC_DP	0x8000006a	Yes	Yes	Double precision vector/SIMD instructions
PAPI_REF_CYC	0x8000006b	Yes	No	Reference clock cycles

-----

Of 108 possible events, 59 are available, of which 18 are derived.  
 ~~~~~>>

e) You can run papi tests from ctest and ftest directories. E.g.,

```
<<~~~~~
(base) [milenska@gauss ~]$ first
Test case 1: Non-overlapping start, stop, read.
-----
Default domain is: 1 (PAPI_DOM_USER)
Default granularity is: 1 (PAPI_GRN_THR)
Using 20000000 iterations of c += a*b
-----
Test type   :          1          2          3          4          5
PAPI_TOT_INS:  200000307  200000287  400000656  600001024  600001024
PAPI_TOT_CYC:  180312849  180170941  360336648  540408903  540408903
-----
Verification:
Row 1 Column 1 at least 20000000
% difference between PAPI_TOT_INS: 1 & 2: 100.00
% difference between PAPI_TOT_CYC 1 & 2: 100.08
Column 1 approximately equals column 2
Column 3 approximately equals 2 * column 2
Column 4 approximately equals 3 * column 2
Column 4 exactly equals column 5
PASSED
~~~~~>>
```

\* To see the source code for this example read first.c, e.g.

```
<<~~~~~
-bash-4.1$ more /apps/arch/papi-6.0.0.1/share/papi/ctests/first.c
~~~~~>>
```

f) To see events supported by your substrate run.

```
<<~~~~~
(base) [milenska@gauss ~]$ papi_native_avail | more
Available native events and hardware information.
-----
PAPI version           : 6.0.0.1
Operating system       : Linux 4.18.0-348.7.1.el8_5.x86_64
Vendor string and code : GenuineIntel (1, 0x1)
Model string and code  : Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz (85, 0x55)
CPU revision           : 7.000000
CPUID                  : Family/Model/Stepping 6/85/7, 0x06/0x55/0x07
CPU Max MHz            : 4100
CPU Min MHz            : 1200
Total cores            : 40
SMT threads per core   : 1
Cores per socket       : 20
Sockets                : 2
Cores per NUMA region  : 20
NUMA regions           : 2
Running in a VM         : no
Number Hardware Counters : 11
Max Multiplex Counters : 384
Fast counter read (rdpmc): yes
-----
=====
```

```

Native Events in Component: perf_event
=====
| ix86arch::UNHALTED_CORE_CYCLES |
|     count core clock cycles whenever the clock signal on the specific |
|     core is running (not halted) |
|     :e=0 |
|     edge level (may require counter-mask >= 1) |
|     :i=0 |
|     invert |
|     :c=0 |
|     counter-mask in range [0-255] |
|     :t=0 |
|     measure any thread |
--More--
~~~~~>>

```

#### 4. To help you get started I prepared a small example

##### a) Copy the example from /apps/arch/arch.tut/test.papi to your home directory

```

<<~~~~~
(base) [milenska@gauss ~]$ cp -R /apps/arch/arch.tut/test.papi .
(base) [milenska@gauss ~]$ cd test.papi/
~~~~~>>

```

##### b) Examine Makefile

```

<<~~~~~
(base) [milenska@gauss ~]$ cat Makefile.arrsum.papi
arrsum_papi: arrsum_papi.o
    gcc -lpapi arrsum_papi.o -o arrsum_papi

arrsum_papi.o: arrsum_papi.c
    gcc -I/apps/arch/papi-6.0.0.1/include -O0 -c arrsum_papi.c

clean:
    rm *.o
    rm arrsum_papi
~~~~~>>

```

\* If you want to learn more about GNU make visit  
<http://www.gnu.org/software/make/manual/make.html>

##### c) Examine the arrsum\_papi.c (sums up elements of an integer array). Observe PAPI events, use of PAPI functions, etc. What metrics do we use in reporting performance? How do we calculate them?

```

<<~~~~~
(base) [milenska@gauss ~]$ more arrsum_papi.c
~~~~~>>

```

##### d) Compile the program.

```

<<~~~~~
(base) [milenska@gauss ~]$ make -f Makefile.arrsum.papi
gcc -I/apps/arch/papi-6.0.0.1/include -O0 -c arrsum_papi.c

```

```
gcc -lpapi arrsum_papi.o -o arrsum_papi
~~~~~>>
```

e) Run the program and study the results (play with different sizes)

```
<<~~~~~
(base) [milenka@gauss test.papi]$ ./arrsum_papi 16384
242201568
TOT_INS for summing up:      114695
TOT_CYC for summing up:      28985
Caclulated CPI: 0.252722
~~~~~>>
```

f) If you want to see the assembly code for this example type in the following:

```
<<~~~~~
(base) [milenka@gauss test.papi]$ gcc -lpapi -S -O0 -I/apps/arch/papi-6.0.0.1/include
-c arrsum_papi.c
~~~~~>>
```

arrsum\_papi.s file will be generated.

g) To probe further:

- Add other events in test program (cycles, cache misses, etc).
- Change compiler optimization (in Makefile replace -O0 with O3)
- Play with other compilers (e.g., Intel's icc).
- Make your own examples

## 5. Using PAPI with parallel programs

To help you get started an example is included in the /apps/arch/arch.tut/test.papi directory. You will find two source files, daxpy.c and daxpy\_omp.c. They implement a serial and an OpenMP version of the double  $a*X + Y$  loop (DAXPY).

- a) Examine the daxpy.c source file. Notice how PAPI instrumentation is controlled by a macro. Observe how we create an event set, start counters, read, and stop counters.

b) Examine Makefile

```
<<~~~~~
(base) [milenka@gauss test.papi]$ cat Makefile.daxpy
all: daxpy.exe daxpy_papi.exe

daxpy.exe: daxpy.o
        gcc daxpy.o -o daxpy.exe

daxpy.o: daxpy.c
        gcc -O2 -c daxpy.c -o daxpy.o

daxpy_papi.exe: daxpy_papi.o
        gcc daxpy_papi.o -o daxpy_papi.exe -lpapi

daxpy_papi.o: daxpy.c
        gcc -Wall -DUSEPAPI -I/apps/arch/papi-6.0.0.1/include -O2 -c daxpy.c -o
daxpy_papi.o

clean:
        rm *.o
```

```
rm daxpy_papi.exe daxpy.exe
```

```
(base) [milenka@gauss test.papi]$ make -f Makefile.daxpy
gcc -O2 -c daxpy.c -o daxpy.o
gcc daxpy.o -o daxpy.exe
gcc -Wall -DUSEPAPI -I/apps/arch/papi-6.0.0.1/include -O2 -c daxpy.c -o daxpy_papi.o
gcc daxpy_papi.o -o daxpy_papi.exe -lpapi
~~~~~>>
```

This makefile creates two executables for the serial `daxpy.c` – one that measures execution time of the critical loop using `clock()` function only and the other that utilizes both the `clock()` function and two PAPI events (`TOT_INS` and `TOT_CYC`).

c) Run the programs as follows.

```
<<~~~~~
(base) [milenka@gauss test.papi]$ ./daxpy_papi.exe 3.14 10000000
Execution time of the daxpy loop: 0.012675 seconds.
20399548.620000
TOT_INS in the daxpy loop:      80001641
TOT_CYC in the daxpy loop:      39154315
Calculated CPI: 0.489419
~~~~~>>
```

What is the loop execution time?

Is it the same for both programs? How many clock cycles does the loop take?

What is the clock cycle time?

Can you derive time from the clock cycles and clock cycle time?

How does it compare with the time from the `clock()` function?

d) Examine the `daxpy_omp.c` source file.

Observe changes relative to the original example in the `daxpy.c` (include `omp.h`; define `PAPI_option_t`, enabling inheriting PAPI events among created threads, for loop parallelization, time measurement using `omp_get_wtime()`, etc).

```
<<~~~~~
(base) [milenka@gauss test.papi]$ more daxpy_omp.c
~~~~~>>
```

e) Compile the program.

```
<<~~~~~
(base) [milenka@gauss test.papi]$ cat Makefile.daxpy_omp
all: daxpy_omp.exe daxpy_omp_papi.exe

daxpy_omp.exe: daxpy_omp.o
    gcc -fopenmp daxpy_omp.o -o daxpy_omp.exe

daxpy_omp.o: daxpy_omp.c
    gcc -O2 -c daxpy_omp.c -o daxpy_omp.o

daxpy_omp_papi.exe: daxpy_omp_papi.o
    gcc -fopenmp daxpy_omp_papi.o -o daxpy_omp_papi.exe -lpapi

daxpy_omp_papi.o: daxpy_omp.c
    gcc -Wall -DUSEPAPI -I/apps/arch/papi-6.0.0.1/include -O2 -fopenmp -c
daxpy_omp.c -o daxpy_omp_papi.o

clean:
```



```
rm *.o
rm daxpy_omp_papi.exe daxpy_omp.exe
```

```
(base) [milenska@gauss test.papi]$ make -f Makefile.daxpy_omp
gcc -O2 -c daxpy_omp.c -o daxpy_omp.o
gcc -fopenmp daxpy_omp.o -o daxpy_omp.exe
gcc -Wall -DUSEPAPI -I/apps/arch/papi-6.0.0.1/include -O2 -fopenmp -c daxpy_omp.c -o
daxpy_omp_papi.o
gcc -fopenmp daxpy_omp_papi.o -o daxpy_omp_papi.exe -lpapi
~~~~~>>
```

Two executables are created: `daxpy_omp.exe` (without PAPI measurements) and `daxpy_omp_papi.exe` (with PAPI measurements).

f) Run the program and study the results.

```
<<~~~~~
(base) [milenska@gauss test.papi]$ ./daxpy_papi.exe 3.14 40000000
Execution time of the daxpy loop: 0.545356 seconds.
688399548.620000
TOT_INS in the daxpy loop: 3200004621
TOT_CYC in the daxpy loop: 1680152297
Calculated CPI: 0.525047
(base) [milenska@gauss test.papi]$ ./daxpy_omp_papi.exe 3.14 40000000
Critical loop execution time: 0.294851 seconds.
688399548.620000
TOT_INS for daxpy: 2806358523
TOT_CYC for daxpy: 1826463421
Calculated CPI: 0.650830
~~~~~>>
```

How do these numbers change with increasing the number of elements in the arrays? Experiment with changing the number of threads? What do you observe?

6. To learn more about

- \* gcc go to: <http://gcc.gnu.org/onlinedocs/gcc/>
- \* GNU make tools: <http://www.gnu.org/software/make/manual/make.html>