



# Foundations of High Performance Computing Prelude

*How to measure the time consumed by your application*

Luca Tornatore – I.N.A.F

# Defining performance in HPC

In your professional life you will have to assess *timing* quite often.

In several cases, the point will be to measure :

- the ***time-to-solution*** that a given code takes to provide a correct result (in general it will be a function of the problem's dimension and of the scalability properties of the code)
- the ***time spent in a code's segment*** if you are profiling it

**“Performance”** is often related to time a code takes to provide the answer, or for a code's segment, to run.

It may also be translated to the amount of operations (Flops, lops, Memory Accesses, cache misses, ...) that have been performed – that relates, although not linearly, to the time it takes to perform them.

NOTE: in this new era, “performance” may refer to other quantities, for instance to the *energy* it takes to arrive at the problem's solution.

# Defining performance in HPC

When “**performance**” is related to *time*, you’re left with the problem of **measuring** the time on the machine you are running on.

So:

- what time do you actually need (or, you have access to)?
- what is the most correct to be used, if many are available ?

# What is “time” ?

Basically, you have access to 3 different types of “time”.

## 1. “wall-clock” time

Basically the same time you can get from the wall-clock in this room.

It is a measure of the “absolute time”.

In **POSIX** systems, it is the amount of the number of seconds elapsed since the start of the Unix epoch at 1 January 1970 00:00:00 UT.

## 2. “system-time”

The amount of time that the whole system spent executing your code. It may include I/O, system calls, etc.

## 3. “process user-time”

The amount of time spent by CPU executing your code’s instructions, strictly speaking.

# What is “time” ?

Basically, you have access to 3 different times:

## 1. “wall-clock” time

Basically the same time you can see on your clock.  
It is a measure of the “absolute time”.  
In **POSIX** systems, it is the amount of time since  
the start of the Unix epoch at 1 January 1970.

## 2. “system-time”

The amount of time that the whole system spends  
may include I/O, system calls, etc.

## 3. “process user-time”

The amount of time spent by CPU  
strictly speaking.

### POSIX systems

POSIX is an ensemble of IEEE standards  
meant to define a standard environment  
and a standard API for applications, as well  
as the applications’ expected behaviour.

It enlarges the C API, for instance, the CLI  
utilities, the shell language and many things.

All **\*NIX** systems are POSIX systems.  
Other compliant systems are

- AIX (*IBM*)
- OSX (*Apple*)
- HP-UX (*HP*)
- Solaris (*Oracle*)

# Where you measure the “time” ?

You can measure all the quoted times :

- ***Outside*** your code
  - you measure the whole code execution
    - you ask the OS to measure the time your code took to execute time:
    - using the `time` command (see `man time`)
    - using `perf` profiler
    - .. discover other ways on your system
- ***Inside*** your code
  - you can measure separate code's section
    - you access system functions to access system's counter

# Where you measure the “time” ?

You can measure all the quoted times :

- ***Outside*** your code
  - you measure the whole code execution
    - you ask the OS to measure the time your code took to execute time:
    - using the `time` command (see `man time`)
    - using `perf` profiler
    - .. discover other ways on your system
- ***Inside*** your code
  - you can measure separate code's section
    - you access system functions to access system's counter

That is the focus for the next few slides

# What type of clock are we looking for ?

- What time do we need ?  
*Real, User, System, ...*
- What precision do we need ?  
*1s, 1ms, 1us, 1ns*
- What wrap-around time do we need ?
- Do we need a *monotonic* clock ?
- Do we need a *portable* function call ?

# How do you measure “time” ?

Baseline: you call the correct system function right before and after the code snippet you’re interested in, and calculate the difference (yes, you’re including the time function’s overhead).

- **gettimeofday( . . )** returns the wall-clock time with  $\mu\text{s}$  precision

Data are given in a `timeval` structure:

```
struct timeval {  
    time_t          tv_sec;    // seconds  
    suseconds_t    tv_usec;   // microseconds  
};
```

- **clock\_t clock()** returns the **user-time + system-time** with  $\mu\text{s}$  precision.

Results must be divided by `CLOCKS_PER_SEC`

# How do you measure “time” ?

- `int clock_gettime(clockid_t clk_id, struct timespec ...)`

`CLOCK_REAL_TIME` system-wide realtime clock;

`CLOCK_MONOTONIC` monotonic time

`CLOCK_PROCESS_CPUTIME` High-resolution **per-process** timing

`CLOCK_THREAD_CPUTIME_ID` high-precision **per-thread** timing

Resolution is **1 ns**

```
struct timespec {  
    time_t      tv_sec;    /* seconds */  
    long        tv_nsec;   /* nanoseconds */  
};
```



32-bits quantity: wraps at 136,19 years

# How do you measure “time” ?

- **int getrusage(int who, struct rusage \*usage)**

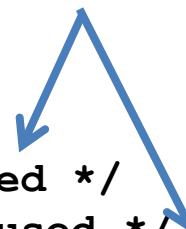
RUSAGE\_SELF process + all threads

RUSAGE\_CHILDREN all the children hierarchy

RUSAGE\_THREAD calling thread

```
struct rusage {  
    struct timeval ru_utime; /* user CPU time used */  
    struct timeval ru_stime; /* system CPU time used */  
    long    ru_maxrss;      /* maximum resident set size */  
    long    ru_ixrss;       /* integral shared memory size */  
    long    ru_idrss;       /* integral unshared data size */  
    long    ru_isrss;       /* integral unshared stack size */  
    long    ru_minflt;     /* page reclaims (soft page faults) */  
    long    ru_majflt;     /* page faults (hard page faults) */  
    long    ru_nswap;       /* swaps */  
    long    ru_inblock;     /* block input operations */  
    long    ru_oublock;     /* block output operations */  
    long    ru_msgsnd;     /* IPC messages sent */  
    long    ru_msgrcv;     /* IPC messages received */  
    long    ru_nsignals;    /* signals received */  
    long    ru_nvcsw;       /* voluntary context switches */  
    long    ru_nivcsw;      /* involuntary context switches */  
};
```

Resolution: 1ns



# In practice..

A possibility on a POSIX system is:

```
#define CPU_TIME (clock_gettime( CLOCK_PROCESS_CPUTIME_ID, &ts ), \
                (double)ts.tv_sec + \
                (double)ts.tv_nsec * 1e-9)

....  
  
Tstart = CPU_TIME ;  
  
// your code segment here  
  
Time = CPU_TIME - Tstart;
```