Time Measurement in C/C++

K. Iglberger, B. Heubeck, C. Jandl

University Erlangen-Nuremberg – System Simulation

December 15th-16th 2008

Meta Programming

Outline

Standard Time Measurement

The time() Function

```
#include <ctime>
time_t time( time_t *time );
```

- returns the current calendar time or -1 if there is an error
- if the argument `time` is given, then the current time is stored in `time`
- only measures the time up to seconds

Standard Time Measurement

The clock() Function

```
#include <ctime>
clock_t clock( void );
```

- returns the processes CPU time (time since the program started)
- returns -1 if that information is not available
- convertion to seconds by division by `CLOCKS_PER_SEC`
- Note: if your compiler is POSIX compliant, then `CLOCKS_PER_SEC` is always defined as 1.000.000
- only measures the time up to seconds

Standard Time Measurement

```
#include <ctime>

// Time stamp before the computations
clock_t start = clock();

... /* Computations to be measured */

// Time stamp after the computations
clock_t end = clock();

double cpu_time = static_cast<double>( end - start ) /
                  CLOCKS_PER_SEC;
```

PAPI Time Measurement

```
int main() {
   ...

   // Time stamp before the computations
   long_long start_usec( PAPI_get_real_usec() );

   // Computations to be measured
   ...

   // Time stamp after the computations
   long_long end_usec( PAPI_get_real_usec() );

   ...
}
```

PAPI Time Measurement

Advantages

- Measures the time up to micro-seconds
- Easy to use ...

Disadvantages

- ... but PAPI library must be available!
- Portability

Native Linux Time Measurement

Measurement of Wall Clock Time

```
#include <sys/time.h>
#include <sys/types.h>

struct timeval tp;
double sec, usec, start, end;

// Time stamp before the computations
gettimeofday( &tp, NULL );
sec   = static_cast<double>( tp.tv_sec  );
usec  = static_cast<double>( tp.tv_usec )/1E6;
start = sec + usec;

// Computations to be measured
...
```

Native Linux Time Measurement

Measurement of Wall Clock Time

```
...

// Time stamp after the computations
gettimeofday( &tp, NULL );
sec  = static_cast<double>( tp.tv_sec  );
usec = static_cast<double>( tp.tv_usec )/1E6;
end  = sec + usec;

// Time calculation (in seconds)
double time = end - start;
```

Native Linux Time Measurement

Measurement of the CPU Time

```
#include <sys/resource.h>
#include <sys/types.h>

struct rusage ruse;
double sec, usec, start, end;

// Time stamp before the computations
getrusage( RUSAGE_SELF, &ruse );
sec   = static_cast<double>( ruse.ru_utime.tv_sec  );
usec  = static_cast<double>( ruse.ru_utime.tv_usec )/1E6;
start = sec + usec;

// Computations to be measured
...
```

Native Linux Time Measurement

Measurement of the CPU Time

```
...

// Time stamp after the computations
getrusage( RUSAGE_SELF, &ruse );
sec  = static_cast<double>( ruse.ru_utime.tv_sec  );
usec = static_cast<double>( ruse.ru_utime.tv_usec )/1E6;
end  = sec + usec;

// Time calculation (in seconds)
double time = end - start;
```

Native Linux Time Measurement

Advantages

- No library required
- Fairly easy to use ...

Disadvantages

- ... but not portable!
- Measures the time only up to seconds

OpenMP Time Measurement

Function signature

```
#include <omp.h>
double omp_get_wtime( void );
```

Measurement of the Wall Clock Time

```
#pragma omp parallel
   {
      // Starting the time measurement
      double start = omp_get_wtime();

      // Computations to be measured
      ...

      // Measuring the elapsed time
      double end = omp_get_wtime();
```

OpenMP Time Measurement

Advantages

- Easy to use!
- Portable (if the compiler supports OpenMP 2.0)

Disadvantages

- Designed to be "per thread" times
- Measures the time only up to seconds

Boost Time Measurement

```
#include <boost/timer.hpp>

...

// Starting the time measurement
boost::timer t;

// Computations to be measured
...

// Measuring the elapsed time
double time = t.elapsed();

...
```

Boost Time Measurement

Advantages

- Very easy to use!
- Portable

Disadvantages

- Measures not accurate up to micro-seconds
- Boost required