Tools of high performance computing 2024

Exercise 2

Return by Wednesday 29.1.2024 23:59 to Moodle.
Exercise session: Friday 30.1.2024

Problem 1. *(6 points)*

When in a program an array is accessed past the upper limit of its index it might go unnoticed. For example (on the left in Fortran and on the right in C):

```
integer :: a(10)                    int a[10];
…                                   …
print *,a(11)                       printf("%d\n",a[10]);
```

There is, however a limit beyond which the array can not be accessed without the program crashing. Investigate this limit by defining the array a as shown above and trying to access elements with larger and larger index. Run the program many times. Comment on your results.

Problem 2. *(6 points)*

Run the program of problem 1 under debugger so that it crashes. After crash, in the debugger, print the array content and the value of index which was last used.

Problem 3. *(6 points)*

Write a program (Fortran or C) that computes the sum

$$\sum_{k=0}^{n} \exp(\sin(k/1000000)) \,.$$

Measure the CPU time used in the computation using 32, 64, and 128-bit[1] floating point numbers[2]. In order to get enough measurable CPU time use large enough values of n. Ascertain that you are really using the mentioned kinds of floating point numbers. To measure CPU time, use the corresponding Fortran or C/C++ functions (see next page). Remember to print the result in the end, so that the compiler doesn't optimize away the loop. Comment on your results.

Problem 4. *(6 points)*

Unpack the attached package `simpleprofiling.zip`. Use either the Fortran or C version in this problem. By using the profiler, investigate what are the times spent in the two innermost loops in the code (the lines `sum1=sum1+…` and

1 Seems that at least GNU ant Intel C/C++ compilers support 128-bit float as `__float128` and not as `long double`.

2 If you are using C/C++ remember to use the math functions specific for the floating point kind: `expf`, `sinf` for `float`, `exp`, `sin` for `double` and `expl`, `sinl` for `__float128`.

`sum2=sum2+…`). Remember to compile the code with options

```
-O0 -pg -g -static
```
[1]

Comment on your results.

Measuring CPU time (problem 3)

Fortran

```
…
real :: t1,t2
…
call cpu_time(t1)
! do the computations
call cpu_time(t2)
print *,'CPU time in seconds: ',t2-t1
…
```

C/C++

```
…
#include <time.h>
…
clock_t t1,t2;
double cputime;
…
t1=clock();
// do the computations
t2=clock();
cputime=(double)(t2-t1)/CLOCKS_PER_SEC;
printf("CPU time in seconds: %g\n",cputime);
…
```

1 The last option links the executable statically; i.e., includes all libraries in the executable file. This way you also get profiling information on the library functions; in this case math functions exp, sin and cos.