ASTR615 Fall 2015 Problem Set #2

Due Mon Oct 5th, 2015

Topics for this problem set include benchmarking, debugging, parallelization and round-off errors.

1. This problem set builds on homework #1 and is designed to familiarize you with arrays, debugging and OpenMP parallelization.

   Hints: In class I have only mentioned quickly the procedure of parallelization using OpenMP. Look at some examples and tutorials on how to use OpenMP to parallelize a code, available on the class web-page. OpenMP allows you to add extra commands to the source code with directives to parallelize loops on a shared memory computers (for example a desktop with multi-processors and/or CPU with multi-cores). When you compile the code you need to use the option -openmp, or equivalent (if you use gcc is -fopenmp: look at the manual page of the compiler you use). Without the openmp option, parallelization directives are treated as comments by the compiler. You need access to a desktop with $N \geq 2$ CPUs or cores. Your laptop may have a dual core processor, galileo.astro.umd.edu (my desktop computer) has 2 quad cores (2x4=8 cores). Try to use the computer that has the largest number of cores! On linux machines you can check how may cores are available on a host by viewing the file /proc/cpuinfo. Have fun, and start early!

   (a) Insert in the benchmarking code in homework #1 the handler to detect floating point exceptions (FPEs) as discussed in class (see C examples). Re-run your code in the debugger (remember to compile with $-g$ option) and increase the number of operations n until a FPE is detected. Explain what happens in the context of data representation in computers (integers and floating points).

   (b) After simplifying the math operation in one of the loops to make it easily parallelizable: e.g., calculate n times `pow(1.1,0.5)`; parallelize your code using OpenMP directives and run the parallel code on 1, 2, 3, or more cores. Plot the execution time t as a function of the number of cores N, or $t \times N$ vs N. Does the speedup scale linearly with the number of cores? What is the loss of speed due to OpenMP overhead?

   (c) Modify your benchmark code as follows. Create a 2-dimensional array $A(N, N)$ and add to each element of the array a constant floating point number of your choice. After compiling without optimizations, time the execution in two cases: (i) with the fast changing index of the array being the columns, and (ii) being the rows. Do this for a range of values of N: for N below a critical number there should be little difference between cases (i) and (ii).

      i. First define the array as a static array (e.g., double A[N][N];). How large can N be before you encounter a problem at runtime ? Explain what happens and how can you increase N further.

      ii. Next, use dynamical allocation of memory (with malloc). Knowing the difference in execution times between case (i) and (ii) for a set of N values, given the CPU clock speed, can you estimate roughly the size of the cache and/or the hit rate? (*Hint: for the hit rate you need to know the latency of the RAM. If you do not know what it is, assume 10 nsec, typical for DDR3 RAM*) Guess what is the largest value of N you can use on your computer, explaining your reasoning.

iii. Finally, compile with -O2 optimization and compare the execution times for the same set of N values, or for a value of N of your choice larger than the critical value. Comment on the results.

2. Write a program that uses the quadratic formula to compute both roots of

$$x^2 - 4999x + 1 = 0$$

in single precision. The program should also recompute the smaller root from the larger, using the fact that the product of the roots must equal 1 in this case. Explain any discrepancy. Which method is preferable? What happens when you repeat this exercise in double precision?