CISC 811: High Performance Computing, Assignment 1

Set January 15th due Jan 29th
Download the LINPACK benchmark from the class website:
`http://www.astro.queensu.ca/~thacker/new/teaching/811/811.html`,
you can use either the C or FORTRAN versions, although the C version reports computational speed in terms of kFlops rather than MFlops. Note, there is no need to unroll the loops at the preprocessor level in the C version. If you do not have access to a workstation with the gprof and gcov profiling tools please email me.

Q1. (a) Compile the code on your workstation (or a machine with a C or FORTRAN compiler) using no optimization flags. Execute the code - what is the MFlop rating returned? What CPU did you execute on?
(b) Now recompile with a `-O2` compilation flag (for optimization) and run the benchmark again. What MFlop rating is returned now and how much higher is it than the previous result?
(c) What additional speed gains can you get beyond -O2? Which compiler flags were used and what optimizations were invoked by using those flags?

Recompile the LINPACK code to include gprof profiling support.

Q2. (a) Execute the code and use gprof to print out a flat profile. Which routine takes the most execution time, and what percentage of the total execution time is that? If you wanted to optimize the code which routines would you focus on?
(b) Use gprof to print out a line-by-line profile. Which line consumes most of the execution time? How well does this particular benchmark relate to the 80:20 rule?
(c) Use gprof to print out the call graph profile. Which subroutine has the most children?

Based upon the profiles (from the function level and line-by-line analysis) it should be clear it is actually fairly hard to optimize the LINPACK benchmark at the C/FORTRAN language level. Vendors typically release linear algebra subroutines that have been optimized at the assembly language level which can be up to twice as fast as routines written in C/FORTRAN.

Recompile the LINPACK code to include gcov coverage analysis.

Q3. (a) What is the coverage value for the LINPACK benchmark? How many times is the line that consumes most of the execution time (Q2 b) actually executed?
(b) What percentage of branches are executed? What percentage of branches are executed at least once?

The importance of linear algebra in numerical analysis makes LINPACK an important benchmark. However, that has not stopped other people attempting to come up with benchmarks that more fully investigate machine architecture. One example is a benchmark called HINT, originally written by John Gustafson, formerly of Ames Lab. The HINT benchmark webpage is now `http://hint.byu.edu/`.

Download the HINT source from the course website, (note that only a C version of this code is available). Compile the benchmark to work at 64 bit precision (`DOUBLE`) using `-O2` optimization.

Execute the benchmark on the same machine you ran the LINPACK benchmark. In a nutshell the HINT benchmark is calculating the area under a graph to higher and higher precision. It requires more memory as higher precision is reached, which will tax different layers of the memory hierarchy. Rather than one number, it will produce a series of numbers including the time taken, the memory used and an indicator of the improvement in the accuracy of calculation per second (QUIPs). Roughly speaking QUIPs tells you about the computational efficiency but it does not necessarily map to MFlops in a consistent way.

Q4. (a) Plot up QUIPs versus the problem size. There should be noticeable plateaus in the QUIPs value for different problem sizes.
(b) Describe how the graph relates to the memory design of the computer you are working on.

For the MFlop ratings for LINPACK we only considered one size of problem.

(c) Alter the LINPACK code to run problem sizes of n=50 to 1000 (1000 is the default) in steps of 10. Note for the smaller problem sizes the timing granularity is too coarse to measure speed and you will have to create a new version of the benchmark with timing around multiple calls of the subroutines (and take an average).
(d) Plot up the Mflop value versus problem size - how much difference in speed is there between the smallest problem size and the largest?
(e) Compare and contrast the performance of the HINT benchmark for different problem sizes with the graph of the LINPACK benchmark you just derived. Explain why the HINT benchmark may not have caught on in the way that was hoped.

Q5. (a) From the top500 website describe what R_{max}, R_{peak}, N_{max} and $N_{1/2}$ stand for.
(b) What are the most significant architectural trends for the last 10 years?
(c) What is the (average) doubling time for performance of the fastest machine on the top500 and when can we expect the first Petaflop machine to be built?
(d) A recent review of trends in currents CPUs suggests that the doubling time for processor performance is now around 4 years (the increase in transistor density seen recently has come from increased memory on the chip rather than more transistors devoted to logic operations). If the aggregate performance of a machine on the top500 is given by the single CPU speed multiplied by the number of CPUs, what is the average doubling time for the number of CPUs in a system on the top500?
(e) Suppose all you cared about was your top500 rating, what kind of machine would you build to optimize the price-performance?