Optimization Lab

Goals:

- See how compiler options help you optimize the performance of hand-coded routines.
- See how performance can be improved by calling numerical libraries.

You will be working with three different versions of a code to solve a system of linear equations via LU factorization. The tarball contains all three codes, together with a makefile to compile them and a script to submit them to the scheduler. Embedded in the script are instructions that time each code and put the timing data into an output file.

Unpack the source code:

```
cd ~
tar xvfz ~tg459572/LABS/ludecomp.tgz
```

The directory ludecomp has three code versions:

- nr.c – uses code copied from Numerical Recipes. It does not use any external libraries.
- gsl.c – calls the GNU Scientific Library. You need to use the module command to link and load this library in the TACC environment.
- lapack.c – calls the standard interface to LAPACK. This call may be linked against any compatible, optimized library that performs linear algebra. We will be using Intel's Math Kernel Library (MKL), which comes with the Intel compilers. The needed modules are already loaded by default on Stampede.

The Makefile has many targets. Here are the two you will want to use the most:

- make - This makes all three versions of the program: nr, gsl, and lapack. If make fails, it is likely because it wants libraries that are not currently loaded.
- make clean - This deletes all binaries you compiled for a clean start if you change your build procedure significantly.

To get started on Stampede, here are the steps to follow:

1. Add GSL to the currently-loaded modules: module load gsl
2. In the directory ~/ludecomp, type: make
3. Submit the job.sh script: sbatch job.sh
4. View the results of your runs: less results.txt

Everybody's results will be recorded on The Eric Chen Scoreboard, http://consultrh5.cac.cornell.edu/intro_to_ranger/ .

Review the three codes and evaluate each based on these criteria:

1. How many lines of code did it take to implement the solution?
2. For each version, how hard would it be to swap in a different algorithm by, for instance, substituting an iterative solver, or using a sparse-matrix solver?
3. Can any of these codes run multithreaded? Can they run distributed, using MPI? You may need to Google the libraries to figure this out.

We've already seen what the compiler can do with -g, the debug suite of options. Next let's try the optimization options, to assess how they affect running times.

1. Edit the first few lines of the Makefile to add some compiler flags. Some possible FFLAGS are listed below.
2. Compile the three codes: make
3. Submit the codes to the scheduler: sbatch job.sh
4. Again examine results.txt.
5. Try some other choices of compiler and optimizations and see what is fastest. For the codes that call libraries, how does your choice of options affect the performance? (Remember, you're not compiling the libraries!)

Here are some compiler options to try (recommended ones in bold):

-O3 - Tries harder to optimize code compared to -O2, but the code may not be faster, and the output may no longer be correct (remember to check).

-ipo - Creates inter-procedural optimizations.

-xhost - Compiles for the architecture of the compiler's host, which on Stampede is the same for all nodes; includes SSE and AVX instructions for the vector unit.

-vec_report[0|..|5] - Controls the amount of vectorizer diagnostic information.

-fast - Includes: -ipo, -O2, -static … [DO NOT USE, static load means every process must load its own copy of instructions from libs, no code sharing]

-g - Produces better debugging information.

-fp-stack-check - Catches errors in the floating point stack, at the cost of speed.

-openmp - Enables OpenMP directives.

-openmp_report[0|1|2] – Sets OpenMP parallelizer diagnostic level.

Extra credit: The MKL library has built-in support for OpenMP multithreading. To enable it, you simply set the following environment variable:

```
export OMP_NUM_THREADS=8
```

Try this! Uncomment the above line in job.sh, and submit the job. Does the time improve or not, when compared to leaving this variable unset? What if you try a different number of threads? Remember, Stampede nodes have 2 processors and a total of 16 cores. (Note, the default value of OMP_NUM_THREADS is 1, so that when 16 MPI processes share a node they won't create chaos by forking 16xN threads.)