Project One — Pthreads Experimentation: Understanding Locality, Load Balancing, and Synchronization Effects

Assigned: Thursday, January 22nd, 2015
Pre-Assignment Due: In class, Thursday, January 29th, 2015
GE Pthreads Due: In class, Tuesday, February 3rd, 2015

The goal of this assignment is to understand the interplay among load balancing, locality, true and false data communication, and coordination, in the steps for parallelization: decomposition, assignment, orchestration, and mapping.

You will be working with an application that exhibits data parallelism: Gaussian Elimination, and in the context of pthreads. Gaussian Elimination is a classical method for reducing an arbitrary matrix into an equivalent upper-diagonal matrix. It can be used as the first step in solving a system of linear equations. The second step is then a back-solver, in which the remaining linear equations are solved one-by-one starting from the bottom.

In this problem, however, we will only be concerned with the first part, the gaussian elimination step. We will look at a particular version, namely Gaussian with partial pivoting, a numerically more stable version of the algorithm. A program implementing the sequential version of the algorithm is available to you. Your task is to create a parallel version of this program using pthreads.

You should hand in a working and documented pthreads version of the program. In addition, you should provide a basic correctness argument for your solution (arguing that the relevant dependences and other issues are taken care of by proper synchronization/communication). You should also describe some of the different versions of your program that you tried, what performance results you got out of them (including what your experimental environment was), and why you think your current version is reasonably efficient (or what more you could do to make it more efficient). To make this more concrete, I will expect that you implement at least two parallelization strategies and/or use two different synchronization primitives and compare and contrast them (explain how they interact with the underlying environment).

An example pthreads implementation of an application (SOR) is available in `/u/cs(2 or 4)58/apps`. The sequential version of gauss is available in `/u/cs(2 or 4)58/apps/gauss`. The example Makefiles should allow you to work on all available types of machines. Please do report any problems and fixes to us/the discussion board so that all can benefit.

One aspect of this work is an understanding of the influence of the underlying architecture on your performance, with well-documented timing results. You should pay attention to providing and analyzing performance on at least two platforms. Measure the time required to solve an $n \times n$ system for $n = 128, 256, 512, 1024$, and 2048, on one to as many processors as you have available on your platform. Make sure to parameterize your program for both the problem size and the number of processors so as to avoid having to recompile for every run. In order to ensure a common base, one of these platforms must be one of the 3 cycle (cycle1, cycle2, cycle3) machines (on both the undergraduate and graduate side).

You should put your solutions in a directory called `cs458_proj1_pthreads` under your home directory. Include a pdf README file with the basic correctness argument and the performance in graphical format. Please use the `TURN_IN` script (in /u/cs(2 or 4)58/bin) to turn in your directory.

Pre-Assignment: So that you get started, your first task is to compile and run the sample program supplied — sor. Please choose a platform, generate results for a range of number of processors for a single input data set, and plot these numbers in the form of a graph. Annotate the graph to indicate the hardware platform, compiler options, and input data size (as outlined below).

Deadlines: The deadlines are listed above. If you haven't already done so, your first task (TODAY) should be to make sure you get yourself an account on either the graduate or undergraduate network.

Notes:

- Useful commands - `/proc/cpuinfo` should give you useful processor information on the linux platforms.
- The example sor application you generated speedup charts for shows how to insert timing tests into your code, including the fact that it is best to make sure all processes are in lockstep before starting timing, by inserting a `barrier`. In addition, code and/or instructions for accessing the high resolution timers on most of the machines is provided and accessible via /u/cs (2 or 4)58/hrTimer/.
- To get accurate timings, you'll need to make sure you have exclusive use of the processor(s). A useful command to determine if there are other users is `top`.
- The cycle (1,2,3) machines on both the graduate and undergraduate networks are multiprocessor machines that you can use for this purpose. We will also send out information on other possible machines you might use. Please use /proc/cpuinfo to find out more about the machines.
- On the graduate side, all the machines you will be using are research machines, which will mean that research use may receive higher priority from time to time, resulting in restricted access to the machine for periods of time. Any misuse can result in revocation of your utilization privileges.
- Make sure to kill all your processes when you are done, and before logging out of the machines. Check using `ps -Af` to ensure that you don't leave any run-away processes behind.

Here are some guidelines for your reports that you should follow. You should include -

- A summary description of the sequential Gaussian Elimination
- Any analysis including profiling to determine where the majority of the time is spent
- Your parallelization strategies.
- Correctness of your parallel implementations (make sure it works for an input matrix other than the one provided as well and indicate how you tested to determine correctness)
- Experimental results -
  - Environment specification - hardware (include processor speed and type, cache size, memory organization, OS), and compiler and optimization flags
  - Application settings - input matrix, problem size
  - Actual performance numbers - include and compare against the best sequential times you could obtain for each problem size and number of processors
  - Analysis of your results - you should use the fine-grain timers to help determine where time is being spent/do a breakdown of the time.