EECS 570 Fall 2003

Homework #1

Due 2:40 PM Wednesday Oct. 1

Read Problem 2.7 in the text. Then do the following problems:

1. (5 pts) Problem 2.7a. Draw a diagram similar to Figure 2.8 for a 4x4 matrix. Just show the dependences within one iteration of the outermost \((k)\) loop; there is no need to show the dependences across iterations.

2. (5 pts) Draw a dataflow diagram for a sample iteration.

3. (20 pts) Problem 3.8b & c. That is, analyze the load balance and communication-to-computation ratio for both the block (contiguous chunks) and cyclic (interleaved) row assignments. Provide equations in terms of \(n\), \(p\), and \(k\).

4. (5 pts) What are the pros and cons of using an element-based rather than a row-based decomposition? Specifically state which decomposition you believe would have higher performance.

5. (65 pts) Implement a shared-address-space version of the pipelined algorithm (see description of 2.9) using Pthreads. Use a row-based decomposition and the assignment that looks better to you based on your answer to the previous problem. (See the notes below for more information.) Run this version on the CAC’s SGI Origin and measure the time taken by each phase (input, elimination, back-substitution, and output) for varying numbers of processors and input sizes.
   a. Plot the speedups you achieve on the elimination step alone.
   b. Plot the speedups you achieve on the overall program.
   c. Use Amdahl’s Law to derive an equation that predicts the speedup you would achieve on a machine with \(p\) processors \((p \gg 16)\). Is this estimate accurate, optimistic, or pessimistic? Can you come up with a more accurate estimate?
   d. What are the current bottlenecks? What steps might you take to eliminate them?
   e. (extra credit) Eliminate at least one of the bottlenecks in your existing implementation and repeat parts a-d. The amount of extra credit you receive will be a function of the final speedup you achieve on the overall program.

Notes

- A serial version of the Gaussian elimination code will be presented in class. You may base your parallel version on this code if you wish.
- Your parallel version should include the back-substitution step and output the final solution vector. However, you need to parallelize only the elimination step.
- Instructions for compiling and running your programs on the CAC machines are available at the CAC web site (see links on the class web page).
- Pthreads programs will run on uniprocessor machines, with the OS time-slicing the individual threads. Pthreads is available on most Unix machines, including CAEN Solaris machines and
Linux. You should test and debug your program on a uniprocessor (e.g., a CAEN workstation) before you attempt to run it on a CPC machine. Note that seemingly correct Pthreads programs can deadlock on a uniprocessor if one thread waits for another thread, but does not yield the CPU so that the other thread can run. You can avoid this behavior by using Pthreads synchronization calls for all your synchronization, or by calling pthread_yield() inside any busy-wait loops you may have.

- You are welcome to discuss the mechanics of writing and executing Pthreads programs and using the SGI machine with your fellow students. I encourage all of you to support one another in this learning process. Feel free to use the course newsgroup for this purpose. However, the program that you write and the results that you turn in are expected to be your individual effort.

- The following site has some useful Pthreads material http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html

- Test matrices can be obtained from /nfs/www.eecs.umich.edu/w/web/courses/eecs570 this works from CAEN and EECS-DCO