G22.3033-003: Architecture and Programming of Parallel Computers
Handout #7: Homework 3 (15 points)
Due: October 22, 1998

This assignment has three objectives: first, to become familiar with the Origin program development and execution environment; second, to develop an understanding of writing message-passing programs using MPI; and finally, to understand the performance impact of how message-passing overheads affect the task granularity that can be efficiently supported in scalable parallel machines.

To provide an incentive for you to form your project groups as early as possible (and to actually start working together), I will allow you to do this assignment in groups (no more than 3 students per group). Everyone in the group will receive the same grade. You are of course free to do the assignment on your own if you so desire.

Particulars

For this assignment you will convert the blocked LU decomposition program from Homework 2 into a message-passing parallel program. This program will be written for the SGI/Cray Origin 2000 using the Message Passing Interface (MPI) communication library (use only the C interface).

The assignment consists of three parts: the first two of which add up to 15 points, and an optional third part (for extra credit):

1. (**10 points**) Starting from the sequential blocked LU decomposition program, construct a message-passing program using calls to the MPI library. You will need to do the following:

   - decide how the matrix is going to be partitioned among the participating processors. Once you have decided this, each processor will need to allocate its local portion of the matrix. I suggest that you use a block-cyclic distribution to get good locality and load-balance.
   - decide how you are going to handle initialization of the matrix, and the final check for correctness. To keep matters simple, I suggest that you do the initialization and correctness check on processor 0. After initialization, you will need to distribute subportions of the matrix to the appropriate processor. Similarly, after computing the decomposition, each processor will need to send its portion to processor 0, which performs the final check.
   - write the main driver loop of the program which iterates over diagonal blocks. The idea is that each such iteration consists of the following steps:
     (a) the owner processor calls `lu0` to factorize the block.
     (b) it then sends the factorized blocks to other processors which use it to modify the row and column blocks (using `bmodd` and `bdiv`) appropriately.
     (c) each processor then sends its modified row and column blocks to processors holding the interior blocks. The interior blocks are modified using `bmod` with the corresponding row and column blocks.

     Note that you will need to allocate a workspace area on each processor, to store the incoming blocks for each iteration. I will leave it to you to decide how big this needs to be.
   - place appropriate barriers as required to separate the various steps.

Note that you are free to choose the implementation (and mode) of send/receive functions; i.e., you can either use the synchronous or the asynchronous options. For the synchronous ones, you may have to use the buffered send option (`MPI_Bsend`) to prevent deadlock.

While you can naturally use collective communication operations (such as `MPI_Gather, MPI_Scatter, and MPI_Broadcast`) and cartesian topology constructors (such as `MPI_Cart*`) for the above program, **I would like you to use only point-to-point operations**. The objective is for you to understand the underlying message-passing operations that are required.

Please verify that your program works with a small number of processors and a small data size before running it on larger input sizes and larger numbers of processors.

For this part of the assignment, hand in a listing of the program (one per group), and the absolute times and speedups obtained on **1, 2, 4, 8, 12, and 16** processors for a matrix size of 1024x1024 with a block size of 16. Compute speedups with respect to only the main driver loop; i.e., you can ignore the time for setting up the matrix and verifying the correctness of the decomposition. As indicated in Handout#6, to get speedup numbers that you can completely trust, you will need to use the dedicated batch queues. However, please do not run anything on these till you verify that your program performs as expected on the interactive and shared batch systems.

Note also that there are several possible ways of writing this program (some of which will give you a slowdown). I suggest that you work on the simplest program that will produce a speedup. Most of the design for such a program (deciding the partitioning, the communication structure, etc.) can actually be done before you write a single line of code.

2. (**5 points**) Replace the communication calls in the main driver loop of the program that you construct above with calls to your own routines which emulate message-passing primitives with higher overhead. To clarify, if you use the `MPI_Send` and `MPI_Recv` functions in your program, replace these calls with calls to your own functions `my_MPI_Send` and `my_MPI_Recv` which look like the following (similarly for `my_MPI_Recv`):

   ```
   void my_MPI_Send( /* arguments of MPI_Send */ ) {
     /* add a delay loop */
     MPI_Send( /* arguments */ );
   }
   ```

   The delay loop can be generated using a null `for` loop: you need to ensure however that the compiler does not eliminate this loop. One way of ensuring this is to put the delay loop in a separate function that is compiled without optimization. The loop bound of this delay loop should be such that you are able to emulate delays of **5, 10, and 20** μ**s**.

   Measure speedups again with this modified program for delay values of **5, 10, and 20** μ**s**. Comment upon the results. How does the performance change when the block size is increased to 32? Can you relate the observed performance to the change in communication-to-computation ratio? For this part of the assignment, hand in the program listing showing changes to the original program, the speedup plots, and accompanying discussion.

3. (**extra credit: 3 points**) In the second part above, you designed an experiment that emulated higher-overhead message-passing primitives. Can you design an experiment where the goal is to emulate message-passing primitives with higher latency than you find on the Origin? As you should recall from class, overhead refers to the portion of communication cost where the processor is busy either sending or receiving messages. On the other hand, latency refers to the time spent in the network interface and the network.

   As with Homework 2, I need only a description of how such an experiment would work.