CS 420: Introduction to Parallel Computing for Scientists and Engineers

Fall 2013

Machine Problem 2 – MPI Jacobi

Due date: October 23, 5:00 PM

Background

For this MP you are given serial code for a Jacobi relaxation method to simulate heat flow in a 2-D grid which you will parallelize using MPI. The goals are for you to learn to recognize potential parallelism in code, develop an efficient data transfer routine, and familiarize yourself with MPI. Your code will be graded on both correctness and speed. Be sure to use the serial code given to you to check that your optimized parallel code produces the same answer. There is an extra part to the assignment for students enrolled in 4 credit hours.

System

You can use any computer with a multi-core processor and the MPI software installed to develop the code for this assignment, but you will need to use Taub to run your code at higher core counts and get running times for this assignment. The code that you will be editing is written in C. Both EWS and Taub have the gcc compiler and MPI available.

Use the included Makefile to automate your compilation. Use the optimization level -O0 when compiling the code. Taub contains dual 6-core processors. You will be asked to run tests using up to four nodes (48 cores) when timing the code to test performance (see **Submission** for details). You can run MPI code using the command

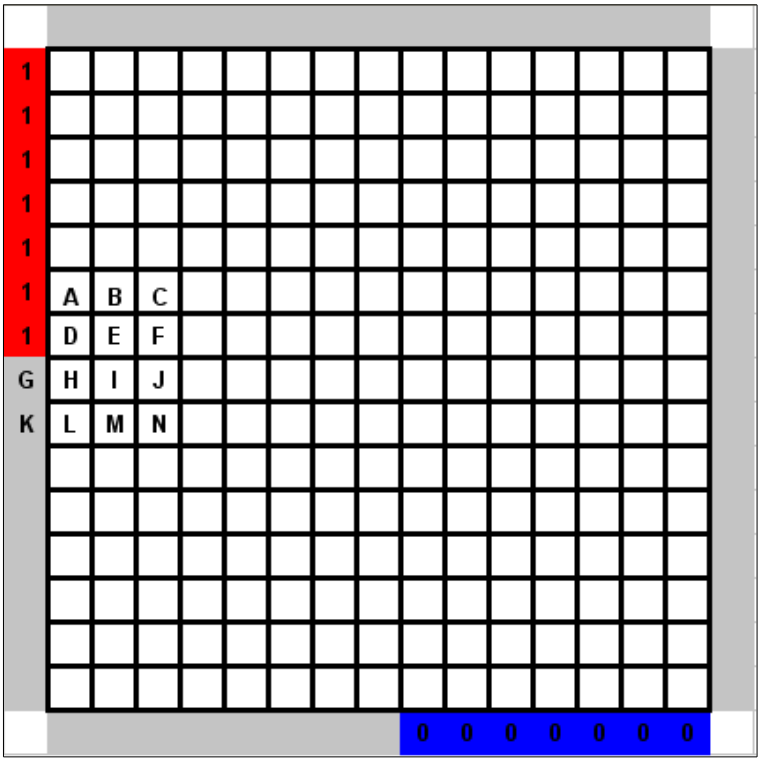
"mpirun -n 48 ./jacobi"

where -n <number> sets the number of processors to use.

A print_grid function is provided to output the grid to the screen. You can use this to check the output of your code. Do not use this function when timing your code. Timer code is also included. Be sure to put the timing functions only around the simulation code, and to ignore the setup and cleanup. Also be sure to put a barrier before beginning to time the code. We recommend that you develop your code using a small number of processors (start with 2 cores on 1 node) and a small grid (such as an 8x8 grid). Once you have a working parallel code that produces the correct answer, start using larger core counts and larger grids to test the performance of the code. You will also want to optimize the code as much as possible to get the best performance.

Similar to MP0, you will need to use batch scripts to run your code on the Taub's compute nodes when testing the performance of your programs. A sample batch script is included on the wiki. You can modify it to run using different numbers of cores. It is currently set up to allow you to run tests using multiple core counts in a single batch script. It is also set up to use up to 48 cores (4 nodes) at once.

Assignment



You will code a parallel MPI implementation of the Jacobi relaxation algorithm to simulate heat flow in two-dimensional space. Consider the square, discretized environment above. The world is divided into N^2 elements. Extending from the upper half of the left side is a panel with fixed temperature 1. Likewise, there is a panel with fixed temperature 0 extending from the right half of the lower side. Both panels have length `N/2` (use integer division; e.g. `81/2 = 40`). You may assume that `N` is evenly divisible by the number of processors `P`.

At each time step, the temperature of each element should be the average of its own temperature and the temperatures of its four neighbors (north, south, east, west) at the previous time step. For example, using the image above, the temperature of the interior element E at time t should be

```
E(t) = 0.2*(E(t-1) + B(t-1) + F(t-1) + I(t-1) + D(t-1)).
```

Treat elements on the world boundary that are not adjacent to the fixed temperature panels as though there were a ghost element just beyond the world that always matched the boundary element's temperature. For example, the update for H would be as follows:

```
H(t) = 0.2*(H(t-1) + D(t-1) + I(t-1) + L(t-1) + H(t-1)).
```

The code initializes the temperature of the grid to 0.5. After each iteration, a difference between the previous and updated temperature is computed for each grid point. The maximum change over all of the grid points should be compared after each iteration to a threshold value supplied in the code.

The program should conclude when either

1. the maximum number of iterations I has been reached, or
2. the maximum change for any point is below the threshold value,

whichever happens first.

Coding Restrictions

You are limited to the use of certain MPI constructs for this assignment. You can only use:

- MPI_Init
- MPI_Comm_size
- MPI_Comm_rank
- MPI_Send
- MPI_Receive
- MPI_Reduce/MPI_Allreduce
- MPI_Barrier
- MPI_Finalize

Communication

Each processor will be responsible for computing the temperature updates for a block of adjacent rows of the discretized space. At each iteration, processors must communicate their boundary values to neighboring processes. For example, process `1` should send its top boundary (row) to process `0`, and its bottom boundary to process 2. Likewise, each process should receive a message from each of its neighbor processes. Each message contains a whole row of data. Use `MPI_Double` to store the temperature values.

Boundary processes may not participate in all stages of communication (i.e., because they have only one neighbor instead of two).

Input

The program should take four command line parameters:

```
./mp2 N I R C
```

Where `N` measures the size of the world, and `I` is the maximum number of iterations to run. R and C are the zero-indexed row and column, respectively, of a single query element.

Output

You should output two values:

1. A single floating-point value for the temperature at cell (R,C) at the end of the program. Output the value up to 8 decimal places.

2. The running time for the main routine.

Example

The following is an example of how the program is used. The value returned in this example is arbitrary and may not match the result you will get.

```
qsub -I -l nodes=1:ppn=12 -walltime 01:00:00
module load mpi
mpirun -n 12 ./pjacobi 64 500 25 32
Results:
Running time=  0.00123906
Value at (R,C)=  0.45600000
```

In this example, there is one node with 12 cores. For this assignment, there will be one process per processor. We encourage you to use the batch system for runs with more than one node, and always use it for timing. Each process is responsible for a contiguous block of data: process 0 has rows 0 to X; process 1 has rows X+1 to 2*X; etc. for some X. Process 0 cannot compute the updates for row X without knowing the values of row X+1 on the previous time step. During the communication phase, process 0 will send row X to process 1 and receive the values of row X+1 from process 1. Process 1 will need to exchange messages with process 0 and process 2. After 500 iterations, the process containing (25,32) prints out the temperature of the element with coordinates (25,32). The other processes do not output anything.

4 Credit Hours

If you are enrolled in 4 credit hours, you will need to also write a version of the code that splits the data up into columns, instead of rows. Parallelize this code, and run the same analysis on it that you run on the row-divided code. Comment on the differences in efficiency and code complexity between the two implementations.

Submission

The grid sizes N, max iteration values I, and row/column indices R and C to test will be announced in a few days once the university computer systems are up and working again after the outage is over this weekend. The grid sizes will be a multiple of all of the processor sizes to allow an even number of rows to fit on each processor core**.**

You should run the MPI jacobi program and record the running times, number of iterations, and final tolerance for all combinations of the following values:

P=1, 12, 24, 36, and 48.

N=...

For each run, you should use the following values for I, R, and C.

I=... R=... C=...

Submit answers to the following items in a pdf file.

1. Create a table containing the tolerance and value of a[R][C] for each combination of P and N above.

2. Plot the running time as a function of P for different values of N. Plot the number of cores (P) as the x-axis , the running time as the y-axis, and make each grid size (N) a separate line in the plot. Always use 10 iterations. For the different values of P, use the minimum number of nodes possible for each processor count—For example, you would use 2 nodes for P=24, since there are 12 cores per node.

3. Create a speedup plot similar to part 3, but instead of plotting running time as the y-axis, plot the speedup $\frac{Time\ on\ 1\ processor}{Time\ on\ p\ processors}$ as the y-axis.

4. If the grid size were not divisible by the number of cores, how would you need to modify your program? How would this impact performance?

5. In this assignment, each process was responsible for computing the updates for a contiguous block of rows in the 2-D space. Describe a 2-D decomposition, where each processor takes a rectangular tile rather than a set of rows. What changes would be required in your code?

6. Discuss the results you obtained. Were you able to gain a linear or near-linear speedup? Why or why not?

Name your file netid_mp2.pdf. It is important that you strictly follow the file naming guidelines for your submission since we will be using automated scripts to help with the grading.

Submit your fastest parallel code in the file jacobi.c.

Copy this report, all of your parallel source code, and your Makefile into a folder named mp2. Either zip the folder (Windows) or tar and gzip it (Linux). Be careful when using tar as you can accidentally overwrite your files if you put the arguments in the wrong order. Submit your .zip or .tar.gz file to the Subversion repository. The timestamp given by Subversion will be used to determine whether an assignment is on time, so be sure to upload it by the deadline.

If you are unsure of how to create a .pdf file, most popular word processors (MS Word, LibreOffice, etc.) have the functionality built in.