CSCI-GA.3033-016

Multicore Processors

Lab Assignment 2

In this lab you will implement a method for solving a group of linear equations using OpenMP, MPI, and a mix of both..

What will your program do?

Given a set of n equations with n unknowns (x_1 to x_n), your program will calculate the values of x_1 to x_n within an error margin of e%.

The format of the input file is:

- line1: #unknowns
- line2: absolute relative error
- Initial values for each unknown
- line 3 till end: the coefficients for each equation. Each equation on a line. On the same line and after all the coefficients you will find the constant of the corresponding equation.

For example, if we want to solve a system of 3 linear equations, you can have a file like this one:

```
3
0.01
2 3 4
5 1 3 6
3 7 2 8
3 6 9 6
```

The above file corresponds to the following set of equations:

$$5X_1 + X_2 + 3X_3 = 6$$
$$3X_1 + 7X_2 + 2X_3 = 8$$
$$3X_1 + 6X_2 + 9X_3 = 6$$

The third line in the file tells us that the initial values for X_1 is 2, for X_2 is 3, and for X_3 is 4. Those values may not be the solution, or are very far from the solution that must be within 1% of the real values (as given by the 0.01 in line 2).

How will your program do that?

We start with a set of n equations and n unknowns, like this:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2$$
$$\vdots \qquad \vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n$$

You are given all a_{ij} and b_1 to b_n. You need to calculate all Xs.

Here are the steps:

1. Rewrite each equation such has the left-hand-side is one of the unknowns.

   Rewriting each equation

   $$x_1 = \frac{c_1 - a_{12}x_2 - a_{13}x_3 \ldots\ldots - a_{1n}x_n}{a_{11}} \longleftarrow \text{From Equation 1}$$

   $$x_2 = \frac{c_2 - a_{21}x_1 - a_{23}x_3 \ldots\ldots - a_{2n}x_n}{a_{22}} \longleftarrow \text{From equation 2}$$

   $$\vdots \qquad \vdots \qquad \vdots$$

   $$x_{n-1} = \frac{c_{n-1} - a_{n-1,1}x_1 - a_{n-1,2}x_2 \ldots\ldots - a_{n-1,n-2}x_{n-2} - a_{n-1,n}x_n}{a_{n-1,n-1}} \longleftarrow \text{From equation n-1}$$

   $$x_n = \frac{c_n - a_{n1}x_1 - a_{n2}x_2 - \ldots\ldots - a_{n,n-1}x_{n-1}}{a_{nn}} \longleftarrow \text{From equation n}$$

   Note: The Cs above refer to the constants, which are the b_1 to b_n.

   In general:

   $$x_i = \frac{c_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}x_j}{a_{ii}}, i = 1,2,\ldots,n.$$

2. Remember that you were given some initial values for the Xs in the input file. The absolute relative error is:

   $$\left|\in_a\right|_i = \left|\frac{x_i^{new} - x_i^{old}}{x_i^{new}}\right| \times 100$$

   Therefore, our goal is to reduce absolute relative error for each unknown to make it less or equal to relative error given in the input file (2nd line of the intput file).

Note: You need to multiply the error given in the file by 100 to match it with the above equation, or to not multiply the above equation by 100.

3. Substitute the initial values in the equation of each X_i to get new value for X_i. Now we have a new set of Xs.
   Important: Let's say you calculated a new X_0. When you calculate X_1 **DO NOT** use the new value for X_0 but the old value, in the current iteration. In the following iteration, use all new values.

4. Calculate the absolute relative errors for each X.

5. If all errors are equal or less the given number (2nd line in the file) then you are done.

6. Otherwise go back to step 3 with the set of new Xs as X_{old}.

What is the input to your program?

The input to your program is a text file named xxxx.txt where xxxx can be any name. We already discussed the file format.

So, if your program is called solve then I must be able to run your program as

mpiexec -n *x* ./solve inputfile.txt (pure MPI)
or
mpiexec -n *x* ./solve inputfile.txt (MPI + OpenMP)
or
./solve inputfile.txt (pure OpenMP)

(x is the number of processes).

What is the output of your program?

Your program must output to a text file with the name x.sol, where x is the number of unknowns. For the above example, your program must generate a file 3.sol that contains:
2
3
4
Where 2 correspond to the value of X_1, 3 corresponds to X_2, and 4 corresponds to X_3, ...
That is, each value on a line.

The number of iterations is printed on the screen:
total number of iterations: 5

What do I do after I finish my program?

We have provided you with a reference program *gsref* so you can check the correctness of your results. We will test your submission against this reference (for correctness of solution not the number of iterations). Execute ./gsref to see the usage.

Make sure your program compiles on crunchy machines on CIMS and that you use the latest version of gcc because OpenMP is embedded within gcc so the more updated version you use the more updated OpenMP. You can do this as follows:

- Once you are logged into a crunchy machine, type: module avail
- Check the latest gcc version.
- Type: module load gcc-9.2.0 (or whatever latest version of gcc you find as long as it is 6 or higher).
- Must do that each time you login and compile

Also, before doing any MPI programming, do the following once you login onto your CIMS account and ssh to one of the computational nodes (e.g. crunchy1, crunchy3, crunchy5, or crunchy6) Type, the following: **module load mpi/openmpi-x86_64**

What do you have to do?

- You are also provided with executable file ./gens to generate equations. Execute ./gengs to see the usage. (Note: If you get command not found when execution ./gsref or ./gengs then change the permissions to 777 for both of them.)
- Start with problems of size (i.e. number of unknowns) 100 then increment logarithmically (i.e. 100, 1000, 10000, …) till you 100,000 or till you run out of space.
- Create the following table:

	100	1000	10,000	100,000
Sequential (OpenMP)	1	1	1	1
Sequential (MPI)				
MPI				
OpenMP				
MPI + OpenMP				

All the entries in the table must be speedup relative to the sequential OpenMP (i.e. Open with one thread). This is why the first row of results is all 1s. **The time measures is the time for calculating the solutions. Do not count the time for reading the input file or writing the output file.**

For the parallel versions, in each entry of the table, and beside the speedup, write the best number of threads (in case of OpenMP) or processes (in case of MPI) or processes and threads per process (in case of hybrid). "The best number" means the one that gives the best performance. So, you will need to do some experimentation.

After you are done filling the table, write few lines explaining the following:

- Are there differences between sequential OpenMP and sequential MPI? And why?
- What is the best configuration for each problem size? Justify. Your justification must not be generic. So something like "problem size is small and we don't have parallelism" is not enough. You need to go deeper and how long each part of your code takes and analyze that. You know how to measure times taken by for each part of your code. There is clock(), there is API in OpenMP, and there is API + reduction (MPI_MAX) for MPI.

What and how to submit?

You need to submit four files:

- solveopenmp.c ← pure openmp
- solvempi.c ← pure MPI
- solvehybrid.c ← MPI+OpenMP
- pdf file that contains the table and your analysis as indicated above.

Put them in one zip file with the name netID.zip where netID is your own NetID.

How will we grade this?

- Each version of your code compiles and execute correctly: 10 points each → 30
- The table → 10 points
- Your explanation → 10 points

Total of 50 points

Penalties

- You will lose points also if your conclusions are like "As we can see from the table, x increasing with y". We need your explanation not your description of what we already see!
- You will also lose points if you don't follow the steps regarding filenames, output, etc.