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 MPI.

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 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

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

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

   M M M

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

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

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

   In general:

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

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

mpirun -n *x* solve inputfile.txt

(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).

Before you can compile your program, do the following two steps:

- ssh to one of the crunchyx (x=1, 3, 5, or 6) machines
- type: module load mpi/openmpi-x86_64

After you finish the parallel version of your program, compile it with:
mpicc -std=c99 -o solve solve.c -lm

Where solve.c is your source code. We provide a skeleton file, **solve.c**, to help you start. Feel free to modify anything in solve.c (or ignore it altogether) as long as you will be able to read the file, and output the results on the screen in a format similar to ./gsref

What do you have to do?

1. Download the file eqns.tar from the course website into your CIMS account.
2. ssh to a crunchyx machine (x can be 1, 3, 5, and 6).
3. type: tar -xvf eqns.tar
4. This will create a directory called eqns
5. cd lab1
6. Try executing ./gens and ./gsref to have an idea
7. Write your MPI program and get it to compile correctly. You can make use of gs.c to help you.
8. Use **./gengs** (provided) to generate several problem sizes. Its command line is ./gengs x y where x is number of variables and y is the error. The program will generate a text file with the name x.txt where x is the number of variables, and the format of that file is described above.
9. Check the solution (the X values) against ./gsref
10. Generate the following Table 1:
    a. The columns represent the number of processes. You need to try for the following number of processes: 1, 2, 10, 20, and 40.
    b. The rows represent the number of unknowns. It goes: 10, 100, 1000, 10000, and 100000.
    c. Keep error rate at 0.001
    d. The entry in the table (for a process number vs problem size) is the output of the time command (that is, your command starts: time mpirun -n…). Use the **real** number (as the command generates 3 numbers: sys, user, and real). You may need to repeat the experiment few times (~5 or so) and take the average as the performance may fluctuate.
    e. **Empty entries in the table:** If the number of processes is bigger than the number of unknowns, do not do this experiment and leave the entry empty.
11. Generate Table 2: Same as Table 1 but the entries contain the speedup relative to number of processes = 1. So, table two can be thought of as the speedup.
12. Using the above two tables, explain:

a. When don't you get speedup (i.e. at what number of processes and problem sizes)?
b. Why you don't get speedup in the case above?
c. When do you get speedup, if at all?
d. Explain c above.

You explanation must be specific and not very generic.

13. Put your results (the two tables and the answer to the above questions) in one file (named results.doc or results.pdf) and put your name and NetID at the top of that file.

What and how to submit?

Through NYU classes, submit: the pdf (or doc) file and your source code solve.c in a single zip file names: lastname.firstname.zip

How will we grade this?

- Correctness: Your code must compile and run correctly on CIMS machines → 5 points
- Table 1 → 5 points
- Table 2 → 5 points
- The report → 5 points
- At least some of your experiments have speedup of 2x and higher over your code with one process

For a total of 30 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.