# Parallel Computing
## Lab Assignment 2

In this lab you will implement a method for solving a group of linear equations using OpenMP.

**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 input to your program is:

- a file: *somename*.txt
- number of threads

The output will be two parts:

- Number of iterations printed on the screen.
- Output file *num*.sol (num = number of unknown variables)

The format of the input file is:

- line1: number of variables (i.e., number of unknowns)
- line2: absolute relative error <= 1
- Initial values for each unknown variable
- 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

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

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

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

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

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

   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 input 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 $X_1$. When you calculate $X_2$ **DO NOT** use the new value for $X_1$ but the old value, of 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 than 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 *somename*.txt where *somename* can be any name. The second input is the number of threads. We already discussed the file format.

Name your program netID.c where netID is your own netID.

Compile with: gcc -std=c99 -f openmp -Wall -o solve netID.c -lm

I must be able to run your program as

./solve inputfile.txt t

(t is the number of threads).

**What is the output of your program?**

Your program must output to a text file with the name num.sol, where num is the number of unknowns. For the example mentioned earlier, your program must generate a text file 3.sol that contains:
2
3
4
Where 2 corresponds 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 *./refoutput* 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).
This is a sequential code not a parallel one. So, to execute it, you just need to input the filename, no need to enter the number of threads.
First, execute: chmod 777 ./refoutput
Second, execute: ./refoutput inputfile.txt

We are also providing you with a program to generate input files. This program is called ./genfile
First, execute: chmod 777 ./genfile
Second, execute: ./genfile num err
num: number of variables
err: the absolute error, and it is between 1 and 0.0001

The output will be a file num.txt that can be used as input to your program and to the refoutput program.

Note: refoutput and genfile are executables not source code. So, no need to compile them. But they will work only on crunchy machines, not your laptop.

**What do you have to do?**

1. Write your openMP file and compare its output to refoutput. Do not compare the number of iterations, just the values of Xs.
2. Generate the following Table 1:
    a. The columns represent the number of threads. You need to try for the following number of threads: 1, 2, 4, and 8. Your program must account for the condition where number of variables is not divisible by the number of threads. However, we will never test your code with a case where the number of variables is smaller than the number of threads.
    b. The rows represent the number of unknowns. It goes: 8, 16, 32, 64, 128, 256, 512, and 1024.
    c. Keep error rate at 0.001
    d. The entry in the table (for a thread number vs problem size) is the time of the parallel part. That is, use omp_get_wtime around the parallel code and do *not* include the time for reading from the file, writing to the file, and any dynamic allocation you make. You may need to repeat the experiment few times (~5 or so) and take the average as the performance may fluctuate.
3. Generate Table 2: Same as Table 1 but the entries contain the speedup relative to number of threads = 1.
4. Generate Table 3: Same as Table 3 but the entries contain the efficiency = speedup (over one-thread execution)/ number of threads.
5. Give the following answers:
    a. What is the trend you see in Tables 1 and 2 (expected to be same trend)?

b. What is your interpretation of that trend?
c. What is the trend you see in Tables 3 (expected to be same trend)?
d. What is your interpretation of that trend?

6. Put your results, and answers, in one file (results.pdf).
7. Generate a zip file that contains your source code (netID.c) and the results and answers (results.pdf). The zip file is called: netID,zip, where netID is your own netID
8. Submit the lab on Brightspace as you did for the previous lab.

**Enjoy!**