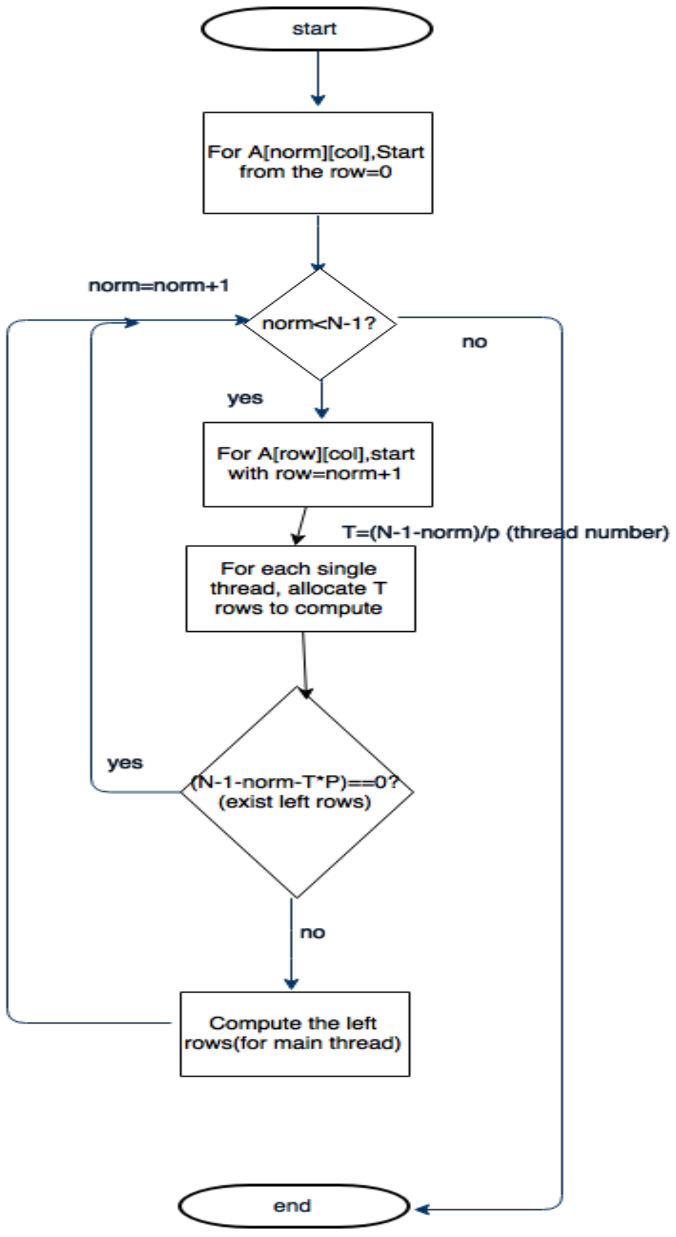
Report

1. Description

In this programming assignment, I implemented two kinds of parallel algorithm one in Pthread and the other in MPI to solve a dense linear equations of the form A*X=B using Gaussian elimination without pivoting. Next, I will introduce how I parallel it using these two methods.

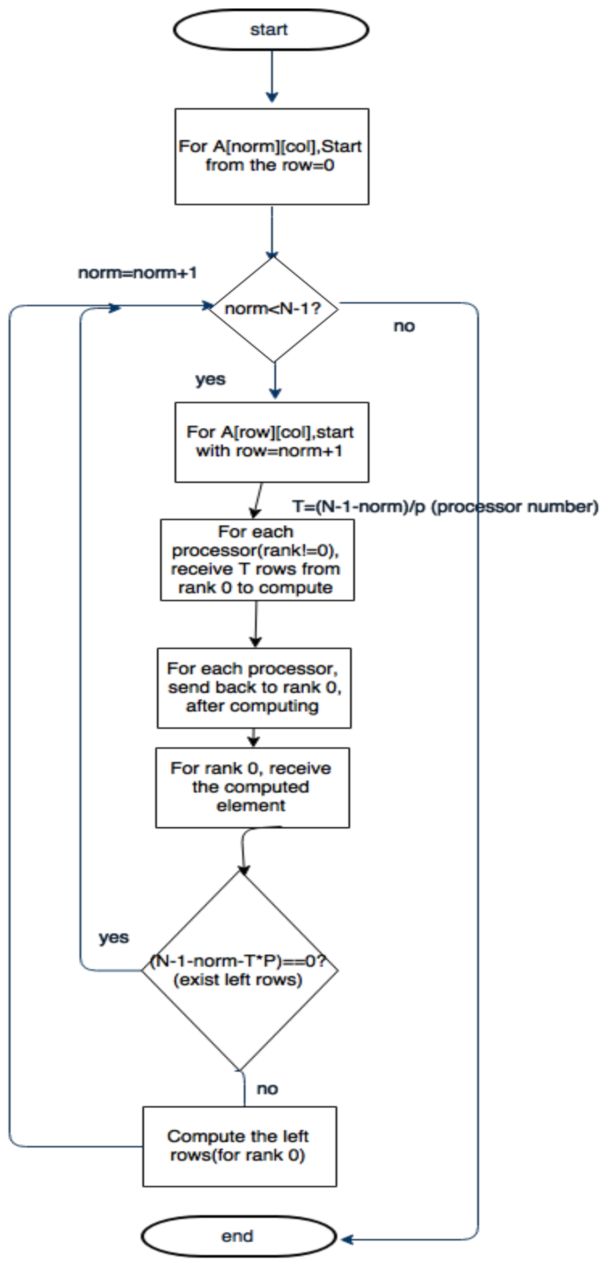
1). Pthread

As we know, In Gaussian elimination, each outer loop (each norm row), the current A[] depend on result of last loop, so we cannot parallel the whole process. However, in one loop, each row, which row number is larger than norm row, is independent, so we can parallel each row computing in one loop. The following figure described my Pthread method:


start
For A[norm][col],Start from the row=0
norm=norm+1
norm<N-1?
no
yes
For A[row][col],start with row=norm+1
T=(N-1-norm)/p (thread number)
For each single thread, allocate T rows to compute
yes
(N-1-norm-T*P)==0? (exist left rows)
no
Compute the left rows(for main thread)
end

2). MPI

For MPI, I use the similar parallel method to Pthread. The following figure show the process:


start
For A[norm][col],Start from the row=0
norm=norm+1
norm<N-1?
no
yes
For A[row][col],start with row=norm+1
T=(N-1-norm)/p (processor number)
For each processor(rank!=0), receive T rows from rank 0 to compute
For each processor, send back to rank 0, after computing
For rank 0, receive the computed element
yes
(N-1-norm-T*P)==0? (exist left rows)
no
Compute the left rows(for rank 0)
end

2. Optimization

my optimization for my former algorithm is that, at first I only allocate one row for each thread, then allocate next row to them after computing. In this way, there is a huge communication time cost, which will lower the performance of this algorithm. Thus, I optimized that algorithm to current one: For each outer loop, I only allocate many rows to threads/procesoors one time, so that it can decrease the time for communication. As a result, the elapsed time decreased a lot.

3. Result Analysis

1) In my Pthread method , I assume that Matrix size = 2000, random seed =1

Thread_Num	1	2	4	8	16
Local	9242	4860	4343	4527	4772
Jarvis	12037	8639	4225	4259	4096

1.The elapsed time in different size of thread

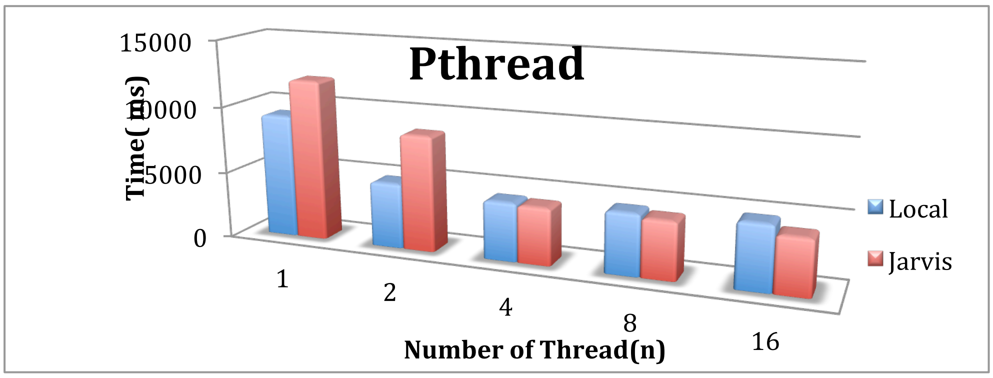


Figure 1. The elapsed time in different size of thread

This figure 1 shows that the computing time is longest in only 1 thread, and with the size of threads increase, the time become less. However, when the number of thread is larger than 4, the time stops to decrease. It is because that in my local machine and Jarvis both have 4 cores CPU, so 4 threads will result in the best performance, but when we use more threads to compute, the performance will not increase but decrease due to more thread exists.

2) Also, in my MPI method, the Matrix size =2000, random seed =1.

Processor_Num	1	2	4	8	16
Local	18292	12973	10916	13378	25518
Jarvis	32546	22100	19894	29970	49020

2. The elapsed time in different size of processor

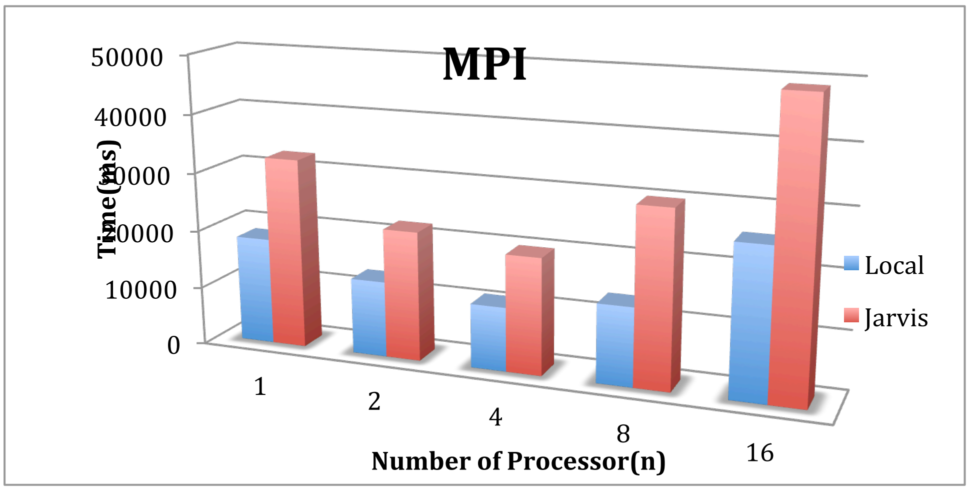
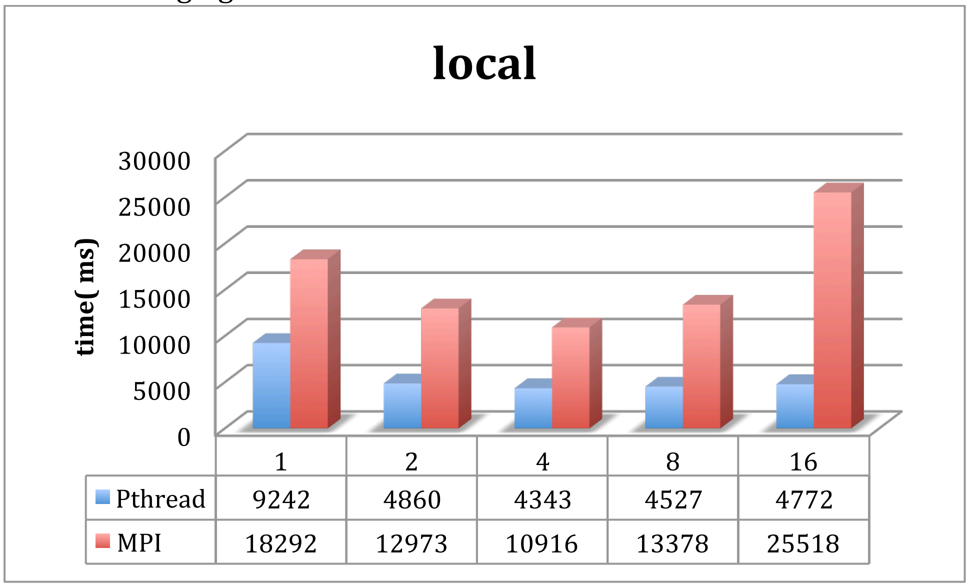


Figure 2. The elapsed time in different size of processor

The figure 2 shows that the best computing performance occurs in that the number of processor is 4, because the cpu cores is 4. However, when we use more processors, the elapsed time increased a lot, that is because the communication cost is so large in MPI.

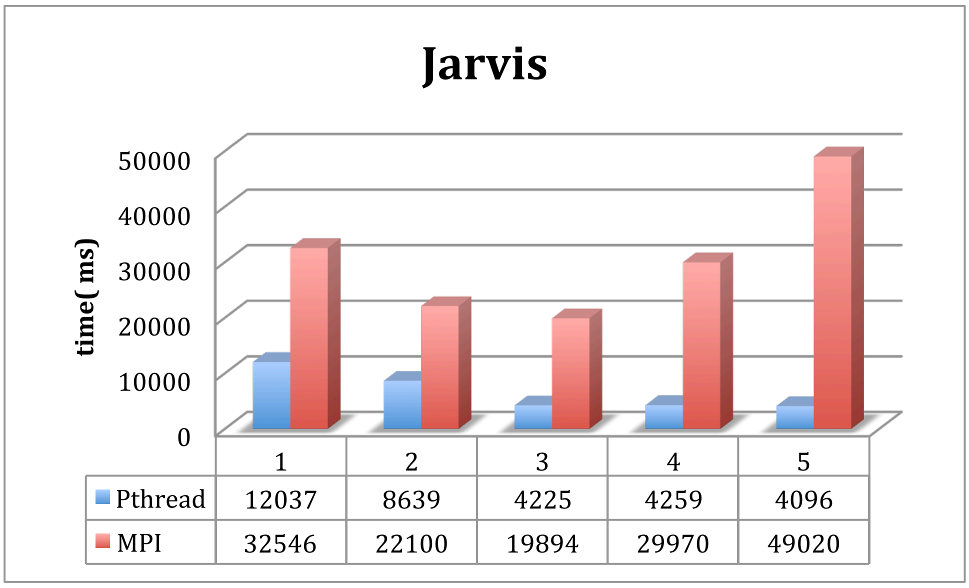
3) Comparisions in Pthread and MPI

The following figures will show the different time in MPI and Pthread method



	1	2	4	8	16
🟦 Pthread	9242	4860	4343	4527	4772
🟥 MPI	18292	12973	10916	13378	25518

Figure 3. Pthread and MPI method elapsed time in local machine



	1	2	3	4	5
🟦 Pthread	12037	8639	4225	4259	4096
🟥 MPI	32546	22100	19894	29970	49020

Figure 4. Pthread and MPI method elapsed time in Jarvis

From the figure abovem we can know that, when the Martix size is 2000*2000, the performance of Pthread method is better than MPI, we cannot discover the advantage of MPI in small scale of data, when the data scale become larger, the performance of MPI will become better.