Xin Xing
Georgia Institute of Technology
School of Mathematics

Chengwei Li
Georgia Institute of Technology
Civil and Environment Engineering

CSE6220 HPC

PROGRAMMING ASSIGNMENT #2

JACOBI METHOD

Project Description

The *Jacobi Method* is an iterative numerical method for solving linear system. Given a full rank $n \times n$ matrix A and a vector $b \in \mathcal{R}^n$, *Jacobi Method* iteratively approximates the solution $x \in \mathcal{R}^n$ for the system:

$$Ax = b \text{ with iteration formula } Dx_{n+1} = -(L+U)x_n + b.$$

The advantage of Jacobi Method over other similar algorithms like Gauss-Seidal or SOR is that, it can be implemented by parallel computing method. For parallelized Jacobi method, we distribute matrix A and $L+U$ among the $q \times q$ processor mesh uniformly. Then use the parallel matrix-vector multiplication to compute $(L+U)x$. The vectors x, and b, and the diagonal elements D of A are distributed among the first column of the processor grid. We can thus update the component of x_{n+1} by $x_{n+1}^{(i)} \leftarrow \frac{1}{a_{ii}}(b_i - (Rx)_i)$ using local operation.

In order to terminate after finite iteration, we calculate the residual $\|Ax_n - b\|_2$ and keep counting the iteration times in each iteration. Given the thresholds max_iteration and l2_norm, we can terminate the iteration when either of the thresholds is met.

Algorithm

Sequential Implementation

The implementation of the sequential Jacobi method is directly following the algorithm to get new approximation x_{n+1} based on the present approximation in each iteration. The basic structure is as,

```
Initialize x_0 = [0 0 ..0]
while (residual > l2_norm && iteration_count < max_iter)
   Update x_{n+1} from x_n;
   Compute residual ||Ax_{n+1} - b||;
   Update iteration_count;
end
```

Parallel Implementation

The key idea for parallelized Jacobi method is to implement the matrix-vector multiplication $(L + U)x_n$ by the parallel algorithm.

To efficiently parallelize the Jacobi algorithm, the devised schemes should achieve data locality, minimize the number of communications, and maximize the overlapping between the communication and the computations. For simplicity, the number p of processes is a perfect square number such that it could form a$\sqrt{p} \times \sqrt{p}$ mesh. The distribution of the blocks of matrix and vector can be performed in a column-wise fashion.

At the beginning, the matrix A and right hand side vector b are both stored in $(0, 0)$ processor in the grid communicator. Then the main step in parallel Jacobi is as following.

1. Distribution of matrix

   Distribute the rows of matrix on the first column of processor using the MPI_Cart_sub function to construct local communicator and the MPI_Scatterv function to distribute the matrix based on the rows to the processors.

   Scatter the columns of the submatrix in each row of processor. Again build up the local communicator using MPI_Cart_sub and MPI_Scatterv to get the submatrix distributed among each processor on the same row.

2. Distribution the vector

   Distribute the vector in [0,0] processor to the processors on the first coloum of the grid.

3. Parallel Jacobi Iteration

   (a) Transpose the vector x on the grid, such that a processor with index (i, j) will end up with the elements that were on processor (j, 0).

   (b) multiply the local vector with the local matrix

   (c) parallel reduction to sum up the resulting vectors along the rows of the grid back onto the processors of the first column.

   (d) update the iteration from x_{n+1} to x_n in the first column and compute the residual $\|Ax_{n+1} - b\|_2$ in the first column.

Runtime

To test the sequential and parallelized Jacobi method, we applied both methods on several different-size matrices with same difficulty (0.5) and with different number of processors to be used. The thresholds for termination are set as l_2-terminate $= 10^{-8}$ and max_iter=100. The results are shown in the following.

First of all, for sequential Jacobi method, the runtime increases nonlinearly along with the matrix size n in Figure 1. And more analytic analysis shows that, with the same difficulty(equals 0.5) which leads to similar convergence speed and iteration times, the runtime is of scale $O(n^2)$. And, in fact, this is exactly the complexity of the matrix-vector multiplication which is used exactly once in each iteration.

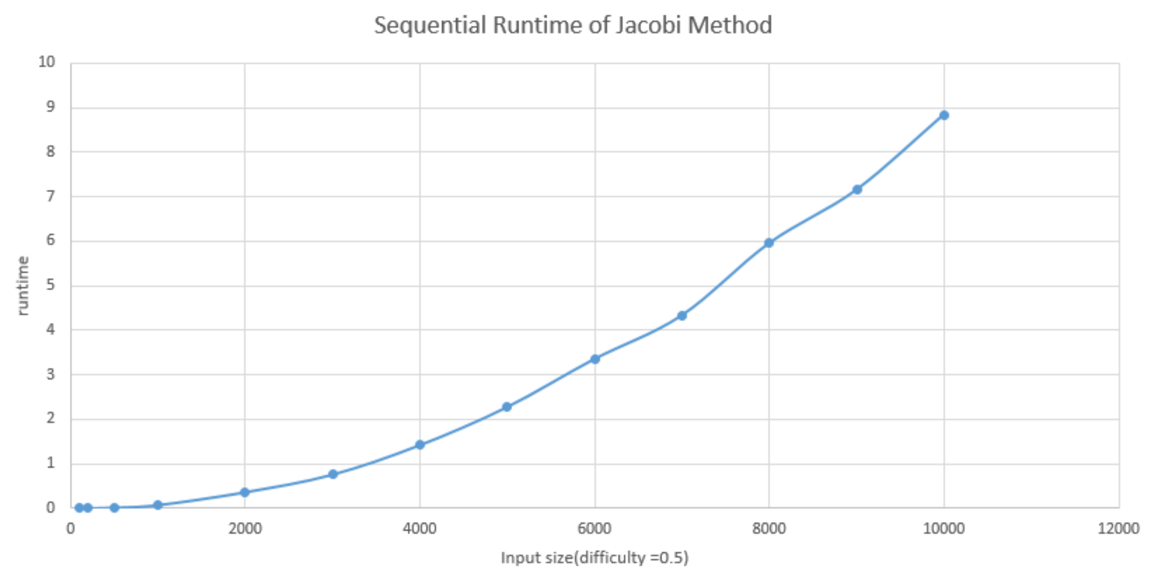

Figure 1: Runtime for Sequential Jacobi Method

For parallel Jacobi method, we could see the varying pattern on the runtime with the change of the problem size in Figure 2. When the matrix size n is small, the runtime for the parallel version would be much larger than that of sequential version. The reason is, when n is small, the communication time will dominate the runtime in the parallel Jacobi. So, the extra communication time in parallel counteracts the computational advantage it has over sequential one. This scenario doesn't disappear until n goes bigger than 1000 in our experiment. And then, the speedup of parallel Jacobi begins to emerges.

The parallel Jacobi's advantage could be better illustrated through the Figure 3. Basically,the parallel Jacobi always runs much faster than the sequential one. However, in our test with matrix with size up to 10000, the parallel Jacobi runs fastest with 4 and 9 processors, in comparison with $p = 16, 25$. The communication time could be one possible explanation for the increased runtime with respect to number of processors, which the computational time should decrease as more processors are involved.

More specifically, the extra communication time of parallel Jacobi method includes,

- Distribution of matrix A among the processor grid.
- In each iteration, scatter the vector x_n in first column to the whole processor grid.
- In each iteration, the summation of the vector component in each row of processors using MPI_Reduce
- In each iteration, the summation of residual in the first column of processor using MPI_Reduce

Hence, it's possible that computation time parallel Jacobi saves cannot make up the additional communicational time it takes as the number of more processors involved. To better illustrate the effect of processor number, much larger matrix test is needed. At last, the runtime for parallel Jacobi should decrease as number of processor increases.

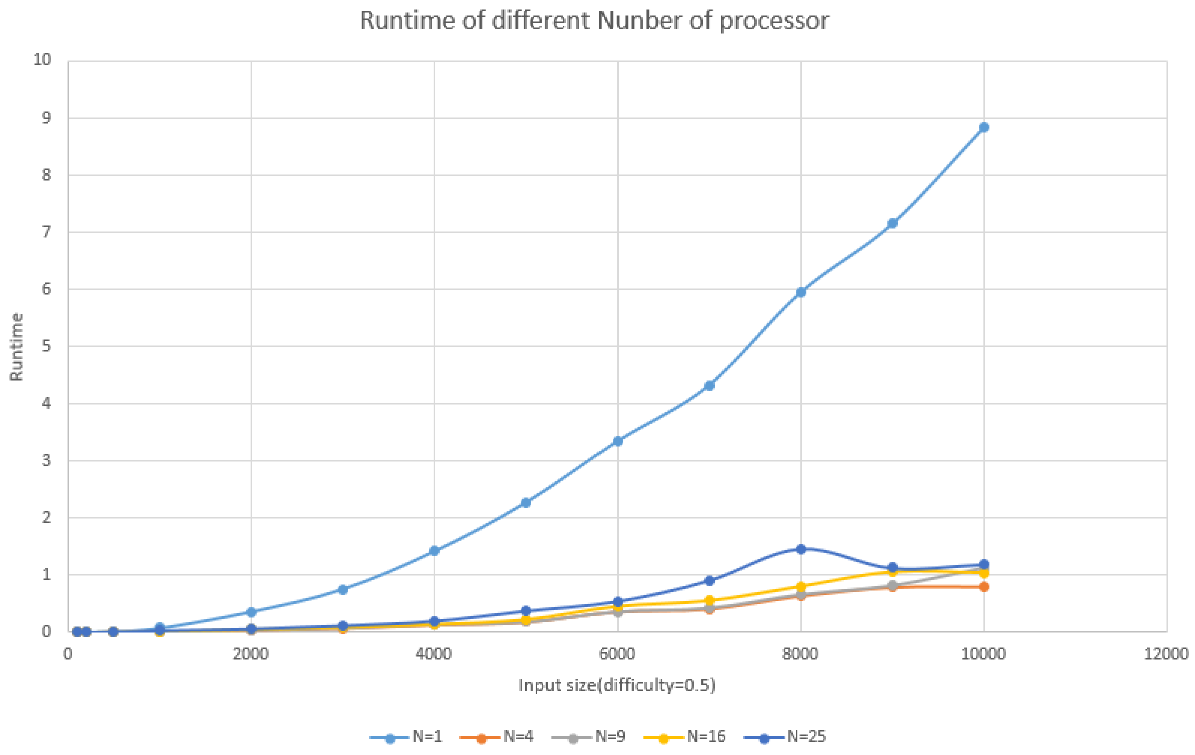

Figure 2: Runtime for Parallel Jacobi Method(Sequential included with $N = 1$)

Speed Up and Efficiency

The speedup and efficiency of the parallel algorithm with different number of processors is shown in Figure 4 and Figure 5 respectively w.r.t. the matrix size. Clearly, we can observe that, with fixed processor number, the basic trend of speedup ratio and efficiency is increasing along with larger matrix size. It exactly shows the advantage of parallel Jacobi that it performs better with large matrix comparing the sequential Jacobi.

Besides, considering the speedup with different processor number, it is consistent with the runtime we've collected before that, due to the additional communication time, the speedup doesn't increase as expected when we increase the processor number.

On the other hand, the initial efficiency is extremely low due to the fact that the parallel version needs the communication between processors. With the problem size increases, the efficiency of the program would be lot better, but still due to the communication between processor and the limit of the bandwidth, the efficiency is still less than 0.4. However, the number of processor actually matters with respect to the efficiency. The efficiency of the algorithm is larger with more processors as we can see from Figure 5.

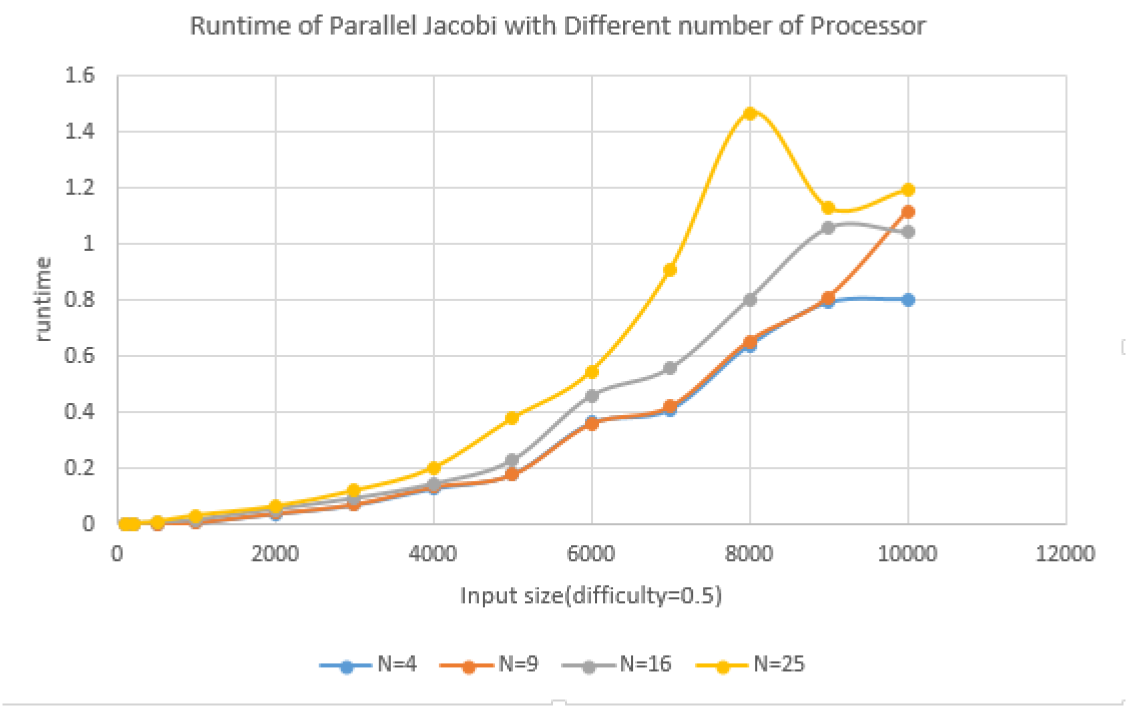

Figure 3: Runtime for Parallel Jacobi method

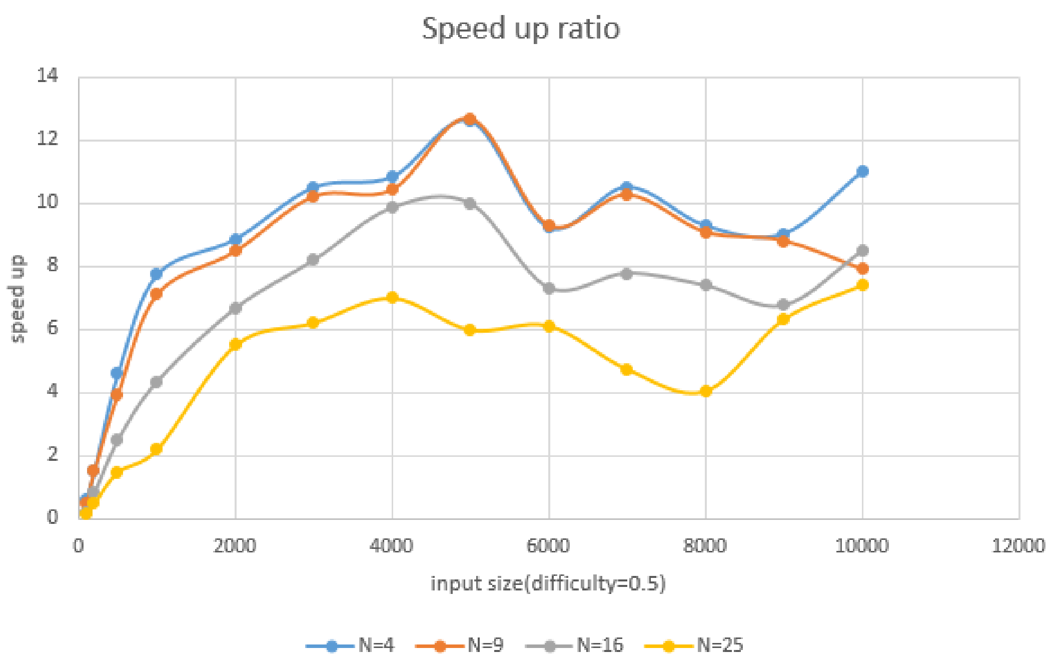

Figure 4: Speed up with different number of processors

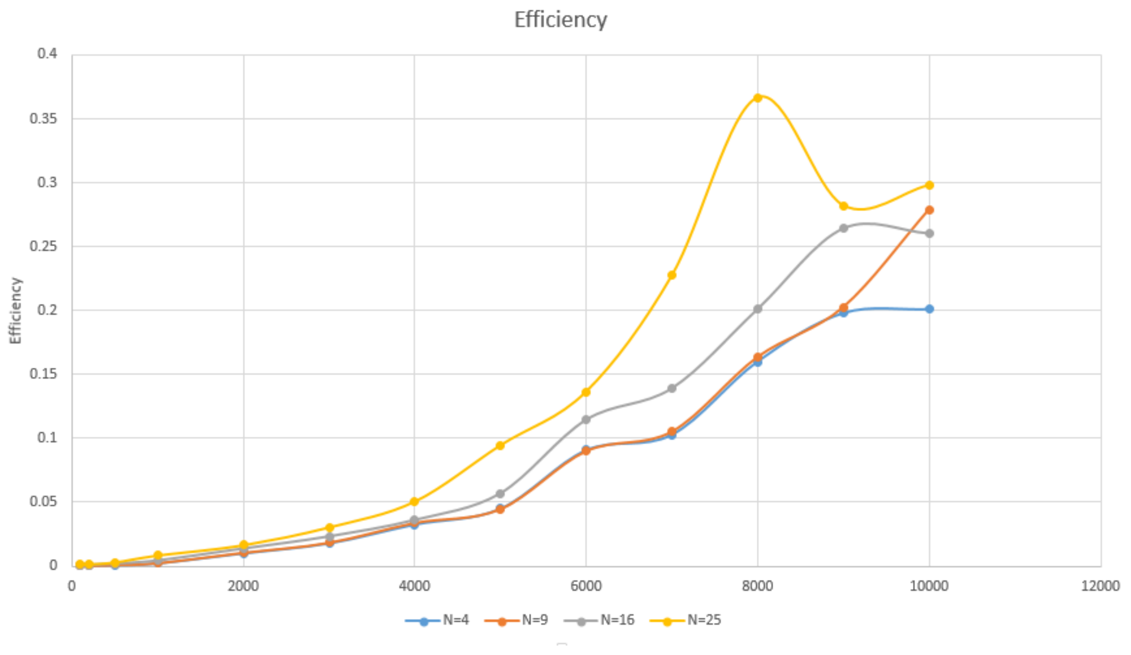

Figure 5: Efficiency of the Parallel Jacobi method

Conclusion and Possible Improvement

We typically run our experiments through changing the size of the input and the number of processor involved to get the results show above. In the parallel version, we could see the impact of the communication between processors on the total runtime of the program, and could see how the number of the processor would actually affect the final results of the experiment.

Just as the result shows above, the communication time is the key component that counteract the speedup and efficiency of the parallel Jacobi. Although the communication time is inevitable in parallel algorithm, we still need to improve our communication protocol in our code to confirm better performance. This is the point where we might be able to make some further improvement over parallel Jacobi.