Implementation and Analysis of Jacobi-Schwarz with RMA

CSE6230 - Final Project

Quan Zhou
Quantitative and Computational Finance
Georgia Institute of Technology
Email: qzhou81@gatech.edu

***Abstract*—Jacobi-Schwarz, a.k.a Schwarz Method, is a parallel algorithm that iteratively solves large scale linear systems for boundary value problems. It has shared regions across subdomains and thus converges faster than Block Jacobi. In this project, Jacobi-Schwarz is implemented with Remote Memory Access and multi-thread Intel Math Kernel Library. Multiple shared windows are used to reduce runtime. Comparing to MPI send/recv, implementation with RMA is three times slower. Result also shows that with different grow values, optimal number of threads differs.**

I. Introduction

A boundary value problem is a differential equation together with a set of additional constraints, called the boundary conditions. Boundary value problems are widely used in several branches in physics[1]. For example, given a temperature distribution in a square room and heat sources and sinks, solve for the temperature at each grid point. In this problem, a linear equation system shows the connection between adjacent grids with boundary conditions that the values on the boundary of the room are known.

There exists many efficient single process solver to solve such a boundary value problem. However, as the scale of the problem grows to the order of millions of grids, it is no longer solvable using singe process because storing the entire linear system requires huge memory and solving time could be extremely long. Thus, parallel solving algorithms such as Jacobi-Schwarz should be used instead.

In this project, I implemented Jacobi-Schwarz in two ways using MPI send/recv and RMA to divide the huge problem into much smaller local problem. Then the local linear equation system is solved by Intel Math Kernel Library with multi-threading.

The rest of the report is organized as following. Section II will introduce the temperature problem in details. Design and optimization of the codes are explained in Section III. Section VI will introduce experiment results, followed by conclusion in Section V.

II. Problem Definition

A. Room Temperature

Given a square room, divide it into n by n grids. Each grid has a number indicating its temperature. The 2D problem is solving the temperature at each grid point.

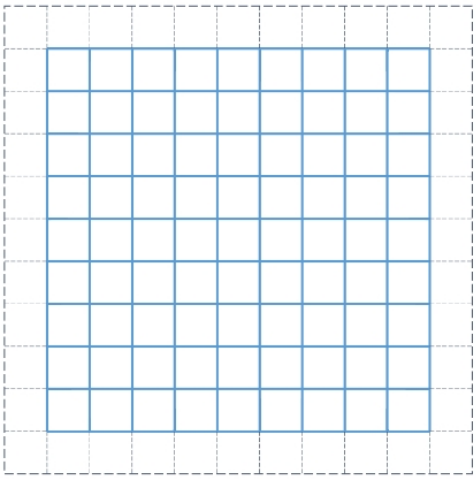

Fig. 1. n by n temperature grids

All values at the boundary are known. In this project, all boundary values are set to zero. To

simplify the problem, the connection between each grid and its neighbors is shown in (1).

$$4u_c = u_N + u_W + u_S + u_E + f_c \tag{1}$$

where u_c denotes the temperature at grid c, u_N, u_W, u_S, u_E denote the temperatures of its neighbors and f_c denotes the source or sink at grid c. Then the 2D temperature problem can be formulated as the following matrix format:

$$Au = f$$

B. Jacobi-Schwarz

Since n could be as large as 10^3, it is much more efficient to use parallel algorithms to solve the problem. Using Jacobi-Schwarz will significantly reduce runtime and optimize memory allocation.

The Jacobi-Schwarz method divides the grids into many subdomains. Each subdomain only stores local values and it is a similar local boundary value problem, sharing the boundaries with adjacent subdomains.

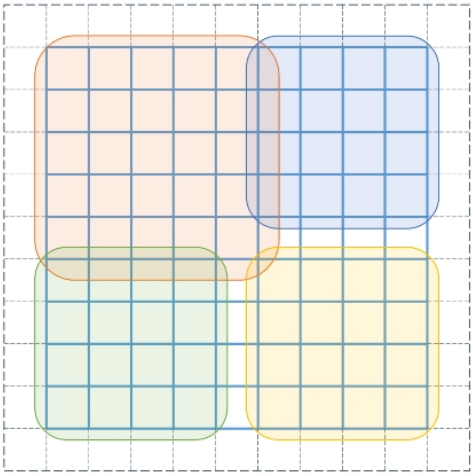

Fig. 2. subdomains in Jacobi-Schwarz

To iteratively solve the entire problem, each subdomain first send and receive the values from its neighbors. Then solve the local linear system. The process ends when residual is small enough or number of iterations reaches a user defined threshold.

III. Parallel Design and Optimization

To parallelize the algorithm, each subdomain is a process and runs its own linear system solver using Intel MKL. Since the boundary values are not locally stored, remote memory access is used to get values from another subdomain. Each subdomain has 4 separate shared windows, which are the top/bottom k rows and left/right k columns, communicating with its 4 neighbors(top, bottom, left, right). k is the grow value. Also, the neighbors in 4 corners can get the values they need from these 4 windows too.

Using multiple windows instead of a single giant window can reduce the cases where multiple processes try to read data from the same window. It means the lock time of the data should be reduces. However, on the other hand, there are more locks to acquire and release. It is a trade-off between the number of locks and the holding time of the locks. More detailed experiment results are shown in Section V.

Figure 3 shows the solving process of each process. After generating a random initial guess, the major iteration starts. It continuously solves the local problem until the global residual is small enough of number of iterations reaches its limit.

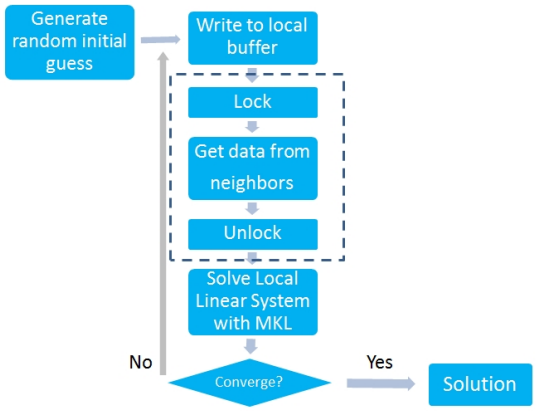

Fig. 3. solving process

Two barriers are used in each iteration. Firstly, make sure that writing local buffers is complete before allowing other processes to read from the buffer. Secondly, start computing global residual norm only when all the processes finish solving their local systems and have the correct local residual.

There two barriers and the locks guarantees that all the data is ready when it is needed and no race condition exists.

IV. Experiment Result

All the experiments were run on Gotham mic4. Problem scale is 10 by 6 processes and each local problem has 100 by 100 grids. Default process pinning was used.

The comparison between using MPI send/recv, single-window RMA and multi-window RMA is shown in Figure 4. X-axis is grow value and y-axis is time per iteration. MPI send/recv was used as time baseline. Each process uses single thread in MKL.

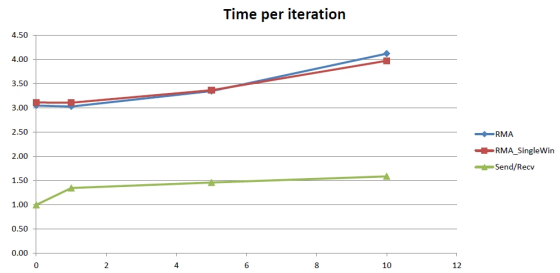

Fig. 4. MPI send/recv, single-window RMA and multi-window RMA

These three methods have exactly the same results because the solving process is the same. The only difference is how to communicate between processes. According to the data, using RMA is approximately three times slower than using MPI send/recv. And using a single window for RMA is better if growth is large. The reason is that if growth is large, the overlapping area at the corners becomes larger, which is less efficient. But larger growth won't affect single-window RMA since all the values are still only stored once.

With multi-threading in MKL, the results are shown in Figure 5. The baseline for time per iteration is using single thread and growth being zero. For different grow value, optimal number of threads are different. When grow=0, using 4 threads has the best performance. When grow=1, using 2 threads has the best performance. When grow=10, using 4 threads has the best performance. To get more accurate runtime, maximum number of iterations is set to 500.

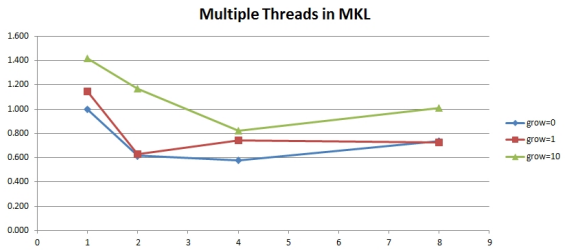

Fig. 5. Multi-threading in MKL

V. Conclusion

In this project, Jacobi-Schwarz method is implemented with RMA and multi-threads in Intel MKL. Compared with MPI send/recv, RMA is less efficient for transferring data between processes. However, it is a more flexible way to access data in another process. When allocating shared window, the process doesn't need to know which process will use its data. On the other hand, MPI send/recv requires a very clear statement of which two processes are communicating.

References

[1] https://en.wikipedia.org/wiki/Boundary_value_problem