Performance Comparison between Blocking and Non-Blocking Communications for a Three-Dimensional Poisson Problem

Guan Wang and Matthias K. Gobbert

Department of Mathematics and Statistics, University of Maryland, Baltimore County

Abstract

The system of linear equations obtained by finite difference discretization of the three-dimensional Poisson problem is solved by a matrix-free parallel implementation of the Jacobi method. After checking the convergence of our implementation, we compare the speed of using blocking `MPI_Ssend/MPI_Recv` and non-blocking `MPI_Isend/MPI_Irecv` communications. The comparisons indicate that the non-blocking communications are slightly faster than the blocking ones but show the same speedup and efficiency behavior. The OpenMPI implementation of MPI is used in conjunction with the Portland Group C compiler.

1 Introduction

Our goal is to solve the Poisson equation with homogeneous Dirichlet boundary conditions on the unit cube in three spatial dimensions numerically. The numerical method we will use is the finite difference method using the conventional seven-point stencil, resulting a large, sparse, and highly structured system of linear equations. In Section 2 this problem will be stated in detail, and Section 3 will introduce the derivation of the iterative method and prepare for the parallel implementation.

All numerical studies for this report were performed on the distributed-memory cluster hpc.rs in the UMBC High Performance Computing Facility (HPCF). The cluster has 33 compute nodes, each with two dual-core AMD Opteron 2.66GHz (1 MB cache per core) and 13 GB memory, connected by a high performance InfiniBand interconnect network. The operating system is RedHat Enterprise Linux 5 distribution. We use the Portland Group C compiler and the OpenMPI implementation of MPI.

2 Problem Statement

Consider 3-D Poisson problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{2.1}$$

with $\Omega = (0,1)^3 \subset \mathbb{R}^3$. To discretize the domain for numerical methods, we discretize the x, y, and z directions by $N+1$ small intervals with equal length $h = 1/(N+1)$. Then we get a square mesh given by

$$\begin{aligned} x_i &= i\,h = \frac{i}{N+1}, \quad i = 0, \dots, N+1 \\ y_j &= j\,h = \frac{j}{N+1}, \quad j = 0, \dots, N+1 \\ z_k &= k\,h = \frac{k}{N+1}, \quad k = 0, \dots, N+1, \end{aligned}$$

There are $N+2$ points on each coordinate direction, and in the interior there are N^3 mesh points. Using finite differences, we approximate the second partial derivatives by

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j, z_k) \approx \frac{u(x_{i-1}, y_j, z_k) - 2u(x_i, y_j, z_k) + u(x_{i+1}, y_j, z_k)}{h^2},$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j, z_k) \approx \frac{u(x_i, y_{j-1}, z_k) - 2u(x_i, y_j, z_k) + u(x_i, y_{j+1}, z_k)}{h^2},$$

$$\frac{\partial^2 u}{\partial z^2}(x_i, y_j, z_k) \approx \frac{u(x_i, y_j, z_{k-1}) - 2u(x_i, y_j, z_k) + u(x_i, y_j, z_{k+1})}{h^2},$$

and then we have

$$\Delta u(x_i, y_j, z_k) = \frac{\partial^2 u}{\partial x^2}(x_i, y_j, z_k) + \frac{\partial^2 u}{\partial y^2}(x_i, y_j, z_k) + \frac{\partial^2 u}{\partial z^2}(x_i, y_j, z_k)$$
$$\approx \frac{1}{h^2}\left(u_{i,j,k-1} + u_{i,j-1,k} + u_{i-1,j,k} - 6u_{i,j,k} + u_{i+1,j,k} + u_{i,j+1,k} + u_{i,j,k+1}\right),$$

as an approximation to the Laplace operator Δu and $u_{i,j,k} \approx u(x_i, y_j, z_k)$. The final discretized form of the equation is

$$-u_{i,j,k-1} - u_{i,j-1,k} - u_{i-1,j,k} + 6u_{i,j,k} - u_{i+1,j,k} - u_{i,j+1,k} - u_{i,j,k+1} = h^2\, f_{i,j,k},$$

for $i, j, k = 1, \ldots, N$, with $f_{i,j,k} = f(x_i, y_j, z_k)$ for short.

3 Jacobi Method

The discretized system can be written in matrix form as

$$Au = b$$

with system matrix

$$A = \begin{bmatrix} S & -I_{N^2} & & & \\ -I_{N^2} & S & -I_{N^2} & & \\ & \ddots & \ddots & \ddots & \\ & & -I_{N^2} & S & -I_{N^2} \\ & & & -I_{N^2} & S \end{bmatrix},$$

where

$$S = \begin{bmatrix} T & -I_N & & & \\ -I_N & T & -I_N & & \\ & \ddots & \ddots & \ddots & \\ & & -I_N & T & -I_N \\ & & & -I_N & T \end{bmatrix}, \quad T = \begin{bmatrix} 6 & -1 & & & \\ -1 & 6 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 6 & -1 \\ & & & -1 & 6 \end{bmatrix},$$

and $I_{N^2} \in \mathbb{R}^{N^2 \times N^2}$ and $I_N \in \mathbb{R}^{N \times N}$ are identity matrices, and the right hand side is

$$b_{i,j,k} = h^2\, f_{i,j,k}.$$

We can see that $A \in \mathbb{R}^{N^3 \times N^3}$, and u and b are column vectors with a length N^3.

The source function in the right-hand side is given by

$$f(x,y,z) = (-2\pi^2)\left(\cos(2\pi x)\sin^2(\pi y)\sin^2(\pi z) + \sin^2(\pi x)\cos(2\pi y)\sin^2(\pi z) + \sin^2(\pi x)\sin^2(\pi y)\cos(2\pi z)\right),$$

and the analytic solution is $u_{\text{true}}(x,y,z) = \sin^2(\pi x)\sin^2(\pi y)\sin^2(\pi z)$; this problem is the generalization of the two-dimensional example used in [1].

Since N is usually very large, and N^3 is even much larger, it is often difficult to use direct methods to solve this linear system. Iterative methods are more often used here. In this report, we choose Jacobi method to compute the numerical solution. The algorithm and derivation of Jacobi method are stated in [4]. And to avoid computing the huge matrix-vector product above, we use a matrix-free implementation to solve the system. In this problem, the Jacobi method can then be explicitly programmed as

$$u_{i,j,k}^{(m+1)} = \frac{1}{6}\left(b_{i,j,k} + u_{i,j,k-1}^{(m)} + u_{i,j-1,k}^{(m)} + u_{i-1,j,k}^{(m)} + u_{i+1,j,k}^{(m)} + u_{i,j+1,k}^{(m)} + u_{i,j,k+1}^{(m)}\right)$$

with iteration counter $m = 0, 1, 2, \ldots$. Here, u is an $N \times N \times N$ array, and we use x-y-z ordering to organize u as an N^3 column vector that $u_{i,j,k} = u_{i+Nj+N^2k}$ for $i, j, k = 1, \ldots, N$.

When we do parallel computing here, we need to split the array u in the z-direction onto the p MPI processes. If we let $l\_N = N/p$, the size of the local arrays l_u are $N \times N \times l\_N$.

4 Numerical Results

4.1 Correctness and Convergence Check

The first thing we need to check is the correctness of our implementation. We set the tolerance $tol = 10^{-6}$, and we start our iteration with an initial guess $u^{(0)} = 0$. We start with a relatively small $N = 16$, which yields a mesh with $18 \times 18 \times 18$ points and $h = 1/17 = 0.058823$. Unlike the 2-D case, we cannot clearly see the graph of the function. So we only compare the $L^\infty(\Omega)$-norm of the error between u and the true solution u_{true}. Table 1 shows a convergence study for some mesh resolutions N that yields degrees of freedom (DOF) N^3 for the three-dimensional $N \times N \times N$ mesh. This tells us that the Jacobi method converges to the true solution, but it takes progressively much larger numbers of iterations `#iter` and progressively longer times of computation (TOC). All test performances in Table 1 are run on only one process. In the following, we will focus on parallel computing.

Parallelization should not change the algorithm at all, so all numerical results should be consistent with the serial case. We take $N = 16$, and let the number of processes p go from 1 to 8,

Table 1: Convergence study of the Jacobi method.

N	DOF	`#iter`	$\|u - u_{\text{true}}\|_{L^\infty(\Omega)}$	TOC (sec.)
16	4,096	728	1.1169278486920731e-02	5.70090e-02
32	32,768	2675	3.0027080376821003e-03	1.71493e+00
64	262,144	10095	7.7305748779854522e-04	8.79645e+01
128	1677,7216	38611	1.9112583060509891e-04	1.83779e+03

Table 2: Consistency of parallelization of the Jacobi method for $N = 16$.

p	`#iter`	$\|u - u_{\text{true}}\|_{L^\infty(\Omega)}$	TOC
1	728	1.1169278403210803e-02	5.10090e-02
2	728	1.1169278403210803e-02	2.25871e-02
4	728	1.1169278403210803e-02	2.08211e-02
8	728	1.1169278403210803e-02	7.08668e-02

and look at `#iter`, $\|u - u_{\text{true}}\|_{L^\infty(\Omega)}$, and the time of computation (TOC). From Table 2 we can observe that `#iter` and $\|u - u_{\text{true}}\|_{L^\infty(\Omega)}$ are highly consistent, but the wall clock time does not show a good regularity here: From $p = 1$ to $p = 2$ and $p = 4$, the time of computation gets faster, but it slows down to $p = 8$, since the time of communication dominates over the time of calculation in this case. For further studies on parallel performance, we need to consider larger N.

4.2 Numerical Results for Blocking Communications

In the parallel computing, on Process id, all interior points of this process's subdomain can be directly computed by (3). To compute the points on the top and bottom faces of the subdomain, points on the bottom face from Process $id + 1$ and points on the top face from Process $id - 1$ are needed, respectively. We call these points from neighboring processes "ghost points". Here we firstly use `MPI_Ssend` and `MPI_Recv` for communications between neighboring processes. The function `MPI_Ssend` sends a message so that the send does not return until the destination begins to receive the message [2]. This is called blocking communication.

Now we test for N from 64 to 512, with N doubled each time. We will look at the performance for p from 1 to 32 for $N = 64$, from 1 to 64 for $N = 128$, and from 1 to 128 for the other. Besides the wall clock time, we compute two more quantities, speedup and efficiency. The speedup $S_p = T_1(N)/T_p(N)$ is the ratio of wall clock time for serial execution over that of parallel execution with p processes, whose optimal value is p. The efficiency $E_p = S_p/p$, whose optimal value is 1, describes how close a parallel execution with p processes is to the optimal value.

Considering the slow rate of convergence of Jacobi method and the cubically increasing DOF, when N is large, the number of iterations for convergence will be unacceptably huge. In this case, we can set a relatively small maximum number of iterations, and look at the time spent on this many iterations, but we do not expect convergence of the solution. Notice that as N is larger, for the same maximum number of iterations, the computation takes a much longer time. So here we set $maxit = 99{,}999$ for $N = 64$, $maxit = 9{,}999$ for $N = 128$, $maxit = 999$ for $N = 256$, and $maxit = 99$ for $N = 512$.

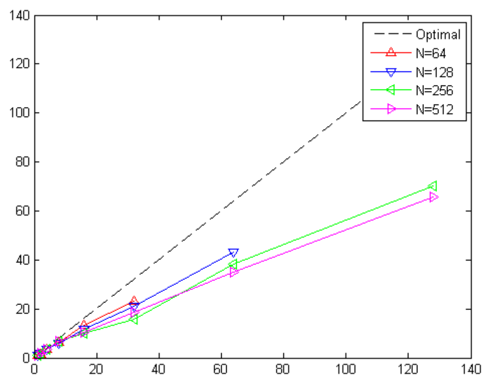
Table 3 and Figure 1 show the timing results of our performances. Table 3 (a) shows the wall clock time for each N. Here for $p = 1$, we can use only 1 process per node, 2 processes per node for $p = 2$, and for $p \geq 4$, we always use 4 processes per node. Since we use different maximum numbers of iterations for each N, the wall clock time in each column are not comparable. In each row of Table 3, we can see for the same N, as p is doubled, the wall clock time becomes approximately half of the previous one, mostly a little more than a half when $p \geq 4$, as a result of more time spent on communications between different processes. Consequently, in Figure 1 (a) the speedup S_p slows down as p increases and in Figure 1 (b) the efficiency E_p also decreases eventually.

Table 3: Timing Results for Blocking Communications.

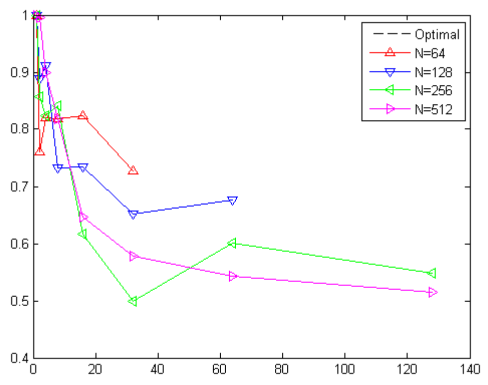
(a) Wall clock time in seconds								
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
64	87.20	57.31	26.59	13.32	6.62	3.75	N/A	N/A
128	1023.38	575.73	280.54	174.49	87.21	49.01	23.67	N/A
256	588.54	343.46	178.57	87.43	59.73	36.89	15.32	8.39
512	396.56	199.20	110.25	60.52	38.34	21.47	11.40	6.03

(b) Observed speedup S_p								
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
64	1.0000	1.5215	3.2791	6.5460	13.1657	23.2466	N/A	N/A
128	1.0000	1.7775	3.6479	5.8650	11.7344	20.8818	43.2381	N/A
256	1.0000	1.7136	3.2958	6.7318	9.8533	15.9542	38.4244	70.1106
512	1.0000	1.9908	3.5968	6.5525	10.3421	18.4699	34.7976	65.8107

(c) Observed efficiency E_p								
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
64	1.0000	0.7607	0.8198	0.8183	0.8229	0.7265	N/A	N/A
128	1.0000	0.8888	0.9120	0.7331	0.7334	0.6526	0.6756	N/A
256	1.0000	0.8568	0.8420	0.8415	0.6158	0.4986	0.6004	0.5477
512	1.0000	0.9954	0.8992	0.8191	0.6464	0.5772	0.5437	0.5141



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 1: Speedup and Efficiency for Blocking Communications.

4.3 Numerical Results for Non-Blocking Communications

Now we want to compare the above blocking results with non-blocking communication with the use of `MPI_Isend` and `MPI_Irecv`. A non-blocking communication is an approach that allows users to start sending and receiving several messages, while proceeding with other operations. It is recognized as a way of compensating for the relatively slow speed of communications as compared to numerical calculations [3]. The arguments for `MPI_Isend/MPI_Irecv` are very similar as those for `MPI_Send/MPI_Recv`, except adding a handle argument used to determine whether an operation has completed. The commands `MPI_Wait` or `MPI_Waitall` are used to wait for one send or all sends complete [2], respectively. To implement this, we just replace where we use `MPI_Send/MPI_Recv` by `MPI_Isend/MPI_Irecv` and add a wait command right before using the ghost points. In between the posting of the `MPI_Isend/MPI_Irecv` and the wait command, we calculate all interior points of the local process's subdomain.

We test with the same parameters as in the blocking case. The results are shown in Table 4 and Figure 2. Looking at Table 4 (a), we can observe that in most case, for the corresponding problem size and number of processes, the non-blocking communication has a little faster speed than the blocking communication. The speedup and the efficiency show a similar tendency as in the blocking case.

Acknowledgments

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

References

[1] Matthias K. Gobbert. Parallel performance studies for an elliptic test problem. Technical Report HPCF–2008–1, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2008.

[2] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, second edition, 1999.

[3] Peter S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, 1997.

[4] David S. Watkins. *Fundamentals of Matrix Computations.* Wiley, second edition, 2002.

Table 4: Timing Results for Non-Blocking Communications.

(a) Wall clock time in seconds

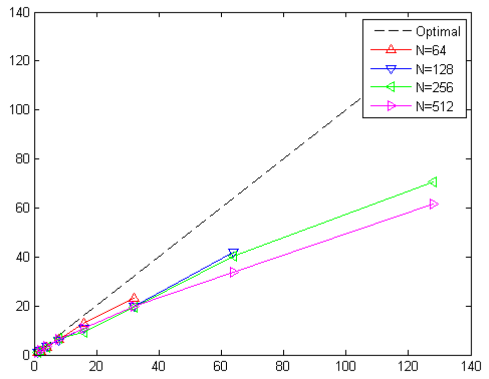
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
64	86.23	55.96	26.68	13.94	6.65	3.71	N/A	N/A
128	922.78	546.66	272.13	158.42	85.04	46.31	22.07	N/A
256	576.77	326.11	176.72	87.02	62.42	29.75	14.28	8.16
512	366.00	189.30	106.08	57.82	33.45	18.46	10.84	5.92

(b) Observed speedup S_p

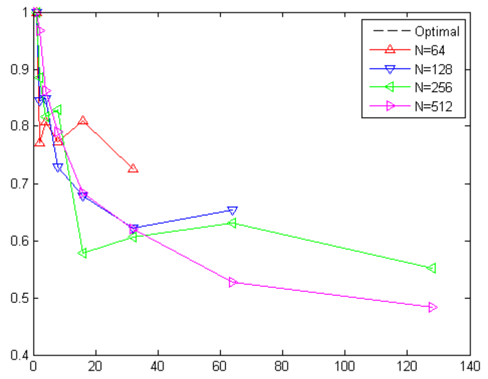
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
64	1.0000	1.5411	3.2327	6.1859	12.9651	23.2334	N/A	N/A
128	1.0000	1.6880	3.3909	5.8249	10.8510	19.9251	41.8022	N/A
256	1.0000	1.7686	3.2637	6.6276	9.2398	19.3880	40.3840	70.6530
512	1.0000	1.9334	3.4502	6.3296	10.9432	19.8247	33.7509	61.8055

(c) Observed efficiency E_p

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
64	1.0000	0.7705	0.8082	0.7732	0.8103	0.7260	N/A	N/A
128	1.0000	0.8440	0.8477	0.7281	0.6782	0.6227	0.6532	N/A
256	1.0000	0.8843	0.8159	0.8284	0.5775	0.6059	0.6310	0.5520
512	1.0000	0.9667	0.8626	0.7912	0.6839	0.6195	0.5274	0.4829



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 2: Speedup and Efficiency for Non-Blocking Communications.