

Parallel Performance Studies for a Parabolic Test Problem on the Cluster tara

Michael Muscedere, Andrew M. Raim, and Matthias K. Gobbert (gobbert@umbc.edu)

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

Technical Report HPCF-2010-4, www.umbc.edu/hpcf > Publications

Abstract

The performance of parallel computer code depends on the intricate interplay of processors, the architecture of the computer nodes, their interconnect network, the numerical algorithm, and its implementation. The solution of large, sparse, highly structured equations of linear equations by an iterative linear solver that requires communication between the parallel processes at every iteration is an instructive test of this interplay. This note considers a parabolic test problem given by a time-dependent, scalar, linear reaction-diffusion equation in three dimensions, whose time-stepping requires the solution of such a system of linear equations at every timestep. The results presented here show excellent performance on the cluster tara with up to 512 parallel processes when using 64 compute nodes. The results support the scheduling policy implemented, since they confirm that it is beneficial to use all eight cores of the two quad-core processors on each node simultaneously, giving us in-effect a computer that can run jobs efficiently with up to 656 parallel processes when using all 82 compute nodes. The cluster tara is an IBM server x iDataPlex purchased in 2009 by the UMBC High Performance Computing Facility (www.umbc.edu/hpcf). It is an 86-node distributed-memory cluster comprised of 82 compute, 2 develop, 1 user and 1 management nodes. Each node features two quad-core Intel Nehalem X5550 processors (2.66 GHz, 8 MB cache), 24 GB memory, and a 120 GB local hard drive. All nodes and the 160 TB central storage are connected by an InfiniBand (QDR) interconnect network.

1 Introduction

The spatial discretization of a test problem comprised of partial differential equations given by a time-dependent linear reaction-diffusion equation in three space dimensions results in a large system of ordinary differential equations (ODEs). This ODE system is solved by the family of numerical differentiation formulas [9]. Since these ODE solvers are implicit, a system of linear equations needs to be solved at every time step. The system matrices of these systems are large, sparse, highly structured, and symmetric positive definite, and we use a matrix-free implementation of the conjugate gradient (CG) method to solve them, as in [8]. The present report generalizes these studies for an elliptic test problem in [8] to a parabolic test problem, although also the spatial dimensions of the test problems (three vs. two) are different and there are important differences in the behavior of the algorithms: The CG method for the elliptic test problem requires very large numbers of iterations, by contrast, the number of CG iterations in each time step of a time-dependent problem is very limited due to the good initial solution guess available from the previous timestep. This is significant, because the key challenge for the parallel interconnect stems from the CG iterations and not from the time stepping. Section 2 details the test problem and describes its parallel implementation including a discussion covering the convergence of the numerical approximation over successively finer mesh sizes.

This report repeats the studies of [6] which examined the parallel performance of the same parabolic test problem for the cluster hpc, which contained two dual-core processors capable of four processes on each node, on the new cluster tara, which contains two quad-core processor capable of eight processes on each node. In detail, tara is an 86-node distributed-memory cluster purchased in 2009 by the UMBC High Performance Computing Facility (www.umbc.edu/hpcf) comprised of 82 compute, 2 develop, 1 user, and 1 management nodes. Each node features two quad-core Intel Nehalem processors (2.66 GHz, 8 MB cache), 24 GB memory, and a 120 GB hard drive, thus up to 8 parallel processes can run simultaneously per node. All nodes and the 160 TB central storage are connected by an InfiniBand (QDR = quad-data rate) interconnect network. The cluster is an IBM System x iDataPlex.¹ An iDataPlex rack uses the same floor space as a conventional 42 U high rack but holds up to 84 nodes, which saves floor space. More importantly, two nodes share a power supply which reduces the power requirements of the rack and makes it potentially run cooler and more environmentally friendly than the standard racks.² For tara, the iDataPlex rack houses the 84 compute and develop nodes and includes all other components associated with the nodes such as power distribution and Ethernet switches. The user and management nodes with their larger form factor are contained second, standard rack along with the InfiniBand switch. The PGI 9.0 C compiler has been used to create the executable which were used in this report.

¹Vendor page www-03.ibm.com/systems/x/hardware/idadaplex/

²Press coverage for instance www.theregister.co.uk/2008/04/23/ibm.idataplex/

Section 3 describes the parallel performance studies in detail and provides the underlying data for the following summary results. Table 1.1 summarizes the key results of the present study by giving the observed wall clock time (total time to execute the code) in HH:MM:SS (hours:minutes:seconds) format. We consider the test problem on six progressively finer meshes, resulting in progressively larger systems of linear equations with system dimensions ranging from about 140,000 to over 4.3 billion equations. The parallel implementation of the numerical method is run on increasing numbers of nodes from 1 to 64 while varying the number of processes per node from 1 to 8. The upper-left entry of each sub-table in Table 1.1 (i.e., 1 process per node on 1 node) represents the serial run of the code. The lower-right entry of each sub-table lists the time using all cores of both quad-core processors in all 64 nodes, for a total of 512 parallel processes working together to solve the problem. Specifically for the $128 \times 128 \times 512$ spatial mesh that results in a system of about 8.5 million equations to be solved, the serial run takes over 46 minutes, while the run using all cores on all 64 nodes takes about 12 seconds. Even more strikingly, one realizes the advantage of parallel computing for the larger $256 \times 256 \times 1024$ mesh with over 67.7 million equations: The serial run of about 11 hours can be reduced to about 2 minutes using 512 parallel processes. Refining the mesh to $512 \times 512 \times 2048$ yields a problem with over 500 million equations. Due to memory limitations, this mesh size can only be solved by using the total memory of 8 or more nodes. This brings out one of the key advantages of parallel computing: pooling the memory from several nodes allows the solution of larger problems than could be solved on any single node, no matter the time needed to solve the problem. Furthermore, refining the mesh to $1024 \times 1024 \times 4096$ yields a problem with over 4.3 billion equations. Due to memory limitations, this mesh size can only be solved by using all 64 nodes. Table 1.1 shows that our largest sized problem is solved in a respectable time of about 5 hours using all cores on the 64 nodes.

The summary results in Table 1.1 are arranged to study two key questions: (i) whether the code scales linearly to 64 nodes, which ascertains the quality of the InfiniBand interconnect network, and (ii) whether it is worthwhile to use multiple processors and cores on each node, which analyzes the quality of the architecture of the nodes and in turn guides the scheduling policy (whether it should be the default to use all cores on a node or not).

- (i) Reading along each row of Table 1.1, speedup in proportion to the number of nodes is observed. This is discussed in detail in Section 3 in terms of the number of parallel processes. The results show some experimental variability with better-than-expected results when using in the finer mesh sizes $256 \times 256 \times 1024$ and $1024 \times 1024 \times 4096$ in the MVAPICH2 code implementation. Interestingly, when reading along the rows of Table 1.2 these better-than-expected results are more frequent in the OpenMPI implementation of the code. But more remarkably, there is a nearly halving of the execution time for the finer meshes $128 \times 128 \times 512$, $256 \times 256 \times 1024$, and $512 \times 512 \times 2048$ in all columns out to final column using 64 nodes. These excellent results successfully demonstrate the scalability of the algorithm and its implementation up to very large number of nodes, as well as highlight the quality of the new quad-data rate InfiniBand interconnect.
- (ii) To analyze the effect of running 1, 2, 4, or 8 parallel processes per node, we compare the results column-wise in each sub-table. It is apparent that the execution time of each problem is in fact roughly halved with doubling the numbers of processes within each node used. These result confirm that it is not just effective to use both processors on each node, but also to use all cores of each quad-core processor simultaneously. Roughly, this shows that the architecture of the IBM nodes purchased in 2009 has sufficient capacity in all vital components to avoid creating any bottlenecks in accessing the memory of the node that is shared by the processes. These results thus justify the purchase of compute nodes with two processors (as opposed to one processor) and of multi-core processors (as opposed to single-core processors). Moreover, these results will guide the scheduling policy implemented on the cluster. On the one hand, it is not disadvantageous to run several serial jobs simultaneously on one node, and on the other hand, for jobs using several nodes, it is advantageous to make use of all cores on the nodes reserved by the scheduler.

The entries of Table 1.1 used the MVAPICH2 implementation of MPI. They may be compared with the results in Table 1.2, where OpenMPI has been used. From the raw timings it is difficult to declare a clear winner between the two implementations, but a careful comparison highlights a few patterns. Fixing our attention to the largest problem size with a complete set of data for all p values ($N_x \times N_y \times N_z = 256 \times 256 \times 1024$), we notice that when more nodes are in use (16, 32, and 64) the MVAPICH2 timings seem to reduce more quickly than the OpenMPI timings as processes-per-node increase. When the number of nodes used is less than 16, the pattern does not seem to be clear. Sometimes, OpenMPI achieves a better time, and sometimes it does not. However, we notice in a few cases that the difference is large. For example, consider the case of 2 nodes and 1 process per node; in this case, MVAPICH2 achieves a time of 04:54:49, and OpenMPI trails behind at 05:21:03. But with 1 node and 1 process per node, the MVAPICH2 time of 10:58:00 is far worse than than the OpenMPI time of 09:49:05. For coarser meshes, the differences in absolute times are usually small. In Section 4, we see that the efficiency of OpenMPI

Table 1.1: Wall clock time in HH:MM:SS on tara using MVAPICH2 for the solution of the parabolic test problem on $N_x \times N_y \times N_z$ meshes using 1, 2, 4, 8, 16, 32, and 64 compute nodes with 1, 2, 4, and 8 processes per node.

(a) Mesh resolution $N_x \times N_y \times N_z = 32 \times 32 \times 128$, DOF = 140,481							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:00:35	00:00:17	00:00:09	00:00:06	00:00:04	00:00:03	00:00:02
2 processes per node	00:00:18	00:00:09	00:00:06	00:00:05	00:00:04	00:00:03	00:00:02
4 processes per node	00:00:11	00:00:06	00:00:05	00:00:04	00:00:04	00:00:02	N/A
8 processes per node	00:00:08	00:00:06	00:00:05	00:00:04	00:00:03	N/A	N/A
(b) Mesh resolution $N_x \times N_y \times N_z = 64 \times 64 \times 256$, DOF = 1,085,825							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:04:33	00:02:35	00:01:10	00:00:40	00:00:21	00:00:12	00:00:07
2 processes per node	00:02:19	00:01:09	00:00:40	00:00:22	00:00:12	00:00:07	00:00:05
4 processes per node	00:01:14	00:00:38	00:00:23	00:00:12	00:00:07	00:00:05	00:00:04
8 processes per node	00:00:42	00:00:21	00:00:12	00:00:07	00:00:04	00:00:04	N/A
(c) Mesh resolution $N_x \times N_y \times N_z = 128 \times 128 \times 512$, DOF = 8,536,833							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:46:14	00:23:44	00:12:09	00:06:37	00:03:23	00:01:45	00:00:52
2 processes per node	00:23:38	00:11:52	00:06:40	00:03:25	00:01:49	00:00:58	00:00:30
4 processes per node	00:12:54	00:06:20	00:03:47	00:02:02	00:00:59	00:00:33	00:00:18
8 processes per node	00:07:07	00:03:53	00:02:00	00:00:59	00:00:32	00:00:20	00:00:12
(d) Mesh resolution $N_x \times N_y \times N_z = 256 \times 256 \times 1024$, DOF = 67,700,225							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	10:58:00	04:54:49	02:45:21	01:22:05	00:41:33	00:20:56	00:10:36
2 processes per node	04:48:32	02:26:29	01:24:19	00:40:52	00:21:10	00:10:53	00:05:40
4 processes per node	03:04:01	01:22:23	00:44:27	00:23:53	00:12:33	00:06:17	00:03:04
8 processes per node	01:27:12	00:43:35	00:22:03	00:11:52	00:06:04	00:03:33	00:01:57
(e) Mesh resolution $N_x \times N_y \times N_z = 512 \times 512 \times 2048$, DOF = 539,233,281							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	N/A	N/A	N/A	18:58:45	09:36:22	05:01:33	02:27:50
2 processes per node	N/A	N/A	N/A	09:26:10	04:46:39	02:24:10	01:12:10
4 processes per node	N/A	N/A	N/A	05:25:31	02:49:06	01:23:08	00:43:20
8 processes per node	N/A	N/A	N/A	02:43:45	01:21:29	00:40:56	00:22:11
(f) Mesh resolution $N_x \times N_y \times N_z = 1024 \times 1024 \times 4096$, DOF = 4,304,410,625							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	N/A	N/A	N/A	N/A	N/A	N/A	N/A
2 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	N/A
8 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	04:50:18

tends to be worse than MVAPICH2 when the mesh resolutions are large, as the number of process scale up. Some of the differences observed here may be due to experimental variability, and more studies (also with other codes) will be useful. Nevertheless, due to its potential to perform better for large problems (for code with large memory requirements) for large number of nodes and, importantly, with all cores on the nodes in use, MVAPICH2 was chosen as the default MPI implementation for tara. This choice potentially allows for the most effective use of the cluster in production, because it optimizes the throughput of jobs using all available computational nodes.

The results of Table 1.1 can also be compared to corresponding entries obtained on the cluster hpc in 2008 as reported in [6, Table 1]. For the resolutions that hpc could fit into memory of one node, that is, for mesh resolutions of $32 \times 32 \times 128$, $64 \times 64 \times 256$, $128 \times 128 \times 512$, and $256 \times 256 \times 1024$, the serial runs on tara are about twice as fast as the hpc results. This shows the core-by-core performance improvement of an Intel Nehalem processor from 2009 compared to an AMD Opteron from 2008. A similar factor of speed improvement can be observed for other directly comparable cases of 2 and 4 processes per node on 2, 4, 8, 16, and 32 nodes used. It is even more striking to compare the performance on a nodal basis, that is using all available cores per node, which is 8 on tara and 4 on hpc. In this comparison, tara running the maximum possible 8 processes per node is often from 3 to 4 times faster than hpc running the maximum possible 4 processes per node, albeit with a significant amount of experimental variability.

Table 1.2: Wall clock time in HH:MM:SS on tara using OpenMPI for the solution of the parabolic test problem on $N_x \times N_y \times N_z$ meshes using 1, 2, 4, 8, 16, 32, and 64 compute nodes with 1, 2, 4, and 8 processes per node.

(a) Mesh resolution $N_x \times N_y \times N_z = 32 \times 32 \times 128$, DOF = 140,481							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:00:34	00:00:18	00:00:10	00:00:07	00:00:05	00:00:03	00:00:03
2 processes per node	00:00:19	00:00:10	00:00:06	00:00:05	00:00:04	00:00:02	00:00:02
4 processes per node	00:00:11	00:00:06	00:00:05	00:00:04	00:00:04	00:00:02	N/A
8 processes per node	00:00:07	00:00:04	00:00:04	00:00:03	00:00:02	N/A	N/A
(b) Mesh resolution $N_x \times N_y \times N_z = 64 \times 64 \times 256$, DOF = 1,085,825							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:05:01	00:02:20	00:01:11	00:00:43	00:00:22	00:00:13	00:00:08
2 processes per node	00:02:33	00:01:11	00:00:36	00:00:21	00:00:12	00:00:08	00:00:05
4 processes per node	00:01:21	00:00:37	00:00:20	00:00:12	00:00:07	00:00:05	00:00:04
8 processes per node	00:00:46	00:00:21	00:00:13	00:00:07	00:00:05	00:00:04	N/A
(c) Mesh resolution $N_x \times N_y \times N_z = 128 \times 128 \times 512$, DOF = 8,536,833							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:51:21	00:24:19	00:11:43	00:06:50	00:03:34	00:01:45	00:00:56
2 processes per node	00:27:13	00:12:26	00:06:13	00:03:23	00:01:46	00:00:57	00:00:31
4 processes per node	00:14:06	00:06:16	00:03:29	00:01:51	00:00:57	00:00:34	00:00:19
8 processes per node	00:07:44	00:03:44	00:01:51	00:00:58	00:00:31	00:00:20	00:00:12
(d) Mesh resolution $N_x \times N_y \times N_z = 256 \times 256 \times 1024$, DOF = 67,700,225							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	09:49:05	05:21:03	02:40:52	01:20:50	00:41:22	00:21:19	00:11:22
2 processes per node	04:57:34	02:50:05	01:25:12	00:44:03	00:22:02	00:11:42	00:05:45
4 processes per node	02:42:38	01:22:08	00:47:14	00:24:26	00:11:37	00:06:20	00:03:10
8 processes per node	01:32:17	00:48:40	00:23:27	00:12:13	00:06:20	00:03:34	00:02:03
(e) Mesh resolution $N_x \times N_y \times N_z = 512 \times 512 \times 2048$, DOF = 539,233,281							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	N/A	N/A	N/A	17:51:23	08:56:18	04:31:41	02:20:59
2 processes per node	N/A	N/A	N/A	09:33:58	04:44:58	02:27:05	01:16:10
4 processes per node	N/A	N/A	N/A	05:03:21	02:38:32	01:20:36	00:40:58
8 processes per node	N/A	N/A	N/A	02:49:08	01:25:27	00:45:35	00:23:42
(f) Mesh resolution $N_x \times N_y \times N_z = 1024 \times 1024 \times 4096$, DOF = 4,304,410,625							
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	N/A	N/A	N/A	N/A	N/A	N/A	N/A
2 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	N/A
8 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	05:30:00

2 The Parabolic Test Problem

We consider the following time-dependent, scalar, linear reaction-diffusion equation in three space dimensions that is a simplification of a multi-species model of calcium flow in heart cells [4, 5]: Find the concentration of the single species $u(x, y, z, t)$ for all $(x, y, z) \in \Omega$ and $0 \leq t \leq T$ such that

$$\begin{aligned} \frac{\partial u}{\partial t} - \nabla \cdot (D \nabla u) &= 0 && \text{in } \Omega \text{ for } 0 < t \leq T, \\ \mathbf{n} \cdot (D \nabla u) &= 0 && \text{on } \partial\Omega \text{ for } 0 < t \leq T, \\ u &= u_{\text{ini}}(x, y, z) && \text{in } \Omega \text{ at } t = 0, \end{aligned} \quad (2.1)$$

with the domain $\Omega = (-X, X) \times (-Y, Y) \times (-Z, Z) \subset \mathbb{R}^3$ with $X = Y = 6.4$ and $Z = 32.0$ in units of micrometers. We set the final time as $T = 100$ ms in the simulation. Here, $\mathbf{n} = \mathbf{n}(x, y, z)$ denotes the unit outward normal vector at the surface point (x, y, z) of the domain boundary $\partial\Omega$. The diagonal matrix $D = \text{diag}(D_x, D_y, D_z)$ consists of the diffusion coefficients in the three coordinate directions. To model realistic diffusion behavior we choose $D_x = D_y = 0.15$ and $D_z = 0.30$ in micrometers squared per milliseconds. The initial distribution is chosen to be

$$u_{\text{ini}}(x, y, z) = \cos^2\left(\frac{\lambda_x x}{2}\right) \cos^2\left(\frac{\lambda_y y}{2}\right) \cos^2\left(\frac{\lambda_z z}{2}\right), \quad (2.2)$$

where $\lambda_x = \pi/X$, $\lambda_y = \pi/Y$ and $\lambda_z = \pi/Z$. To get an intuitive feel for the solution behavior over time, we observe that the partial differential equation (PDE) in (2.1) has no source term and that no-flow boundary conditions are prescribed over the entire boundary. Hence, the molecules present initially at $t = 0$ will diffuse through the domain without escaping. Since the system conserves mass, the system will approach a steady state with a constant concentration throughout the domain as $t \rightarrow \infty$. This problem has been used as a test problem before in [3, 4] and its true solution can in fact be computed using separation of variables and Fourier analysis to be

$$u(x, y, z, t) = \frac{1 + \cos(\lambda_x x)e^{-D_x \lambda_x^2 t}}{2} \frac{1 + \cos(\lambda_y y)e^{-D_y \lambda_y^2 t}}{2} \frac{1 + \cos(\lambda_z z)e^{-D_z \lambda_z^2 t}}{2}. \quad (2.3)$$

The true solution confirms that the system evolves from the non-uniform initial distribution $u_{\text{ini}}(x, y, z)$ to the constant steady state solution $u_{SS} \equiv 1/8$; we do not reach this steady state with our final simulation time of $T = 100$ ms.

A method of lines discretization of (2.1) using finite elements with tri-linear nodal basis functions results in a stiff, large system of ordinary differential equations (ODEs) referred to as the so-called semi-discrete problem corresponding to (2.1) [5]. This ODE system is solved by the family of numerical differentiation formulas (NDF k , $1 \leq k \leq 5$) [9], which are generalizations of the well-known backward differentiation formulas (BDF k) (see, e.g., [2]). Since these ODE solvers are implicit, a system of linear equations needs to be solved at every time step. These systems are large, sparse, highly structured, and symmetric positive definite, and we use the conjugate gradient (CG) method to solve them. In a careful implementation, the conjugate gradient method requires in each iteration exactly two inner products between vectors, three vector updates, and one matrix-vector product involving the system matrix A . In fact, this matrix-vector product is the only time in which A enters into the algorithm. We avoid the storage cost of A by using a so-called matrix-free implementation of the CG method, in which no matrix is created or stored, but rather the needed matrix-vector products are computed directly by a user-supplied function [1]. The parallel implementation of the CG algorithm uses the MPI function `MPI_Allreduce` for the inner products and the technique of interleaving calculations and communications by non-blocking MPI communications commands `MPI_Isend` and `MPI_Irecv` in the matrix-free matrix-vector products.

Table 2.1 summarizes several key parameters of the numerical method and its implementation. The first two columns show the spatial mesh resolutions $N_x \times N_y \times N_z$ considered in the studies and their associated numbers of unknowns that need to be computed at every time step, commonly referred to as degrees of freedom (DOF). The column `nsteps` lists the number of time steps taken by the ODE solver. Due to the linearity of the problem (2.1), this number turns out to be independent of the mesh resolution, even though the ODE solver uses automatic time step and method order selection. The observed wall clock time for a serial run of the code is listed in hours:minutes:seconds (HH:MM:SS) and in seconds, indicating the rapid increase for finer meshes. The final two columns list the memory usage in MB, both predicted by counting variables in the algorithm and by observation using the Linux command `top` on the compute node being used.

The error in the finite element solution $u_h(\cdot, t)$ of the semi-discrete problem resulting from discretizing (2.1) is measured in the $L^2(\Omega)$ -norm. Under standard assumptions [7, 10], the finite element error has the form

$$\|u_h(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)} \leq Ch^q \quad \text{as } h \rightarrow 0 \quad (2.4)$$

Table 2.1: Sizing study (using MVAPICH2) listing the mesh resolution $N_x \times N_y \times N_z$, the number of degrees of freedom (DOF), the number of ODE steps to final time, the time in HH:MM:SS and in seconds, and the predicted and observed memory usage in MB for a one-processor run.

$N_x \times N_y \times N_z$	DOF	nsteps	wall clock time		memory usage (MB)	
			HH:MM:SS	seconds	predicted	observed
$32 \times 32 \times 128$	140,481	208	00:00:35	34.76	23	33
$64 \times 64 \times 256$	1,085,825	208	00:04:33	273.34	174	177
$128 \times 128 \times 512$	8,536,833	208	00:46:14	2773.53	1,368	1,379
$256 \times 256 \times 1024$	67,700,225	208	10:58:00	39479.70	10,846	10,859
^a $512 \times 512 \times 2048$	539,233,281	208	N/A	N/A	86,394	86,544
^b $1024 \times 1024 \times 4096$	4,304,410,625	208	N/A	N/A	689,640	707,080

^a A serial run was not possible due to memory limitations. This result uses 8 nodes with one process per node. The timing is not shown here, since it should not be compared to the serial runs.

^b This result uses 64 nodes with eight processes per node.

Table 2.2: Convergence study (using MVAPICH2) listing the $L^2(\Omega)$ -norm of the finite element error from (2.4) and estimated convergence order $q^{(est)}$ from (2.5) in parentheses at the times $t = 30$, $t = 40$, and $t = 50$ ms.

$N_x \times N_y \times N_z$	$t = 30$		$t = 40$		$t = 50$	
$32 \times 32 \times 128$	2.6954e-02		2.1525e-02		1.7019e-02	
$64 \times 64 \times 256$	6.7459e-03	(1.9984)	5.3837e-03	(1.9994)	4.2556e-03	(1.9997)
$128 \times 128 \times 512$	1.6843e-03	(2.0019)	1.3451e-03	(2.0009)	1.0636e-03	(2.0004)
$256 \times 256 \times 1024$	4.1852e-04	(2.0088)	3.3541e-04	(2.0037)	2.6569e-04	(2.0012)
$512 \times 512 \times 2048$	1.0213e-04	(2.0349)	8.3029e-05	(2.0143)	6.6226e-05	(2.0043)
$1024 \times 1024 \times 4096$	2.3358e-05	(2.1284)	2.0048e-05	(2.0501)	1.6401e-05	(2.0136)

at any point in time $0 \leq t \leq T$, where $q > 0$ for convergence and C denotes a generic constant of moderate size independent of the maximum mesh spacing $h := \max\{\Delta x, \Delta y, \Delta z\}$. Our problem with tri-linear nodal basis functions satisfies (2.4) with $q = 2$ at every point in time t . Therefore, we expect the $L^2(\Omega)$ -norm of the numerical solution against the true solution to decrease by a factor 4, whenever each of the mesh spacings Δx , Δy , Δz is halved. To formally estimate the convergence order q in (2.4) from numerically observed errors, one can use the estimation formula

$$q^{(est)} = \log_2 \left(\frac{\|u_{2h}(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)}}{\|u_h(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)}} \right). \quad (2.5)$$

At three representative points in time, Table 2.2 shows the $L^2(\Omega)$ -norm of the error $u_h(\cdot, t) - u(\cdot, t)$ and in parentheses the results of this formula for $q^{(est)}$. In all cases, the norms of the finite element errors decrease by a factor of about 4 each time the mesh is refined by a factor of 2 and $q^{(est)}$ approaches 2, both showing that the finite element method is second order convergent as predicted by the numerical theory. The fact that second order convergence is attained for the spatial discretization also confirms that both the tolerance of the iterative linear solver is tight enough to ensure a sufficiently accurate solution of the linear system and the tolerance on ODE error is small enough to ensure that the time error does not dominate.

3 Performance Studies on tara with MVAPICH2

The run times for the finer meshes observed for serial runs in Table 2.2 bring out one key motivation for parallel computing: The run times for a problem of a given, fixed size can be potentially dramatically reduced by spreading the work across a group of parallel processes. More precisely, the ideal behavior of code for a fixed problem size using p parallel processes is that it be p times as fast. If $T_p(N)$ denotes the wall clock time for a problem of a fixed size parametrized by N using p processes, then the quantity $S_p = T_1(N)/T_p(N)$ measures the speedup of the code from 1 to p processes, whose optimal value is $S_p = p$; for the finest resolutions, where data are only available starting with, for instance, $p = 8$, this definition is extended by the formula $S_p = 8T_8(N)/T_p(N)$, or analogously if the starting point is $p = 16$, 32, or 64. The efficiency $E_p = S_p/p$ characterizes in relative terms how close a run

with p parallel processes is to this optimal value, for which $E_p = 1$. The behavior described here for speedup for a fixed problem size is known as strong scalability of parallel code.

The results given in this section are based on the MVAPICH2 implementation of MPI. Table 3.1 lists the results of a performance study for strong scalability. Each row lists the results for one problem size, parametrized by the mesh resolution in the z -direction as N_z . Each column corresponds to the number of parallel processes p used in the run. The runs for Table 3.1 distribute these processes as widely as possible over the available nodes, that is, each process is run on a different node up to a maximum number of 64 nodes. In other words, up to $p = 64$, seven of the eight cores available on each node are idling, and only one core performs calculations. For $p = 128$, $p = 256$, and $p = 512$, this cannot be accommodated on 64 nodes, thus 2 processes are run on each node for $p = 128$, 4 processes per node for $p = 256$, and 8 processes per node for $p = 512$. Comparing adjacent columns in the raw timing data in Table 3.1 (a) indicates that using twice as many processes speeds up the code by a factor of two approximately, at least up to $p = 32$. To quantify this point more clearly, the speedup is computed in Table 3.1 (b) showing near-optimal speedup $S_p \approx p$ for all cases except $N_z = 128$ up to $p = 64$, which is expressed in terms of efficiency $E_p \approx 1$ in Table 3.1 (c). The customary representation of speedup and efficiency are presented in Figure 3.1 (a) and (b), respectively. Figure 3.1 (a) shows very clearly the excellent speedup up to $p = 32$ parallel processes for all mesh sizes which is maintained up to $p = 128$ for mesh size $N_z = 1024$. The efficiency plotted in Figure 3.1 (b) is directly derived from the speedup, but the plot is still useful because it details interesting features for small values of p that are hard to discern in the speedup plot. Here, we notice the variability of the results for small p is visible. In fact, we notice here, as in the table, that a number of results show apparently better than optimal behavior, with efficiency greater than 1.0. This can happen due to experimental variability of the runs, for instance, if the single-process timing $T_1(N)$ used in computation of $S_p = T_1(N)/T_p(N)$ happens to be slowed down in some way. Another reason for excellent performance can also be that runs on many processes result in local problem sizes that entirely fit or nearly fit into the cache of the individual processors, leading to fewer cache misses and thus potentially dramatic improvement in run times, beyond those expected by merely distributing the calculations to more processes. It is customary in results of fixed problem size that the speedup is better for larger problems. Examining column $p = 64$ in Table 3.1 (b) across all problem sizes N_z we see this is in fact the case. The conclusions discussed so far apply to up to $p = 64$ parallel processes. In each case, only 1 parallel process is run on each node, with the other seven cores available to handle all other operating system or other duties. For $p = 128$, $p = 256$, and $p = 512$, 2, 4, or 8 processes share each node necessarily, as 64 nodes are available, thus one expects slightly degraded performance as we go from $p = 64$ to $p = 128$, $p = 256$, and $p = 512$. This is born out by all data in Table 3.1 as well as clearly visible in Figures 3.1 (a) and (b) for $p > 64$. However, the times in Table 3.1 (a) for all finer meshes $N_z = 512$ and $N_z = 1024$ clearly demonstrate an improvement by using more cores, just not at the optimal rate of halving the wall clock time as p doubles.

To analyze the impact of using more than one core per node, we study the speedup for 2, 4, and 8 processes per node. We run 2 processes per node in Table 3.2 and Figure 3.2, 4 processes per node in Table 3.3 and Figure 3.3, and, 8 processes per node in Table 3.4 and Figure 3.4, wherever possible. That is, when studying 2 processes per node in Table 3.2, 4 and 8 processes per node are required for the $p = 256$ and $p = 512$ cases respectively since 64 nodes are available. When 4 processes per node are studied in Table 3.3, $p = 2$ is computed using a two-process job running on a single node while 8 processes per node are still required for $p = 512$. Lastly, when 8 processes per node are studied with Table 3.4, $p = 2$ is again computed using a two-process job running on a single node, while $p = 4$ is computed using a four-process job running on a single node. Note that the $p = 1$ serial case which is computed on a dedicated node (i.e., running the entire job on a single process on a single node) is also recorded in each of the tables. The results in the efficiency plots of Tables 3.2 (b), 3.3 (b), and 3.4 (b) show generally good efficiency for the larger problems sizes $N_z \geq 512$. However, it is curious that when we compare across tables we see a decrease in efficiency for $N_z = 1024$. Specifically, concentrating on $p = 64$ and $N_z = 1024$ across all the of tables, we see that the efficiency values in Tables 3.1 and 3.2 are excellent and compare very closely (0.9707 and 0.9443). However, Tables 3.3 and 3.4 in comparison have degraded efficiency of (0.8188 and 0.8670). This drop in efficiency for large problem sizes when more of the $p = 64$ processes are executed locally within a node suggests that it is beneficial to spread the processes over all the allocated nodes to increase efficiency. Keeping our attention on $N_z = 1024$, we see in each table that the efficiency decreases as the number of processes p increases, but focusing on the raw timings in Table 3.1 (a), 3.2 (a), 3.3 (a), and 3.4 (a), we see that increasing the number of parallel processes is very effective for reducing the required wall time to solve the problem. We have shown the performance of solving a very fine mesh $N_z = 2048$ as well. We see in Table 3.1 (c) that the efficiency is excellent but the optimal halving of run times as p increases is not occurring, for example, when increasing from $p = 128$ to $p = 256$ the time only decreases from 72 minutes to 43 minutes. However, this result would likely have taken over 6 days if it could have been run in serial. To make this estimate we assumed an ideal speedup from $p = 1$ to $p = 8$ for $N_z = 2048$ in Table 3.1 (a), which yields 8×19 hours = 152 hours.

Table 3.1: MVAPICH2 performance on tara by number of processes used with 1 process per node, except for $p = 128$ which uses 2 processes per node, $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:35	00:00:17	00:00:09	00:00:06	00:00:04	00:00:03	00:00:02	00:00:02	N/A	N/A
256	00:04:33	00:02:35	00:01:10	00:00:40	00:00:21	00:00:12	00:00:07	00:00:05	00:00:04	N/A
512	00:46:14	00:23:44	00:12:09	00:06:37	00:03:23	00:01:45	00:00:52	00:00:30	00:00:18	00:00:12
1024	10:57:60	04:54:49	02:45:21	01:22:05	00:41:33	00:20:56	00:10:36	00:05:40	00:03:04	00:01:57
2048	N/A	N/A	N/A	18:58:45	09:36:22	05:01:33	02:27:50	01:12:10	00:43:20	00:22:11
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	04:50:18
(b) Observed speedup S_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	2.0852	3.6979	5.6890	7.9361	11.3595	14.9185	14.2459	N/A	N/A
256	1.0000	1.7654	3.9307	6.8523	12.7372	22.8736	40.9805	52.4645	72.8907	N/A
512	1.0000	1.9476	3.8050	6.9919	13.6742	26.3368	53.6882	91.6566	152.1410	236.0451
1024	1.0000	2.2319	3.9794	8.0166	15.8360	31.4249	62.1219	116.2672	214.3423	336.5990
2048	N/A	N/A	N/A	8.0000	15.8060	30.2098	61.6209	126.2299	210.2193	410.5573
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000
(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.0426	0.9245	0.7111	0.4960	0.3550	0.2331	0.1113	N/A	N/A
256	1.0000	0.8827	0.9827	0.8565	0.7961	0.7148	0.6403	0.4099	0.2847	N/A
512	1.0000	0.9738	0.9512	0.8740	0.8546	0.8230	0.8389	0.7161	0.5943	0.4610
1024	1.0000	1.1159	0.9949	1.0021	0.9897	0.9820	0.9707	0.9083	0.8373	0.6574
2048	N/A	N/A	N/A	1.0000	0.9879	0.9441	0.9628	0.9862	0.8212	0.8019
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

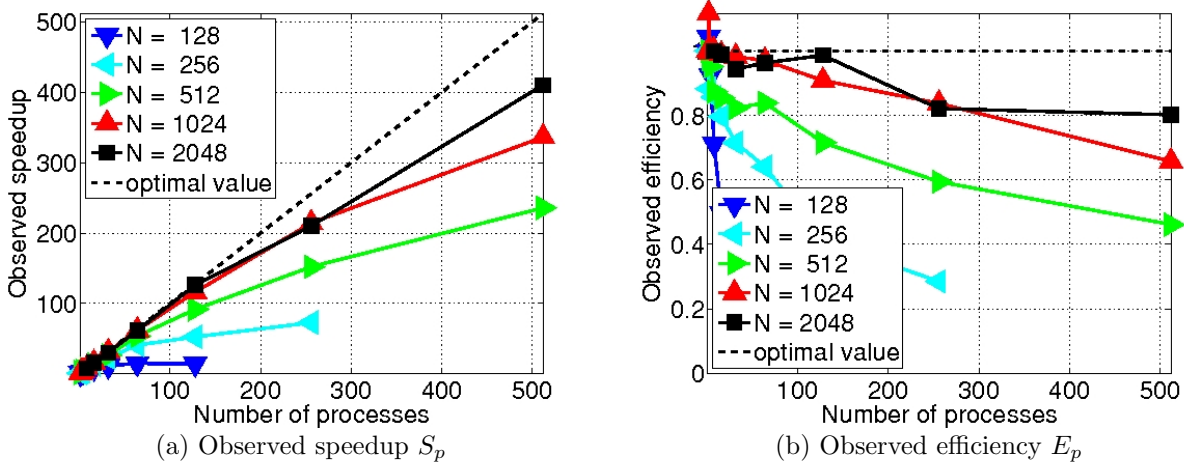


Figure 3.1: MVAPICH2 performance on tara by number of processes used with 1 process per node, except for $p = 128$ which uses 2 processes per node $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

Table 3.2: MVAPICH2 performance on tara by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node, $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:35	00:00:18	00:00:09	00:00:06	00:00:05	00:00:04	00:00:03	00:00:02	N/A	N/A
256	00:04:33	00:02:19	00:01:09	00:00:40	00:00:22	00:00:12	00:00:07	00:00:05	00:00:04	N/A
512	00:46:14	00:23:38	00:11:52	00:06:40	00:03:25	00:01:49	00:00:58	00:00:30	00:00:18	00:00:12
1024	10:57:60	04:48:32	02:26:29	01:24:19	00:40:52	00:21:10	00:10:53	00:05:40	00:03:04	00:01:57
2048	N/A	N/A	N/A	N/A	09:26:10	04:46:39	02:24:10	01:12:10	00:43:20	00:22:11
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	04:50:18
(b) Observed speedup S_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.8840	3.6705	5.5350	7.4433	9.1715	13.3180	14.2459	N/A	N/A
256	1.0000	1.9672	3.9597	6.8029	12.4359	23.6249	37.6501	52.4645	72.8907	N/A
512	1.0000	1.9566	3.8952	6.9314	13.5261	25.3591	48.1181	91.6566	152.1410	236.0451
1024	1.0000	2.2805	4.4922	7.8040	16.1042	31.0935	60.4358	116.2672	214.3423	336.5990
2048	N/A	N/A	N/A	N/A	16.0000	31.6010	62.8340	125.5174	209.0327	408.2400
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000
(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.9420	0.9176	0.6919	0.4652	0.2866	0.2081	0.1113	N/A	N/A
256	1.0000	0.9836	0.9899	0.8504	0.7772	0.7383	0.5883	0.4099	0.2847	N/A
512	1.0000	0.9783	0.9738	0.8664	0.8454	0.7925	0.7518	0.7161	0.5943	0.4610
1024	1.0000	1.1403	1.1230	0.9755	1.0065	0.9717	0.9443	0.9083	0.8373	0.6574
2048	N/A	N/A	N/A	N/A	1.0000	0.9875	0.9818	0.9806	0.8165	0.7973
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

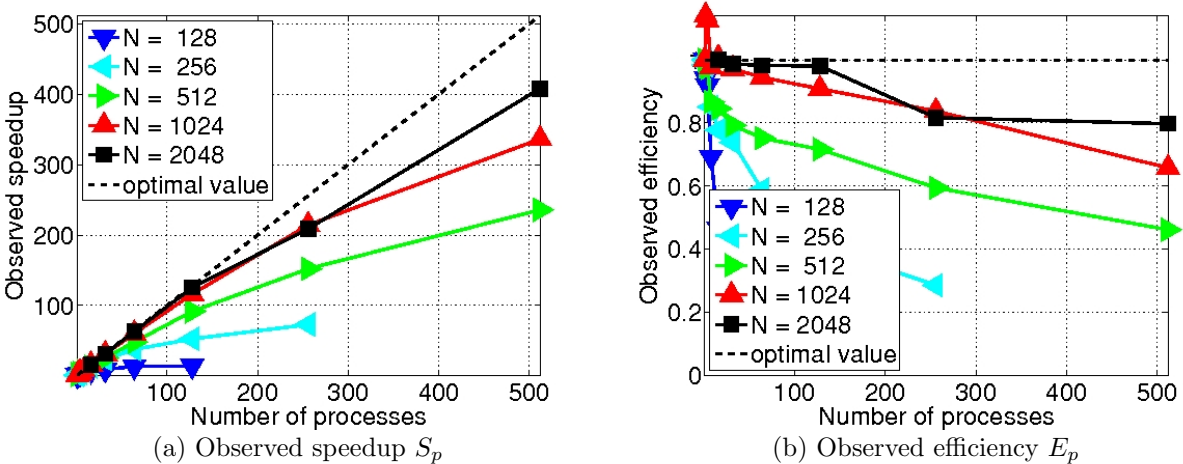


Figure 3.2: MVAPICH2 performance on tara by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node, $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

Table 3.3: MVAPICH2 performance on tara by number of processes used with 4 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:35	00:00:18	00:00:11	00:00:06	00:00:05	00:00:04	00:00:04	00:00:02	N/A	N/A
256	00:04:33	00:02:19	00:01:14	00:00:38	00:00:23	00:00:12	00:00:07	00:00:05	00:00:04	N/A
512	00:46:14	00:23:38	00:12:54	00:06:20	00:03:47	00:02:02	00:00:59	00:00:33	00:00:18	00:00:12
1024	10:57:60	04:48:32	03:04:01	01:22:23	00:44:27	00:23:53	00:12:33	00:06:17	00:03:04	00:01:57
2048	N/A	N/A	N/A	N/A	N/A	05:25:31	02:49:06	01:23:08	00:43:20	00:22:11
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	04:50:18
(b) Observed speedup S_p										
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.8840	3.2126	5.9828	7.4433	9.4201	9.6022	17.1232	N/A	N/A
256	1.0000	1.9672	3.7103	7.2716	12.1323	22.7594	39.6720	59.0367	72.8907	N/A
512	1.0000	1.9566	3.5850	7.3026	12.2112	22.8049	47.1849	85.1821	152.1410	236.0451
1024	1.0000	2.2805	3.5758	7.9871	14.8051	27.5563	52.4048	104.6152	214.3423	336.5990
2048	N/A	N/A	N/A	N/A	N/A	32.0000	61.6012	125.2904	240.3674	469.4364
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000
(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.9420	0.8031	0.7478	0.4652	0.2944	0.1500	0.1338	N/A	N/A
256	1.0000	0.9836	0.9276	0.9090	0.7583	0.7112	0.6199	0.4612	0.2847	N/A
512	1.0000	0.9783	0.8963	0.9128	0.7632	0.7127	0.7373	0.6655	0.5943	0.4610
1024	1.0000	1.1403	0.8940	0.9984	0.9253	0.8611	0.8188	0.8173	0.8373	0.6574
2048	N/A	N/A	N/A	N/A	N/A	1.0000	0.9625	0.9788	0.9389	0.9169
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

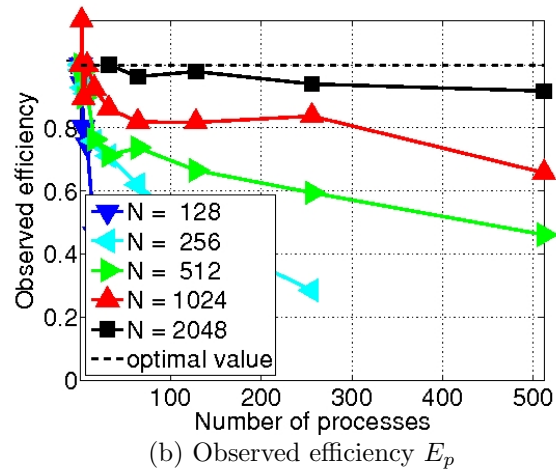
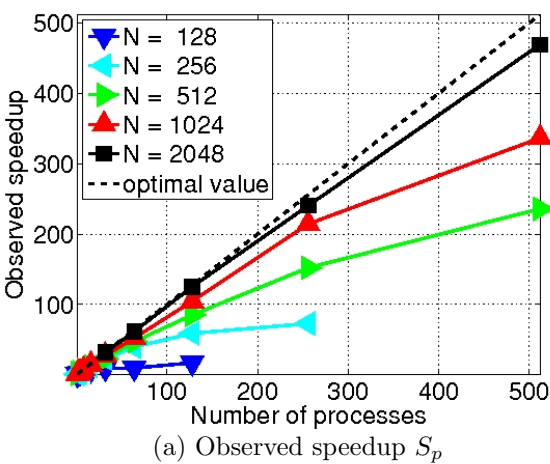


Figure 3.3: MVAPICH2 performance on tara by number of processes used with 4 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 512$ which uses 8 processes per node.

Table 3.4: MVAPICH2 performance on tara by number of processes used with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:35	00:00:18	00:00:11	00:00:08	00:00:06	00:00:05	00:00:04	00:00:03	N/A	N/A
256	00:04:33	00:02:19	00:01:14	00:00:42	00:00:21	00:00:12	00:00:07	00:00:04	00:00:04	N/A
512	00:46:14	00:23:38	00:12:54	00:07:07	00:03:53	00:01:60	00:00:59	00:00:32	00:00:20	00:00:12
1024	10:57:60	04:48:32	03:04:01	01:27:12	00:43:35	00:22:03	00:11:52	00:06:04	00:03:33	00:01:57
2048	N/A	N/A	N/A	N/A	N/A	N/A	02:43:45	01:21:29	00:40:56	00:22:11
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	04:50:18
(b) Observed speedup S_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.8840	3.2126	4.5737	6.3200	7.4914	9.1234	10.2235	N/A	N/A
256	1.0000	1.9672	3.7103	6.4942	12.8812	22.8736	38.7716	61.2870	76.9972	N/A
512	1.0000	1.9566	3.5850	6.4960	11.8954	23.1939	46.6453	87.9090	136.9644	236.0451
1024	1.0000	2.2805	3.5758	7.5459	15.0986	29.8361	55.4849	108.3387	185.3420	336.5990
2048	N/A	N/A	N/A	N/A	N/A	N/A	64.0000	128.6165	256.0646	472.3221
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000
(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.9420	0.8031	0.5717	0.3950	0.2341	0.1426	0.0799	N/A	N/A
256	1.0000	0.9836	0.9276	0.8118	0.8051	0.7148	0.6058	0.4788	0.3008	N/A
512	1.0000	0.9783	0.8963	0.8120	0.7435	0.7248	0.7288	0.6868	0.5350	0.4610
1024	1.0000	1.1403	0.8940	0.9432	0.9437	0.9324	0.8670	0.8464	0.7240	0.6574
2048	N/A	N/A	N/A	N/A	N/A	N/A	1.0000	1.0048	1.0003	0.9225
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

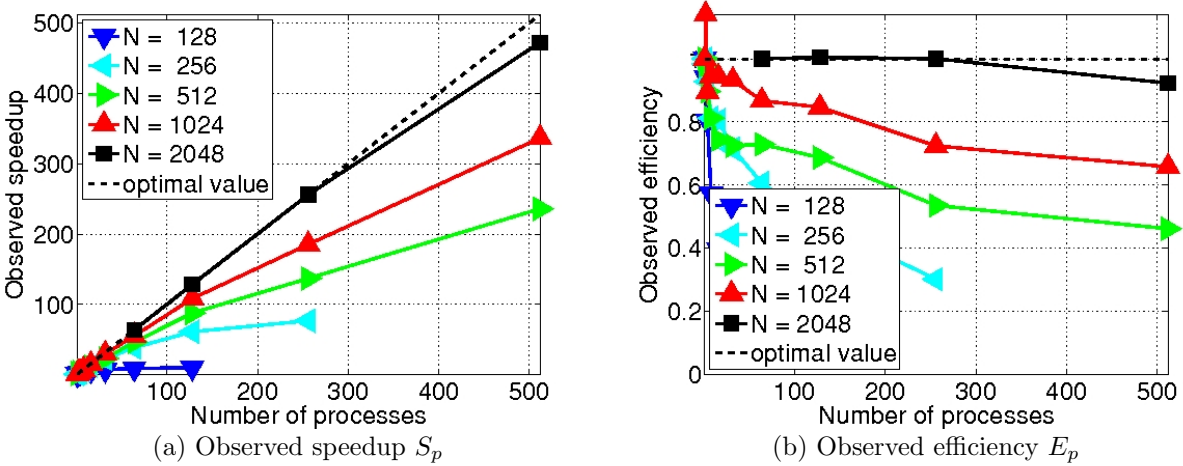


Figure 3.4: MVAPICH2 performance on tara by number of processes used with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

4 Performance Studies on tara with OpenMPI

This section summarizes the results of analogous studies to the previous section, this time using the OpenMPI implementation of the MPI standard. Otherwise, the same cluster and compiler have been used. The goal of this study is to answer the question of whether MVAPICH2 or OpenMPI should be used as the default MPI implementation on the cluster tara.

Analogous studies for the sizing and convergence studies of the method as reported in Tables 2.1 and 2.2, respectively, were performed using OpenMPI. Identical numerical results were observed, confirming the correctness of the studies. Table 1.2 in the Introduction summarizes the raw timing results for our OpenMPI studies, analogously to Table 1.1 in the Introduction. Reading the data row-wise (varying number of nodes) or column-wise (varying process per node), we again observe excellent scalability. Tables 4.1, 4.2, 4.3, and 4.4, with Figures 4.1, 4.2, 4.3, and 4.4 show detailed performance results including speedup and efficiency. These results at first glance appear very similar to the 1, 2, 4, and 8 process per node results from Section 3. However, comparing the two sets of data carefully reveals major differences. Considering Table 4.4, we notice the lower efficiency values using OpenMPI for $N_z = 1024$ as p increases compared to corresponding MVAPICH2 efficiency values shown in Table 3.4. Two results that are impacting the efficiency calculation. First, The execution time for the OpenMPI serial case finishes much faster than the MVAPICH2 serial case (9:49:05 vs. 10:57:50). Secondly, for $p = 8$, MVAPICH2 has an execution time about 5 minutes faster than OpenMPI. Moreover, comparing execution times for $p > 8$, MVAPICH2 is always faster, albeit the difference favors MVAPICH2 by only 5 seconds for $p = 512$ on all nodes. The results for 1, 2, and 4 processes per node also show that for larger values of p , the MVAPICH2 implementation nearly always exhibits faster execution times. So, we conclude that for multiple processes on the allocated nodes, defaulting to MVAPICH2 as the parallel coding implementation is preferred. For problems requiring less than 8 processes, an OpenMPI implementation may result in the shortest run time.

Table 4.1: OpenMPI performance on tara by number of processes used with 1 process per node, except for $p = 128$ which uses 2 processes per node, $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:34	00:00:18	00:00:10	00:00:07	00:00:05	00:00:03	00:00:03	00:00:02	N/A	N/A
256	00:05:01	00:02:20	00:01:11	00:00:43	00:00:22	00:00:13	00:00:08	00:00:05	00:00:04	N/A
512	00:51:21	00:24:19	00:11:43	00:06:50	00:03:34	00:01:45	00:00:56	00:00:31	00:00:19	00:00:12
1024	09:49:05	05:21:03	02:40:52	01:20:50	00:41:22	00:21:19	00:11:22	00:05:45	00:03:10	00:02:03
2048	N/A	N/A	N/A	17:51:23	08:56:18	04:31:41	02:20:59	01:16:10	00:40:58	00:23:42
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	05:29:60
(b) Observed speedup S_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.8893	3.4323	5.2934	7.3632	10.5706	11.7211	14.6638	N/A	N/A
256	1.0000	2.1570	4.2552	7.0720	13.7161	23.9460	39.3064	60.4168	83.7444	N/A
512	1.0000	2.1113	4.3820	7.5085	14.4287	29.3971	54.8135	99.3075	164.9984	248.8304
1024	1.0000	1.8349	3.6618	7.2870	14.2426	27.6261	51.8036	102.5788	186.4853	287.9394
2048	N/A	N/A	N/A	8.0000	15.9821	31.5485	60.7930	112.5363	209.2239	361.5913
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000
(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.9446	0.8581	0.6617	0.4602	0.3303	0.1831	0.1146	N/A	N/A
256	1.0000	1.0785	1.0638	0.8840	0.8573	0.7483	0.6142	0.4720	0.3271	N/A
512	1.0000	1.0557	1.0955	0.9386	0.9018	0.9187	0.8565	0.7758	0.6445	0.4860
1024	1.0000	0.9174	0.9154	0.9109	0.8902	0.8633	0.8094	0.8014	0.7285	0.5624
2048	N/A	N/A	N/A	1.0000	0.9989	0.9859	0.9499	0.8792	0.8173	0.7062
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

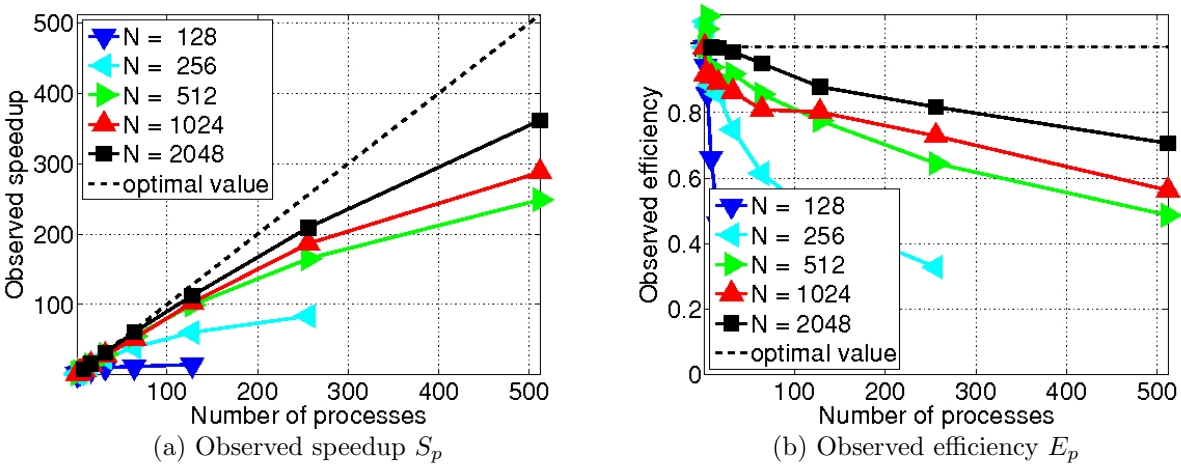


Figure 4.1: OpenMPI performance on tara by number of processes used with 1 process per node, except for $p = 128$ which uses 2 processes per node $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

Table 4.2: OpenMPI performance on tara by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node, $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:34	00:00:19	00:00:10	00:00:06	00:00:05	00:00:04	00:00:02	00:00:02	N/A	N/A
256	00:05:01	00:02:33	00:01:11	00:00:36	00:00:21	00:00:12	00:00:08	00:00:05	00:00:04	N/A
512	00:51:21	00:27:13	00:12:26	00:06:13	00:03:23	00:01:46	00:00:57	00:00:31	00:00:19	00:00:12
1024	09:49:05	04:57:34	02:50:05	01:25:12	00:44:03	00:22:02	00:11:42	00:05:45	00:03:10	00:02:03
2048	N/A	N/A	N/A	N/A	09:33:58	04:44:58	02:27:05	01:16:10	00:40:58	00:23:42
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	05:29:60
(b) Observed speedup S_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.7681	3.5636	5.3426	7.3319	9.3388	14.2397	14.6638	N/A	N/A
256	1.0000	1.9765	4.2444	8.4236	14.2275	25.1233	39.4092	60.4168	83.7444	N/A
512	1.0000	1.8863	4.1312	8.2672	15.2072	29.1882	53.8364	99.3075	164.9984	248.8304
1024	1.0000	1.9797	3.4634	6.9134	13.3727	26.7296	50.3792	102.5788	186.4853	287.9394
2048	N/A	N/A	N/A	N/A	16.0000	32.2255	62.4400	120.5762	224.1714	387.4244
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000
(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.8840	0.8909	0.6678	0.4582	0.2918	0.2225	0.1146	N/A	N/A
256	1.0000	0.9883	1.0611	1.0529	0.8892	0.7851	0.6158	0.4720	0.3271	N/A
512	1.0000	0.9432	1.0328	1.0334	0.9504	0.9121	0.8412	0.7758	0.6445	0.4860
1024	1.0000	0.9898	0.8659	0.8642	0.8358	0.8353	0.7872	0.8014	0.7285	0.5624
2048	N/A	N/A	N/A	N/A	1.0000	1.0070	0.9756	0.9420	0.8757	0.7567
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

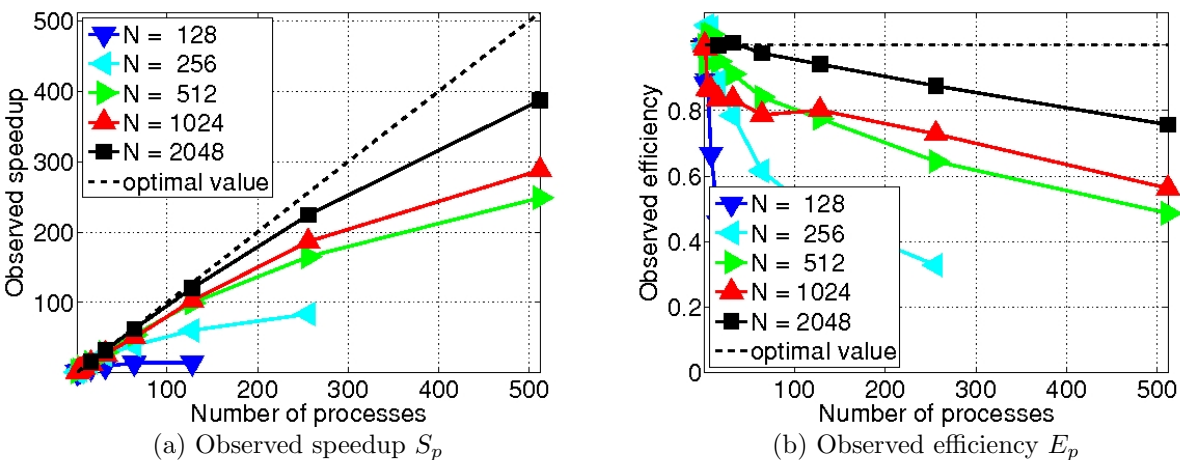


Figure 4.2: OpenMPI performance on tara by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node, $p = 256$ which uses 4 processes per node, and $p = 512$ which uses 8 processes per node.

Table 4.3: OpenMPI performance on tara by number of processes used with 4 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:34	00:00:19	00:00:11	00:00:06	00:00:05	00:00:04	00:00:04	00:00:02	N/A	N/A
256	00:05:01	00:02:33	00:01:21	00:00:37	00:00:20	00:00:12	00:00:07	00:00:05	00:00:04	N/A
512	00:51:21	00:27:13	00:14:06	00:06:16	00:03:29	00:01:51	00:00:57	00:00:34	00:00:19	00:00:12
1024	09:49:05	04:57:34	02:42:38	01:22:08	00:47:14	00:24:26	00:11:37	00:06:20	00:03:10	00:02:03
2048	N/A	N/A	N/A	N/A	N/A	05:03:21	02:38:32	01:20:36	00:40:58	00:23:42
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	05:29:60

(b) Observed speedup S_p										
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.7681	3.2357	5.6307	7.5736	8.7684	9.5989	17.4040	N/A	N/A
256	1.0000	1.9765	3.7257	8.1813	15.2340	25.4628	43.6928	62.2893	83.7444	N/A
512	1.0000	1.8863	3.6412	8.1831	14.7584	27.6702	53.6116	89.7065	164.9984	248.8304
1024	1.0000	1.9797	3.6222	7.1725	12.4707	24.1168	50.7103	93.0193	186.4853	287.9394
2048	N/A	N/A	N/A	N/A	N/A	32.0000	61.2332	120.4349	236.9574	409.5218
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000

(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.8840	0.8089	0.7038	0.4734	0.2740	0.1500	0.1360	N/A	N/A
256	1.0000	0.9883	0.9314	1.0227	0.9521	0.7957	0.6827	0.4866	0.3271	N/A
512	1.0000	0.9432	0.9103	1.0229	0.9224	0.8647	0.8377	0.7008	0.6445	0.4860
1024	1.0000	0.9898	0.9055	0.8966	0.7794	0.7536	0.7923	0.7267	0.7285	0.5624
2048	N/A	N/A	N/A	N/A	N/A	1.0000	0.9568	0.9409	0.9256	0.7998
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

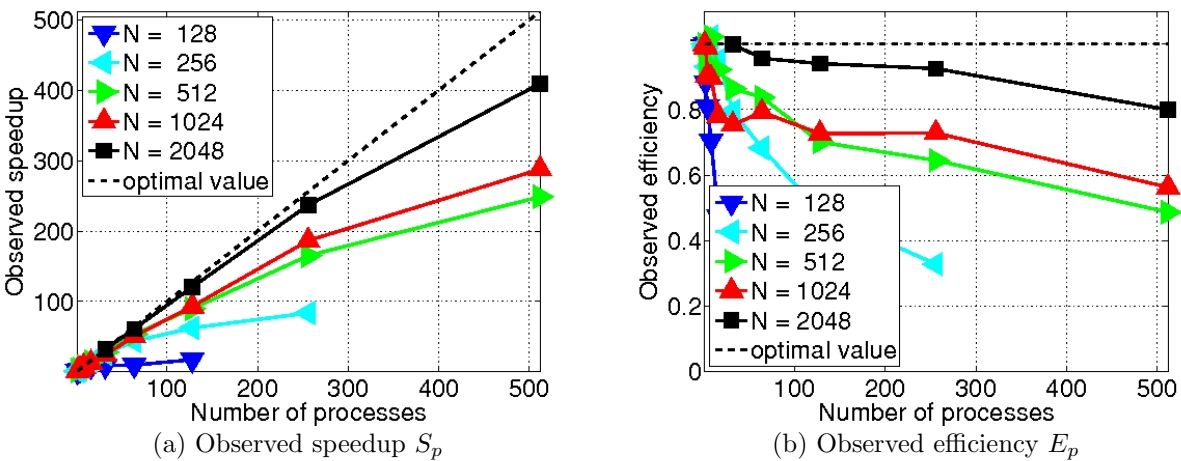


Figure 4.3: OpenMPI performance on tara by number of processes used with 4 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 512$ which uses 8 processes per node.

Table 4.4: OpenMPI performance on tara by number of processes used with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

(a) Wall clock time in HH:MM:SS										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	00:00:34	00:00:19	00:00:11	00:00:07	00:00:04	00:00:04	00:00:03	00:00:02	N/A	N/A
256	00:05:01	00:02:33	00:01:21	00:00:46	00:00:21	00:00:13	00:00:07	00:00:05	00:00:04	N/A
512	00:51:21	00:27:13	00:14:06	00:07:44	00:03:44	00:01:51	00:00:58	00:00:31	00:00:20	00:00:12
1024	09:49:05	04:57:34	02:42:38	01:32:17	00:48:40	00:23:27	00:12:13	00:06:20	00:03:34	00:02:03
2048	N/A	N/A	N/A	N/A	N/A	N/A	N/A	02:49:08	01:25:27	00:23:42
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	05:29:60

(b) Observed speedup S_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	1.7681	3.2357	5.0826	8.5935	9.8177	11.4106	19.5795	N/A	N/A
256	1.0000	1.9765	3.7257	6.5582	14.2275	23.0666	43.2539	61.5265	82.3716	N/A
512	1.0000	1.8863	3.6412	6.6406	13.7542	27.6404	52.7035	98.7979	151.4513	248.8304
1024	1.0000	1.9797	3.6222	6.3837	12.1060	25.1244	48.2138	92.9044	165.2387	287.9394
2048	N/A	N/A	N/A	N/A	N/A	N/A	64.0000	126.6684	237.4226	456.6458
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	512.0000

(c) Observed efficiency E_p										
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
128	1.0000	0.8840	0.8089	0.6353	0.5371	0.3068	0.1783	0.1530	N/A	N/A
256	1.0000	0.9883	0.9314	0.8198	0.8892	0.7208	0.6758	0.4807	0.3218	N/A
512	1.0000	0.9432	0.9103	0.8301	0.8596	0.8638	0.8235	0.7719	0.5916	0.4860
1024	1.0000	0.9898	0.9055	0.7980	0.7566	0.7851	0.7533	0.7258	0.6455	0.5624
2048	N/A	N/A	N/A	N/A	N/A	N/A	1.0000	0.9896	0.9274	0.8919
4096	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0000

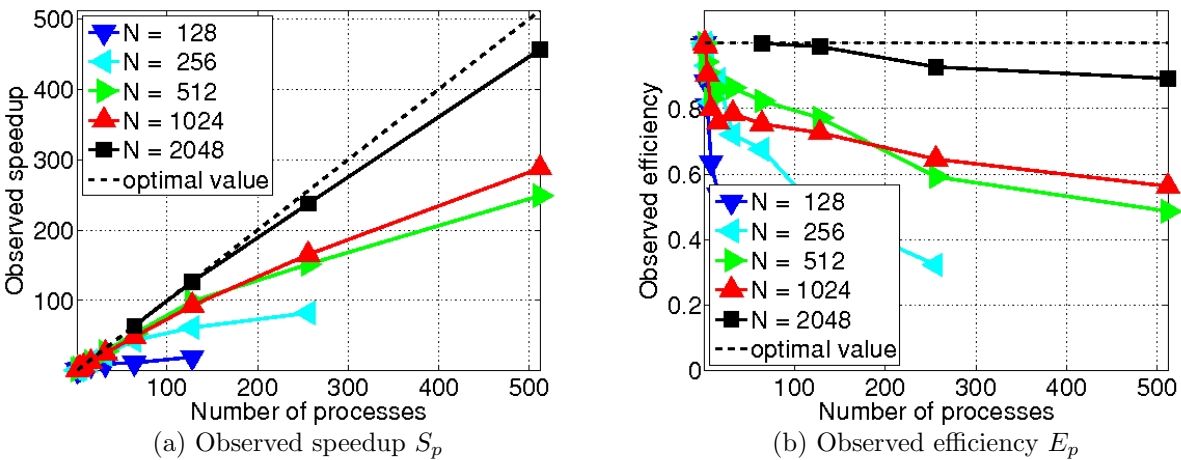


Figure 4.4: OpenMPI performance on tara by number of processes used with 8 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, and $p = 4$ which uses 4 processes per node.

5 Conclusions

The results presented still support the traditional observation that the best performance improvements, in the sense of halving the time when doubling the number of processes, are achieved by only running one parallel process on each node. But for production runs, we are not interested in this improvement being optimal. Rather, we are interested in the run time being the smallest on a given number of nodes. Thus, given a fixed number of nodes, the question is if one should run 1, 2, 4, or 8 processes per node. This is answered by the data organized in the form of Table 1.1 in the Introduction. It is these results, which are the same raw timing data as in Tables 3.1, 3.2, 3.3, and 3.1, that make it clear that using 8 processes per node is the best way to use the cluster tara. In fact, the improvement of timings for large problems is nearly optimal by doubling the number of processes per node, as well. This is an excellent result leading us to select a scheduling policy that uses all eight cores of the two quad-core processors on all assigned node simultaneously. These results use the MVAPICH2 implementation of the MPI standard. As discussed in Section 4, the OpenMPI implementation of MPI performs in the same way, but some MVAPICH2 runs are faster than corresponding OpenMPI runs, supporting the choice of MVAPICH2 as the default MPI implementation on tara.

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. Andrew Raim additionally acknowledges financial support as HPCF RA.

References

- [1] Kevin P. Allen and Matthias K. Gobbert. Coarse-grained parallel matrix-free solution of a three-dimensional elliptic prototype problem. In Vipin Kumar, Marina L. Gavrilova, Chih Jeng Kenneth Tan, and Pierre L'Ecuyer, editors, *Computational Science and Its Applications—ICCSA 2003*, vol. 2668 of *Lecture Notes in Computer Science*, pp. 290–299. Springer-Verlag, 2003.
- [2] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.
- [3] Matthias K. Gobbert. Configuration and performance of a Beowulf cluster for large-scale scientific simulations. *Comput. Sci. Eng.*, vol. 7, pp. 14–26, March/April 2005.
- [4] Matthias K. Gobbert. Long-time simulations on high resolution meshes to model calcium waves in a heart cell. *SIAM J. Sci. Comput.*, vol. 30, no. 6, pp. 2922–2947, 2008.
- [5] Alexander L. Hanhart, Matthias K. Gobbert, and Leighton T. Izu. A memory-efficient finite element method for systems of reaction-diffusion equations with non-smooth forcing. *J. Comput. Appl. Math.*, vol. 169, no. 2, pp. 431–458, 2004.
- [6] Michael Muscedere and Matthias K. Gobbert. Parallel performance studies for a parabolic test problem. Technical Report HPCF-2008-2, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2008.
- [7] Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*, vol. 23 of *Springer Series in Computational Mathematics*. Springer-Verlag, 1994.
- [8] Andrew M. Raim and Matthias K. Gobbert. Parallel performance studies for an elliptic test problem on the cluster tara. Technical Report HPCF-2010-2, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2010.
- [9] Lawrence F. Shampine and Mark W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, vol. 18, no. 1, pp. 1–22, 1997.
- [10] Vidar Thomée. *Galerkin Finite Element Methods for Parabolic Problems*, vol. 25 of *Springer Series in Computational Mathematics*. Springer-Verlag, second edition, 2006.