Parallel Performance Studies for a Three-Species Application Problem on the Cluster tara

David W. Trott and Matthias K. Gobbert ({dtrott1,gobbert}@umbc.edu)

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

Technical Report HPCF–2010–11, www.umbc.edu/hpcf > Publications

Abstract

High performance parallel computing depends on the interaction of a number of factors including the processors, the architecture of the compute nodes, their interconnect network, and the numerical code. In this note, we present performance and scalability studies on the cluster tara using a well established parallelized code for a three-species application problem. This application problem requires long-time simulations on a fine mesh, thus posing a very computationally intensive problem. The speedup of run times afforded by parallel computing makes the difference between simply unacceptably long runs to obtain the results (e.g., several days or weeks) and practically feasible studies (e.g., overnight runs). The results also 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

An important, practical approach to testing the real-life performance of a computer is to perform studies using reliable high performance code that is already being used in production. Performance tests of this nature not only provide a tool for gauging the effectiveness of a specific hardware setup, but they can also provide guidance to selecting a particular usage policy for clusters as well as give concrete experience in the expected length of production runs on the specific cluster. This note is part of a sequence of performance studies conducted on the cluster tara purchased in 2009 by the UMBC High Performance Computing Facility (HPCF, www.umbc.edu/hpcf). It was shown in [4] that an elliptic test problem given by the stationary Poisson equation, whose code uses a parallel, matrix-free implementation of the conjugate gradient (CG) linear solver, provides an excellent test problem since it tests two important types of parallel communications, namely collective communications involving all participating parallel processes and point-to-point communications between certain pairs of processes. The report [3] extends the stationary Poisson equation to a parabolic test problem given by the scalar, time-dependent, linear heat equation. The code for this problem involves implicit time-stepping with a linear solve using CG at every time step, and thus CG remains the key challenge for the parallel interconnect. This note extends the time-dependent parabolic problem to a system of three non-linear reaction-diffusion equations for the application problem of simulating calcium waves in heart cells [1]. This three-species application problem provides a substantially more computationally intensive test of the cluster, since the solution requires many more time steps than the test problem and it involves non-linearities and additional terms. Section 2 provides a brief introduction to the three-species application problem and the numerical algorithm used.

The studies in this note test the 86-node distributed-memory cluster tara 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. The studies use the MVAPICH2 implementation of the MPI standard.

[1]Vendor page www-03.ibm.com/systems/x/hardware/idataplex/

[2]Press 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 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 four progressively finer meshes, resulting in progressively larger systems of equations to be solved with system dimensions ranging from about 56,000 to over 25.6 million 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 $64 \times 64 \times 256$ spatial mesh that results in a system of over 3 million equations to be solved, the serial run takes over 57 hours, while the runs using either all 8 cores on 32 nodes or 4 cores on 64 nodes takes under 30 minutes. In this sub-table and in the previous ones with coarser resolutions, the "N/A" indicates that the run is not possible; this results from the mesh being split across the processes in the z-dimension, hence the number of nodes cannot be larger than the number of mesh points in the z-direction. For the $128 \times 128 \times 512$ spatial mesh that results in a system of over 25.6 million equations to be solved, the run with 512 parallel processes using all 8 cores on all 64 nodes takes about $2\frac{1}{2}$ hours; based on this, we predict that a serial run might take about 512 times that long or up to two months. Since that time-frame is excessive and the available results clearly demonstrate the power of parallel computing, we did not complete the performance study for this mesh, as indicated by the blank entries in this sub-table.

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 for most data points. This is discussed in detail in Section 3 in terms of the number of parallel processes. 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. We note that going from the next-to-last to the last entry in each row of the table does show sub-optimal speedup. However, this is for the case of having only one x-y-plane of mesh points on each process due to the splitting of the z-planes across all processes, thus there is actually too little calculation on each process left to perform well.

(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 previous reports in this sequence of performance studies, [4] and [3], used significantly finer meshes resulting in larger systems of linear equations than this report. These reports refined the mesh until running out of memory and thus demonstrated one key advantage of parallel computing: Larger problems can be solved by pooling the memory from several compute nodes. By contrast, the problem in this report is an actual application problem with physical effects and modeling requirements that necessarily require a much larger final time (1,000 ms vs. 100 ms in [3]), as discussed in Section 2. Additionally, the non-linearities in the terms require more time steps per unit of time simulated, and each time step is more expensive due to the non-linearities, the larger number of terms that need to be evaluated, and the three species. The long run times for the present problem thus restrict the mesh resolutions that can be computed within reasonable amount of time, as seen in Table 1.1. For the present problem, even the finest mesh possible within reasonable run times, $128 \times 128 \times 512$, does not put a strain on the memory of a node of the given cluster (see Table 2.1 in Section 2). Thus, this combination of facts brings out another key advantage of parallel computing: For efficient implementations of appropriate algorithms, problems can be solved significantly faster by pooling the processing power of several compute nodes. As the results of Table 1.1 demonstrate, this speedup can make the difference between simply unacceptably long runs to obtain the results (e.g., several days or weeks) and practically feasible studies (e.g., overnight runs).

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

(a) Mesh resolution $N_x \times N_y \times N_z = 16 \times 16 \times 64$, DOF = 56,355

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:30:09	00:15:29	00:08:19	00:04:25	00:02:36	00:01:56	00:01:51
2 processes per node	00:15:41	00:08:01	00:04:25	00:02:31	00:01:45	00:01:31	N/A
3 processes per node	00:08:04	00:04:19	00:02:31	00:01:39	00:01:25	N/A	N/A
4 processes per node	00:04:27	00:02:34	00:01:36	00:01:10	N/A	N/A	N/A

(b) Mesh resolution $N_x \times N_y \times N_z = 32 \times 32 \times 128$, DOF = 421,443

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	05:16:13	02:40:35	01:21:26	00:42:09	00:22:29	00:13:13	00:07:45
2 processes per node	02:41:35	01:20:33	00:41:27	00:21:46	00:12:56	00:07:44	00:05:24
3 processes per node	01:24:06	00:43:23	00:21:60	00:12:06	00:07:38	00:05:14	N/A
4 processes per node	00:45:17	00:23:13	00:12:39	00:07:46	00:05:09	N/A	N/A

(c) Mesh resolution $N_x \times N_y \times N_z = 64 \times 64 \times 256$, DOF = 3,257,475

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	57:14:46	29:32:28	14:34:19	07:27:06	03:51:04	01:59:36	01:05:30
2 processes per node	29:08:07	14:39:54	07:21:31	03:46:57	02:00:56	01:04:30	01:43:39
3 processes per node	16:06:40	08:05:01	04:03:45	02:05:39	01:08:09	00:38:44	00:25:21
4 processes per node	08:19:29	04:14:48	02:10:41	01:10:29	00:41:40	00:27:47	N/A

(d) Mesh resolution $N_x \times N_y \times N_z = 128 \times 128 \times 512$, DOF = 25,610,499

	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	23:41:45	11:18:17
2 processes per node	N/A	N/A	N/A	N/A	23:45:20	11:09:53	06:43:25
3 processes per node	N/A	N/A	N/A	26:42:05	12:12:05	06:35:41	03:53:24
4 processes per node	N/A	N/A	23:44:55	12:30:28	06:53:45	03:57:07	02:33:53

2 Three-Species Application Problem

The three-species application problem models the flow of calcium on the scale of one heart cell. Calcium ions enter into the cell at release units distributed throughout the cell and then diffuse. At each release unit, the probability for calcium to be released increases along with the concentration of calcium, thus creating a feedback loop of waves re-generating themselves repeatedly. An accurate model of such waves is useful since they are part of the normal functioning of the heart, but can also trigger abnormal arrhythmias. This model requires simulations on the time scale of several repeated waves and on the spatial scale of the entire cell. This requires long-time studies on spatial meshes that need to have a high resolution to resolve the positions of the calcium release units throughout the entire cell [1]. The problem can be modeled by a system of n_s time-dependent reaction-diffusion equations coupled by non-linear reaction terms $r^{(i)}$ and involving problem-specific additional terms J in the calcium species ($i = 0$), together with no-flux boundary conditions and a given set of initial conditions

$$\frac{\partial u^{(i)}}{\partial t} - \nabla \cdot (D_i(\mathbf{x})\nabla u^{(i)}) = r^{(i)}(u^{(0)}, \ldots, u^{(n_s-1)}) + J(u^{(0)}, \mathbf{x}, t)\delta_{i0} \quad \text{for } x \in \Omega,\ 0 < t \leq t_{\text{end}} \tag{2.1}$$

$$\mathbf{n} \cdot (D_i(\mathbf{x})\nabla u^{(i)}) = 0 \qquad \text{for } x \in \partial\Omega,\ 0 < t \leq t_{\text{end}} \tag{2.2}$$

$$u^{(i)}(\mathbf{x}, 0) = u^{(i)}_{\text{ini}}(\mathbf{x}) \quad \text{for } x \in \Omega,\ t = 0 \tag{2.3}$$

where the concentrations $u^{(i)}(\mathbf{x}, t)$ of the $n_s = 3$ chemical species $i = 0, 1, 2$ are functions of space $\mathbf{x} \in \Omega$ and time $0 \leq t \leq t_{\text{end}}$ [1, 2]. In particular, we consider the rectangular spatial domain

$$\Omega = (-6.4\ \mu\text{m}, 6.4\ \mu\text{m}) \times (-6.4\ \mu\text{m}, 6.4\ \mu\text{m}) \times (-32.0\ \mu\text{m}, 32.0\ \mu\text{m}) \subset \mathbb{R}^3.$$

The left-hand side of (2.1) models the diffusive transport of each chemical species with diffusivity given by the positive definite matrices $D_i \in \mathbb{R}^{3\times3}$. The non-linear reaction terms $r^{(i)}$ are responsible for the coupling between the equations in (2.1). The source and sink terms for the calcium generation within the heart cell are contained within the term $J(u^{(0)}, \mathbf{x}, t))$. This term also houses the stochastic aspect of the model since the calcium release units (CRUs) which are arranged discretely on a three-dimensional lattice each have a probability of opening depending on the concentration of calcium present at that site. The simulations in this note use the same parameter values for the problem as described in [1].

The spatial discretization of the three-species application problem with tri-linear nodal finite elements results in a large system of ordinary differential equations (ODEs). This ODE system is solved by the family of numerical differentiation formulas [5]. Since these ODE solvers are fully implicit, it is necessary to solve the fully coupled non-linear system of equations at every time step. The Newton method with an analytically supplied Jacobian is used for the non-linear solver. The linear solver makes use of the iterative QMR method with matrix-free matrix-vector multiplies. Table 2.1 summarizes several key parameters of the numerical method and its implementation. The first three columns show the spatial mesh resolution of $N_x \times N_y \times N_z$ finite elements, the number of mesh points $N = (N_x + 1)(N_y + 1)(N_z + 1)$, and and their associated numbers of unknowns $n_s N$ for the n_s species that need to be computed at every time step, commonly referred to as degrees of freedom (DOF). The following column lists the number of time steps taken by the ODE solver, which are significant and which increase with finer resolutions. The final two columns list the memory usage in MB, both predicted by counting variables in the algorithm and by observation provided in a memory log file produced from the performance run. We notice that even the finest resolution fits comfortably in the memory of one node of the cluster used.

Table 2.1: Sizing study (using MVAPICH2) listing the mesh resolution $N_x \times N_y \times N_z$, the number of mesh points $N = (N_x + 1) \times (N_y + 1) \times (N_z + 1)$, the number of degrees of freedom (DOF $= n_s N$), the number of time steps taken by the ODE solver, and the predicted and observed memory usage in MB for a one-process run. The run for the mesh resolution $128 \times 128 \times 512$ only ran briefly to observe memory usage and did not complete to final time.

$N_x \times N_y \times N_z$	N	DOF	number of time steps	memory usage (MB) predicted	memory usage (MB) observed
$16 \times 16 \times 64$	18,785	56,355	19,828	9	20
$32 \times 32 \times 128$	140,481	421,443	25,342	68	79
$64 \times 64 \times 256$	1,085,825	3,257,475	32,588	522	533
$128 \times 128 \times 512$	8,536,833	25,610,499	38,553	4,103	4,115

3 Performance Studies on tara with MVAPICH2

The run times for the finer meshes observed for serial runs in Table 1.1 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 resolution, where data are only available starting with $p = 32$, this definition is extended by the formula $S_p = 32T_{32}(N)/T_p(N)$. 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.

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. To quantify this point more clearly, the speedup is computed in Table 3.1 (b) showing very good speedup $S_p \approx p$ for all cases except $N_z = 64$ up to $p = 32$, degrading somewhat erratically for larger values of p. This is expressed in terms of efficiency E_p greater than about 90% for all cases except $N_z = 64$ up to $p = 32$ in Table 3.1 (c). It is customary in results of fixed problem size that the speedup is better for larger problems. Examining column $p = 32$ in Tables 3.1 (b) and (c) across all problem sizes N_z, we see this is in fact the case. The customary representation of speedup and efficiency are presented in Figure 3.1 (a) and (b), respectively. Figure 3.1 (a) shows the excellent speedup up to $p = 32$ parallel processes for all mesh sizes which is maintained up to $p = 64$ for mesh size $N_z = 256$. 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 consistency of most results for small p.

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 for 2, 4, and 8 processes per node in Tables 3.2, 3.3, and 3.4, respectively, are seen to be very comparable, with noticable experimental variability, to those for 1 process per node in Table 3.1. This is an excellent result that confirms the conclusion from Section 1 regarding the scheduling policy on tara that uses all eight cores of the two quad-core processors on all assigned node simultaneously for production runs.

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

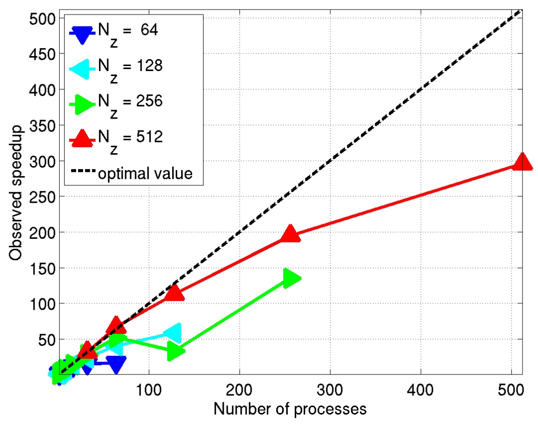
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	00:30:09	00:15:29	00:08:19	00:04:25	00:02:36	00:01:56	00:01:51	N/A	N/A	N/A
128	05:16:13	02:40:35	01:21:26	00:42:09	00:22:29	00:13:13	00:07:45	00:05:24	N/A	N/A
256	57:14:46	29:32:28	14:34:19	07:27:06	03:51:04	01:59:36	01:05:30	01:43:39	00:25:21	N/A
512						23:41:45	11:18:17	06:43:25	03:53:24	02:33:53

(b) Observed speedup $S_p = T_1/T_p$

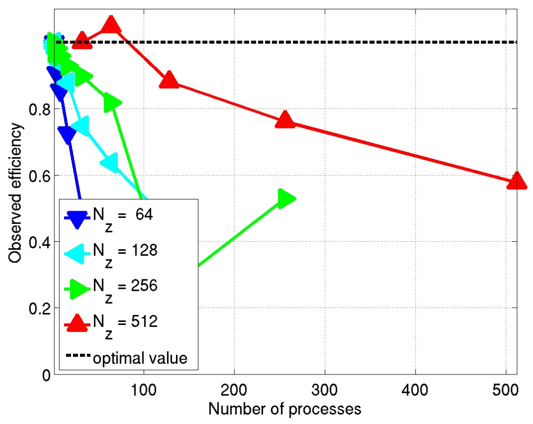
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.00	1.95	3.63	6.83	11.60	15.57	16.35	N/A	N/A	N/A
128	1.00	1.97	3.88	7.50	14.06	23.93	40.77	58.57	N/A	N/A
256	1.00	1.94	3.93	7.68	14.87	28.72	52.43	33.14	135.46	N/A
512						32.00	67.08	112.78	194.93	295.64

(c) Observed efficiency $E_p = S_p/p$

N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.0000	0.9741	0.9073	0.8540	0.7251	0.4865	0.2554	N/A	N/A	N/A
128	1.0000	0.9846	0.9709	0.9379	0.8789	0.7479	0.6370	0.4576	N/A	N/A
256	1.0000	0.9689	0.9821	0.9603	0.9291	0.8975	0.8193	0.2589	0.5291	N/A
512						1.0000	1.0481	0.8811	0.7614	0.5774



(a) Observed speedup S_p



(b) Observed efficiency E_p

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

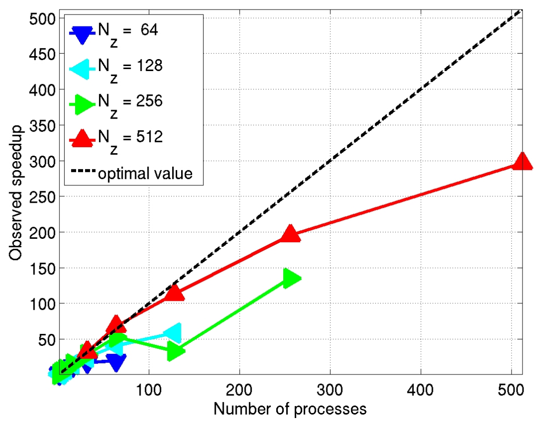
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	00:30:09	00:15:41	00:08:01	00:04:25	00:02:31	00:01:45	00:01:31	N/A	N/A	N/A
128	05:16:13	02:41:35	01:20:33	00:41:27	00:21:46	00:12:56	00:07:44	00:05:24	N/A	N/A
256	57:14:46	29:08:07	14:39:54	07:21:31	03:46:57	02:00:56	01:04:30	01:43:39	00:25:21	N/A
512						23:45:20	11:09:53	06:43:25	03:53:24	02:33:53

(b) Observed speedup $S_p = T_1/T_p$

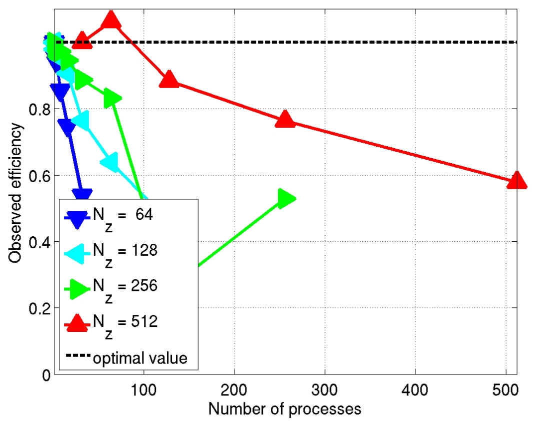
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.00	1.92	3.76	6.83	11.97	17.20	19.81	N/A	N/A	N/A
128	1.00	1.96	3.93	7.63	14.53	24.46	40.88	58.57	N/A	N/A
256	1.00	1.96	3.90	7.78	15.13	28.40	53.25	33.14	135.46	N/A
512						32.00	68.09	113.06	195.42	296.39

(c) Observed efficiency $E_p = S_p/p$

N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.0000	0.9610	0.9395	0.8542	0.7482	0.5376	0.3095	N/A	N/A	N/A
128	1.0000	0.9785	0.9815	0.9535	0.9079	0.7645	0.6388	0.4576	N/A	N/A
256	1.0000	0.9824	0.9759	0.9724	0.9459	0.8875	0.8320	0.2589	0.5291	N/A
512						1.0000	1.0639	0.8833	0.7634	0.5789



(a) Observed speedup S_p



(b) Observed efficiency E_p

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, and $p = 512$ which uses 8 processes per node.

(a) Wall clock time in HH:MM:SS

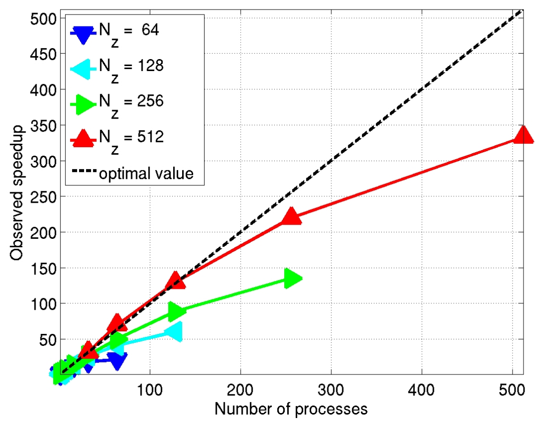
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	00:30:09	00:15:41	00:08:04	00:04:19	00:02:31	00:01:39	00:01:25	N/A	N/A	N/A
128	05:16:13	02:41:35	01:24:06	00:43:23	00:21:60	00:12:06	00:07:38	00:05:14	N/A	N/A
256	57:14:46	29:08:07	16:06:40	08:05:01	04:03:45	02:05:39	01:08:09	00:38:44	00:25:21	N/A
512						26:42:05	12:12:05	06:35:41	03:53:24	02:33:53

(b) Observed speedup $S_p = T_1/T_p$

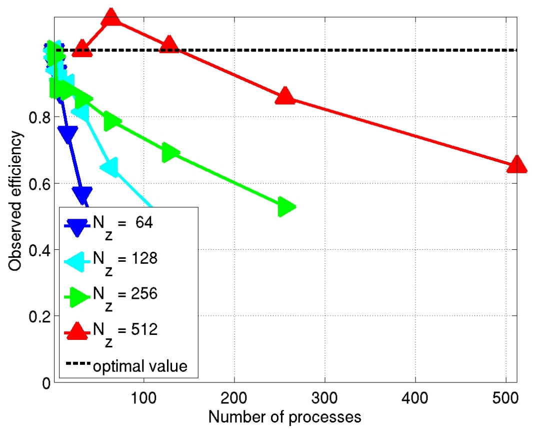
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.00	1.92	3.74	6.99	12.01	18.19	21.35	N/A	N/A	N/A
128	1.00	1.96	3.76	7.29	14.38	26.12	41.42	60.52	N/A	N/A
256	1.00	1.96	3.55	7.08	14.09	27.34	50.40	88.66	135.46	N/A
512						32.00	70.03	129.57	219.66	333.14

(c) Observed efficiency $E_p = S_p/p$

N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.0000	0.9610	0.9353	0.8732	0.7505	0.5685	0.3336	N/A	N/A	N/A
128	1.0000	0.9785	0.9401	0.9113	0.8985	0.8162	0.6473	0.4728	N/A	N/A
256	1.0000	0.9824	0.8883	0.8852	0.8807	0.8543	0.7876	0.6927	0.5291	N/A
512						1.0000	1.0942	1.0122	0.8580	0.6507



(a) Observed speedup S_p



(b) Observed efficiency E_p

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

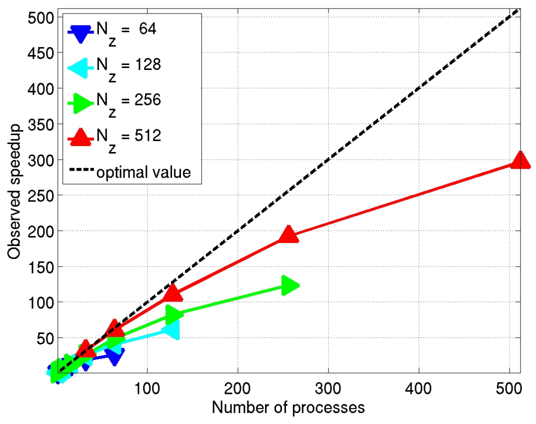
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	00:30:09	00:15:41	00:08:04	00:04:27	00:02:34	00:01:36	00:01:10	N/A	N/A	N/A
128	05:16:13	02:41:35	01:24:06	00:45:17	00:23:13	00:12:39	00:07:46	00:05:09	N/A	N/A
256	57:14:46	29:08:07	16:06:40	08:19:29	04:14:48	02:10:41	01:10:29	00:41:40	00:27:47	N/A
512						23:44:55	12:30:28	06:53:45	03:57:07	02:33:53

(b) Observed speedup $S_p = T_1/T_p$

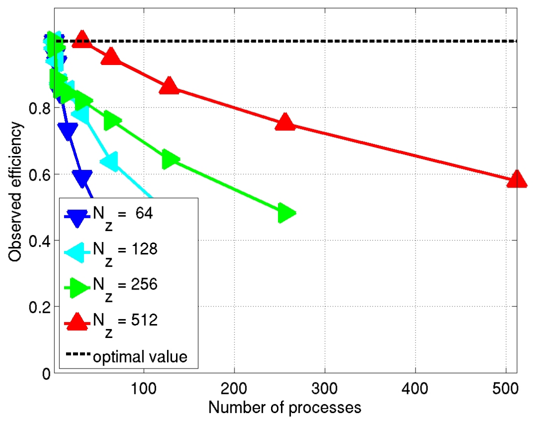
N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.00	1.92	3.74	6.77	11.73	18.92	25.77	N/A	N/A	N/A
128	1.00	1.96	3.76	6.98	13.62	25.01	40.76	61.33	N/A	N/A
256	1.00	1.96	3.55	6.88	13.48	26.28	48.73	82.44	123.63	N/A
512						32.00	60.76	110.20	192.30	296.30

(c) Observed efficiency $E_p = S_p/p$

N_z	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
64	1.0000	0.9610	0.9353	0.8460	0.7333	0.5912	0.4027	N/A	N/A	N/A
128	1.0000	0.9785	0.9401	0.8730	0.8515	0.7816	0.6368	0.4791	N/A	N/A
256	1.0000	0.9824	0.8883	0.8596	0.8425	0.8214	0.7614	0.6441	0.4829	N/A
512						1.0000	0.9494	0.8610	0.7512	0.5787



(a) Observed speedup S_p



(b) Observed efficiency E_p

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.

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. David Trott additionally acknowledges financial support as HPCF RA.

References

[1] 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.

[2] 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.

[3] Michael Muscedere, Andrew M. Raim, and Matthias K. Gobbert. Parallel performance studies for a parabolic test problem on the cluster tara. Technical Report HPCF–2010–4, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2010.

[4] 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.

[5] Lawrence F. Shampine and Mark W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, vol. 18, no. 1, pp. 1–22, 1997.