Parallel Performance Studies for a Parabolic Test Problem

Michael Muscedere and Matthias K. Gobbert

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

{mmusce1,gobbert}@math.umbc.edu

Abstract

The performance of a parallel computer depends on an intricate interplay of the processors, the architecture of the compute nodes, their interconnect network, the numerical algorithm, and the scheduling policy used. This note considers a parabolic test problem given by a time-dependent linear reaction-diffusion equation in three space dimensions, whose spatial discretization results in a large system of ordinary differential equations. These are integrated in time by the family of numerical differentiation formulas, which requires the solution of a system of linear equations at every time step. The results presented here show excellent performance on the cluster hpc in the UMBC High Performance Computing Facility and confirm that it is beneficial to use all four cores of the two dual-core processors on each node simultaneously, giving us in effect a computer that can run jobs efficiently with up to 128 parallel processes.

1 Introduction

This report generalizes the previous studies in [6] for an elliptic test problem to a parabolic test problem given by a time-dependent linear reaction-diffusion equation in three space dimensions. The spatial discretization of the parabolic partial differential equation results in a large system of ordinary differential equations (ODEs). This ODE system is solved by the family of numerical differentiation formulas. 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, and we use a matrix-free implementation of the conjugate gradient method to solve them, as in [6]. In this sense, the present report generalizes the studies in [6], although the spatial dimensions of the test problems 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, but contrast, the number of CG iterations in each time step is very limited. This is significant, because the key challenge for the parallel interconnect stems from the CG iterations and not from the time stepping. Section 2 states the test problem and describes its numerical approximation in more detail.

This parabolic test problem considered here has been used as a test problem before [4, 5]. Differences in the algorithmic details include that [4] uses the QMR method as iterative linear solver, while [5] used the implicit Euler method (i.e., BDF1) as ODE solver. Another notable difference between the present parallel performance study and those in [4, 5] is that the latter ones focused solely on the performance of the interconnect network and not on the nodal hardware, as described in the following.

This note now considers the parallel performance of the distributed-memory cluster hpc in the UMBC High Performance Computing Facility (`www.umbc.edu/hpcf`) with InfiniBand interconnect and with each compute node having two dual-core processors (AMD Opteron 2.6 GHz with 1024 kB cache and 13 GB of memory per node) for a total of up to four parallel processes to be run simultaneously per node. Section 3 describes the parallel scalability results in detail and provides the underlying data for the following summary results. Table 1 summarizes the key results of the present study by giving the wall clock time (total time to execute the code) in hours:minutes:seconds (HH:MM:SS) format. We consider the test problem on four progressively finer meshes, resulting in problems with progressively larger complexity as indicated by the numbers of degrees of freedom (DOF) ranging up to over 67.7 million equations. The parallel implementation of the numerical method is run on different numbers of nodes from 1 to 32 with different numbers of processes per node used. Specifically, the upper-left entry of each sub-table with 1 process per node represents the serial run of the code, which takes 66 seconds (1:06 minute) for the $32 \times 32 \times 128$ spatial mesh. The lower-right entry of each sub-table lists the time using both cores of both dual-core processors on all 32 nodes for a total of 128 parallel processes working together to solve the problem, which is 6 seconds for this mesh. More strikingly, one realizes the advantage of parallel computing for the large $256 \times 256 \times 1024$ mesh with over 67.7 million equations: The serial run of about $22\frac{1}{4}$ hours can be reduced to about 14 minutes using 128 parallel processes.

The results in Table 1 are arranged to study two key questions: (i) “Does the code scale optimally to all 32 nodes?” and (ii) “Is it worthwhile to use multiple processors and cores on each node?” The first question addresses the quality of the throughput of the InfiniBand interconnect network. The second question sheds light on the quality of the architecture within the nodes and cores of each processor. The answer to second questions will guide the scheduling policy by determining whether it should be the default to use all cores per node.

Table 1: Wall clock time in HH:MM:SS for the solution of problems on $N_x \times N_y \times N_z$ meshes using 1, 2, 4, 8, 16, 32 compute nodes with 1, 2, and 4 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
1 process per node	00:01:06	00:00:36	00:00:18	00:00:09	00:00:05	00:00:04
2 processes per node	00:00:36	00:00:18	00:00:09	00:00:06	00:00:04	00:00:04
4 processes per node	00:00:17	00:00:10	00:00:06	00:00:05	00:00:05	00:00:06
(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
1 process per node	00:09:43	00:04:55	00:02:30	00:01:17	00:00:40	00:00:22
2 processes per node	00:05:30	00:02:47	00:01:25	00:00:43	00:00:23	00:00:13
4 processes per node	00:02:42	00:01:27	00:00:42	00:00:23	00:00:14	00:00:10
(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
1 process per node	01:40:34	00:51:30	00:25:41	00:12:53	00:06:39	00:03:25
2 processes per node	00:56:50	00:28:59	00:14:29	00:07:13	00:03:42	00:01:57
4 processes per node	00:29:51	00:14:01	00:08:45	00:04:09	00:02:13	00:01:08
(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
1 process per node	22:15:28	11:20:36	05:25:21	02:41:47	01:22:08	00:41:15
2 processes per node	11:27:51	05:46:46	03:00:15	01:30:59	00:45:37	00:23:00
4 processes per node	06:43:00	03:05:25	01:34:06	00:50:34	00:27:54	00:14:10

(i) Reading along each row of Table 1, the wall clock time nearly halves as the number of nodes used doubles for all but the coarsest meshes, for which the performance improvement stops by about 16 nodes used. That is, by being essentially proportional to the number of nodes used, the speedup is nearly optimal for all meshes of significant size, which are the cases for which parallel computing is relevant. This is discussed in more detail in Section 3 in terms of the number of parallel processes.

(ii) To analyze the effect of running 1, 2, or 4 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 per node for all but the coarsest meshes. These results are excellent and confirm that it is not just effective to use both processors on each node, but also to use both cores of each dual-core processor simultaneously. Roughly, this shows that the architecture of the IBM nodes purchased in 2008 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 dual-core processors (as opposed to single-core processors). Moreover, these results will guide the scheduling policy implemented on the cluster: Namely, 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 are compared to those in Table 7 in the appendix for the cluster kali purchased in 2003 also from IBM with a Myrinet interconnect network and two (single-core) processors per node. We observe that the execution times for Table 1 are roughly half of the the value of those recorded in Table 7 for corresponding entries, that is with the same number of nodes and parallel processes per node. For instance, we see that kali can solve the finest mesh resolution $256 \times 256 \times 1024$ when using 16 nodes, and the best observed time is about 1:39 hours (99 minutes) with 2 parallel processes on each node. By comparison, hpc requires about 46 minutes for this problem using 16 nodes and 2 processes per node. But the optimal time on hpc, when using 16 nodes, is in fact 28 minutes, when using all 4 cores on each node. This shows the benefit of having 4 cores per node concretely. Looking at the comparison between the machines in a different way, then we see that only 4 nodes instead of 16 are required to solve the problem in approximately the same amount of wall clock time, namely 1:34 hours.

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, 7]: 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

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

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

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

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 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 $t \to \infty$. This problem has been used as a test problem before in [5, 4] and its the true solution can in fact be computed using separation of variables and Fourier analysis to be

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

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) [7]. This ODE system is solved by the family of numerical differentiation formulas (NDFk, $1 \leq k \leq 5$) [8], which are generalizations of the well-known backward differentiation formulas (BDFk) (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, 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 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 good agreement between predicted and observed memory usage indicates that the implementation of the code does not have any unexpected memory usage. The wall clock times and the memory usages for these serial runs indicate for which mesh resolutions this parabolic test problem becomes challenging computationally. Notice that the finer

Table 2: Sizing study 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	00:01:06	65.53	23	28
$64 \times 64 \times 256$	1,085,825	208	00:09:43	583.29	174	187
$128 \times 128 \times 512$	8,536,833	208	01:40:34	6034.38	1368	1433
$256 \times 256 \times 1024$	67,700,225	208	22:15:28	80128.13	10846	11366

meshes do show significant run times and memory usage more than 10 GB giving the parallel computing an opportunities to decrease run times as well as to decrease memory usage per process by spreading the problem over the parallel processes.

3 Performance Studies on hpc

The run times for the finer meshes observed for serial runs in Table 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 parallel 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 the number 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 $S_p = p$. 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 $E_p = 1$. This behavior described here for speedup for a fixed problem size is known as strong scalability of parallel code.

Table 3 lists the results of a performance study for strong scalability. Each row lists the results for one problem size, parametrized by the mesh resolution $N = N_x \times N_y \times N_z$. Each column corresponds to the number of parallel processes p used in the run. The runs for Table 3 distribute these processes as widely as possible over the available nodes, that is, each process is run on a different node up to the available number of 32 nodes. In other words, up to $p = 32$, three of the four cores available on each node are idling, and only one core performs calculations. For $p = 64$ and $p = 128$, this cannot be accommodated on 32 nodes, thus 2 processes run on each node for $p = 64$ and 4 processes per node for $p = 128$. Comparing adjacent columns in the raw timing data in Table 3 (a) indicates that using twice as many processes speeds up the code by nearly a factor of two, at least up to $p = 32$ for all but the coarsest mesh size. To quantify this more clearly, the speedup in Table 3 (b) is computed, which shows near-optimal with $S_p \approx p$ for all cases up to $p = 32$, which is expressed in terms of efficiency $0.84 \leq E_p \leq 1$ in Table 3 (c) for all but the coarsest mesh size.

The customary visualizations of speedup and efficiency are presented in Figure 1 (a) and (b), respectively. Figure 1 (a) shows very clearly the very good speedup up to $p = 32$ parallel processes for the three finest meshes. The efficiency plotted in Figure 1 (b) is directly derived from the speedup, but the plot is still useful because it can better bring out any interesting features for small values of p that are hard to tell in a speedup plot. Here, we notice that the variability of the results for small p is visible. In fact, for the finest mesh $256 \times 256 \times 1024$ the table shows a number of results 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 the 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 problems that fit or nearly fit into the cache of the processor, which leads to fewer cache misses and thus potentially dramatic improvement of the run time, beyond merely distributing the calculations to more processes. It is customary in results for fixed problem sizes that the speedup is better for larger problems, since the increased communication time for more parallel processes does not dominate over the calculation time as quickly as it does for small problems. Thus, the progression in speedup performance from smaller to larger mesh resolutions seen in Table 3 (b) is expected. To see this clearly, it is vital to have the precise data in Table 3 (b) and (c) available and not just their graphical representation in Figure 3.

Table 3: Performance by number of processes used with 1 process per node, except for $p = 64$ which uses 2 processes per node and $p = 128$ which uses 4 processes per node.

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

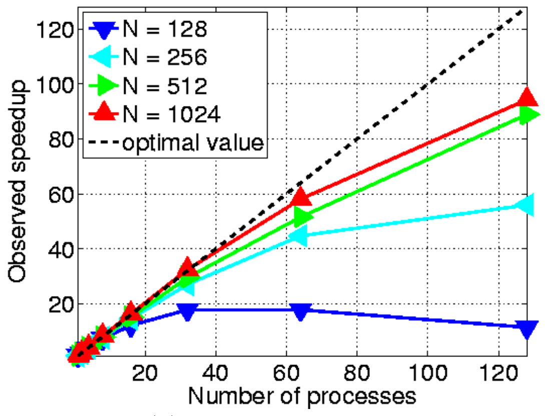
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	00:01:06	00:00:36	00:00:18	00:00:09	00:00:05	00:00:04	00:00:04	00:00:06
$64 \times 64 \times 256$	00:09:43	00:04:55	00:02:30	00:01:17	00:00:40	00:00:22	00:00:13	00:00:10
$128 \times 128 \times 512$	01:40:34	00:51:30	00:25:41	00:12:53	00:06:39	00:03:25	00:01:57	00:01:08
$256 \times 256 \times 1024$	22:15:28	11:20:36	05:25:21	02:41:47	01:22:08	00:41:15	00:23:00	00:14:10

(b) Observed speedup S_p

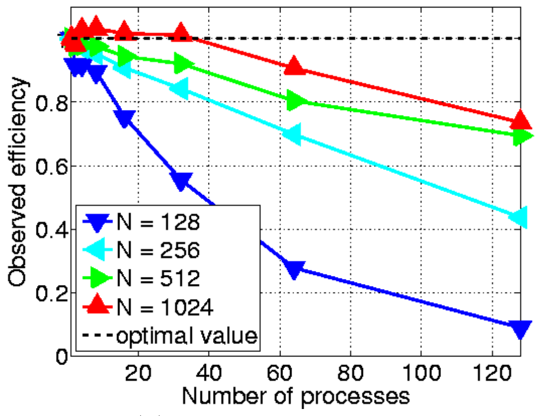
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	1.0000	1.8371	3.6753	7.1696	12.0681	17.7588	17.7588	11.3570
$64 \times 64 \times 256$	1.0000	1.9744	3.8889	7.6187	14.5386	26.9667	44.6966	55.9243
$128 \times 128 \times 512$	1.0000	1.9530	3.9162	7.8087	15.1135	29.4777	51.4835	88.8716
$256 \times 256 \times 1024$	1.0000	1.9622	4.1048	8.2548	16.2599	32.3771	58.0500	94.2651

(c) Observed efficiency E_p

$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	1.0000	0.9186	0.9188	0.8962	0.7543	0.5550	0.2775	0.0887
$64 \times 64 \times 256$	1.0000	0.9872	0.9722	0.9523	0.9087	0.8427	0.6984	0.4369
$128 \times 128 \times 512$	1.0000	0.9765	0.9790	0.9761	0.9446	0.9212	0.8044	0.6943
$256 \times 256 \times 1024$	1.0000	0.9811	1.0262	1.0318	1.0162	1.0118	0.9070	0.7364



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 1: Performance by number of processes used with 1 process per node, except for $p = 64$ which uses 2 processes per node and $p = 128$ which uses 4 processes per node.

Table 4: Performance by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node and $p = 128$ which uses 4 processes per node.

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

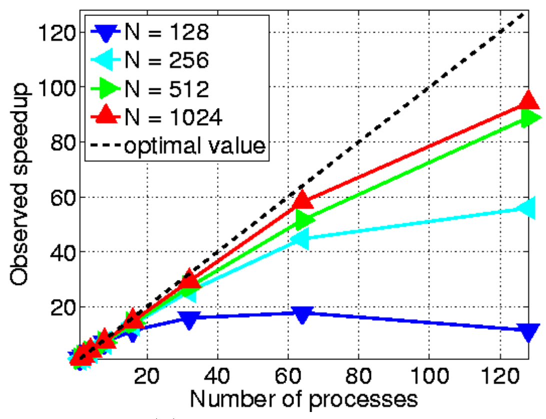
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	00:01:06	00:00:36	00:00:18	00:00:09	00:00:06	00:00:04	00:00:04	00:00:06
$64 \times 64 \times 256$	00:09:43	00:05:30	00:02:47	00:01:25	00:00:43	00:00:23	00:00:13	00:00:10
$128 \times 128 \times 512$	01:40:34	00:56:50	00:28:59	00:14:29	00:07:13	00:03:42	00:01:57	00:01:08
$256 \times 256 \times 1024$	22:15:28	11:27:51	05:46:46	03:00:15	01:30:59	00:45:37	00:23:00	00:14:10

(b) Observed speedup S_p

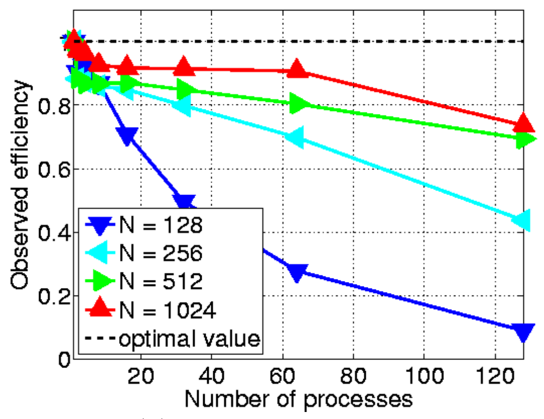
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	1.0000	1.8132	3.6939	6.9639	11.3374	15.8668	17.7588	11.3570
$64 \times 64 \times 256$	1.0000	1.7655	3.5020	6.8833	13.5870	25.5717	44.6966	55.9243
$128 \times 128 \times 512$	1.0000	1.7695	3.4694	6.9421	13.9336	27.1586	51.4835	88.8716
$256 \times 256 \times 1024$	1.0000	1.9415	3.8512	7.4090	14.6777	29.2724	58.0500	94.2651

(c) Observed efficiency E_p

$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	1.0000	0.9066	0.9235	0.8705	0.7086	0.4958	0.2775	0.0887
$64 \times 64 \times 256$	1.0000	0.8827	0.8755	0.8604	0.8492	0.7991	0.6984	0.4369
$128 \times 128 \times 512$	1.0000	0.8847	0.8674	0.8678	0.8709	0.8487	0.8044	0.6943
$256 \times 256 \times 1024$	1.0000	0.9707	0.9628	0.9261	0.9174	0.9148	0.9070	0.7364



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 2: Performance by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node and $p = 128$ which uses 4 processes per node.

Table 5: Performance by number of processes used with 4 processes per node, except for $p = 1$ which uses 1 process per node and $p = 2$ which uses 2 processes per node.

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

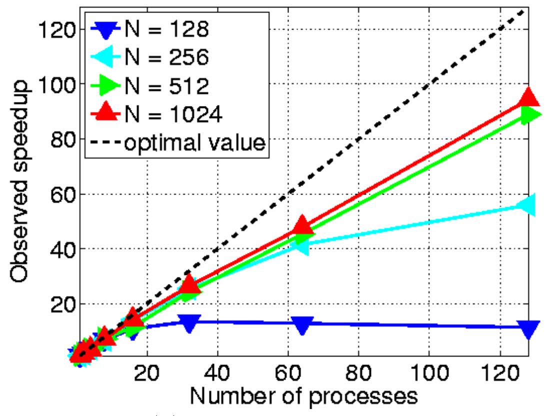
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	00:01:06	00:00:36	00:00:17	00:00:10	00:00:06	00:00:05	00:00:05	00:00:06
$64 \times 64 \times 256$	00:09:43	00:05:30	00:02:42	00:01:27	00:00:42	00:00:23	00:00:14	00:00:10
$128 \times 128 \times 512$	01:40:34	00:56:50	00:29:51	00:14:01	00:08:45	00:04:09	00:02:13	00:01:08
$256 \times 256 \times 1024$	22:15:28	11:27:51	06:43:00	03:05:25	01:34:06	00:50:34	00:27:54	00:14:10

(b) Observed speedup S_p

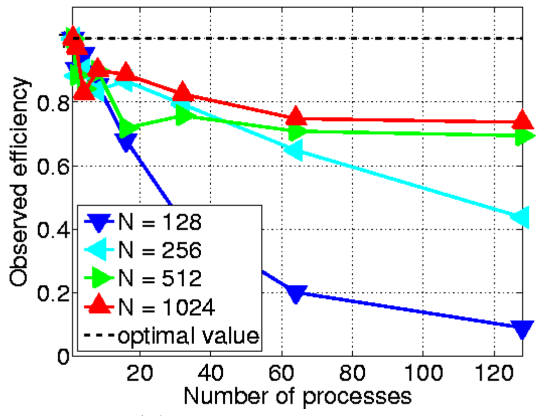
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	1.0000	1.8132	3.8077	6.8332	10.8854	13.4835	12.7988	11.3570
$64 \times 64 \times 256$	1.0000	1.7655	3.6059	6.6991	13.8680	25.3825	41.4858	55.9243
$128 \times 128 \times 512$	1.0000	1.7695	3.3684	7.1759	11.4899	24.2374	45.3304	88.8716
$256 \times 256 \times 1024$	1.0000	1.9415	3.3139	7.2028	14.1912	26.4136	47.8743	94.2651

(c) Observed efficiency E_p

$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$32 \times 32 \times 128$	1.0000	0.9066	0.9519	0.8541	0.6803	0.4214	0.2000	0.0887
$64 \times 64 \times 256$	1.0000	0.8827	0.9015	0.8374	0.8668	0.7932	0.6482	0.4369
$128 \times 128 \times 512$	1.0000	0.8847	0.8421	0.8970	0.7181	0.7574	0.7083	0.6943
$256 \times 256 \times 1024$	1.0000	0.9707	0.8285	0.9004	0.8870	0.8254	0.7480	0.7364



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 3: Performance by number of processes used with 4 processes per node, except for $p = 1$ which uses 1 process per node and $p = 2$ which uses 2 processes per node.

The conclusions discussed so far apply to up to $p = 32$ parallel processes. In each case, only 1 parallel process is run on each node, with the other three cores available to handle all other operating system or other duties. For $p = 64$ and $p = 128$, 2 or 4 processes share each node necessarily, as only 32 nodes are available, thus one expects slightly degraded performance as we go from $p = 32$ to $p = 64$ and $p = 128$. This is born out by all data in Table 3 as well as clearly visible in Figures 3 (a) and (b) for $p > 32$. However, the times in Table 3 (a) for all finer meshes 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 run 2 processes per node in Table 4 and Figure 2, and we run 4 processes per node in Table 5 and Figure 3, wherever possible. That is, for $p = 128$ in Table 4 and Figure 2 entries require 4 processes per node since only 32 nodes are available. On the other hand, in Table 5 and Figure 3 $p = 1$ is always computed on a dedicated node, i.e., running the entire job on a single process on a single node, and $p = 2$ is computed using a two-process job running on a single node. The results in the efficiency plots of Figures 2 (b) and 3 (b) show clearly that there is a significant loss of efficiency when going from $p = 1$ (always on a dedicated node) to $p = 2$ (with both processes on one node) to $p = 4$ (with 4 processes on one node).

The detailed timing data in Tables 3 (a), 4 (a) and 5 (a) confirm this. For example, we observe taking the $p = 32$ case in each table for the finest mesh $256 \times 256 \times 1024$, we get execution times of 41:15 minutes, 45:37 minutes, and 50:34 minutes, respectively, showing an approximate 4 to 5 minutes increase in execution going from 1 process to 2 and then from 2 to 4 processes per node.

The results presented so far indicate clearly the well-known conclusion that best performance improvements, in the sense of halving the time when doubling the number of processes, is achieved by only running one parallel process on each node. But for production runs, we are not interested in this improvement being optimal, but 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, or 4 processes per node. This is answered by the data organized in the form of Table 1 in the Introduction. Namely, extending our observations discussed in the previous paragraph for the finest mesh $256 \times 256 \times 1024$ and assuming now the number of available nodes to be fixed at 8 (as opposed to the number of parallel processes p fixed), we compare the times 2:41:47 hours from column $p = 8$ of Table 3 (a), 1:30:59 hours from column $p = 16$ of Table 4 (a), and again 50:34 minutes from column $p = 32$ of Table 5 (a) to see that running the maximum possible number of 4 processes per node gives us the shortest execution time. This is exactly the observation seen in the column for 8 nodes in Table 1.

A Performance Studies on kali

This appendix summarizes results of analogous studies to the previous sections performed on the cluster kali purchased in 2003. This cluster had originally 32 nodes, each with two (single-core) processors (Intel Xeon 2.0 GHz with 512 kB cache) and 1 GB of memory, connected by a Myrinet interconnect network; only 27 of the 32 nodes are connected by the Myrinet network at present (2008), hence only 16 nodes are available for parallel performance study, when considering only powers of 2 for convenience.

Table 6 collects results which determine the sizing data of the problem. The finest mesh $256 \times 256 \times 1024$ can only be accomodated using (at least) 16 nodes on kali. We note that the number of ODE steps is the same for each mesh size as in Table 2. Comparing the serial raw timings in Table 6 with Table 2, we see that the new processors (using only one core) are roughly two times faster than the old processors.

Table 7 is a summary table of raw timing results analogous to Table 1 in the Introduction. Reading the data row-wise, we observe good speedup confirming that the Myrinet network works well communicating between nodes.

Table 8 and Figure 4 summarize and visualize the underlying performance results for the case of running only 1 process on each node, except $p = 32$ with 2 processes, and are analogous to Table 3 and Figure 1. We note here that the necessary memory for the finest mesh requires at least 16 nodes, therefore speedup is redefined to use only the available data as $S_p := 16T_{16}(N)/T_p(N)$ for the $256 \times 256 \times 1024$ mesh. Comparing the corresponding efficiency data in Table 3 (c) with Table 8 (c), we notice that the efficiency demonstrated by kali is better than that seen in the new IBM machine up to $p = 16$ for all meshes where data is available. However, while considering this speedup result we must recall that the new IBM machine completes the task nearly in half the time as kali for every value of p.

Table 9 and Figure 5 summarize and visualize the performance results for the case of running 2 processes on each node, except $p = 1$ with 1 process. Once more we redefine speedup for the finest mesh $256 \times 256 \times 1024$ to use only the available data as $S_p := 32T_{32}(N)/T_p(N)$. The efficiency plots in Figure 5 (b) and Figure 2 (b) very clearly demonstrate that the performance degradation occurs from $p = 1$ to $p = 2$, that is, it is associated with using both processors per node instead of one process only. Comparing Table 4 (c) with Table 9 (c), we see the new IBM machine demonstrates noticeably better efficiency for p up to $p = 16$ than kali while maintaining the advantage of demonstrating about half the execution time.

Finally, we notice in Table 8 (c) that the efficiency drops off as we go from $p = 16$ to $p = 32$ for every available mesh size. Recalling that the data for $p = 32$ of Table 8 is attained by running 16 nodes with 2 processors leads us to compare Tables 8 (c) and 9 (c) column-wise for each available mesh resolution. We see clearly that using both processors per node noticeably reduces the speedup over using only one processor with the second one idling. This is an observation that used to be accepted fact and is the basis for the statement that "standard dual processor PC's will not provide better performance when the second processor is used" [3, FAQ "What kind of parallel computers or clusters are needed to use PETSc?"]. It is interesting to note that this effect is far less pronounced on the new cluster for this test problem.

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] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. Portable, extensible toolkit for scientific computation (PETSc). `www.mcs.anl.gov/petsc`. Version 2.3.3, released May 23, 2007.

[4] Matthias K. Gobbert. Long-time simulations on high resolution meshes to model calcium waves in a heart cell. *SIAM J. Sci. Comput.*, in press (2008).

[5] 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.

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

[7] 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.

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

Table 6: Sizing study on kali 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.

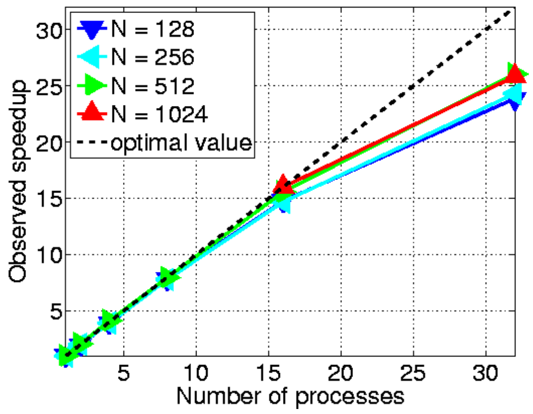
$N_x \times N_y \times N_z$	DOF	`nsteps`	wall clock time		memory usage (MB)	
			HH:MM:SS	seconds	predicted	observed
$32 \times 32 \times 128$	140,481	208	00:02:18	138.40	23	24
$64 \times 64 \times 256$	1,085,825	208	00:20:44	1243.67	174	167
$128 \times 128 \times 512$	8,536,833	208	03:32:22	12742.34	1368	1304
$256 \times 256 \times 1024$	67,700,225	208	N/A	N/A	10847	N/A

Table 7: Performance on kali. Wall clock time in HH:MM:SS for the solution of problems on $N_x \times N_y \times N_z$ meshes using 1, 2, 4, 8, 16 compute nodes with 1 and 2 processes per node.

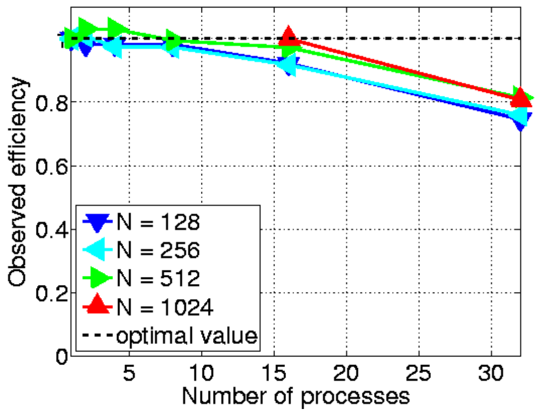
(a) Mesh resolution $N_x \times N_y \times N_z = 32 \times 32 \times 128$, DOF = 140,481					
	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	00:02:18	00:01:11	00:00:35	00:00:18	00:00:09
2 processes per node	00:01:23	00:00:41	00:00:19	00:00:10	00:00:06
(b) Mesh resolution $N_x \times N_y \times N_z = 64 \times 64 \times 256$, DOF = 1,085,825					
	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	00:20:44	00:10:17	00:05:19	00:02:40	0:01:25
2 processes per node	00:12:07	00:06:05	00:03:07	00:01:37	0:00:51
(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
1 process per node	03:32:22	01:42:59	00:51:32	00:26:42	00:13:39
2 processes per node	02:11:44	01:03:32	00:32:15	00:16:12	00:08:09
(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
1 process per node	N/A	N/A	N/A	N/A	02:39:24
2 processes per node	N/A	N/A	N/A	N/A	01:38:44

Table 8: Performance on kali by number of processes used with 1 process per node, except for $p = 32$ which uses 2 processes per node.

(a) Wall clock time in HH:MM:SS						
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$32 \times 32 \times 128$	00:02:18	00:01:11	00:00:35	00:00:18	00:00:09	00:00:06
$64 \times 64 \times 256$	00:20:44	00:10:17	00:05:19	00:02:40	00:01:25	00:00:51
$128 \times 128 \times 512$	03:32:22	01:42:59	00:51:32	00:26:42	00:13:39	00:08:09
$256 \times 256 \times 1024$	N/A	N/A	N/A	N/A	02:39:24	01:38:44
(b) Observed speedup S_p						
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$32 \times 32 \times 128$	1.0000	1.9623	3.9285	7.8547	14.7548	23.8621
$64 \times 64 \times 256$	1.0000	2.0169	3.8984	7.7856	14.6815	24.3047
$128 \times 128 \times 512$	1.0000	2.0622	4.1210	7.9541	15.5578	26.0526
$256 \times 256 \times 1024$	N/A	N/A	N/A	N/A	16.0000	25.8561
(c) Observed efficiency E_p						
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$32 \times 32 \times 128$	1.0000	0.9811	0.9821	0.9818	0.9222	0.7457
$64 \times 64 \times 256$	1.0000	1.0084	0.9746	0.9732	0.9176	0.7595
$128 \times 128 \times 512$	1.0000	1.0311	1.0303	0.9943	0.9724	0.8141
$256 \times 256 \times 1024$	N/A	N/A	N/A	N/A	1.0000	0.8080



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 4: Performance on kali by number of processes used with 1 process per node, except for $p = 32$ which uses 2 processes per node.

Table 9: Performance on kali by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node.

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

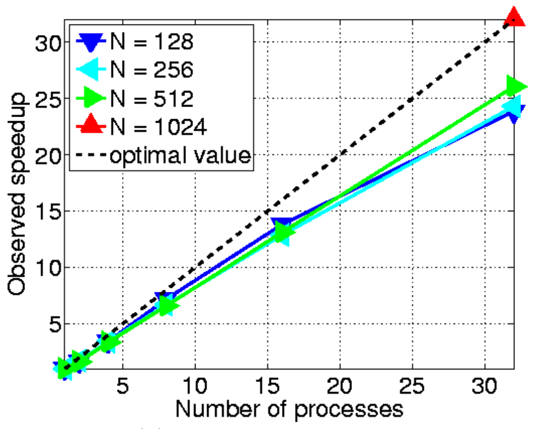
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$32 \times 32 \times 128$	00:02:18	00:01:23	00:00:41	00:00:19	00:00:10	00:00:06
$64 \times 64 \times 256$	00:20:44	00:12:07	00:06:05	00:03:07	00:01:37	00:00:51
$128 \times 128 \times 512$	03:32:22	02:11:44	01:03:32	00:32:15	00:16:12	00:08:09
$256 \times 256 \times 1024$	N/A	N/A	N/A	N/A	N/A	01:38:44

(b) Observed speedup S_p

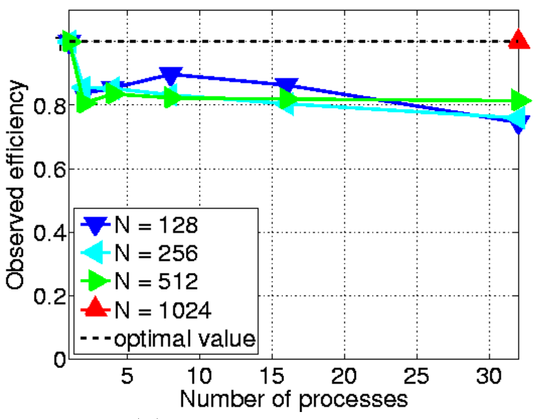
$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$16 \times 16 \times 128$	1.0000	1.6751	3.4148	7.1784	13.8124	23.8621
$32 \times 32 \times 256$	1.0000	1.7116	3.4117	6.6595	12.8678	24.3047
$64 \times 64 \times 512$	1.0000	1.6122	3.3424	6.5839	13.1075	26.0526
$128 \times 128 \times 1024$	N/A	N/A	N/A	N/A	N/A	32.0000

(c) Observed efficiency E_p

$N_x \times N_y \times N_z$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$32 \times 32 \times 128$	1.0000	0.8376	0.8537	0.8973	0.8633	0.7457
$64 \times 64 \times 256$	1.0000	0.8558	0.8529	0.8324	0.8042	0.7595
$128 \times 128 \times 512$	1.0000	0.8061	0.8356	0.8230	0.8192	0.8141
$256 \times 256 \times 1024$	N/A	N/A	N/A	N/A	N/A	1.0000



(a) Observed speedup S_p



(b) Observed efficiency E_p

Figure 5: Performance on kali by number of processes used with 2 processes per node, except for $p = 1$ which uses 1 process per node.