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

Jonathan Graf and Matthias K. Gobbert (gobbert@umbc.edu)

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

Technical Report HPCF–2015–7, www.umbc.edu/hpcf > Publications

Abstract

We report parallel performance studies on each of the three uniform portions, from 2009, 2010, and 2013, of the cluster maya in the UMBC High Performance Computing Facility (HPCF) for a parabolic test problem given by a time-dependent, scalar, linear reaction-diffusion equation in three dimensions. The results show very good performance up to 64 compute nodes on all portions and support several key conclusions: (i) The newer nodes are faster per core as well as per node, however, for most serial production code using one of the 2010 nodes with 2.8 GHz is a good default. (ii) The high-performance interconnect supports parallel scalability on at least 64 nodes near-optimally. (iii) It is often faster to use all cores on modern multi-core nodes but it is useful to track memory usage to determine if this is the case for memory-bound code.

1 Introduction

The UMBC High Performance Computing Facility (HPCF) is the community-based, interdisciplinary core facility for scientific computing and research on parallel algorithms at UMBC. Started in 2008 by more than 20 researchers from ten academic departments and research centers from all three colleges, it is supported by faculty contributions, federal grants, and the UMBC administration. The facility is open to UMBC researchers at no charge. Researchers can contribute funding for long-term priority access. System administration is provided by the UMBC Division of Information Technology, and users have access to consulting support provided by dedicated full-time graduate assistants. See www.umbc.edu/hpcf for more information on HPCF and the projects using its resources.

Released in Summer 2014, the current machine in HPCF is the 240-node distributed-memory cluster maya. The newest components of the cluster are the 72 nodes in maya 2013 with two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs and 64 GB memory that include 19 hybrid nodes with two state-of-the-art NVIDIA K20 GPUs (graphics processing units) designed for scientific computing and 19 hybrid nodes with two cutting-edge 60-core Intel Phi 5110P accelerators. These new nodes are connected along with the 84 nodes in maya 2009 with two quad-core 2.6 GHz Intel Nehalem X5550 CPUs and 24 GB memory by a high-speed quad-data rate (QDR) InfiniBand network for research on parallel algorithms. The remaining 84 nodes in maya 2010 with two quad-core 2.8 GHz Intel Nehalem X5560 CPUs and 24 GB memory are designed for fastest number crunching and connected by a dual-data rate (DDR) InfiniBand network. All nodes are connected via InfiniBand to a central storage of more than 750 TB. The studies in this report use the Intel C compiler version 14.0 (compiler options -std=c99 -Wall -O3) together with Intel MPI version 4.1. This is the default on maya. All results in this report use dedicated nodes with remaining cores idling

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 maya in HPCF. It was shown in [8] that an elliptic test problem given by the stationary Poisson equation in two space dimensions, 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. This report extends the stationary Poisson equation to a time-dependent parabolic problem given by one scalar, time-dependent, linear reaction-diffusion equation in three space dimensions on a rectangular box domain with homogeneous Neumann boundary conditions. Note that 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 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 iterations and not from the time stepping. Section 2 provides a brief introduction to the problem and the numerical algorithm used. The last report of the series [6] extends the time-dependent parabolic test problem to a system of three non-linear advection-reaction-diffusion equations for the application problem of simulating calcium induced

Table 1.1: Observed wall clock time in HH:MM:SS on maya 2013 for mesh resolution $N_x \times N_y \times N_z = 128 \times 128 \times 512$ with 8,536,833 degrees of freedom.

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	02:21:29	01:12:21	00:36:16	00:19:05	00:10:35	00:06:52	00:04:37
2 processes per node	01:11:24	00:35:57	00:18:27	00:09:53	00:05:51	00:04:04	00:03:02
4 processes per node	00:37:56	00:20:15	00:10:39	00:06:17	00:04:05	00:03:16	00:02:44
8 processes per node	00:24:16	00:13:10	00:07:20	00:04:40	00:03:23	00:03:13	00:02:52
16 processes per node	00:18:30	00:10:03	00:05:56	00:04:15	00:03:37	00:03:29	N/A

calcium release (CICR) in a heart cell [3, 5, 11]. This non-linear three-species application problem provides a substantially more computationally intensive test of the cluster, since the model involves additional terms, the solution requires many more time steps than the problem in this report, it involves non-linearities requiring a Newton solver at every time step, and a linear solve inside every Newton iteration.

The parallel, matrix-free implementation involves necessarily communications both collectively among all parallel processes and between pairs of processes in every iteration. Therefore, this method provides an excellent test problem for the overall, real-life performance of a parallel computer. The problem in this report has also been used in [7] and [11] in which the mesh was refined until running out of memory thus demonstrating one key advantage of parallel computing: larger problems can be solved by pooling the memory from several compute nodes. In this report we rather demonstate another key adavntage of parallel computing: for effecient implementations of appropriate algorithms, problems can be solved significantly faster by pooling the processing power of several compute nodes. This report can also be considered an update to [2, 9, 10] in which this problem was used to analyze previous clusters. However, among other differences, simulations in [10] are only run to a final time of 100 ms while the studies in this report use a final time of 1,000 ms, hence the significantly longer run times in this report. In Table 6.1 of Section 6 we show historical comparison of the clusters in the UMBC HPCF using this test problem.

Table 1.1 contains an excerpt of the performance results reported in Table 3.1 of Section 3 for the studies on the newest portion maya 2013 of the cluster. This excerpt reports the results for one mesh resolution and using the default compiler and MPI implementation. Table 1.1 reports the observed wall clock time in HH:MM:SS (hours:minutes:seconds) for all possible combinations of numbers of nodes and processes per node (that are powers of 2), that is, for $1, 2, 4, 8, 16, 32,$ and 64 nodes and $1, 2, 4, 8,$ and 16 processes per node. It is conventional to restrict studies to powers of 2, since this makes it easy to judge if timings are halved when the number of parallel processes is doubled. N/A indicates that the case is not feasible due to $p > (N_z + 1)$, where $N_z + 1$ is the number of finite volume cells on the z-direction for spatial mesh resolution of $N_x \times N_y \times N_z$. We observe that by simply using all cores on one node we can reduce the runtime from approximately 2.5 hours to 18.5 minutes and by using 8 cores on 64 nodes we can reduce the runtime to under 3 minutes. This table demonstrates the power of parallel computing, in which by pooling the memory of several compute nodes to solve larger problems and to dramatically speed up the solution time. But it also demonstrates the potential for further advances: The studies in this report only used the CPUs of the compute nodes; using accelerators such as the GPUs and the Intel Phi has the potential to shorten the runtimes even more.

The remainder of this report is organized as follows: Section 2 details the test problem and discusses the parallel implementation in more detail, Section 3 contains the complete parallel performance studies on maya 2013, from which Table 1.1 was selected. Sections 4 and 5 contain the performance studies on maya 2010 and 2009, respectively. Finally, Section 6 provides a historical comparison of performance of maya and the previous clusters in HPCF. The results for maya 2013 are already contained in the report [4].

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 nos. CNS–0821258 and CNS–1228778) 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.

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 [3, 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

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

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 μm. We set the final time as $T = 1{,}000$ 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 $\mu\text{m}^2/ms$. The initial distribution is chosen to be $u_{\text{ini}}(x, y, z) = \cos^2(\lambda_x x/2)\cos^2(\lambda_y y/2)\cos^2(\lambda_z z/2)$, 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 zero-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 is used as test problem in [2, 3, 7, 11] and its true solution is

$$u(x, y, x, t) = \frac{1 + \cos(\lambda_x x) e^{-D_x \lambda_x^2 t}}{2} \, \frac{1 + \cos(\lambda_y y) e^{-D_y \lambda_y^2 t}}{2} \, \frac{1 + \cos(\lambda_z z) e^{-D_z \lambda_z^2 t}}{2}. \tag{2.2}$$

We are able to reach this steady state with our final simulation time of $T = 1{,}000$ ms. In fact, this steady state can be reached in significantly less time, but we use $T = 1{,}000$ ms here to emulate the long time simulations needed in our related calcium flow in heart cell application.

A method of lines discretization of (2.1) using the finite volume method results in a stiff, large system of ordinary differential equations (ODEs) referred to as the so-called semi-discrete problem [5]. This ODE system is solved by the family of numerical differentiation formulas (NDFk, $1 \leq k \leq 5$) [12]. Since these ODE solvers are implicit, a system of linear equations needs to be solved at every time step. Krylov subspace methods such as BiCGSTAB require the system matrix A only in matrix-vector products. We avoid the storage cost of A by using a so-called matrix-free implementation, 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 BiCGSTAB 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 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 essentially 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 HH:MM:SS (hours:minutes:seconds) 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. For a convergence study for this problem see [7].

Table 2.1: Sizing study (using Intel MPI).

$N_x \times N_y \times N_z$	DOF	`nsteps`	wall clock time		memory usage (MB)	
			HH:MM:SS	seconds	predicted	observed
$16 \times 16 \times 64$	18,785	2015	00:00:41	41.46	9	21
$32 \times 32 \times 128$	140,481	2018	00:01:39	99.41	21	39
$64 \times 64 \times 256$	1,085,825	2020	00:12:17	736.56	143	176
$128 \times 128 \times 512$	8,536,833	2019	02:21:29	8489.12	1,106	1,256
$256 \times 256 \times 1024$	67,700,225	2014	31:07:03	112022.65	10,431	10,127

3 Performance Studies on maya 2013

This section describes the parallel performance studies for the solution of the test problem on the 2013 portion of maya. The 72 nodes of this portion are set up as 67 compute nodes, 2 develop nodes, 1 user node, and 1 management node. A maya 2013 node consists of two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs. Each core of each CPU has dedicated 32 kB of L1 and 256 kB of L2 cache. All eight cores of each CPU share 20 MB of L3 cache. The 64 GB of the node's memory is formed by eight 8 GB DIMMs, four of which are connected to each CPU. The two CPUs of a node are connected to each other by two QPI (quick path interconnect) links. The nodes in maya 2013 are connected by a quad-data rate InfiniBand interconnect.

Table 3.1 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 for the default setup on maya. We consider the test problem for progressively finer meshes of 16 × 16 × 64, 32 × 32 × 128, 64 × 64 × 256, 128 × 128 × 512, and 256 × 256 × 1024. N/A indicates that the case is not feasible due to $p > (N_z + 1)$, where $N_z + 1$ is the number of finite volume cells on the z-direction for spatial mesh resolution of $N_x \times N_y \times N_z$. We can see this applies to the 64 node, 16 processes per node run on the 128 × 128 × 512 mesh since the run would require 1024 parallel processes while the z-direction of the mesh is only 512 units in size. The upper-left entry of each sub-table contains the runtime for the serial run of the code for that particular mesh. The lower-right entry of each sub-table lists the runtime using all cores of both 8-core processors on 64 nodes for a total of 1024 parallel processes working together to solve the problem. We observe the advantage of parallel computing for instance for the 128 × 128 × 512 mesh, where the serial run of about 2.5 hours can be reduced to approximately 3 minutes by using 512 parallel processes.

Reading along each row of the table, we observe that the doubling the number of nodes used, and thus also doubling the number of parallel processes, we approximately halve the runtime to at least 16 nodes. For instance, if we take 1 process per node on the 128 × 128 × 512 mesh, we observe that doubling the number of nodes from 1 node to 2 nodes results in an improvement in runtime from 02:21:29 to 01:12:21, an improvement by a factor of 1.96. This continues along the row with factors of improvement of 1.99 from 2 to 4 nodes and 1.90 from 4 to 8 nodes. Theses factors decrease to 1.80 from 8 to 16 nodes, 1.54 from 16 to 32 nodes, and only 1.48 from 32 to 64 nodes.

In order to observe the effect of running different numbers of processes per node, we read along each column of a sub-table. We observe that in most columns the runtime is approximately halved by doubling the processes per node from 1 to 2. We also observe that the runtime is approximately halved by doubling the processes per node from 2 to 4. However in most cases we observe only a small improvement in runtime by doubling the processes per node from 4 to 8. We also observe only a small improvement in runtime in most cases doubling the process per node from 8 to the maximum 16.

In a few select cases, we observe an increase in total runtime after increasing the number of parallel processes. For runtimes under a minute, we are not very concerned as in the 16 × 16 × 64 and the 32 × 32 × 128 meshes. For the larger mesh sizes in the case of 16 nodes we observe small increases in runtime from 8 process per node to 16 process per node. When avaiable, we see the same behavior from 8 to 16 process per node using 32 nodes. Still, these are small increases in runtime. The case that is most concerning is the 256 × 256 × 1024 mesh's 64 node case where going from 8 to 16 processes per node results in a significant increase in runtime from 00:20:00 to 44:57:49.

Table 3.2 collects the results of the performance study by number of processes. Each row lists the results for one problem size. Each column corresponds to the number of parallel processes p used in the run. Data is based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$, where not all of the 16 cores of one node are utilized. This table is intended to demonstrate strong scalability, which is also 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 $S_p = p$. 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 $E_p = 1$.

Table 3.2 (b) shows the speedup observed. The speedup S_p is increasing significantly as we increase the number of processes. However, the ratio over the optimal value of speedup p decreases as we increase the number of processes. We also observe that the speedup is better for larger problems. Table 3.2 (c) shows the observed efficiency E_p. The primary decrease of efficiency is between $p = 8$ and $p = 16$, similar to studies in [8] but not as severe. This suggests the bottle neck of CPU memory channels we observed in [8] may still be affecting the scalability of this problem. The fundamental reason for the speedup and efficiency to trail off is that simply too little work is performed on each process. Due to the one-dimensional split in the z-direction into as many subdomains as parallel processes p, eventually only one or two x-y-planes of data are located on each process. This is not enough calculation work to justify the cost of communicating between the processes. In effect, this leads to a recommendation how many nodes to use for a particular $N_x \times N_y \times N_z$ mesh, with more nodes being justifiable for larger meshes.

The customary graphical representations of speedup and efficiency are presented in Figure 3.1 (a) and (b), respectively. Figure 3.1 (a) shows the speedup pattern as we observed in Table 3.2 (b) but more intuitively. 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.

Table 3.1: Observed wall clock time in HH:MM:SS on maya 2013 by the number of nodes and processes per node using the Intel compiler with Intel MPI.

(a) Mesh resolution $N_x \times N_y \times N_z = 16 \times 16 \times 64$, DOF = 18,785

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:00:41	00:00:33	00:00:28	00:00:27	00:00:25	00:00:15	00:00:17
2 processes per node	00:00:35	00:00:31	00:00:28	00:00:25	00:00:22	00:00:15	N/A
4 processes per node	00:00:34	00:00:30	00:00:29	00:00:24	00:00:24	N/A	N/A
8 processes per node	00:00:35	00:00:31	00:00:31	00:00:27	N/A	N/A	N/A
16 processes per node	00:00:28	00:00:32	00:00:35	N/A	N/A	N/A	N/A

(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	00:01:39	00:01:06	00:00:49	00:00:37	00:00:29	00:00:21	N/A
2 processes per node	00:01:04	00:00:47	00:00:39	00:00:31	00:00:28	00:00:18	N/A
4 processes per node	00:00:50	00:00:32	00:00:36	00:00:30	00:00:25	00:00:21	N/A
8 processes per node	00:00:28	00:00:37	00:00:37	00:00:30	00:00:26	N/A	N/A
16 processes per node	00:00:31	00:00:32	00:00:31	00:00:39	N/A	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	00:12:17	00:06:22	00:03:29	00:02:04	00:01:23	00:00:58	00:00:54
2 processes per node	00:06:07	00:03:24	00:02:04	00:01:17	00:00:58	00:00:43	00:00:45
4 processes per node	00:03:23	00:02:01	00:01:28	00:00:59	00:00:58	00:00:41	00:00:46
8 processes per node	00:02:07	00:01:20	00:01:09	00:00:58	00:00:54	00:00:46	N/A
16 processes per node	00:01:35	00:01:04	00:00:56	00:00:58	00:01:10	N/A	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	02:21:29	01:12:21	00:36:16	00:19:05	00:10:35	00:06:52	00:04:37
2 processes per node	01:11:24	00:35:57	00:18:27	00:09:53	00:05:51	00:04:04	00:03:02
4 processes per node	00:37:56	00:20:15	00:10:39	00:06:17	00:04:05	00:03:16	00:02:44
8 processes per node	00:24:16	00:13:10	00:07:20	00:04:40	00:03:23	00:03:13	00:02:52
16 processes per node	00:18:30	00:10:03	00:05:56	00:04:15	00:03:37	00:03:29	N/A

(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	31:07:03	14:42:57	07:22:12	04:03:18	02:09:59	01:12:50	00:44:08
2 processes per node	14:36:21	07:21:57	04:00:44	02:07:55	01:08:40	00:40:30	00:25:51
4 processes per node	07:48:55	04:13:18	02:13:59	01:12:19	00:43:01	00:28:20	00:20:43
8 processes per node	04:52:40	02:40:52	01:28:22	00:52:26	00:33:19	00:24:17	00:20:00
16 processes per node	03:47:31	02:04:31	01:11:32	00:45:10	00:30:43	00:24:42	44:57:49

Table 3.2: Intel compiler with Intel MPI performance on maya 2013 by number of processes used with 16 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, $p = 4$ which uses 4 processes per node, and $p = 8$ which uses 8 processes per node.

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

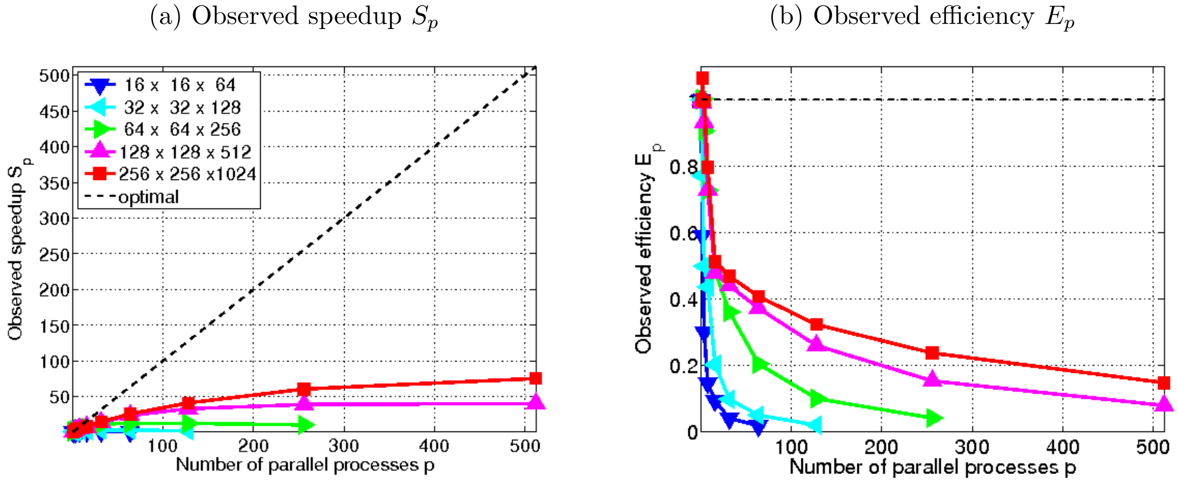
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	00:00:41	00:00:35	00:00:34	00:00:35	00:00:28	00:00:32	00:00:35	N/A	N/A	N/A
32	00:01:39	00:01:04	00:00:50	00:00:28	00:00:31	00:00:32	00:00:31	00:00:39	N/A	N/A
64	00:12:17	00:06:07	00:03:23	00:02:07	00:01:35	00:01:04	00:00:56	00:00:58	00:01:10	N/A
128	02:21:29	01:11:24	00:37:56	00:24:16	00:18:30	00:10:03	00:05:56	00:04:15	00:03:37	00:03:29
256	31:07:03	14:36:21	07:48:55	04:52:40	03:47:31	02:04:31	01:11:32	00:45:10	00:30:43	00:24:42

(b) Observed speedup S_p

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	1.00	1.18	1.20	1.18	1.50	1.29	1.18	N/A	N/A	N/A
32	1.00	1.54	1.99	3.49	3.24	3.14	3.20	2.56	N/A	N/A
64	1.00	2.01	3.62	5.82	7.75	11.54	13.09	12.81	10.60	N/A
128	1.00	1.98	3.73	5.83	7.65	14.07	23.82	33.26	39.20	40.57
256	1.00	2.13	3.98	6.38	8.21	14.99	26.10	41.33	60.77	75.58

(c) Observed efficiency E_p

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	1.00	0.59	0.30	0.15	0.09	0.04	0.02	N/A	N/A	N/A
32	1.00	0.77	0.50	0.44	0.20	0.10	0.05	0.02	N/A	N/A
64	1.00	1.00	0.91	0.73	0.48	0.36	0.20	0.10	0.04	N/A
128	1.00	0.99	0.93	0.73	0.48	0.44	0.37	0.26	0.15	0.08
256	1.00	1.07	1.00	0.80	0.51	0.47	0.41	0.32	0.24	0.15


(a) Observed speedup S_p

(b) Observed efficiency E_p

Figure 3.1: Intel compiler with Intel MPI performance on maya 2013 by number of processes used with 16 processes per node, except for $p = 1$ which uses 1 process per node, $p = 2$ which uses 2 processes per node, $p = 4$ which uses 4 processes per node, $p = 8$ which uses 8 processes per node.

4 Performance Studies on maya 2010

This section describes the parallel performance studies on maya 2010 for the solution of the parabolic test problem. This portion consists of 84 nodes with two quad-core 2.8 GHz Intel Nehalem X5560 CPUs. Each CPU is connected to 12 GB of DDR3 memory through three memory channels. Nodes are connected to by a dual-data rate Ininiband network. The results use the default distribution of MPI processes to the cores on the node.

Table 4.1 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 for the default set up on maya. We consider the test problem for progressively finer meshes of $16 \times 16 \times 64$, $32 \times 32 \times 128$, $64 \times 64 \times 256$, $128 \times 128 \times 512$, and $256 \times 256 \times 1024$. The upper-left entry of each sub-table contains the runtime for the serial run of the code for that particular mesh. The lower-right entry of each sub-table lists the runtime using all cores of both 4-core processors on 64 nodes for a total of 512 parallel processes working together to solve the problem. N/A indicates that the case is not feasible due to $p > (N_z + 1)$, where $N_z + 1$ is the number of finite volume cells on the z-direction for spatial mesh resolution of $N_x \times N_y \times N_z$. We observe the advantage of parallel computing for the $256 \times 256 \times 1024$ mesh where the serial run of about 37 hours can be reduced to approximately 30 minutes by using 512 parallel processes.

Reading along each row of the table, we observe that the doubling the number of nodes used, and thus also doubling the number of parallel process, we approximately halve the runtime. For the $256 \times 256 \times 1024$ mesh we observe that doubling the number of nodes from 1 node to 2 nodes results in an improvement in runtime from 37:26:19 to 19:08:30, an improvement by a factor of 1.96. This continues along the row with factors of improvement of 2.04 from 2 to 4 nodes, 1.80 from 4 to 8 nodes, 1.82 from 8 to 16 nodes, 1.79 from 16 to 32 nodes, and 2.10 from 32 to 64 nodes.

In order to observe the effect of running different numbers of processes per node, we read along each column of a sub-table. We observe that in most columns the runtime is approximately halved by doubling the processes per node from 1 to 2. But perfomance does degrade as the number of processes increases. We observe that the runtime is less than halved by doubling the processes per node from 2 to 4. In most cases we observe a small improvement in runtime by doubling the processes per node from 4 to 8. For instance, if we take 1 process per node on the $256 \times 256 \times 1024$ mesh, we observe that doubling the number of processes per node from 1 to 2 processes results in an improvement in runtime from 37:26:19 to 19:10:41, an improvement by a factor of 1.95. Doubling the processes per node from 2 to 4 processes the factor of improvement is 1.54. Further we observe that performance degrades more going from 4 to 8 processes per node with improvement by only a factor 1.08.

In all 32 and 64 node available cases, we observe an increase in total runtime when increasing the number of processes per node from 4 to 8. We also observe this increase in the 2 node case for the $128 \times 128 \times 512$ mesh. For the most part these are small increases. The largest of these occurs for the mesh $256 \times 256 \times 1024$ with 64 nodes and using all available processess per node as in the maya 2013 results. However here the increase is much smaller than that of maya 2013. We observe an increase in runtime from 00:23:25 to 00:31:38 rather than 00:20:00 to 44:57:49 which was observed for maya 2013.

Table 4.2 collects the results of the performance study by number of processes. Each row lists the results for one problem size. Each column corresponds to the number of parallel processes p used in the run. Data is based on 8 processes per node, except for the cases $p = 1, 2, 4$, where not all of the 8 cores of one node are utilized. This table is intended to demonstrate strong scalability, which is also 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 $S_p = p$. 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 $E_p = 1$.

Table 4.2 (b) shows the speedup observed. The speedup S_p is increasing significantly as we increase the number of processes. However, the ratio over the optimal value of speedup p decreases as we increase the number of processes. We also observe that the speedup is better for larger problems. Table 4.2 (c) shows the observed efficiency E_p. Efficiency does not decrease significantly between $p = 8$ and $p = 16$ as we observed in Table 3.2 (c). The graphical representation of speedup and efficiency are presented in Figure 4.1 (a) and (b), respectively.

Table 4.1: Observed wall clock time in HH:MM:SS on maya 2010 using the Intel compiler with Intel MPI.

(a) Mesh resolution $N_x \times N_y \times N_z = 16 \times 16 \times 64$, DOF = 18,785

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:00:27	00:00:36	00:00:22	00:00:23	00:00:20	00:00:21	N/A
2 processes per node	00:00:22	00:00:30	00:00:22	00:00:22	00:00:21	00:00:10	N/A
4 processes per node	00:00:17	00:00:28	00:00:19	00:00:18	00:00:17	N/A	N/A
8 processes per node	00:00:28	00:00:28	00:00:19	00:00:18	N/A	N/A	N/A

(b) 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	00:01:50	00:01:14	00:00:43	00:00:32	00:00:26	00:00:24	00:00:20
2 processes per node	00:01:02	00:00:50	00:00:32	00:00:25	00:00:32	00:00:23	00:00:20
4 processes per node	00:00:39	00:00:30	00:00:23	00:00:21	00:00:20	00:00:19	N/A
8 processes per node	00:00:31	00:00:22	00:00:22	00:00:21	00:00:30	N/A	N/A

(c) 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	00:16:29	00:08:19	00:04:15	00:02:16	00:01:22	00:00:59	00:00:46
2 processes per node	00:08:15	00:04:14	00:02:18	00:01:25	00:00:55	00:00:43	00:00:40
4 processes per node	00:04:57	00:03:30	00:01:24	00:00:58	00:00:42	00:00:38	00:00:37
8 processes per node	00:03:35	00:01:51	00:01:00	00:00:46	00:01:09	00:00:44	N/A

(d) 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	03:03:33	01:37:03	00:45:04	00:33:57	00:19:11	00:09:18	00:03:49
2 processes per node	01:35:21	00:56:45	00:23:20	00:15:13	00:13:05	00:07:09	00:02:39
4 processes per node	00:58:25	00:33:12	00:14:48	00:11:02	00:09:41	00:03:12	00:02:09
8 processes per node	00:39:36	00:38:52	00:11:20	00:05:55	00:07:02	00:03:18	00:03:50

(e) Mesh resolution $N_x \times N_y \times N_z = 256 \times 256 \times 1024$, DOF = 67,700,225

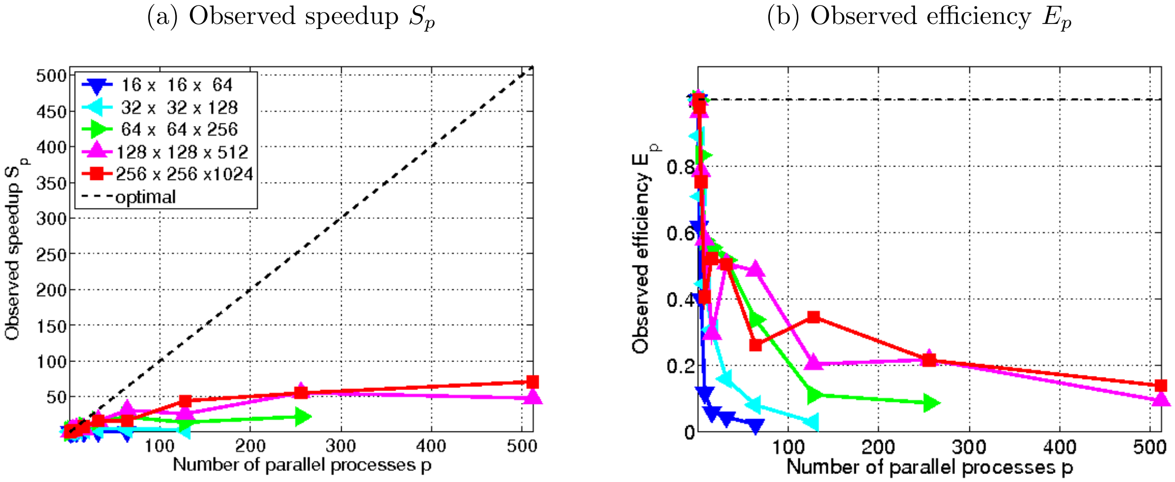
	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	37:26:19	19:08:30	09:21:53	05:11:29	02:50:54	01:35:18	00:45:19
2 processes per node	19:10:41	09:26:16	04:49:58	03:07:53	01:29:54	00:52:59	00:29:27
4 processes per node	12:25:36	05:53:19	03:00:34	02:16:09	01:00:57	00:39:36	00:23:25
8 processes per node	11:31:35	04:28:41	02:19:10	02:14:24	00:50:46	00:40:38	00:31:38

Table 4.2: Intel compiler with Intel MPI performance on maya 2010 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	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	00:00:27	00:00:22	00:00:17	00:00:28	00:00:28	00:00:19	00:00:18	N/A	N/A	N/A
32	00:01:50	00:01:02	00:00:39	00:00:31	00:00:22	00:00:22	00:00:21	00:00:30	N/A	N/A
64	00:16:29	00:08:15	00:04:57	00:03:35	00:01:51	00:01:00	00:00:46	00:01:09	00:00:44	N/A
128	03:03:33	01:35:21	00:58:25	00:39:36	00:38:52	00:11:20	00:05:55	00:07:02	00:03:18	00:03:50
256	37:26:19	19:10:41	12:25:36	11:31:35	04:28:41	02:19:10	02:14:24	00:50:46	00:40:38	00:31:38

(b) Observed speedup S_p										
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	1.00	1.24	1.60	0.94	0.95	1.43	1.45	N/A	N/A	N/A
32	1.00	1.78	2.83	3.57	4.92	5.09	5.16	3.73	N/A	N/A
64	1.00	2.00	3.33	4.61	8.88	16.55	21.64	14.26	22.23	N/A
128	1.00	1.93	3.14	4.64	4.72	16.19	31.04	26.07	55.57	47.97
256	1.00	1.95	3.01	3.25	8.36	16.14	16.71	44.25	55.28	71.00

(c) Observed efficiency E_p										
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	1.00	0.62	0.40	0.12	0.06	0.04	0.02	N/A	N/A	N/A
32	1.00	0.89	0.71	0.45	0.31	0.16	0.08	0.03	N/A	N/A
64	1.00	1.00	0.83	0.58	0.56	0.52	0.34	0.11	0.09	N/A
128	1.00	0.96	0.79	0.58	0.30	0.51	0.48	0.20	0.22	0.09
256	1.00	0.98	0.75	0.41	0.52	0.50	0.26	0.35	0.22	0.14


(a) Observed speedup S_p

(b) Observed efficiency E_p

Figure 4.1: Intel compiler with Intel MPI performance on maya 2010 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 user 4 processes per node.

5 Performance Studies on maya 2009

This section describes the parallel performance studies on maya 2009 for the solution of the parabolic test problem. This portion consists of 84 nodes. Each of these nodes consists of two quad-core 2.6 GHz Intel Nehalem X5550 CPUs. The 24 GB of the node's memory is the combination of six 4 GB DIMMs, three of which are connected to each CPU. Nodes are connected to by a quad-data rate InfiniBand interconnect.

Table 5.1 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 for the default set up on maya. We consider the test problem for progressively finer meshes of $16 \times 16 \times 64$, $32 \times 32 \times 128$, $64 \times 64 \times 256$, $128 \times 128 \times 512$, and $256 \times 256 \times 1024$. The upper-left entry of each sub-table contains the runtime for the serial run of the code for that particular mesh. The lower-right entry of each sub-table lists the runtime using all cores of both 4-core processors on 64 nodes for a total of 512 parallel processes working together to solve the problem. N/A indicates that the case is not feasible due to $p > (N_z + 1)$, where $N_z + 1$ is the number of finite volume cells on the z-direction for spatial mesh resolution of $N_x \times N_y \times N_z$. We observe the advantage of parallel computing for the $256 \times 256 \times 1024$ mesh where the serial run of almost 41 hours can be reduced to approximately 16 minutes by using 512 parallel processes.

Reading along each row of the table, we observe that the doubling the number of nodes used, and thus also doubling the number of parallel process, we approximately halve the runtime. For the $256 \times 256 \times 1024$ mesh we observe that doubling the number of nodes from 1 node to 2 nodes results in an improvement in runtime from 40:59:00 to 19:32:22, an impressive improvement by a factor of 2.10. This continues along the row with factors of improvement of 2.02 from 2 to 4 nodes, 1.96 from 4 to 8 nodes, 1.95 from 8 to 16 nodes, 1.89 from 16 to 32 nodes, and 1.82 from 32 to 64 nodes.

In order to observe the effect of running different numbers of processes per node, we read along each column of a sub-table. We observe that in most columns the runtime is approximately halved by doubling the processes per node from 1 to 2. But perfomance does degrade as the number of processes increases. We observe that the runtime is less than halved by doubling the processes per node from 2 to 4. In almost every case we observe a small improvement in runtime by doubling the processes per node from 4 to 8. For instance, if we take 1 process per node on the $256 \times 256 \times 1024$ mesh, we observe that doubling the number of processes per node from 1 to 2 processes results in an improvement in runtime from 40:59:00 to 19:58:58, an impressive improvement by a factor of 2.05. Doubling the processes per node from 2 to 4 processes the factor of improvement is 1.66. Further we observe that performance degrades more going from 4 to 8 processes per node with improvement by only a factor 1.35.

In almost all 32 and 64 node available cases, we observe a slight speedup when increasing to the number of processes per node, in this case from 4 to 8. The only exceptions are the small increases in runtime for 32 nodes on the $64 \times 64 \times 256$ mesh and 64 nodes on the $128 \times 128 \times 512$ mesh. This is in contrast to the increases in runtime from 4 to 8 processes per node that were observed in maya 2010 and maya 2013 results. On the $256 \times 256 \times 1024$ mesh with 64 nodes and using all available processess per node resulted in an increase from 00:20:00 to 44:57:49 on maya 2013 and an increase from 00:23:25 to 00:31:38 on maya 2010. On maya 2009 we observe a decrease in runtime from 00:16:55 to 00:15:55.

Table 5.2 collects the results of the performance study by number of processes. Each row lists the results for one problem size. Each column corresponds to the number of parallel processes p used in the run. Data is based on 8 processes per node, except for the cases $p = 1, 2, 4$, where not all of the 8 cores of one node are utilized. This table is intended to demonstrate strong scalability, which is also 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 $S_p = p$. 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 $E_p = 1$.

Table 5.2 (b) shows the speedup observed. The speedup S_p is increasing significantly as we increase the number of processes. However, the ratio over the optimal value of speedup p decreases as we increase the number of processes. We also observe that the speedup is better for larger problems. Table 5.2 (c) shows the observed efficiency E_p. The graphical representation of speedup and efficiency are presented in Figure 5.1 (a) and (b), respectively.

Table 5.1: Observed wall clock time in HH:MM:SS on maya 2009 using the Intel compiler with Intel MPI.

(a) Mesh resolution $N_x \times N_y \times N_z = 16 \times 16 \times 64$, DOF = 18,785

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:00:28	00:00:21	00:00:19	00:00:18	00:00:17	00:00:17	00:00:17
2 processes per node	00:00:39	00:00:21	00:00:18	00:00:17	00:00:18	00:00:17	N/A
4 processes per node	00:00:18	00:00:18	00:00:15	00:00:15	00:00:16	N/A	N/A
8 processes per node	00:00:16	00:00:16	00:00:16	00:00:16	N/A	N/A	N/A

(b) 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	00:01:57	00:01:07	00:00:42	00:00:29	00:00:23	00:00:21	00:00:20
2 processes per node	00:01:06	00:00:43	00:00:29	00:00:23	00:00:21	00:00:19	00:00:20
4 processes per node	00:00:43	00:00:30	00:00:22	00:00:19	00:00:18	00:00:17	N/A
8 processes per node	00:00:30	00:00:23	00:00:20	00:00:18	00:00:18	N/A	N/A

(c) 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	00:17:18	00:08:45	00:04:29	00:02:21	00:01:22	00:00:53	00:00:39
2 processes per node	00:08:35	00:04:21	00:02:21	00:01:20	00:00:52	00:00:39	00:00:33
4 processes per node	00:05:05	00:02:36	00:01:24	00:00:52	00:00:38	00:00:32	00:00:35
8 processes per node	00:03:33	00:01:49	00:01:01	00:00:41	00:00:33	00:00:34	N/A

(d) 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	03:01:17	01:32:07	00:46:51	00:24:09	00:12:32	00:06:32	00:03:41
2 processes per node	01:34:18	00:47:50	00:24:33	00:12:27	00:06:31	00:03:39	00:02:20
4 processes per node	00:56:11	00:28:57	00:15:05	00:07:41	00:04:10	00:02:25	00:01:50
8 processes per node	00:40:13	00:21:19	00:11:25	00:05:52	00:03:18	00:02:14	00:02:01

(e) 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 processes per node	40:59:00	19:32:22	09:41:39	04:57:31	02:32:53	01:21:01	00:44:33
2 processes per node	19:58:58	09:50:24	05:02:33	02:34:05	01:20:55	00:43:44	00:24:17
4 processes per node	12:02:21	06:03:56	03:04:53	01:35:44	00:51:09	00:28:37	00:16:55
8 processes per node	08:57:03	04:30:39	02:19:31	01:13:59	00:44:36	00:25:47	00:15:55

Table 5.2: Intel compiler with Intel MPI performance on maya 2009 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

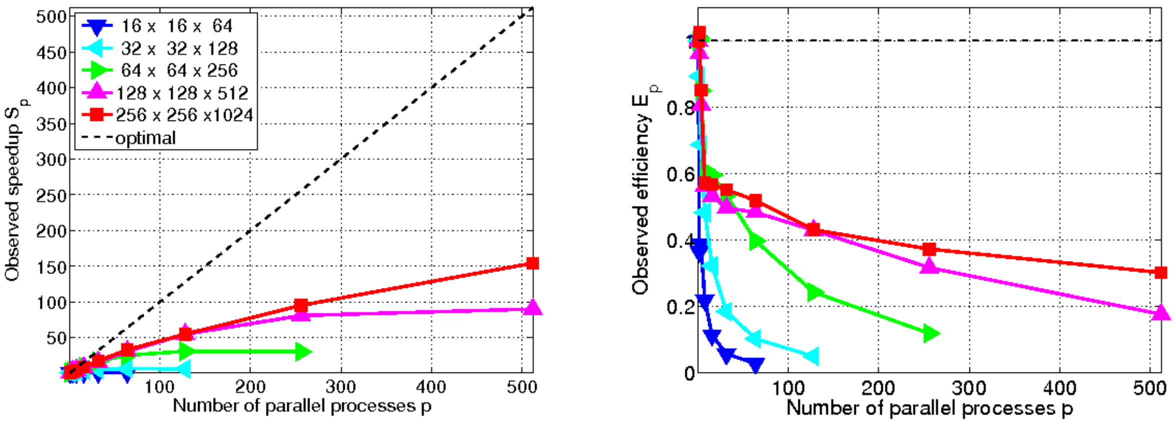
N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	00:00:28	00:00:39	00:00:18	00:00:16	00:00:16	00:00:16	00:00:16	N/A	N/A	N/A
32	00:01:57	00:01:06	00:00:43	00:00:30	00:00:23	00:00:20	00:00:18	00:00:18	N/A	N/A
64	00:17:18	00:08:35	00:05:05	00:03:33	00:01:49	00:01:01	00:00:41	00:00:33	00:00:34	N/A
128	03:01:17	01:34:18	00:56:11	00:40:13	00:21:19	00:11:25	00:05:52	00:03:18	00:02:14	00:02:01
256	40:59:00	19:58:58	12:02:21	08:57:03	04:30:39	02:19:31	01:13:59	00:44:36	00:25:47	00:15:55

(b) Observed speedup S_p

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	1.00	0.73	1.55	1.75	1.82	1.80	1.82	N/A	N/A	N/A
32	1.00	1.79	2.75	3.86	5.15	5.90	6.56	6.36	N/A	N/A
64	1.00	2.02	3.40	4.88	9.52	17.11	25.45	31.11	30.45	N/A
128	1.00	1.92	3.23	4.51	8.50	15.88	30.93	54.96	81.09	90.16
256	1.00	2.05	3.40	4.58	9.09	17.63	33.24	55.13	95.39	154.51

(c) Observed efficiency E_p

N	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
16	1.00	0.36	0.39	0.22	0.11	0.06	0.03	N/A	N/A	N/A
32	1.00	0.89	0.69	0.48	0.32	0.18	0.10	0.05	N/A	N/A
64	1.00	1.01	0.85	0.61	0.60	0.53	0.40	0.24	0.12	N/A
128	1.00	0.96	0.81	0.56	0.53	0.50	0.48	0.43	0.32	0.18
256	1.00	1.03	0.85	0.57	0.57	0.55	0.52	0.43	0.37	0.30



(a) Observed speedup S_p

(b) Observed efficiency E_p

Figure 5.1: Intel compiler with Intel MPI performance on maya 2009 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 user 4 processes per node.

6 Comparisons and Conclusions

Table 6.1 contains a summary of results obtained on the three portions of the cluster maya as well as a comparison to previous HPCF clusters. The table reports results for the historical parabolic scalar (1 species) linear test problem (heat equation) with mesh resolution of $N_x \times N_y \times N_z = 64 \times 64 \times 256$, which was the largest resolution that could be solved on kali in 2003 (using the extended memory of 4 GB on the storage node). Note that for comparability, the results in this section are for a solution for final time of $T = 100$ ms, in contrast to the results the previous sections for a final time of $T = 1{,}000$ ms. Also, to maintain backward compability, this table is restricted to 32 nodes, since the old clusters kali and hpc had fewer nodes than maya. The first row of the table contains the results for the cluster kali from 2003. This cluster was a 33-node distributed-memory cluster with 32 compute nodes including a storage node (with extended memory of 4 GB), containing the 0.5 TB central storage, each with two (single-core) Intel Xeon 2.0 GHz processors and 1 GB of memory, connected by a Myrinet interconnect, plus 1 combined user/management node. Note that for the case of all cores on 1 node, that is, for the case of both (single-core) CPUs used simultaneously, the performance was worse than for 1 CPU and hence the results were not recorded at the time. The second row of the table contains results for the cluster hpc from 2008 which was a 35-node distributed-memory cluster with 33 compute nodes plus 1 development and 1 combined user/management node, each equipped with two dual-core AMD Opteron processors and at least 13 GB of memory, connected by a DDR InfiniBand network and with an InfiniBand-accessible 14 TB parallel file system. The third row contains results for the cluster tara from 2009, which was an 86-node distributed-memory cluster with two quad-core Intel Nehalem processors and 24 GB per node, a QDR InfiniBand interconnect, and 160 TB central storage. This cluster is now part of the cluster maya as maya 2009, and its QDR InfiniBand network extends to the newest portion maya 2013. The fourth row of the table contains results for maya 2009, which recomputes the results from tara using the current default compiler and MPI implementation. The second to last row of the table contains results for the DDR InfiniBand connected portion maya 2010, and the last row contains results for the QDR InfiniBand connected portion maya 2013. It must be noted that for this problem size a 32 node 16 processes per node run is not possible so the 32 node 8 processes per node timing is used.

On the cluster kali, we observed a factor of approximately 28 speedup by increasing the number of nodes from 1 to 32. By using both cores on each node we only see a factor of approximately 34 speedup. We do not observe the expected 64 factor speedup, since both CPUs on the node share a bus connection to the memory, which leads to contention in essentially synchronized algorithms like those in this problem. For more analysis see [2]. Note that there are four cores on each node of cluster hpc compared to just two on the cluster kali, since the CPUs are dual-core. We observe approximately fourfold speedup that we would expect by running it on four cores rather than one. By running on 32 nodes with one core per node we observe less than the expected speedup of approximately 27. Finally, by using all cores on 32 nodes we observe a speedup of 58.30, less than the optimal speedup of 128 [9]. On the cluster tara, we observe a less than optimal speedup of 6.5 by running on all 8 cores rather than on one, caused by the cores of a CPU competing for memory access. By running on 32 nodes with one core per node we observe a speedup of approximately 23. Finally, by using all 8 cores on 32 nodes we observe a speedup of 68, much less than the optimal speedup of 256 [10]. On maya 2009, 2010, and 2013 the run times for this problem are simply too short to make performance conclusions. This is another reason why the studies in this report use a final time of 1000 rather than the 100 final time used in this comparison table. From the maya 2009, 2010, and 2013 results we can observe that the 2013 hardware is the fastest and that the 2009 and 2010 performance are essentially identical in this case.

The results for maya 2010 and 2013 in Table 6.1 demonstrate several typical features of strong scalability studies: One node of the newer hardware is faster in absolute terms. The speedup when using 32 nodes with one core only

Table 6.1: Runtimes (speedup) for $N_x \times N_y \times N_z = 64 \times 64 \times 256$ on the clusters kali, hpc, tara, and maya.

Cluster (year)	method	serial	(1 node)	32 node	32 node
		1 core	all cores	1 core per node	all cores
		time	time (speedup)	time (speedup)	time (speedup)
kali (2003) [2]	FEM	02:19:23	N/A (N/A)	00:05:07 (27.51)	00:04:09 (33.56)
hpc (2008) [9]	FEM	00:09:43	00:02:42 (3.60)	00:00:22 (26.50)	00:00:10 (58.30)
tara (2009) [10]	FEM	00:04:33	00:00:42 (6.50)	00:00:12 (22.75)	00:00:04 (68.25)
maya (2009)	FVM	00:01:36	00:00:19 (4.99)	00:00:05 (19.44)	00:00:03 (28.11)
maya (2010)	FVM	00:01:32	00:00:19 (4.84)	00:00:05 (17.25)	00:00:03 (26.86)
maya (2013)	FVM	00:01:09	00:00:11 (6.09)	00:00:06 (12.29)	00:00:04 (16.12)

has the chance to be better for the 2010 nodes with the worse serial result and the DDR InfiniBand interconnect. This advantage becomes even more pronounced when using all cores on 32 nodes, since the QDR InfiniBand does not profit from its bandwidth advantage together with run times that are just too small for the newer hardware. Even though speedup is less than optimal when using all cores in a node, it is often still faster to use all cores, as opposed to idling some, on the modern multi-core nodes but memory usage should be tracked to determine if this is the case for memory-bound code. Finally, it is obvious and expected that the newer nodes are faster per core as well as per node, however, for most serial production code, that uses only 1 core, using one of the 2010 nodes with 2.8 GHz is a good default, and its DDR InfiniBand interconnect is no disadvantage for serial jobs.

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] 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.

[3] 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.

[4] Jonathan Graf and Matthias K. Gobbert. Parallel performance studies for a parabolic test problem on maya 2013. Technical Report HPCF–2014–7, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

[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] Xuan Huang and Matthias K. Gobbert. Parallel performance studies for a three-species application problem on the cluster maya. Technical Report HPCF–2015–8, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2015.

[7] Xuan Huang, Matthias K. Gobbert, Bradford E. Peercy, Stefan Kopecz, Philipp Birken, and Andreas Meister. Order investigation of scalable memory-efficient finite volume methods for parabolic advection-diffusion-reaction equations with point sources. In preparation (2015).

[8] Samuel Khuvis and Matthias K. Gobbert. Parallel performance studies for an elliptic test problem on the cluster maya. Technical Report HPCF–2015–6, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2015.

[9] 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.

[10] 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.

[11] Jonas Schäfer, Xuan Huang, Stefan Kopecz, Philipp Birken, Matthias K. Gobbert, and Andreas Meister. A memory-efficient finite volume method for advection-diffusion-reaction systems with non-smooth sources. *Numer. Methods Partial Differential Equations*, vol. 31, no. 1, pp. 143–167, 2015.

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