Comparison of Parallel Performance between MVAPICH2 and OpenMPI Applied to a Hyperbolic Test Problem

Michael J. Reid

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

mic7@umbc.edu

Abstract

During the manufacture of integrated circuits, the process of atomic layer deposition (ALD) is used to deposit a uniform seed layer of solid material atop the surface of a silicon wafer. The process can be modeled on the molecular level by a system of transient, linear integro-partial differential Boltzmann equations, coupled with a non-linear surface reaction model, together called the kinetic transport and reaction model (KTRM). Each Boltzmann equation can be approximated by discretizing the velocity space, which yields a system of transient hyperbolic conservation laws that only involve the position vector and time as independent variables. The system can then be solved with DG, a computer implementation of the discontinuous Galerkin method. Due to the large size of the systems being solved and large number of time steps required, it is necessary to use parallel computing to obtain a solution in a reasonable amount of time. We analyze the performance of the DG code on multiple mesh resolutions by measuring its speedup and efficiency on UMBC's new distributed-memory cluster, hpc (`www.umbc.edu/hpcf`). We also compare the performance of DG when it is compiled using the MVAPICH2 and OpenMPI implementations of MPI, the most prevalent parallel communication library today. Testing on a variety of mesh sizes shows that the MVAPICH2 implementation runs as fast or faster than OpenMPI in all cases. This senior thesis is part of undergraduate research conducted under the direction of Dr. Matthias K. Gobbert.

1 Introduction

During the manufacture of integrated circuits, a process called atomic layer deposition (ALD) is used to deposit a uniform seed layer of solid material on the surface of a silicon wafer. ALD consists of several steps, repeated thousands of times, involving reactions between two gaseous species, which adsorb, desorb, and react at the wafer surface. Depending on the gases chosen, however, the process may have unintended results, necessitating a computer simulation. ALD can be modeled on the molecular level by a system of linear Boltzmann equations as transport model, coupled with a general, non-linear surface reaction model, called the kinetic transport and reaction model (KTRM) [2, 1].

The Boltzmann equations in the KTRM are transient, linear integro-partial differential equations. Characteristic of kinetic model, their unknown kinetic densities of all reactive chemical species have to be computed as functions of position vector, velocity vector, and time as independent variables. To affect a numerical solution, each linear Boltzmann equation is approximated by discretizing the velocity space, giving a system of transient hyperbolic conservation laws that only involve the position vector and time as independent variables. The latter system can be posed in standard form, allowing for the solution by a program, DG, which implements the discontinuous Galerkin method [5].

Because of the large number of equations and time steps involved, solving this type of problem, even on a modern personal computer, would take an exorbitant amount of time, in the realm of hundreds of hours. To reduce the amount of computation time needed, we utilize the power of parallel computing, which involves distributing the work done from one processor to multiple processors which communicate with one another during the solution process. All communication done between processes in a distributed-memory cluster are done explicitly in the code through the use of functions specified by the Message Passing Interface (MPI) standard, the dominant communications protocol in high-performance computing today. There are multiple implementations of MPI available; for this paper, we compare the speed of the OpenMPI 1.2.6 and MVAPICH2 1.0.3 implementations.

This paper performs parallel performance studies on the distributed-memory cluster hpc (`www.umbc.edu/hpcf`), purchased in 2008 by UMBC. Use of the DG code makes for an excellent test of the capabilities of the cluster, as it involves both point-to-point and collective communications at every time step, allowing for thorough testing of the interconnect network. The results of this study will teach us how to run this code most efficiently; that is, whether to run one, two, or four processes per node.

Section 2 below specifies the reaction model used for our particular application, and Section 3 states the transport model in more detail and explains the numerical method used to solve it. Section 4 collects a set of representative results for the application problem and presents the parallel performance study on the cluster hpc.

From the results of the parallel performance study in Section 4, we observe that the DG code, when using MVAPICH2, exhibits excellent speedup in all cases. However, use of OpenMPI causes significant scalability problems with certain configurations of nodes and the number of cores used per node. In absolute times, MVAPICH2 matches or beats OpenMPI in nearly every case, sometimes by a significant amount.

2 The Reaction Model

The goal of atomic layer deposition (ALD) is to deposit a uniform layer onto the surface through reactions between a precursor gas, denoted by A, and a reactant gas, denoted by B. The intended reaction pathway calls for A to adsorb to the solid surface and in a next step for B to react with the adsorbed A to form one uniform monolayer of solid on the wafer surface. This is expressed by the surface reaction model

$$\mathrm{A} + \mathrm{v} \rightleftharpoons \mathrm{A_v}, \tag{2.1a}$$

$$\mathrm{B} + \mathrm{A_v} \rightarrow \text{solid} + \text{by-product} + \mathrm{v}, \tag{2.1b}$$

where v denotes a vacant site on the surface and $\mathrm{A_v}$ denotes A attached to a site on the surface [2]. The corresponding reaction rates for these equations are given by

$$R_1 = k_1^f \left(S_\mathrm{T} - S_\mathrm{A}\right) \eta_1 - k_1^b \, S_\mathrm{A}, \tag{2.2a}$$

$$R_2 = k_2^f \, S_\mathrm{A} \, \eta_2, \tag{2.2b}$$

where k_k^f is the forward reaction rate for reaction k in (2.1), k_k^b is the backward reaction rate for reaction k, S_T is the total area on the wafer surface, S_A is the molar concentration of A adsorbed on the surface, and η_i is the flux of species i to the surface.

However, if hydrogen radicals H are used for the reactant B, two additional reactions may occur: some H may adsorb to the surface, blocking A from adsorbing and preventing the layer of solid from forming at that site; and some gaseous H may interact with an $\mathrm{H_v}$ that has adsorbed to the surface, resulting in a gaseous $\mathrm{H_2}$ molecule and making the H unavailable for reaction on the surface.

The reaction model for A and H is then

$$\mathrm{A} + \mathrm{v} \rightleftharpoons \mathrm{A_v}, \tag{2.3a}$$

$$\mathrm{H} + \mathrm{A_v} \rightarrow \text{solid} + \text{by-product} + \mathrm{v}, \tag{2.3b}$$

$$\mathrm{H} + \mathrm{v} \rightleftharpoons \mathrm{H_v}, \tag{2.3c}$$

$$\mathrm{H} + \mathrm{H_v} \rightarrow \mathrm{H_2} + \mathrm{v}. \tag{2.3d}$$

and the reaction rates are given by

$$R_1 = k_1^f \left(S_\mathrm{T} - S_\mathrm{A} - S_\mathrm{H}\right) \eta_1 - k_1^b \, S_\mathrm{A}, \tag{2.4a}$$

$$R_2 = k_2^f \, S_\mathrm{A} \, \eta_2, \tag{2.4b}$$

$$R_3 = k_3^f \left(S_\mathrm{T} - S_\mathrm{A} - S_\mathrm{H}\right) \eta_2 - k_3^b \, S_\mathrm{H}, \tag{2.4c}$$

$$R_4 = k_4^f \, S_\mathrm{H} \, \eta_2, \tag{2.4d}$$

where the additional parameter S_H is the molar concentration of H adsorbed on the surface. We then can write the time-evolution model of the number of surface sites as

$$\frac{dS_\mathrm{A}(\mathbf{x},t)}{dt} = R_1 - R_2, \quad S_\mathrm{A}(\mathbf{x},0) = S_\mathrm{A}^{ini}(\mathbf{x}), \tag{2.5a}$$

$$\frac{dS_\mathrm{H}(\mathbf{x},t)}{dt} = R_3 - R_4, \quad S_\mathrm{H}(\mathbf{x},0) = S_\mathrm{H}^{ini}(\mathbf{x}), \tag{2.5b}$$

where $\mathbf{x}$ is any point on the wafer surface and $S_{\mathrm{A}}^{\mathrm{ini}}(\mathbf{x})$ and $S_{\mathrm{H}}^{\mathrm{ini}}(\mathbf{x})$ are the initial concentration of A and H at $\mathbf{x}$.

We then non-dimensionalize the reaction model with respect to the reference flux η^*. This results in the reaction rates $\hat{R}_i$, given by

$$\hat{R}_1 = \frac{R_1}{\eta^*} = k_1^f \, S_{\mathrm{T}} \, (1 - \vartheta_{\mathrm{A}} - \vartheta_{\mathrm{H}}) \, \frac{\eta_1}{\eta^*} - \frac{k_1^b}{\eta^*} \, S_{\mathrm{T}} \, \vartheta_{\mathrm{A}}, \tag{2.6a}$$

$$\hat{R}_2 = \frac{R_2}{\eta^*} = k_2^f \, S_{\mathrm{T}} \, \vartheta_{\mathrm{A}} \, \frac{\eta_2}{\eta^*}, \tag{2.6b}$$

$$\hat{R}_3 = \frac{R_3}{\eta^*} = k_3^f \, S_{\mathrm{T}} \, (1 - \vartheta_{\mathrm{A}} - \vartheta_{\mathrm{H}}) \, \frac{\eta_2}{\eta^*} - \frac{k_3^b}{\eta^*} \, S_{\mathrm{T}} \, \vartheta_{\mathrm{H}}, \tag{2.6c}$$

$$\hat{R}_4 = \frac{R_4}{\eta^*} = k_4^f \, S_{\mathrm{T}} \, \vartheta_{\mathrm{A}} \, \frac{\eta_2}{\eta^*}. \tag{2.6d}$$

We simplify these equations by introducing the fractional surface coverages $\vartheta_{\mathrm{A}} := S_{\mathrm{A}}/S_{\mathrm{T}}$ and $\vartheta_{\mathrm{H}} := S_{\mathrm{H}}/S_{\mathrm{T}}$, resulting in the equations

$$\hat{R}_1 = \gamma_1^f \, (1 - \vartheta_{\mathrm{A}} - \vartheta_{\mathrm{H}}) \, \hat{\eta}_1 - \gamma_1^b \, \vartheta_{\mathrm{A}}, \tag{2.7a}$$

$$\hat{R}_2 = \gamma_2^f \, \vartheta_{\mathrm{A}} \, \hat{\eta}_2, \tag{2.7b}$$

$$\hat{R}_3 = \gamma_3^f \, (1 - \vartheta_{\mathrm{A}} - \vartheta_{\mathrm{H}}) \, \hat{\eta}_2 - \gamma_3^b \, \vartheta_{\mathrm{H}}, \tag{2.7c}$$

$$\hat{R}_4 = \gamma_4^f \, \vartheta_{\mathrm{H}} \, \hat{\eta}_2. \tag{2.7d}$$

Here we have introduced the corresponding dimensionless reaction coefficients $\gamma_\ell^f := k_\ell^f S_{\mathrm{T}}$ for $\ell = 1, \ldots, 4$ and $\gamma_\ell^b := k_\ell^b S_{\mathrm{T}}$ for $\ell = 1, 3$. Similarly, the differential equations governing the evolution of the surface sites (2.5) becomes a non-dimensionalized model for the evolution of fractional coverage

$$\frac{d\vartheta_{\mathrm{A}}(\hat{t})}{d\hat{t}} = \alpha_p \, (\hat{R}_1 - \hat{R}_2), \quad \vartheta_{\mathrm{A}}(\hat{\mathbf{x}}, 0) = \vartheta_{\mathrm{A}}^{\mathrm{ini}}(\hat{\mathbf{x}}) \tag{2.8a}$$

$$\frac{d\vartheta_{\mathrm{H}}(\hat{t})}{d\hat{t}} = \alpha_p \, (\hat{R}_3 - \hat{R}_4), \quad \vartheta_{\mathrm{H}}(\hat{\mathbf{x}}, 0) = \vartheta_{\mathrm{H}}^{\mathrm{ini}}(\hat{\mathbf{x}}) \tag{2.8b}$$

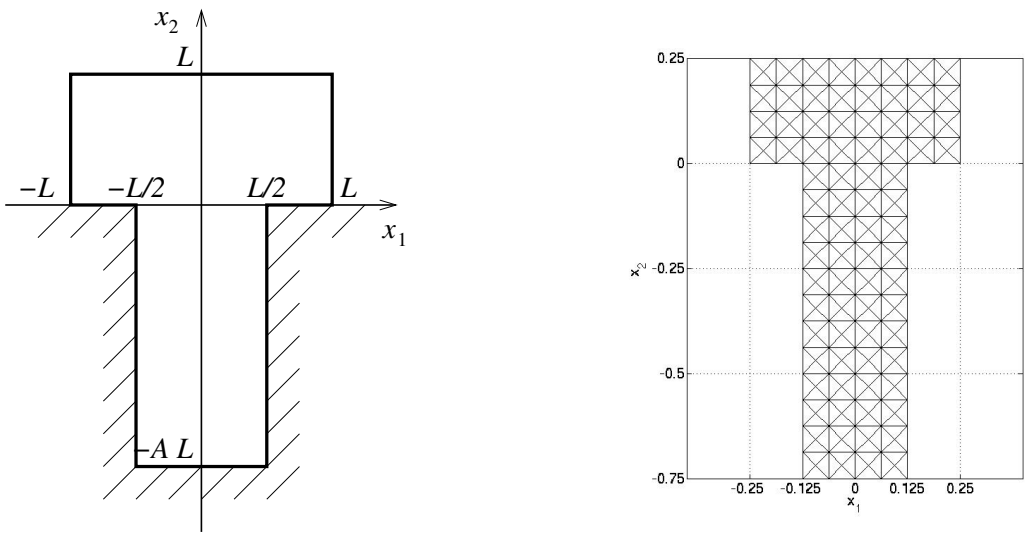
with the dimensionless prefactor $\alpha_p = \frac{\eta^* t^*}{S_{\mathrm{T}}}$.

For purposes of this paper, we only deal with the adsorption step of ALD. In the adsorption step, there is no initial quantity of A or H in the feature, but A is being fed into the top of the feature. Thus, while we are dealing with a two-species model, A is the only species of interest when we look at the results. We also only look at the model on the feature scale, so the domain of our problem is the gaseous area inside and just above a cross-section of one individual feature on the wafer surface. Such a domain is shown in Figure 1 (a) with feature width L and feature aspect ratio A (the ratio of depth over width of the feature). The wafer surface is indicated by the hash marks, and its top surface is at $x_2 = 0$. The pre-cursor species A is fed into the domain from $x_2 = L$. In our studies, we use a feature width $L = 0.25$ μm and a feature aspect ratio $A = 3$.

3 The Transport Model

After making the appropriate simplifications to the equations used to model these reactions, we attain a system of dimensionless Boltzmann equations for the kinetic densities $f^{(i)}(\mathbf{x}, \mathbf{v}, t)$ of the n_s reactive species

$$\frac{\partial f^{(i)}(\mathbf{x}, \mathbf{v}, t)}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f^{(i)}(\mathbf{x}, \mathbf{v}, t) = \frac{1}{\mathrm{Kn}} Q_i(f^{(i)}(\mathbf{x}, \mathbf{v}, t)), \quad i = 1, \ldots, n_s, \tag{3.1}$$



(a) (b)

Figure 1: (a) Cross-section of a feature and domain Ω for the mathematical model with feature width $L = 0.25$ μm and feature aspect ratio $A = 3$. (b) Sample domain mesh used for the numerical method with uniform mesh spacing $h = 1/16 = 0.0625$ μm.

Table 1: Sizing study listing the velocity resolution K, the spatial mesh spacing h, the number of spatial mesh elements N_e, the number of degrees of freedom (DOF), the final time t_{final}, the constant time step Δt, the number of time steps N_t, and the observed wall clock time in HH:MM:SS for a serial run in each test case.

	K	h	N_e	DOF	t_{final}	Δt	N_t	Wall time
Case 1	8×8	0.031250	320	163,840	10.0	$1.00534 \cdot 10^{-3}$	9947	01:10:13
Case 2	8×8	0.015625	1280	655,360	10.0	$5.02669 \cdot 10^{-4}$	19894	09:23:19
Case 3	16×16	0.031250	320	655,360	10.0	$6.28373 \cdot 10^{-4}$	15915	17:08:18
Case 4	16×16	0.015625	1280	2,621,440	10.0	$3.14187 \cdot 10^{-4}$	31829	136:43:01

where n_s is the number of species in the model, Kn is the Knudsen number, and the linear collision operators $Q_i(f^{(i)})$ are given by

$$Q_i(f^{(i)}) = \int_{\mathbb{R}^2} \sigma_i(\mathbf{v}, \mathbf{v}') \left[M^{(i)}(\mathbf{v}') f^{(i)}(\mathbf{x}, \mathbf{v}, t) - M^{(i)}(\mathbf{v}') f^{(i)}(\mathbf{x}, \mathbf{v}, t) \right] d\mathbf{v}', \tag{3.2}$$

where $M^{(i)}$ is the Maxwellian of species i

$$M^{(i)}(\mathbf{v}) = \frac{1}{\pi (v_i^{\text{ref}})^2} \exp\left(\frac{|\mathbf{v}|^2}{2(v_i^{\text{ref}})^2} \right), \tag{3.3}$$

where v_i^{ref} is the reference velocity of species i [1]. We only deal with a 2-D/2-D kinetic model in this paper, so the position $\mathbf{x} = (x_1, x_2)^T \in \Omega \in \mathbb{R}^2$ and the velocity $\mathbf{v} = (v_1, v_2)^T \in \mathbb{R}^2$. Then, for $\mathbf{x}$ on the wafer surface Γ_w, the boundary condition is

$$f^{(1)}(\mathbf{x}, \mathbf{v}, t) = C_1 \left[\eta_1(\mathbf{x}, t) - R_1(\mathbf{x}, t) \right] M^{(1)}(\mathbf{v}), \tag{3.4a}$$

$$f^{(2)}(\mathbf{x}, \mathbf{v}, t) = C_2 \left[\eta_2(\mathbf{x}, t) - R_2(\mathbf{x}, t) - R_3(\mathbf{x}, t) - R_4(\mathbf{x}, t) \right] M^{(2)}(\mathbf{v}), \tag{3.4b}$$

$$\mathbf{x} \in \Gamma_w, \quad \mathbf{n} \cdot \mathbf{v} < 0, \tag{3.4c}$$

with the scaling factors $C_i = 2\sqrt{\pi}/v_i^{\text{ref}}$ and $\mathbf{n} = \mathbf{n}(\mathbf{x})$ the unit outward normal vector at $\mathbf{x} \in \partial\Omega$. At the top of the domain Γ_t, the boundary condition is

$$f^{(i)}(\mathbf{x}, \mathbf{v}, t) = c_i^{\text{top}}(\mathbf{x}, t) M^{(i)}(\mathbf{v}), \quad \mathbf{x} \in \Gamma_t, \quad \mathbf{n} \cdot \mathbf{v} < 0 \tag{3.5}$$

where c_i^{top} is the inflow concentration of species i at the top of the reactor. For the remaining boundaries on the vertical sides of the trench Γ_s, we use specular reflection, so that

$$f^{(i)}(\mathbf{x}, \mathbf{v}, t) = f^{(i)}(\mathbf{x}, \mathbf{v}', t), \quad \mathbf{x} \in \Gamma_s, \quad \mathbf{n} \cdot \mathbf{v} < 0, \tag{3.6}$$

with $\mathbf{v} = \mathbf{v}' - 2\,(\mathbf{v}' \cdot \mathbf{n})\,\mathbf{n}$.

We then discretize in velocity space by approximating each $f^{(i)}(\mathbf{x}, \mathbf{v}, t)$ by the expansion $f_K^{(i)}(\mathbf{x}, \mathbf{v}, t) = \sum_{\ell=0}^{K-1} f_\ell^{(i)}(\mathbf{x}, t)\, \varphi_\ell(\mathbf{v})$, where the basis functions $\varphi_\ell(\mathbf{v})$ in velocity space are products of a Maxwellian and Hermite polynomials in each dimension [3]. Each linear Boltzmann equation in (3.1) is then discretized by substituting $f_K^{(i)}(\mathbf{x}, \mathbf{v}, t)$ for $f^{(i)}(\mathbf{x}, \mathbf{v}, t)$ and testing against the basis functions $\varphi_k(\mathbf{v})$, $k = 0, \ldots, K-1$, resulting in a system of K transient linear first-order hyperbolic transport equations

$$\frac{\partial F^{(i)}}{\partial t} + A^{(1)} \frac{\partial F^{(i)}}{\partial x_1} + A^{(2)} \frac{\partial F^{(i)}}{\partial x_2} = \frac{1}{\text{Kn}} B^{(i)} F^{(i)}, \quad i = 1, \ldots, n_s, \tag{3.7}$$

in space $\mathbf{x} = (x_1, x_2)^T$ and time t for the vector of K coefficient functions $F^{(i)}(\mathbf{x}, t) := (f_0^{(i)}(\mathbf{x}, t), \ldots, f_{K-1}^{(i)}(\mathbf{x}, t))^T$. The $K \times K$ matrices $A^{(1)}, A^{(2)}$, and $B^{(i)}$ are constant due to the linearity of (3.1); moreover, due to choice of basis functions, $A^{(1)}$ and $A^{(2)}$ are also diagonal [3]. Using the diagonality of these matrices, the system (3.7) can be re-formulated in the standard form of hyperbolic conservation laws, suitable for the solution by the discontinuous Galerkin method. We use the implementation in DG [5], a C++ program using MPI (the Message Passing Interface) as the library for the communications between the parallel processes.

To test the performance of our code, we look at four cases: Case 1 with velocity resolution $K = 8 \times 8 = 64$ and uniform spatial mesh spacing $h = 1/32 = 0.031250$; Case 2 with $K = 8 \times 8 = 64$ and $h = 1/64 = 0.015625$; Case 3

with $K = 16 \times 16 = 256$ and $h = 1/32 = 0.031250$; and Case 4 with $K = 16 \times 16 = 256$ and $h = 1/64 = 0.015625$. The complexity of these problems is measured by the number of degrees of freedom (DOF), given by the number of solution components that must be computed at every time step. We use discontinuous bi-linear nodal finite elements on a quadrilateral mesh with four local degrees of freedom (the solution value at every vertex). Thus, the DOF at every time step can be calculated by the formula $4n_sN_eK$, where n_s is the number of species, N_e is the number of spatial finite elements in Ω, and K is the size of the velocity mesh. Figure 1 (b) shows a sample domain mesh with a uniform mesh spacing $h = 1/16$, selected coarser for a clearer plot. The degrees of freedom of each case are collected in Table 1. The time steps are automatically computed at run-time to be the largest possible value that still guarantees stability of the method. As a result, the number of time steps to reach the final time is different in each case; this can be seen as a consequence of the size of the problem, since larger values for K and N_e require smaller time steps to guarantee stability.

To parallelize the problem, the domain Ω is split into sub-domains using a graph partitioning utility. Each process used is assigned one sub-domain. All processes then perform computations for the update of the solution on their respective sub-domains in parallel at each time step. At each time step, there are both point-to-point communications between adjacent sub-domains, such as `MPI_Send` and `MPI_Recv` to send and receive values at the common boundary between sub-domains, as well as collective communications between all processes, such as `MPI_Allreduce` to calculate inner products and similar.

In addition to doing performance studies on various mesh sizes, we also do performance studies on each case with two copies of the DG code: one compiled using the OpenMPI implementation of MPI, and one using the MVAPICH2 implementation.

4 Results

4.1 Application Results

We first look at plots of the concentration of species A during the adsorption step of ALD, shown in Figure 2 for times $t = 0, 2, 4, 6, 8, 10$ ns. For this model, we set our reaction rates $\gamma_1^f = 10^{-2}, \gamma_1^b = 10^{-4}, \gamma_2^f = 10^{-2}, \gamma_3^f = \gamma_3^b = \gamma_4^f = 0$, and the Knudsen number Kn = 100. While the parameters have all been non-dimensionalized [3], the time has been re-dimensionalized in nanoseconds. From the concentration plots we can see that the concentration of A quickly fills the area near the top of the feature, but it takes much longer for the concentration at the bottom of the feature to increase.

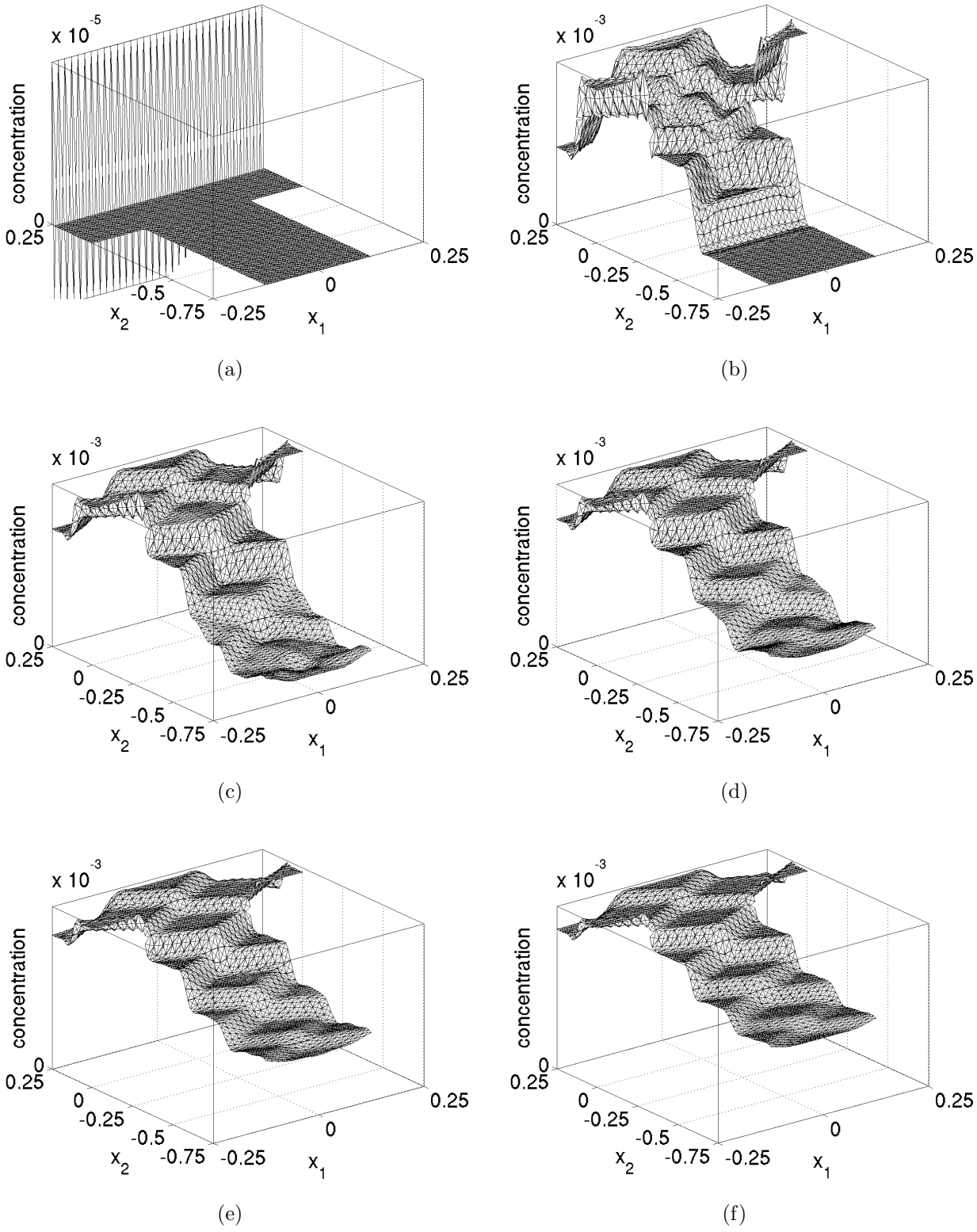


Figure 2: Concentration plot of species A at times (a) 0 ns, (b) 2 ns, (c) 4 ns, (d) 6 ns, (e) 8 ns, and (f) 10 ns. The results are calculated using a velocity resolution of $K = 16 \times 16$ and domain mesh with uniform spacing $h = 0.015625$.

4.2 Parallel Performance Study Using MVAPICH2

For both parallel performance studies, we use the parallel computing cluster hpc in the UMBC High Performance Computing Facility (`www.umbc.edu/hpcf`). This distributed-memory cluster has 32 computational nodes, each with two dual-core AMD Opteron processors (2.6 GHz, 1 MB cache per core) and 13 GB of memory. These nodes are connected via a state-of-the-art high performance InfiniBand interconnect network.

To test parallel performance, we vary both the number of nodes used and the number of parallel processes run on each node. The number of nodes used are 1, 2, 4, 8, 16, and 32. On each node, either one, two, or four processes are run; if a node does not use all four processes, the unused processes are kept idle. We measure parallel performance by the speedup and efficiency of the results. If $T_p(N)$ is defined as the wall clock time for a problem of fixed size N using p processes, the speedup of the code from 1 to p processes is defined as $S_p = T_1(N)/T_p(N)$. Since ideally a run on p processes is p times as fast as the run on 1 process, S_p has the optimal value $S_p = p$. The efficiency is then defined as $E_p = S_p/p$, and thus has an optimal value of 1 [4, 3].

Table 2 summarizes the results of using only one process per node for each of the four cases, with the exception of $p = 64$, which uses two processes per node, and $p = 128$, which utilizes all four processes per node. Reading across, we see that by doubling the number of processes used for the problem, we cut the time taken by the code approximately in half up to $p = 8$, which shows good speedup for the problem. This can be further seen in the efficiency chart, which shows an efficiency $E_p > 0.8$ for all cases up to $p = 8$. There is some efficiency degradation when more processes are used, but this is not unexpected, especially with the cases with fewer DOF. The corresponding speedup plot in Figure 3 (a) and efficiency plot in Figure 3 (b) give a graphical representation of the speedup and efficiency for each process.

Tables 3 and 4 summarize similar results using two and four processes per node, respectively. The most important reuslt to take is that there is no difference between corresponding times for different numbers of processes per node; that is, for any fixed number of processes p, the code will run equally as fast, regardless of the number of nodes between which the processes are divided. This result tells us that to obtain the absolute minimum time needed to run the code, we should use as many processes as we have cores available to us; indeed, looking at Table 4, we see that using the maximum number of cores available—$p = 64$ for $h = 0.031250$ and $p = 128$ for $h = 0.015625$, with the difference being due to the maximum number of subdomains into which the spatial mesh can be split—gives us the smallest execution time in each case.

Table 5 reorganizes the data to make it easier to analyze the scalability of the code when using MVAPICH2. We see that by moving horizontally across the table, doubling the number of nodes, we approximately halve the wall clock time; similarly, we see that by moving downwards, doubling the number of cores per node, we approximately halve the wall clock time. Combined, these results show that the code exhibits good scalability, and that it is most efficient to run the code with the maximum number of cores available.

Table 2: Performance using MVAPICH2 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

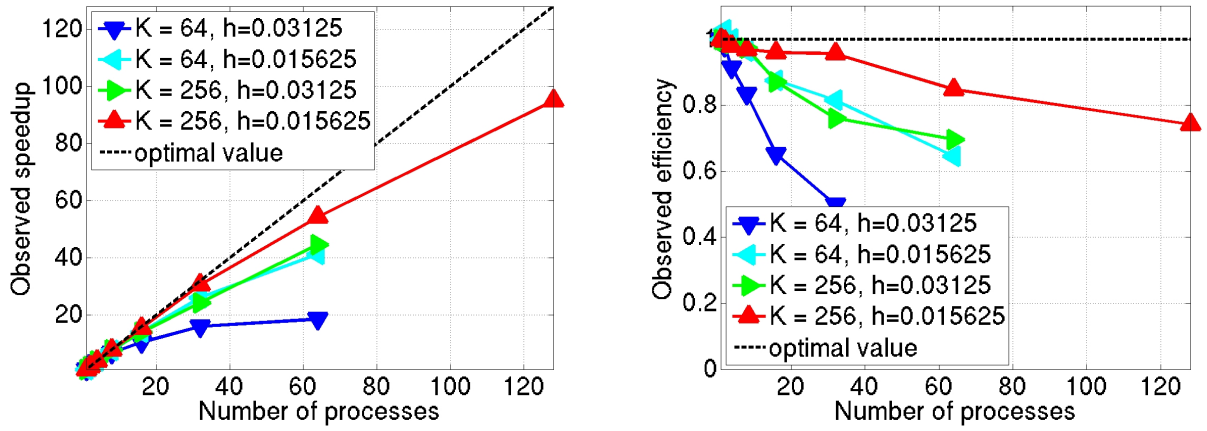
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	01:10:13	00:35:51	00:19:12	00:10:32	00:06:44	00:04:24	00:03:47	N/A
$K = 64$, $h = 0.015625$	09:23:19	04:33:45	02:20:42	01:13:04	00:40:11	00:21:35	00:13:38	N/A
$K = 256$, $h = 0.031250$	17:08:18	08:36:25	04:20:25	02:12:07	01:13:46	00:42:16	00:23:03	N/A
$K = 256$, $h = 0.015625$	136:43:01	68:41:39	34:53:16	17:38:10	08:54:06	04:27:59	02:31:02	01:26:25

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	1.9588	3.6565	6.6694	10.4215	15.9668	18.5570	N/A
$K = 64$, $h = 0.015625$	1.0000	2.0578	4.0035	7.7101	14.0199	26.0912	41.3241	N/A
$K = 256$, $h = 0.031250$	1.0000	1.9912	3.9487	7.7837	13.9393	24.3280	44.6091	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9902	3.9188	7.7521	15.3585	30.6107	54.3109	94.9290

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	0.9794	0.9141	0.8337	0.6513	0.4990	0.2900	N/A
$K = 64$, $h = 0.015625$	1.0000	1.0289	1.0009	0.9638	0.8762	0.8153	0.6457	N/A
$K = 256$, $h = 0.031250$	1.0000	0.9956	0.9872	0.9730	0.8712	0.7603	0.6970	N/A
$K = 256$, $h = 0.015625$	1.0000	0.9951	0.9797	0.9690	0.9599	0.9566	0.8486	0.7416



(a) Observed speedup S_p (b) Observed efficiency E_p

Figure 3: Performance using MVAPICH2 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 3: Performance using MVAPICH2 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

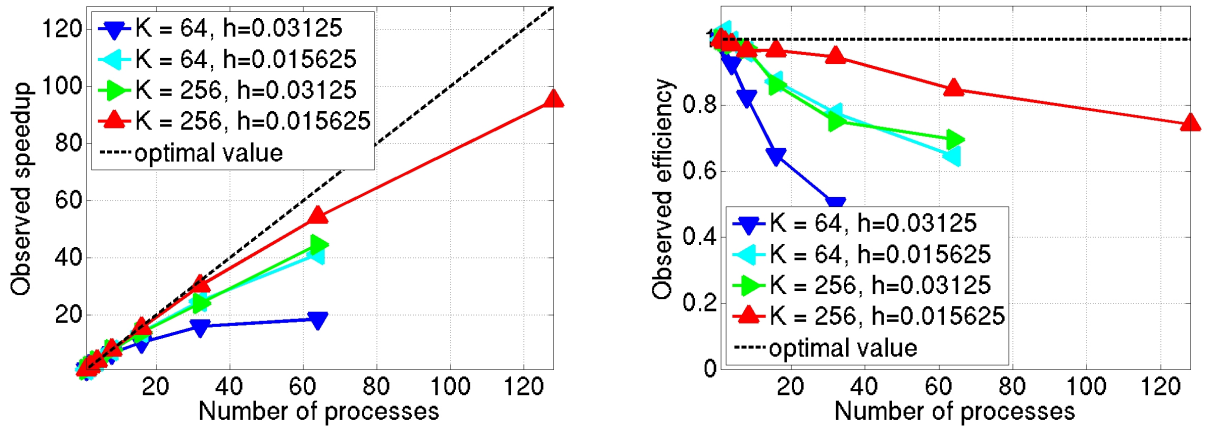
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	01:10:13	00:36:02	00:18:56	00:10:38	00:06:46	00:04:23	00:03:47	N/A
$K = 64$, $h = 0.015625$	09:23:19	04:34:42	02:21:39	01:13:11	00:40:21	00:22:41	00:13:38	N/A
$K = 256$, $h = 0.031250$	17:08:18	08:37:35	04:20:36	02:12:19	01:14:36	00:42:45	00:23:03	N/A
$K = 256$, $h = 0.015625$	136:43:01	68:53:33	34:42:25	17:41:56	08:50:45	04:30:56	02:31:02	01:26:25

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	1.9486	3.7074	6.6056	10.3886	16.0233	18.5570	N/A
$K = 64$, $h = 0.015625$	1.0000	2.0506	3.9770	7.6979	13.9586	24.8370	41.3241	N/A
$K = 256$, $h = 0.031250$	1.0000	1.9868	3.9459	7.7712	13.7831	24.0567	44.6091	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9845	3.9392	7.7246	15.4555	30.2778	54.3109	94.9290

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	0.9743	0.9268	0.8257	0.6493	0.5007	0.2900	N/A
$K = 64$, $h = 0.015625$	1.0000	1.0253	0.9943	0.9622	0.8724	0.7762	0.6457	N/A
$K = 256$, $h = 0.031250$	1.0000	0.9934	0.9865	0.9714	0.8614	0.7518	0.6970	N/A
$K = 256$, $h = 0.015625$	1.0000	0.9922	0.9848	0.9656	0.9660	0.9462	0.8486	0.7416



(a) Observed speedup S_p (b) Observed efficiency E_p

Figure 4: Performance using MVAPICH2 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 4: Performance using MVAPICH2 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

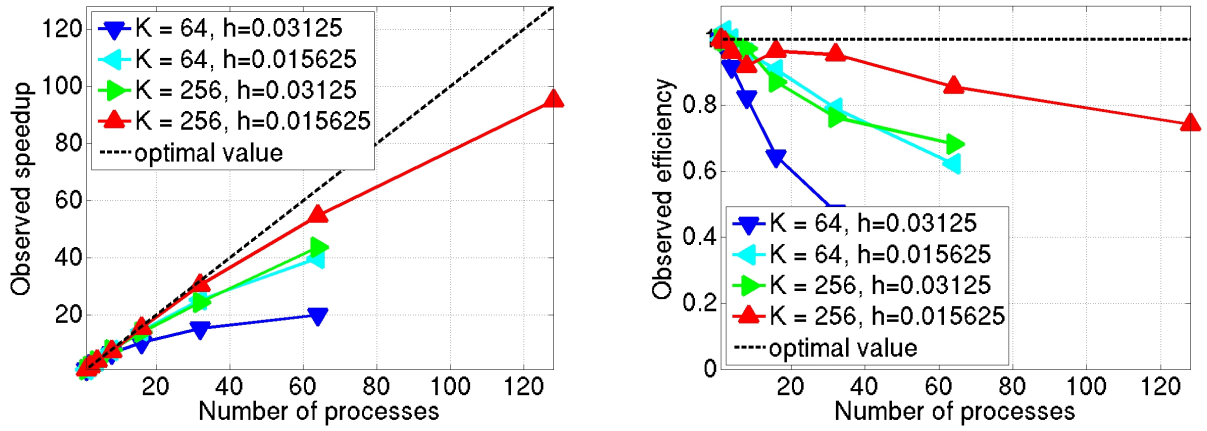
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	01:10:13	00:36:02	00:19:10	00:10:40	00:06:48	00:04:36	00:03:31	N/A
$K = 64$, $h = 0.015625$	09:23:19	04:34:42	02:20:45	01:13:28	00:38:47	00:22:16	00:14:09	N/A
$K = 256$, $h = 0.031250$	17:08:18	08:37:35	04:20:04	02:12:13	01:13:48	00:42:06	00:23:32	N/A
$K = 256$, $h = 0.015625$	136:43:01	68:53:33	35:34:47	18:35:31	08:51:48	05:12:34	02:29:47	01:26:25

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	1.9486	3.6639	6.5878	10.3171	15.2728	20.0143	N/A
$K = 64$, $h = 0.015625$	1.0000	2.0506	4.0021	7.6683	14.5248	25.3043	39.8329	N/A
$K = 256$, $h = 0.031250$	1.0000	1.9868	3.9539	7.7772	13.9328	24.4234	43.6899	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9845	3.8425	7.3535	15.4248	30.5144	54.7667	94.9290

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	0.9743	0.9160	0.8235	0.6448	0.4773	0.3127	N/A
$K = 64$, $h = 0.015625$	1.0000	1.0253	1.0005	0.9585	0.9078	0.7908	0.6224	N/A
$K = 256$, $h = 0.031250$	1.0000	0.9934	0.9885	0.9722	0.8708	0.7632	0.6827	N/A
$K = 256$, $h = 0.015625$	1.0000	0.9922	0.9606	0.9192	0.9641	0.9536	0.8557	0.7416



(a) Observed speedup S_p (b) Observed efficiency E_p

Figure 5: Performance using MVAPICH2 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.

Table 5: Wall clock time in HH:MM:SS using MVAPICH2 for the solution of four cases of velocity and spatial meshes using 1, 2, 4, 8, 16, and 32 compute nodes with 1, 2, and 4 processes per node.

(a) Coarse spatial mesh with $h = 0.031250$, coarse velocity resolution $K = 8 \times 8$, DOF = 163,840

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	01:10:13	00:35:51	00:19:12	00:10:32	00:06:44	00:04:24
2 processes per node	00:36:02	00:18:56	00:10:38	00:06:46	00:04:23	00:03:47
4 processes per node	00:19:10	00:10:40	00:06:48	00:04:36	00:03:31	N/A

(b) Fine spatial mesh with $h = 0.015625$, coarse velocity resolution $K = 8 \times 8$, DOF = 655,360

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	09:23:19	04:33:45	02:20:42	01:13:04	00:40:11	00:21:35
2 processes per node	04:34:42	02:21:39	01:13:11	00:40:21	00:22:41	00:13:38
4 processes per node	02:20:45	01:13:28	00:38:47	00:22:16	00:14:09	N/A

(c) Coarse spatial mesh with $h = 0.031250$, fine velocity resolution $K = 16 \times 16$, DOF = 655,360

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	17:08:18	08:36:25	04:20:25	02:12:07	01:13:46	00:42:16
2 processes per node	08:37:35	04:20:36	02:12:19	01:14:36	00:42:45	00:23:03
4 processes per node	04:20:04	02:12:13	01:13:48	00:42:06	00:23:32	N/A

(d) Fine spatial mesh with $h = 0.015625$, fine velocity resolution $K = 16 \times 16$, DOF = 2,621,440

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	136:43:01	68:41:39	34:53:16	17:38:10	08:54:06	04:27:59
2 processes per node	68:53:33	34:42:25	17:41:56	08:50:45	04:30:56	02:31:02
4 processes per node	35:34:47	18:35:31	08:51:48	04:28:49	02:29:47	01:26:25

4.3 Parallel Performance Study Using OpenMPI

As in Section 4.2, we test each of the four cases by using one, two, and four processes per node, where unused processes are kept idle on each node used. The results of using one process per node are given in Table 6. By reading horizontally, we immediately see that there are significant problems regarding the first two ($K = 64$) cases; doubling the number of processes from $p = 1$ to $p = 2$ yields an insignificant decrease in the time taken to run the code. As a result, the efficiency for the first two cases is extremely poor. The second two ($K = 256$) cases, however, show results very close to those seen when using MVAPICH2, as per Table 2.

Table 7 summarizes results of using two processes per node. While this fixes the problem in the first two cases when going from $p = 1$ to $p = 2$, there is a similar, though less pronounced, problem when moving from $p = 4$ to $p = 8$. The first case has an efficiency of 0.7543, while the second has an efficiency of 0.5445. These are significantly below the corresponding values for MVAPICH2 given in Table 3, which are 0.8257 and 0.9622, respectively. Again, the latter two cases exhibit speedup similar to that when using MVAPICH2.

Table 8 summarizes results when using all four processes per node. The results here are significantly better than the previous results when using OpenMPI, though the times when using MVAPICH2 are still slightly lower for the first two cases.

Table 9 reorganizes the data to make it easier to see the previously mentioned problems with scalability when using OpenMPI. We can also compare this to Table 5 to see that in nearly every case, MVAPICH2 yields a lower wall clock time than OpenMPI, sometimes very significantly so.

Table 6: Performance using OpenMPI 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

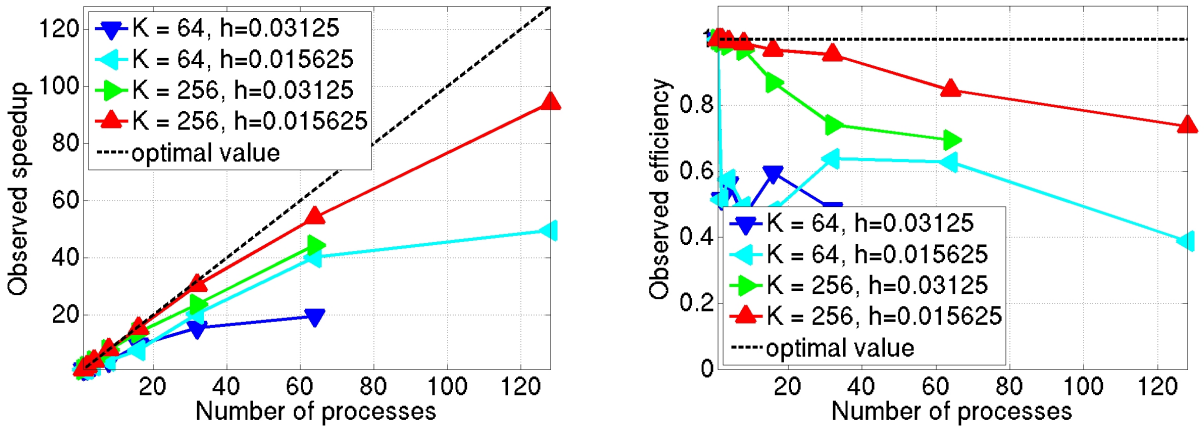
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	01:09:33	01:07:28	00:30:53	00:18:52	00:07:19	00:04:29	00:03:32	N/A
$K = 64$, $h = 0.015625$	09:04:22	08:50:10	03:56:18	02:18:05	01:11:05	00:26:40	00:13:33	00:10:58
$K = 256$, $h = 0.031250$	17:07:52	08:37:50	04:21:24	02:12:45	01:13:57	00:43:21	00:23:07	N/A
$K = 256$, $h = 0.015625$	136:46:48	68:24:00	34:25:21	17:18:34	08:50:00	04:28:56	02:31:36	01:27:07

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	1.0309	2.2515	3.6873	9.5154	15.5003	19.6543	N/A
$K = 64$, $h = 0.015625$	1.0000	1.0268	2.3036	3.9421	7.6577	20.4192	40.1831	49.6300
$K = 256$, $h = 0.031250$	1.0000	1.9849	3.9322	7.7429	13.9006	23.7121	44.4525	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9997	3.9736	7.9020	15.4840	30.5152	54.1368	94.2054

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	0.5154	0.5629	0.4609	0.5947	0.4844	0.3071	N/A
$K = 64$, $h = 0.015625$	1.0000	0.5134	0.5759	0.4928	0.4786	0.6381	0.6279	0.3877
$K = 256$, $h = 0.031250$	1.0000	0.9925	0.9830	0.9679	0.8688	0.7410	0.6946	N/A
$K = 256$, $h = 0.015625$	1.0000	0.9999	0.9934	0.9878	0.9678	0.9536	0.8459	0.7360



(a) Observed speedup S_p (b) Observed efficiency E_p

Figure 6: Performance using OpenMPI 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 7: Performance using OpenMPI 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

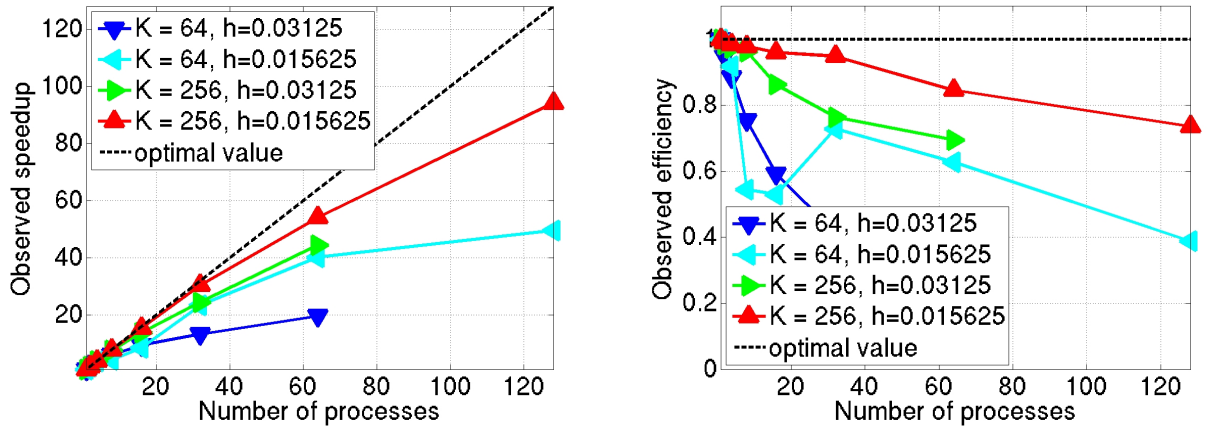
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	01:09:27	00:36:39	00:19:39	00:11:32	00:07:21	00:05:16	00:03:32	N/A
$K = 64$, $h = 0.015625$	09:04:22	04:36:41	02:28:12	02:04:58	01:04:20	00:23:20	00:13:33	00:10:58
$K = 256$, $h = 0.031250$	17:07:52	08:42:29	04:23:45	02:13:24	01:14:28	00:42:03	00:23:07	N/A
$K = 256$, $h = 0.015625$	136:46:48	69:04:26	34:43:21	17:28:31	08:54:06	04:30:17	02:31:36	01:27:07

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	1.8979	3.5403	6.0341	9.4686	13.2262	19.6543	N/A
$K = 64$, $h = 0.015625$	1.0000	1.9675	3.6731	4.3563	8.4614	23.3285	40.1831	49.6300
$K = 256$, $h = 0.031250$	1.0000	1.9672	3.8970	7.7055	13.8037	24.4480	44.4525	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9802	3.9392	7.8271	15.3655	30.3643	54.1368	94.2054

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	0.9490	0.8851	0.7543	0.5918	0.4133	0.3071	N/A
$K = 64$, $h = 0.015625$	1.0000	0.9837	0.9183	0.5445	0.5288	0.7290	0.6279	0.3877
$K = 256$, $h = 0.031250$	1.0000	0.9836	0.9743	0.9632	0.8627	0.7640	0.6946	N/A
$K = 256$, $h = 0.015625$	1.0000	0.9901	0.9848	0.9784	0.9603	0.9489	0.8459	0.7360



(a) Observed speedup S_p (b) Observed efficiency E_p

Figure 7: Performance using OpenMPI 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 8: Performance using OpenMPI 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

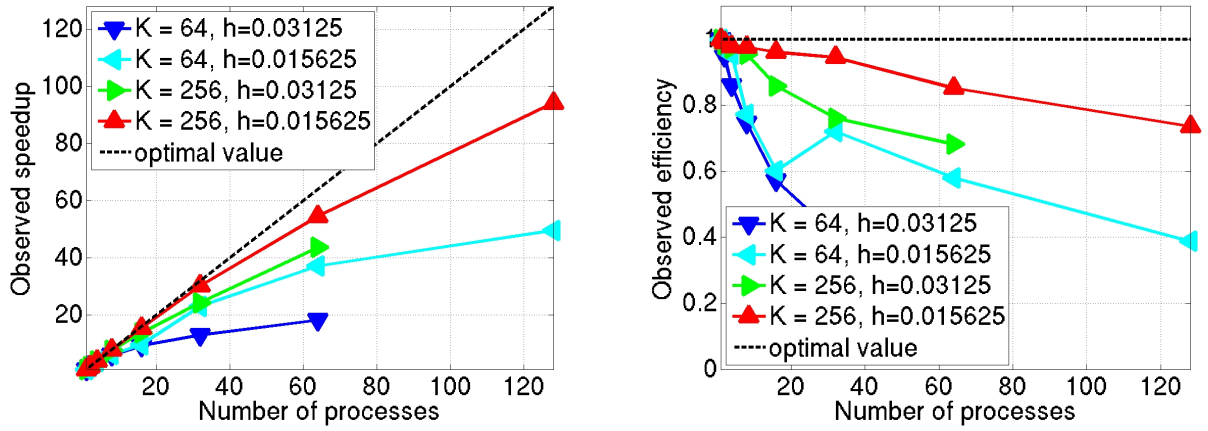
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	01:09:33	00:36:39	00:20:14	00:11:39	00:07:36	00:05:21	00:03:49	N/A
$K = 64$, $h = 0.015625$	09:04:22	04:36:41	02:22:47	01:28:04	00:56:35	00:23:37	00:14:40	00:10:57
$K = 256$, $h = 0.031250$	17:07:52	08:42:29	04:23:53	02:14:37	01:14:50	00:42:13	00:23:32	N/A
$K = 256$, $h = 0.015625$	136:46:48	69:04:26	34:56:30	17:31:03	08:53:42	04:31:19	02:30:27	01:27:07

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	1.8979	3.4379	5.9706	9.1531	12.9858	18.2347	N/A
$K = 64$, $h = 0.015625$	1.0000	1.9675	3.8127	6.1813	9.6218	23.0447	37.1168	49.6300
$K = 256$, $h = 0.031250$	1.0000	1.9672	3.8952	7.6357	13.7354	24.3428	43.6665	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9802	3.9145	7.8082	15.3771	30.2486	54.5462	94.2054

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.031250$	1.0000	0.9490	0.8595	0.7463	0.5721	0.4058	0.2849	N/A
$K = 64$, $h = 0.015625$	1.0000	0.9837	0.9532	0.7727	0.6014	0.7201	0.5799	0.3877
$K = 256$, $h = 0.031250$	1.0000	0.9836	0.9738	0.9545	0.8585	0.7607	0.6823	N/A
$K = 256$, $h = 0.015625$	1.0000	0.9901	0.9786	0.9760	0.9611	0.9453	0.8523	0.7360



(a) Observed speedup S_p (b) Observed efficiency E_p

Figure 8: Performance using OpenMPI 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.

Table 9: Wall clock time in HH:MM:SS using OpenMPI for the solution of four cases of velocity and spatial meshes using 1, 2, 4, 8, 16, and 32 compute nodes with 1, 2, and 4 processes per node.

(a) Coarse spatial mesh with $h = 0.031250$, coarse velocity resolution $K = 8 \times 8$, DOF = 163,840

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	01:09:33	01:07:28	00:30:53	00:18:52	00:07:19	00:04:29
2 processes per node	00:36:39	00:19:39	00:11:32	00:07:21	00:05:16	00:03:32
4 processes per node	00:20:14	00:11:39	00:07:36	00:05:21	00:03:49	N/A

(b) Fine spatial mesh with $h = 0.015625$, coarse velocity resolution $K = 8 \times 8$, DOF = 655,360

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	09:04:22	08:50:10	03:56:18	02:18:05	01:11:05	00:26:40
2 processes per node	04:36:41	02:28:12	02:04:58	01:04:20	00:23:20	00:13:33
4 processes per node	02:22:47	01:28:04	00:56:35	00:23:37	00:14:40	00:10:57

(c) Coarse spatial mesh with $h = 0.031250$, fine velocity resolution $K = 16 \times 16$, DOF = 655,360

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	17:07:52	08:37:50	04:21:24	02:12:45	01:13:57	00:43:21
2 processes per node	08:42:29	04:23:45	02:13:24	01:14:28	00:42:03	00:23:07
4 processes per node	04:23:45	02:13:24	01:14:28	00:42:03	00:23:07	N/A

(d) Fine spatial mesh with $h = 0.015625$, fine velocity resolution $K = 16 \times 16$, DOF = 2,621,440

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes
1 process per node	136:46:48	68:24:00	34:25:21	17:18:34	08:50:00	04:28:56
2 processes per node	69:04:26	34:43:21	17:28:31	08:54:06	04:30:17	02:31:36
4 processes per node	34:56:30	17:31:03	08:53:42	04:31:19	02:30:27	01:27:07

Acknowledgements

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

I also thank UMBC for providing assistance for this research through an Undergraduate Research Award.

References

[1] Matthias K. Gobbert and Timothy S. Cale. A kinetic transport and reaction model and simulator for rarefied gas flow in the transition regime. *J. Comput. Phys.*, vol. 213, pp. 591–612, 2006.

[2] Matthias K. Gobbert, Vinay Prasad, and Timothy S. Cale. Modeling and simulation of atomic layer deposition at the feature scale. *J. Vac. Sci. Technol. B*, vol. 20, no. 3, pp. 1031–1043, 2002.

[3] Matthias K. Gobbert, Samuel G. Webster, and Timothy S. Cale. A Galerkin method for the simulation of the transient 2-D/2-D and 3-D/3-D linear Boltzmann equation. *J. Sci. Comput.*, vol. 30, no. 2, pp. 237–273, 2007.

[4] Michael J. Reid and Matthias K. Gobbert. Parallel performance studies for a hyperbolic test problem. Technical Report HPCF–2008–3, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2008.

[5] Jean-François Remacle, Joseph E. Flaherty, and Mark S. Shephard. An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Rev.*, vol. 45, no. 1, pp. 53–72, 2003.