Parallel Performance Studies for a Numerical Simulator of Atomic Layer Deposition

Michael J. Reid

Section 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 in one cycle, with each cycle 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 to understand the effects. 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, together called the kinetic transport and reaction model (KTRM) [3, 4].

The Boltzmann equations in the KTRM are transient, linear integro-partial differential equations. Characteristic of a 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 [8].

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. Depending on the size of the problem, it may not even be able to run on a single computer due to insufficient memory. 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.

This paper specifically performs a parallel performance study on the cluster hpc in the UMBC High Performance Computing Facility (www.umbc.edu/hpcf). This distributed-memory cluster, purchased in 2008 by UMBC, 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. The code DG offers 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. Though using all 4 available cores on all 32 nodes yields the greatest raw processing power, it may not be the most efficient way to run the code, as memory limitations and bus and network bandwidth bottlenecks might adversely impact its overall performance. Moreover, the most effective use of a cluster that is shared by many users is for each user to use all cores on as few computational nodes as possible, so that other users have as many nodes available as possible. Thus, the results of this study will teach us how to run this code most efficiently and how to utilize the cluster most effectively, by studying, 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 as well as presents the parallel performance study on the cluster hpc. Section 5 draws the conclusions from the studies.

Section 2: The Reaction Model

The goal of atomic layer deposition (ALD) is to deposit a uniform layer of solid material onto the surface of the wafer through reactions between a pre-cursor 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

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

where v denotes a vacant site on the surface and A_v denotes A attached to a site on the surface [4]. 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 H_v that has adsorbed to the surface, resulting in a gaseous H_2 molecule and making the H unavailable for reaction on the surface. The reaction model for A and H is then

$$A + v \leftrightarrow A_v, \tag{2.2a}$$
$$H + A_v \rightarrow \text{solid} + \text{by-product} + v, \tag{2.2b}$$
$$H + v \leftrightarrow H_v, \tag{2.2c}$$
$$H + H_v \rightarrow H_2 + v. \tag{2.2d}$$

The dimensionless forward reaction coefficient for the kth equation is given by γ_k^f for each of the four reactions, and the backward reaction coefficient by γ_k^b for each of the two reversible reactions.

The ultimate goal of this project is to analyze the inhibiting effect of the use of hydrogen radicals as reactant. For purposes of this paper that focuses on the performance of the parallel code used in the future studies, we only consider the adsorption step of ALD. In the adsorption step, there is no initial quantity of A or H in the feature, and only 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 solid wafer surface is indicated by the hash marks, and its top surface is located at $x_2 = 0$. The pre-cursor species A is fed into the domain from $x_2 = L$. In our studies, we use a width $L = 0.25\ \mu$m and an aspect ratio $A = 3$.

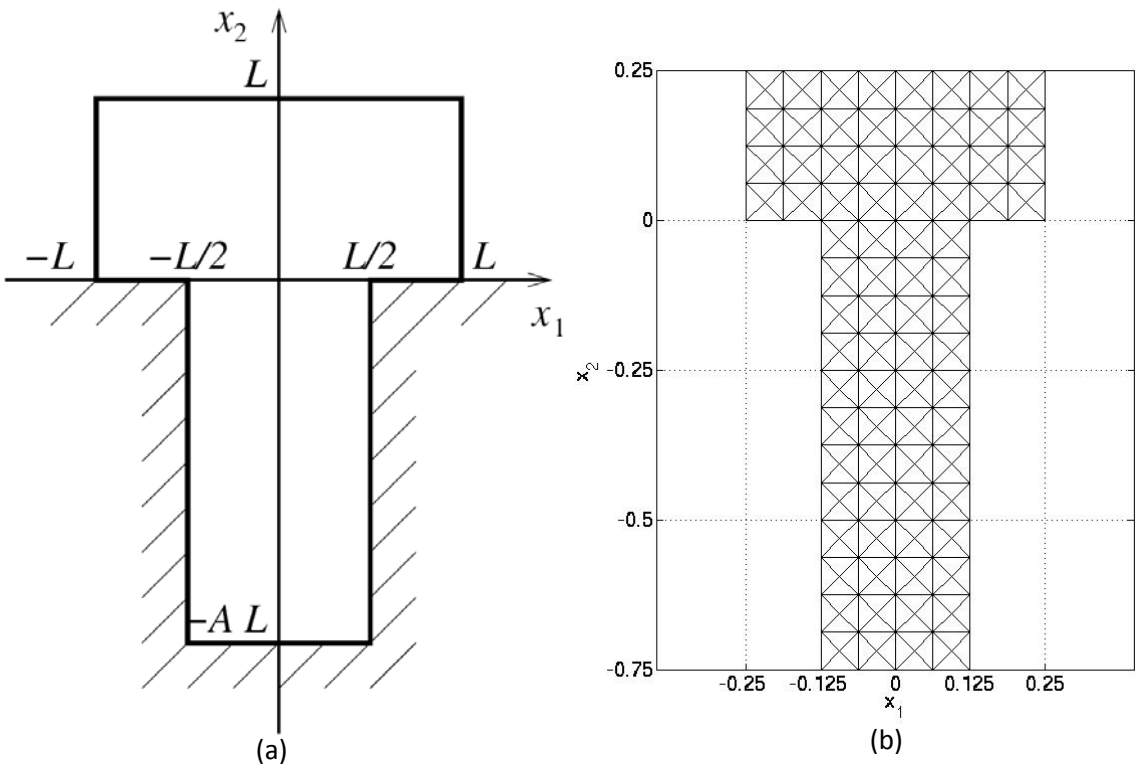


Figure 1: (a) Cross-section of a feature and domain for the mathematical model with feature width $L = 0.25\ \mu$m and feature aspect ratio $A = 3$. (b) Sample domain mesh used for the numerical method with uniform mesh spacing $h = 1/16 = 0.0625\ \mu$m.

Section 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

$$\frac{\partial f^{(i)}(\mathbf{x},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}} f^{(i)}(\mathbf{x},\mathbf{v},t) = \frac{1}{\mathrm{Kn}} Q_i(f^{(i)}(\mathbf{x},\mathbf{v},t)), \qquad i = 0,\ldots,n_s, \qquad (3.1)$$

where $Q_i(f^{(i)})$ is an operator that factors in the collisions between the reactive species and the inert background gas, and Kn is the Knudsen number, a physical parameter of the system [3]. We only deal with a 2-D/2-D kinetic model in this paper, so the position $\mathbf{x} = (x_1, x_2)^{\mathrm{T}} \in \Omega \subset \mathbb{R}^2$ and the velocity $\mathbf{v} = (v_1, v_2)^{\mathrm{T}} \in \mathbb{R}^2$. Approximating $f^{(i)}(\mathbf{x},\mathbf{v},t)$ by $f_K^{(i)}(\mathbf{x},\mathbf{v},t) = \sum_{\ell=0}^{K-1} f_\ell^{(i)}(\mathbf{x},t)\varphi_\ell(\mathbf{v})$

for each species i with carefully chosen basis functions of velocity space $\varphi_\ell(\mathbf{v})$ results in a system of K transient linear first-order hyperbolic transport equations

$$\frac{\partial F^{(i)}}{\partial t} + A^{(1)} \frac{\partial F^{(i)}}{\partial x_1} + A^{(2)} \frac{\partial F^{(i)}}{\partial x_2} = \frac{1}{\mathrm{Kn}} B^{(i)} F^{(i)}, \qquad i = 0, \ldots, n_s, \qquad (3.2)$$

in space $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$ and time t for the vector of K coefficient functions $F^{(i)}(\mathbf{x},t) := (f_0^{(i)}(\mathbf{x},t), \ldots, f_{K-1}^{(i)}(\mathbf{x},t))^{\mathrm{T}}$. The $K \times K$ matrices $A^{(1)}, A^{(2)}$, and $B^{(i)}$ are constant due to the linearity of (3.1); moreover, due to the choice of basis functions, $A^{(1)}$ and $A^{(2)}$ are also diagonal [5]. Using the diagonality of these matrices, the system (3.2) 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 the code DG [8], a C++ program that uses MPI (the Message Passing Interface) as the library for the communications between the parallel processes. We use the MVAPICH2 implementation of MPI for the studies here.

An important aspect of our numerical testing is to recognize the complexity of the problem with which we are working. The degrees of freedom (DOF) of the problem denote the number of unknowns that must be computed at every time step. In this paper, we look only at the case of a domain mesh with 1,280 quadrilateral spatial elements with uniform mesh spacing h = 1/64 = 0.015625 and a velocity mesh with resolution of $K = 16 \times 16 = 256$ discrete velocities; this is Case 4 in the studies in [6]. Figure 1 (b) shows an example of a domain mesh with the coarser spacing of h = 1/16 for clearer visibility. The DOF of the computational problem can be calculated by multiplying the four local degrees of freedom of the discontinuous bi-linear finite elements on each quadrilateral spatial element, the number of spatial elements, the size of the velocity mesh, and the number of reactive species. This yields a value of (4)(1,280)(256)(2) = 2,621,440; this means that over 2.6 million solution components have to be computed at each time step. For our study, we set a final time $t_{\mathrm{final}} = 10$ nanoseconds. The constant time step used, calculated to be the largest possible while still guaranteeing stability, is $\Delta t \approx 3 \cdot 10^{-4}$, resulting in a total of 31,829 time steps.

Section 4: Results

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 nanoseconds (ns). For this model, we set our dimensionless reaction coefficients $\gamma_1^f = 10^{-2}, \gamma_1^b = 10^{-4}, \gamma_2^f = 10^{-2}, \gamma_3^f = \gamma_3^b = \gamma_4^f = 0,$ and the Knudsen number Kn = 100. Because of this high value of the Knudsen number, the discrete numerical velocities interact only insignificantly, which explains the appearance of distinct plateaus in the concentration plots of the figure. The small unphysical undershoot of values in the plot for 0 ns is typical for methods with discontinuous finite elements. From Figure 2, 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.

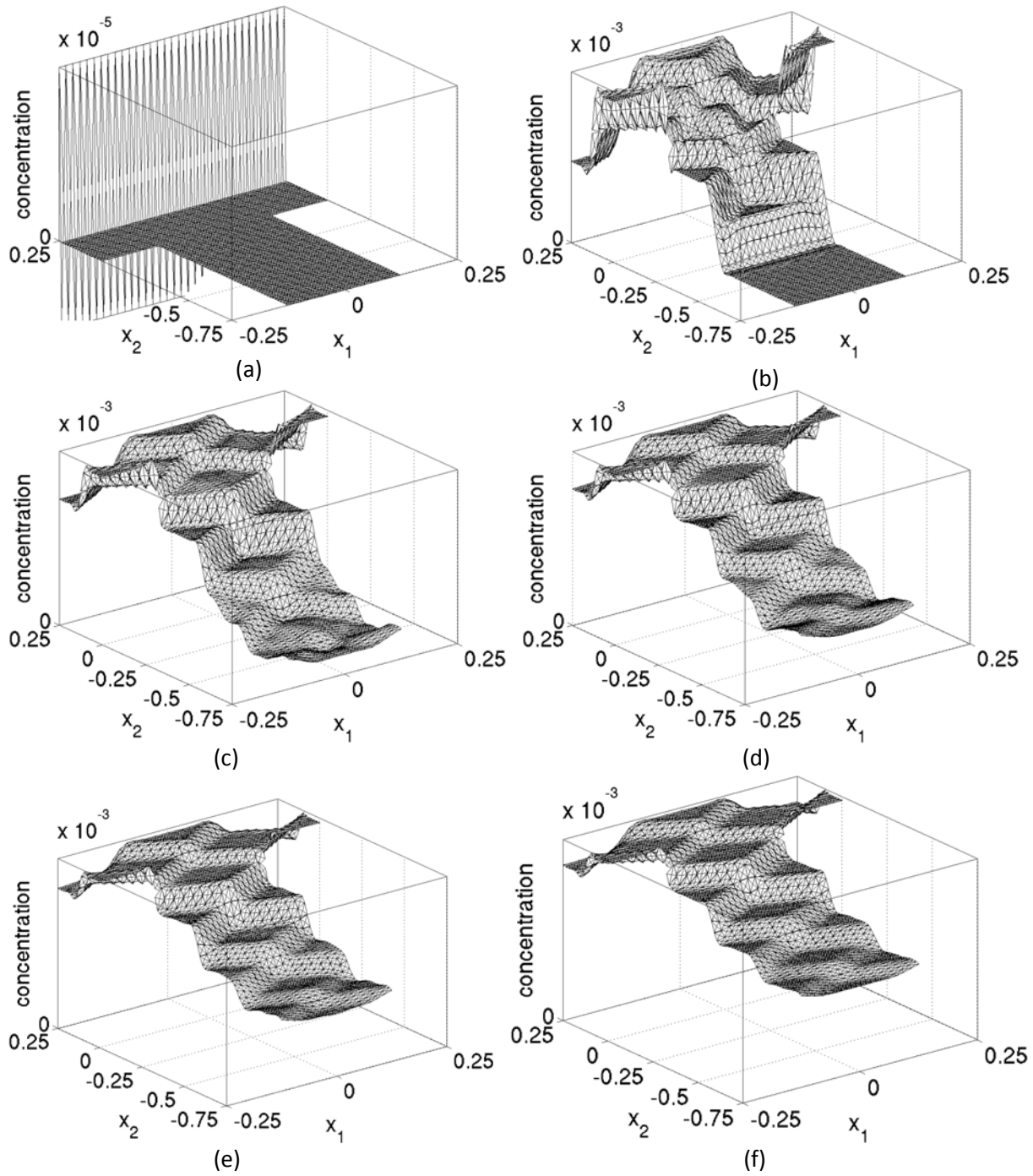


Figure 2: Concentration plot of species A at times (a) 0 ns, (b) 2 ns, (c) 4 ns, (d) 6 ns, (e) 8 ns, and (f) 10 ns.

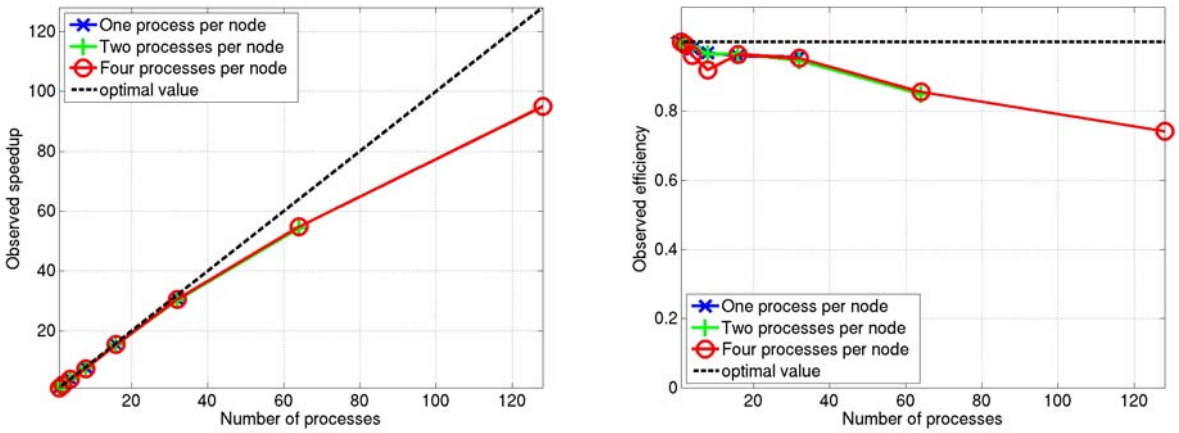
Table 1: Wall clock time in HH:MM:SS for the code DG on hpc using MVAPICH2, with $h = 1/64$, $K = 16 \times 16$, and $t_{\text{final}} = 10$.

	1 process per node	2 processes per node	4 processes per node
p = 1	136:43:01	N/A	N/A
p = 2	68:41:39	68:53:33	N/A
p = 4	34:53:16	34:42:25	35:34:47
p = 8	17:38:10	17:41:56	18:35:31
p = 16	08:54:06	08:50:45	08:51:48
p = 32	04:27:59	04:30:56	04:28:49
p = 64	N/A	02:31:02	02:29:47
p = 128	N/A	N/A	01:26:25

Parallel Performance Study: To study the parallel performance of the code on the cluster hpc described in Section 1, we vary both the number of nodes used and the number of parallel processes run on each node through all possible combinations of their values. The numbers 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 cores are kept idle. The product of nodes used and processes per node then gives the number *p* of parallel MPI processes of the run. The wall clock times in units of hours:minutes:seconds (HH:MM:SS) for these runs are shown in Table 1 with N/A marking those cases where the combination of nodes and processes does not exist on the system. Reading along each column in the table, we see that the run times approximately halve for each doubling of the number of parallel processes; this is the ideal behavior of parallel code: Running code with twice as many parallel processes should be twice as fast. Reading along each row, we see that the run times for any number *p* is independent of the number of processes run per node; to understand the significance of this observation, consider the example of a run that uses $p = 32$ parallel MPI processes: If we run only one process per node, we need to use all 32 nodes of the cluster, and thus no other user can run any code on the machine. But if we run two processes per node, our run only requires 16 nodes. Or with four processes per node, we only need 8 nodes. Thus, many other users can still run their jobs simultaneously to our job. To justify the usage policy of the cluster that is designed in this way, it is important for the users that their jobs do not suffer any significant performance degradation from running more than one process per node.

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 as listed in Table 1, 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, the optimal value for S_p is *p*. The **efficiency** is then defined as $E_p = S_p / p$, and thus has an optimal value of 1 [2, 5]. As a visual representation of the results of Table 1, speedup and efficiency plots are shown in Figure 3, with markers distinguishing the three cases of number of processes per node. We notice that the markers overlie each other for the value of parallel processes *p*, for which data exists. This is an excellent result, which confirms that the scalability of the code is independent of the number of processes run per node. Notice in the efficiency plot that the performance only degrades slightly with the increasing number of parallel processes all the way up to $p = 128$; the efficiency plot

indicates only slightly less than 80% efficiency at $p = 128$. These are excellent results for a real-life application code!



(a) (b)

Figure 3: Comparison of (a) observed speedup and (b) observed efficiency when running one, two, or four processes per node.

Section 5: Conclusions

The excellent performance results presented in the previous section for the cluster hpc with 32 nodes and InfiniBand interconnect [6] confirm corresponding studies on a cluster with 32 nodes and Myrinet interconnect [7] as well as a study with an earlier version of the code using 8 nodes connected by a departmental ethernet connection [1].

Most importantly, we can conclude from the presented results that there is no downside for the users to running more than one process per node, as evidenced by the plots for one, two, and four processes per node being essentially identical. Therefore, one should run as many parallel processes per node as possible in order to run the code in the smallest amount of time as well as to allow as many users as possible to run code simultaneously on the system.

For these performance test runs, a smaller final time of 10 ns was used; in actual ALD simulations, the final times might be approximately 40 ns [5], or four times longer than the runs presented here. We can extrapolate from the results here that if we were to use only one process, a 40 ns final time would take over 540 hours to simulate; however, if we utilize all 128 processes possible, the time taken will be a much more reasonable 6 hours. This illustrates both the power and necessity of parallel computing in modeling ALD or similar processes.

The cluster hpc, on which the results of this paper were obtained, is currently being replaced the new cluster tara. The new cluster has computational nodes with two quad-core Intel Nehalem processors. Since these nodes have twice as many cores per node than hpc, the results of this paper are more important already and will become even more relevant for future systems with even more cores per node.

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. Follow the link under the Systems tab for more specific information on the cluster hpc. I also thank UMBC for providing assistance for this research through an Undergraduate Research Award.

References

[1] Steven C. Foster. Performance studies for the discontinuous Galerkin method applied to the scalar transport equation. *UMBC Review: Journal of Undergraduate Research and Creative Works*, vol. 4, pp. 36-47, 2003.

[2] Matthias K. Gobbert. Parallel performance studies for an elliptic test problem. Technical Report HPCF-12008-1, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2008.

[3] 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.

[4] 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.

[5] 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.

[6] Michael J. Reid. Comparison of parallel performance between MVAPICH2 and OpenMPI applied to a hyperbolic test problem, Senior Thesis, Department of Mathematics and Statistics. Technical Report HPCF-2008-9, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2008.

[7] 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.

[8] 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.