Parallel Performance Studies for a Hyperbolic Test Problem

Michael J. Reid and Matthias K. Gobbert

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

{mic7@umbc.edu,gobbert@math.umbc.edu}

Abstract

The performance of parallel computer code depends on an intricate interplay of the processors, the architecture of the compute nodes, their interconnect network, the numerical algorithm, and the scheduling policy used. This note considers a case study of a solver of a system of transient hyperbolic conservation laws which utilizes both point-to-point and collective communications between parallel processes at each time step. The solver is already known to scale well to many parallel processes on distributed-memory clusters with a high performance interconnect network. The results presented here show excellent overall performance of the new cluster hpc with InfiniBand interconnect and confirm that is beneficial to use the maximum number of cores possible on every node, allowing a total of 128 parallel processes on the 32 compute nodes.

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. The behavior of the reactive species can be modeled by a kinetic transport and reaction model (KTRM) whose mathematical representation is a system of transient linear Boltzmann equations combined with a surface reaction model [3, 4, 5]. Each Boltzmann equation is discretized in velocity space by an expansion in terms of specially chosen velocity basis function resulting in a system of transient linear hyperbolic conservation laws [6]. These transient partial differential equations are solved by the discontinuous Galerkin method [7]. At each time step, there are both point-to-point and collective communications between all parallel processes, making the code an excellent way to test the parallel performance of the network. Section 2 explains the application problem and its mathematical model in more detail, while Section 3 documents the numerical method used.

Past results using the KTRM and its implementation [2, 6], show that the code scales extremely well on parallel computing clusters with a high performance interconnect network. These results were obtained on a distributed-memory cluster purchased in 2003 from IBM with two (single-core) processors per node and a Myrinet interconnect. This note considers the performance of the new distributed-memory cluster hpc, purchased in 2008 also from IBM, with each node having two dual-core AMD Opteron 2.6 GHz processors with 1024 kB cache per core and 13 GB of memory per node, connected by a state-of-the-art InfiniBand interconnect network. The processors on each node allow for up to four processes to be run simultaneously per node. Section 4 evaluates the performance results in detail.

Table 1 summarizes the key results of the study by giving the total time to execute the code in units of hours:minutes:seconds (HH:MM:SS). We use coarse and fine meshes for both the spatial domain and velocity discretization, resulting a total of four test cases. The number of degrees of freedom (DOF) specifies the number of unknowns that need to be computed in every time step and can be used as an indication of the complexity of each case. The parallel code is run on different numbers of nodes, ranging from 1 to 32, running either 1, 2, or 4 processes per node. The upper-left entry of each sub-table corresponds to the serial run of the code, using only one process on one node. Similarly, the lower-right entry of each sub-table corresponds to running 4 processes on all 32 nodes, or 128 processes in total. Note that this entry does not exist for the first and third cases due to the particular way in which the domain is split, and thus the maximum number of nodes that can be used is given by the two table entries adjacent to the lower-right for these cases. As we see by looking at the case of the fine spatial mesh with mesh spacing $h = 0.015625$ together with the fine velocity resolution $K = 16 \times 16$, we can use parallel computing to reduce the time taken to run the code from around 2:43 hours (163 minutes) to under 2 minutes.

The results in Table 1 are organized such that two key questions may be answered: (i) if the code scales linearly to all 32 nodes, which is a test of the quality of the InfiniBand interconnect network, and (ii) whether multiple processors and cores per node should be used.

(i) Reading along each row of Table 1 we see mixed results. In several cases, the run time is not halved as the number of processes doubles, as one would expect. Particularly, for the first two cases of the coarser

Table 1: Wall clock time in HH:MM:SS 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 $h = 0.03125$, 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	00:01:27	00:01:23	00:00:32	00:00:17	00:00:09	00:00:06
2 processes per node	00:00:47	00:00:29	00:00:15	00:00:12	00:00:06	00:00:07
4 processes per node	00:00:26	00:00:15	00:00:09	00:00:07	00:00:06	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	00:10:53	00:10:39	00:05:40	00:02:53	00:01:29	00:00:42
2 processes per node	00:05:47	00:03:01	00:02:39	00:01:28	00:00:41	00:00:19
4 processes per node	00:03:01	00:01:36	00:01:27	00:00:39	00:00:20	00:00:18
(c) Coarse spatial mesh with $h = 0.03125$, 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	00:20:33	00:10:26	00:05:17	00:02:43	00:01:29	00:01:01
2 processes per node	00:10:26	00:05:19	00:02:42	00:01:30	00:00:52	00:00:30
4 processes per node	00:05:20	00:02:45	00:01:33	00:00:54	00:00:32	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	02:43:39	01:23:02	00:42:08	00:21:27	00:10:51	00:05:36
2 processes per node	01:23:18	00:42:09	00:21:28	00:10:55	00:05:36	00:03:08
4 processes per node	00:42:31	00:21:30	00:11:00	00:05:37	00:03:06	00:01:50

velocity resolution $K = 8 \times 8$, while there is a clear trend toward speedup overall, we observe that going from 1 process on 1 node to 1 process on 2 nodes produces almost no speedup, despite the expectation that the latter would be twice as fast. In the next two sub-tables for the finer velocity resolution $K = 16 \times 16$, there is an observable speedup in proportion to the number of nodes used, and the algorithm and its implementation make effective use of the cluster.

(ii) To analyze the results of running 1, 2, or 4 parallel processes per node, we compare the results column-wise on each sub-table. It is here that we see the clear halving (approximately) of times when doubling the number of processes used per node. This is an excellent result, and reinforces the previous knowledge that the code should be run with as many processes per node as available. This confirms also the usage policy currently implemented on hpc that uses all cores on a node simultaneously by default.

Comparing the results of Table 1 to the previously obtained results given in Table 7, we see a vast difference in wall clock times for corresponding cases. For instance, on a serial run of the finest mesh, we improve from a time of 9.5 hours to approximately 2.75 hours, which is about 3.5 times faster.

2 The Hyperbolic Test Problem

Many steps during the manufacture of integrated circuits involve the deposition of a seed layer of solid material onto the surface of a silicon wafer through a process called atomic layer deposition (ALD). The goal of 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 [3, 4, 5]. However, if the gas used for B is hydrogen radicals, denoted by H, it is believed that the deposition of the desired solid

may be inhibited in two ways: (i) Some H may adsorb to the surface as H_v and thus block A from adsorbing at that surface site. (ii) Some gaseous H may react with adsorbed H_v to form a gaseous H_2 and thus would not be available for the desired surface reaction. The proposed reaction model between A and H is given by

$$\begin{aligned} A + v &\rightleftharpoons A_v, && (2.2a)\\ H + A_v &\rightarrow \text{solid} + \text{by-product} + v, && (2.2b)\\ H + v &\rightleftharpoons H_v, && (2.2c)\\ H + H_v &\rightarrow H_2 + v, && (2.2d) \end{aligned}$$

where the last two reactions express the two inhibition pathways. The forward reaction rate for the ith equation is given by γ_i^f for each of the four reactions, and the backward reaction rate is given by γ_i^b for each of the two reversible reactions.

The mathematical model for the flow of the gaseous species through the spatial domain $\Omega \subset \mathbb{R}^2$ that encompasses the neighborhood of the wafer surface is given by a system of linear Boltzmann equations

$$\frac{\partial f^{(i)}(\mathbf{x},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}} f^{(i)}(\mathbf{x},\mathbf{v},t) = \frac{1}{\mathrm{Kn}} Q_i(f^{(i)}(\mathbf{x},\mathbf{v},t)), \quad i = 1,\ldots,n_s, \tag{2.3}$$

for $\mathbf{x} = (x_1, x_2)^T \in \Omega$, $\mathbf{v} \in \mathbb{R}^2$, and $0 < t \le t_{\text{final}}$, 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

$$Q_i(f^{(i)}) = \int_{\mathbb{R}^2} \sigma_i(\mathbf{v},\mathbf{v}') \left[M^{(i)}(\mathbf{v}') f^{(i)}(\mathbf{x},\mathbf{v},t) - M^{(i)}(\mathbf{v}') f^{(i)}(\mathbf{x},\mathbf{v},t) \right] d\mathbf{v}', \tag{2.4}$$

where $M^{(i)}$ is the Maxwellian of species i. Together with the above reaction model, this flow model comprises the kinetic transport and reaction model KTRM [2].

3 The Numerical Method

The solution $f^{(i)}(\mathbf{x},\mathbf{v},t)$ of the KTRM in (2.3) needs to be computed for all species $i = 1,\ldots,n_s$ at all spatial points $\mathbf{x} = (x_1,x_2)^T \in \Omega$, for all velocities $\mathbf{v} \in \mathbb{R}^2$, and for all times $0 \le t \le t_{\text{final}}$. The key of the numerical method is to discretize in velocity space first by approximating $f^{(i)}(\mathbf{x},\mathbf{v},t)$ 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 [6]. Each Boltzmann equation in (2.3) is then discretized by inserting $f_K^{(i)}(\mathbf{x},\mathbf{v},t)$ for $f^{(i)}(\mathbf{x},\mathbf{v},t)$ and testing against the K basis functions $\varphi_k(\mathbf{v})$, $k = 0,\ldots,K-1$. This results in a system of K transient linear hyperbolic conservation laws

$$\frac{\partial F^{(i)}}{\partial t} + A^{(1)}\frac{\partial F^{(i)}}{\partial x_1} + A^{(2)}\frac{\partial F^{(i)}}{\partial x_2} = \frac{1}{\mathrm{Kn}} B^{(i)} F^{(i)} \tag{3.1}$$

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 (2.3). Moreover, the specific basis functions constructed in [6] guarantee that the matrices $A^{(1)}$, $A^{(2)}$ are diagonal. This means that the coupling between species comes through the wafer surface boundary condition that implements the crucial surface reactions (2.2).

For simplicity, we only deal with the 2-D/2-D kinetic model in this report, so that our spatial domain $\Omega \subset \mathbb{R}^2$ and velocity space $\mathbb{R}^2$ are both two-dimensional. We additionally assume we are dealing with the ideal case, wherein the gaseous H neither adsorbs to the surface nor forms H_2; in this case, $\gamma_3^f = \gamma_3^b = \gamma_4^f = 0$. While this means the third and fourth reactions never occur, they are still computed in our code, and thus the computational complexity still reflects the full four reaction model.

To test the performance of the code, we use four cases: Case 1 with velocity resolution $K = 8 \times 8 = 64$ and spatial mesh spacing $h = 0.03125$; Case 2 with $K = 8 \times 8 = 64$ and $h = 0.015625$; Case 3 with $K = 16 \times 16 = 256$ and $h = 0.03125$; and Case 4 with $K = 16 \times 16 = 256$ and $h = 0.015625$. We measure the complexity of these problems by the number of degrees of freedom (DOF), which is given by the number of solution components that have to be computed at every time step. The systems of K hyperbolic conservation laws are exactly the type

Table 2: 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	0.03125	320	163,840	0.20	$1.00500 \cdot 10^{-3}$	199	00:01:27
Case 2	8×8	0.015625	1280	655,360	0.20	$5.02700 \cdot 10^{-4}$	398	00:10:53
Case 3	16×16	0.03125	320	655,360	0.20	$6.28370 \cdot 10^{-4}$	319	00:20:33
Case 4	16×16	0.015625	1280	2,621,440	0.20	$3.14187 \cdot 10^{-4}$	637	02:43:39

of problem that the discontinuous Galerkin finite element method [7] was designed for. We use two-dimensional quadrilateral elements with four local degrees of freedom (the solution value at every vertex) and a uniform mesh spacing h. 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, and K is the size of the velocity mesh. The degrees of freedom of each case are collected in Table 2. 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 parallel process is assigned one of the sub-domains. The processes perform the computations for the update of the solution on all sub-domains in parallel at each time step. At each time step, there are both point-to-point communications between pairs of processes that hold adjacent sub-domains and collective communications among all processes.

4 Performance Studies on hpc

Because finer velocity and spatial meshes cause an increased number of degrees of freedom, which must be solved at a large number of time steps, parallel computing can be used to dramatically decrease the time taken to solve the problem by distributing the work load using multiple processes. If $T_p(N)$ is defined as the wall clock time for a problem of fixed size N by using p processes, then the quantity $S_p = T_1(N)/T_p(N)$ is the speedup of the code from 1 to p processes, with an optimal value $S_p = p$. The efficiency $E_p = S_p/p$ measures how close a run is to optimal speedup, in which case $E_p = 1$ [1].

Table 3 summarizes the results of running only one process on each node, that is, we use only one core of the two dual-core processors on each node with the remaining three cores idling. Since we have 32 nodes total, this distribution cannot be used for $p > 32$; thus, we run 2 processes per node for $p = 64$ and 4 processes per node for $p = 128$. Each row lists the results for one problem size, and each column corresponds to the number of parallel processes p used in that run. We notice no improvement in the wall clock time from $p = 1$ to $p = 2$ for the first two cases. This is unexpected and could possibly be attributed to some overhead with using MPI dominating over the computational cost of the coarse velocity mesh in these cases. For larger values of p as well as for the latter two cases, good speedup is exhibited all the way to $p = 32$, as using twice as many processes speeds up the code by approximately a factor two each time. This is evidenced by the corresponding speedup plot in Figure 1 (a) and associated efficiency plot in Figure 1 (b), where the efficiency for these cases remains above $E_p = 0.6$ throughout.

We then analyze the impact of using more than one core per node; the results of using two cores per node are shown in Table 4 and Figure 2, while the results of using four cores per node are given in Table 5 and Figure 3. Note that two and four cores are only used when possible: In both tables, one process per node is used for $p = 1$; in Table 4, four processes per node must be used for $p = 128$; and in Table 5, two processes per node are used for $p = 2$. The most important difference to notice is that there is not the huge efficiency drop-off from $p = 1$ to $p = 2$ for the first two cases, as there was with using one core per node. For the other two cases, we see difference between timing values for using one, two, or four cores is negligible. This indicates that for this algorithm and its implementation we can use as many cores on as many processors as we have available without any disadvantage. Indeed, looking at Table 5, we see that using the maximum number of cores available—$p = 64$ for $h = 0.03125$ and $p = 128$ for $h = 0.015625$—gives us the smallest execution time in each case.

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

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

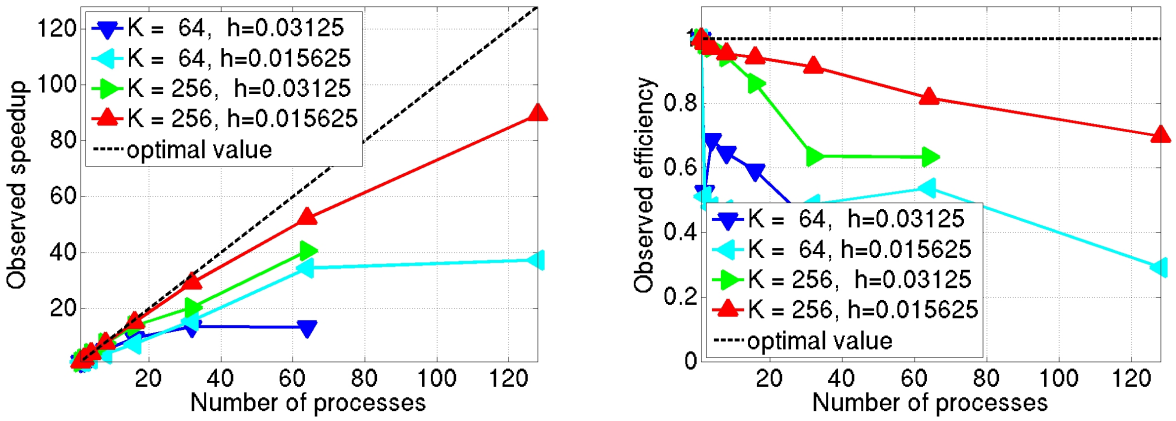
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	00:01:27	00:01:23	00:00:32	00:00:17	00:00:09	00:00:06	00:00:07	N/A
$K = 64$, $h = 0.015625$	00:10:53	00:10:39	00:05:40	00:02:53	00:01:29	00:00:42	00:00:19	00:00:18
$K = 256$, $h = 0.03125$	00:20:33	00:10:26	00:05:17	00:02:43	00:01:29	00:01:01	00:00:30	N/A
$K = 256$, $h = 0.015625$	02:43:39	01:23:02	00:42:08	00:21:27	00:10:51	00:05:36	00:03:08	00:01:50

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	1.0000	1.0510	2.7419	5.1693	9.4668	13.4884	13.3231	N/A
$K = 64$, $h = 0.015625$	1.0000	1.0225	1.9193	3.7648	7.3190	15.5922	34.4046	37.3143
$K = 256$, $h = 0.03125$	1.0000	1.9681	3.8923	7.5570	13.7904	20.3600	40.5726	N/A
$K = 256$, $h = 0.015625$	1.0000	1.9709	3.8836	7.6311	15.0716	29.2145	52.2454	89.3367

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	1.0000	0.5255	0.6855	0.6462	0.5917	0.4215	0.2082	N/A
$K = 64$, $h = 0.015625$	1.0000	0.5113	0.4798	0.4706	0.4574	0.4873	0.5376	0.2915
$K = 256$,$h = 0.03125$	1.0000	0.9840	0.9731	0.9446	0.8619	0.6362	0.6339	N/A
$K = 256$,$h = 0.015625$	1.0000	0.9855	0.9709	0.9539	0.9420	0.9130	0.8163	0.6979



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

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

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

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

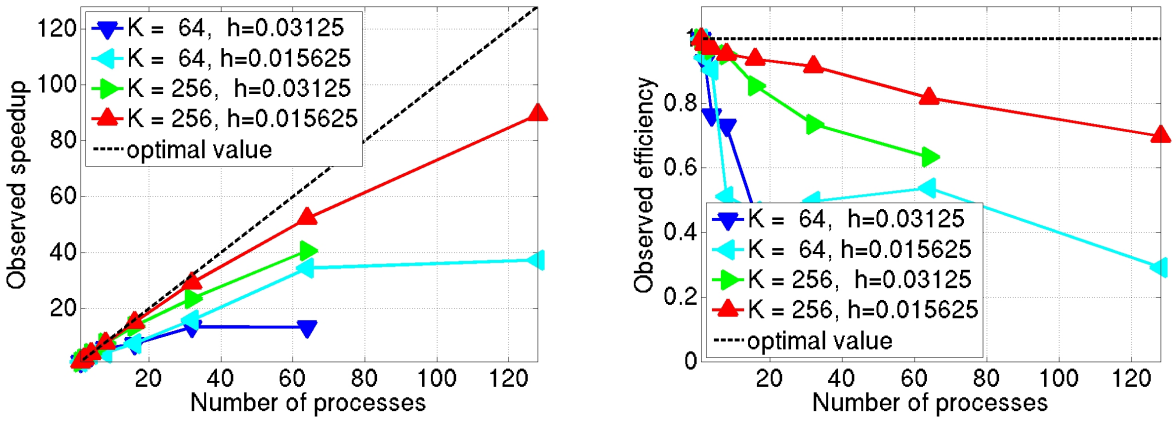
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	00:01:27	00:00:47	00:00:29	00:00:15	00:00:12	00:00:06	00:00:07	N/A
$K = 64$, $h = 0.015625$	00:10:53	00:05:47	00:03:01	00:02:39	00:01:28	00:00:41	00:00:19	00:00:18
$K = 256$,$h = 0.03125$	00:20:33	00:10:26	00:05:19	00:02:42	00:01:30	00:00:52	00:00:30	N/A
$K = 256$,$h = 0.015625$	02:43:39	01:23:18	00:42:09	00:21:28	00:10:55	00:05:36	00:03:08	00:01:50

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	1.0000	1.8610	3.0494	5.8389	7.2987	13.4052	13.3231	N/A
$K = 64$, $h = 0.015625$	1.0000	1.8840	3.6151	4.0946	7.4535	15.8765	34.4046	37.3143
$K = 256$,$h = 0.03125$	1.0000	1.9681	3.8662	7.6008	13.6681	23.5305	40.5726	N/A
$K = 256$,$h = 0.015625$	1.0000	1.9646	3.8832	7.6212	14.9817	29.2537	52.2454	89.3367

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	1.0000	0.9305	0.7624	0.7299	0.4562	0.4189	0.2082	N/A
$K = 64$, $h = 0.015625$	1.0000	0.9420	0.9038	0.5118	0.4658	0.4961	0.5376	0.2915
$K = 256$,$h = 0.03125$	1.0000	0.9841	0.9665	0.9501	0.8543	0.7353	0.6339	N/A
$K = 256$,$h = 0.015625$	1.0000	0.9823	0.9708	0.9526	0.9364	0.9142	0.8163	0.6979



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

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

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

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

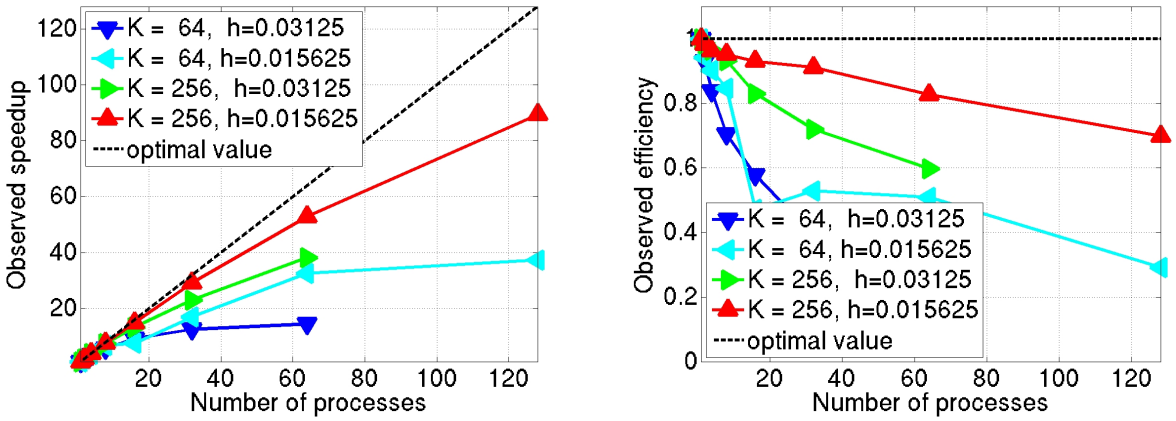
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	00:01:27	00:00:47	00:00:26	00:00:15	00:00:09	00:00:07	00:00:06	N/A
$K = 64$, $h = 0.015625$	00:10:53	00:05:47	00:03:01	00:01:36	00:01:27	00:00:39	00:00:20	00:00:18
$K = 256$,$h = 0.03125$	00:20:33	00:10:26	00:05:20	00:02:45	00:01:33	00:00:54	00:00:32	N/A
$K = 256$,$h = 0.015625$	02:43:39	01:23:18	00:42:31	00:21:30	00:11:00	00:05:37	00:03:06	00:01:50

(b) Observed speedup S_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	1.0000	1.8610	3.3474	5.6420	9.2357	12.5000	14.4279	N/A
$K = 64$, $h = 0.015625$	1.0000	1.8840	3.6083	6.7760	7.5448	16.9346	32.6011	37.3143
$K = 256$,$h = 0.03125$	1.0000	1.9681	3.8571	7.4605	13.2881	23.0424	38.2563	N/A
$K = 256$,$h = 0.015625$	1.0000	1.9646	3.8498	7.6136	14.8847	29.1573	52.9269	89.3367

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$K = 64$, $h = 0.03125$	1.0000	0.9305	0.8369	0.7053	0.5772	0.3906	0.2254	N/A
$K = 64$, $h = 0.015625$	1.0000	0.9420	0.9021	0.8470	0.4715	0.5292	0.5094	0.2915
$K = 256$,$h = 0.03125$	1.0000	0.9841	0.9643	0.9326	0.8305	0.7201	0.5978	N/A
$K = 256$,$h = 0.015625$	1.0000	0.9823	0.9625	0.9517	0.9303	0.9112	0.8270	0.6979



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

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

A Performance Studies on kali

This appendix contains performance results for the same study as described above performed on kali, a distributed-memory cluster purchased in 2003. The studies in [6] were run on this cluster and demonstrated excellent parallel performance of the algorithm and its implementation. The cluster kali has currently 27 nodes connected through a Myrinet network available. Each node contains two (single-core) Intel Xeon 2.0 GHz chips with 512 kB cache and 1 GB memory per node.

Table 6 lists the results of a serial sizing study analogous to Table 2. The results for the time stepping are identical to the ones obtained on hpc, which confirms that the code gives correct results on both clusters. The serial times show that the new cluster hpc is about a factor 3.5 faster than kali.

Table 7 is a summary of raw timing results, analogous to Table 1 in Section 1. The observed speedup both row-wise and column-wise is as expected, demonstrating the quality of the high performance Myrinet interconnect network.

Table 8 and Figure 4 show performance results when running only one process per node, up to the maximum possible $p = 16$. In all cases, we see excellent speedup, though the cases with more degrees of freedom exhibit slightly better speedup. Table 9 and Figure 5 summarize performance results when using two processes on each node, up to $p = 32$. Again, there is excellent speedup throughout, with the more complex cases slightly more efficient. As is the case with hpc, the difference in performance when using two processors per node is negligible, due to the low memory demands of the code being tested.

References

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

[2] 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.

[3] 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.

[4] Matthias K. Gobbert, Vinay Prasad, and Timothy S. Cale. Predictive modeling of atomic layer deposition on the feature scale. *Thin Solid Films*, vol. 410, pp. 129–141, 2002.

[5] Matthias K. Gobbert, Samuel G. Webster, and Timothy S. Cale. Transient adsorption and desorption in micrometer scale features. *J. Electrochem. Soc.*, vol. 149, no. 8, pp. G461–G473, 2002.

[6] 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.

[7] 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.

Table 6: Sizing study on kali 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	0.03125	320	163,840	0.20	$1.00500 \cdot 10^{-3}$	199	00:05:42
Case 2	8×8	0.015625	1280	655,360	0.20	$5.02700 \cdot 10^{-4}$	398	00:41:47
Case 3	16×16	0.03125	320	655,360	0.20	$6.28370 \cdot 10^{-4}$	319	01:10:04
Case 4	16×16	0.015625	1280	2,621,440	0.20	$3.14187 \cdot 10^{-4}$	637	09:31:52

Table 7: Performance on kali. Wall clock time in HH:MM:SS for the solution of four cases of velocity and spatial meshes using 1, 2, 4, 8, and 16 compute nodes with 1, 2, and 4 processes per node.

(a) Coarse spatial mesh with $h = 0.03125$, coarse velocity resolution $K = 8 \times 8$					
	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	00:05:42	00:03:03	00:01:36	00:00:51	00:00:27
2 processes per node	00:03:05	00:01:38	00:00:53	00:00:29	00:00:18
(b) Fine spatial mesh with $h = 0.015625$, coarse velocity resolution $K = 8 \times 8$					
	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	00:41:47	00:22:14	00:11:24	00:05:48	00:02:59
2 processes per node	00:22:06	00:11:23	00:05:47	00:03:00	00:01:36
(c) Coarse spatial mesh with $h = 0.03125$, fine velocity resolution $K = 16 \times 16$					
	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	01:10:04	00:36:02	00:18:19	00:09:23	00:04:52
2 processes per node	00:36:06	00:18:21	00:09:22	00:04:54	00:02:50
(d) Fine spatial mesh with $h = 0.015625$, fine velocity resolution $K = 16 \times 16$					
	1 node	2 nodes	4 nodes	8 nodes	16 nodes
1 process per node	09:31:52	04:57:15	02:35:38	01:21:30	00:41:29
2 processes per node	04:57:47	02:36:02	01:21:34	00:41:14	00:21:24

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

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

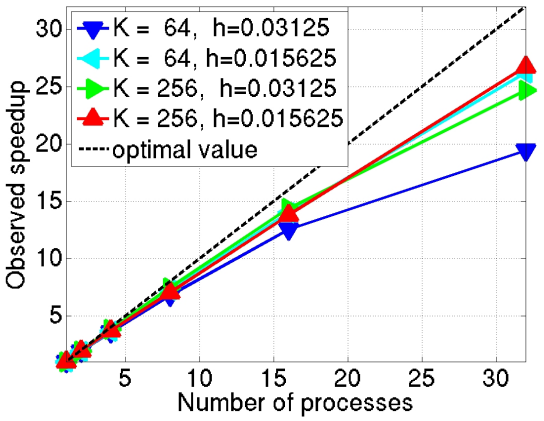
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$K = 64$, $h = 0.03125$	00:05:42	00:03:03	00:01:36	00:00:51	00:00:27	00:00:18
$K = 64$, $h = 0.015625$	00:41:47	00:22:14	00:11:24	00:05:48	00:02:59	00:01:36
$K = 256$, $h = 0.03125$	01:10:04	00:36:02	00:18:19	00:09:23	00:04:52	00:02:50
$K = 256$, $h = 0.015625$	09:31:52	04:57:15	02:35:38	01:12:30	00:41:29	00:21:24

(b) Observed speedup S_p

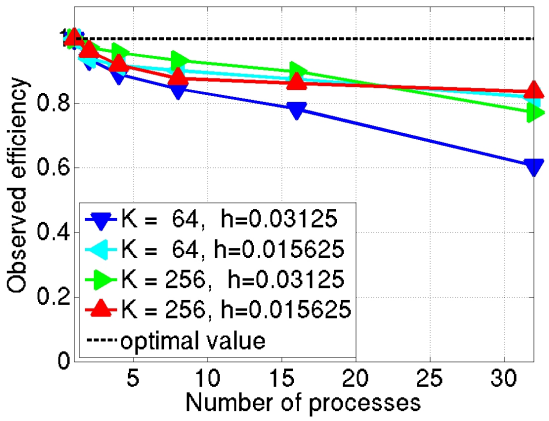
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$K = 64$, $h = 0.03125$	1.0000	1.8724	3.5618	6.7562	12.5137	19.4318
$K = 64$, $h = 0.015625$	1.0000	1.8788	3.6669	7.2115	13.9985	26.2019
$K = 256$, $h = 0.03125$	1.0000	1.9447	3.8260	7.4657	14.3731	24.6786
$K = 256$, $h = 0.015625$	1.0000	1.9239	3.6746	7.0173	13.7835	26.7325

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$K = 64$, $h = 0.03125$	1.0000	0.9362	0.8904	0.8445	0.7821	0.6072
$K = 64$, $h = 0.015625$	1.0000	0.9394	0.9167	0.9014	0.8749	0.8188
$K = 256$, $h = 0.03125$	1.0000	0.9723	0.9565	0.9332	0.8983	0.7712
$K = 256$, $h = 0.015625$	1.0000	0.9620	0.9186	0.8772	0.8615	0.8354



(a) Observed speedup S_p



(b) Observed efficiency E_p

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

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

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

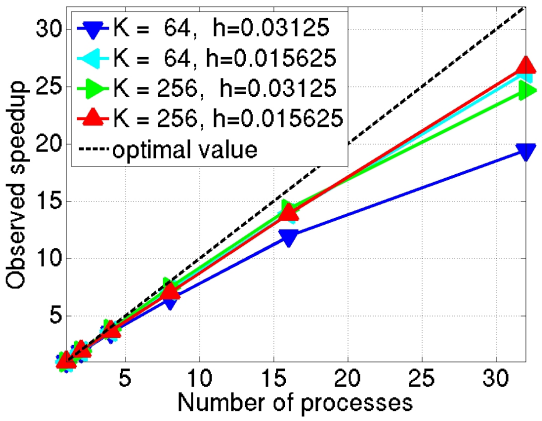
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$K = 64$, $h = 0.03125$	00:05:42	00:03:05	00:01:38	00:00:53	00:00:29	00:00:18
$K = 64$, $h = 0.015625$	00:41:47	00:22:06	00:11:23	00:05:47	00:03:00	00:01:36
$K = 256$, $h = 0.03125$	01:10:04	00:36:06	00:18:21	00:09:22	00:04:54	00:02:50
$K = 256$, $h = 0.015625$	09:31:52	04:57:47	02:36:02	01:21:34	00:41:14	00:21:24

(b) Observed speedup S_p

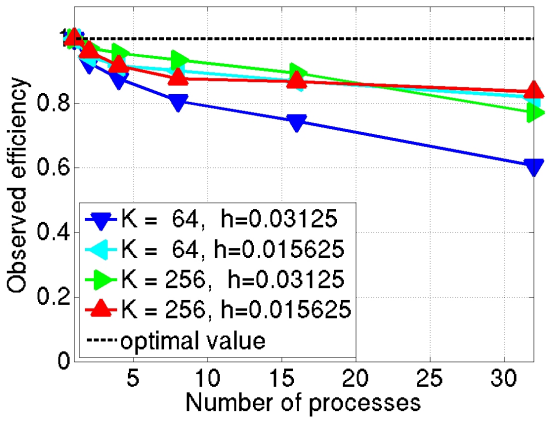
	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$K = 64$, $h = 0.03125$	1.0000	1.8503	3.5055	6.4540	11.9164	19.4318
$K = 64$, $h = 0.015625$	1.0000	1.8904	3.6647	7.2148	13.8923	26.2019
$K = 256$, $h = 0.03125$	1.0000	1.9411	3.8196	7.4755	14.3037	24.6786
$K = 256$, $h = 0.015625$	1.0000	1.9204	3.6651	7.0108	13.8685	26.7325

(c) Observed efficiency E_p

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$K = 64$, $h = 0.03125$	1.0000	0.9252	0.8764	0.8068	0.7448	0.6072
$K = 64$, $h = 0.015625$	1.0000	0.9452	0.9162	0.9019	0.8683	0.8188
$K = 256$, $h = 0.03125$	1.0000	0.9706	0.9549	0.9344	0.8940	0.7712
$K = 256$, $h = 0.015625$	1.0000	0.9602	0.9163	0.8764	0.8668	0.8354



(a) Observed speedup S_p



(b) Observed efficiency E_p

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