Performance Analysis of the Lattice Boltzmann Model Beyond Navier-Stokes

Performance Analysis of the Lattice Boltzmann Model Beyond Navier-Stokes

Amanda Peters Randles*, Vivek Kale†, Jeff Hammond‡, William Gropp†, and Efthimios Kaxiras*
*School of Engineering and Applied Sciences
Harvard University, Cambridge, Massachusetts 02138
Contact Email: apeters@fas.harvard.edu
‡Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439
†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 61801

***Abstract*—The lattice Boltzmann method is increasingly important in facilitating large-scale fluid dynamics simulations. To date, these simulations have been built on discretized velocity models of up to 27 neighbors. Recent work has shown that higher order approximations of the continuum Boltzmann equation enable not only recovery of the Navier-Stokes hydrodynamics, but also simulations for a wider range of Knudsen numbers, which is especially important in micro- and nanoscale flows. These higher-order models have significant impact on both the communication and computational complexity of the application. We present a performance study of the higher-order models as compared to the traditional ones, on both the IBM Blue Gene/P and Blue Gene/Q architectures. We study the tradeoffs of many optimizations methods such as the use of deep halo level ghost cells that, alongside hybrid programming models, reduce the impact of extended models and enable efficient modeling of extreme regimes of computational fluid dynamics.**

***Keywords*-fluid dynamics; lattice Boltzmann; multicore optimization;**

I. Introduction

With the increasing demand for micron scale simulations for devices such as those used for microfluidics, there is an urgent need for models that can accurately model fluid flow beyond the continuum regime, and for the development of optimization techniques that will enable these models to achieve strong performance on current and future computer architectures. The objective of this work is to study the performance impact of improving the accuracy of a computational fluid dynamics (CFD) model and to identify methods to mitigate this cost, thus making the simulation of extreme regimes of CFD tractable.

We have developed a multiscale fluid dynamics simulation that models flow in complicated geometries from microfluidic devices to patient-specific arterial geometries obtained from computed tomography (CT) scans [1], [2]. Our initial models have focused on flow in the coronary arteries where the diameters are on the order of millimeters as shown in Fig. 1. We are currently extending the use of this application to other domains in fluid dynamics such as the study of clogging in a microfluidic device. In expanding the use of this application to modeling gaseous flows in such devices, the model must be extended to accurately simulate flows beyond the continuum regime. This will enable us to study situations in which the traditional model may also fail for liquids, as in the case of modeling the plasma flow between the particulates in blood.

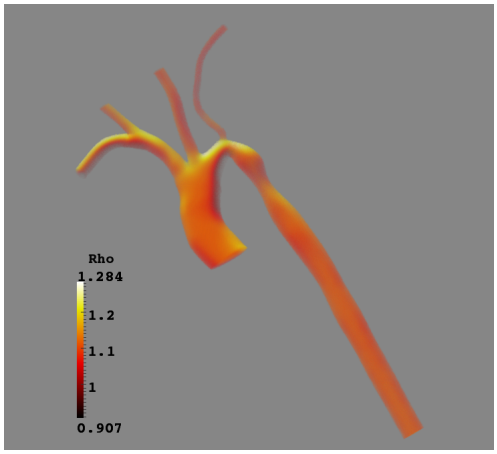


Figure 1. Fluid density in the aorta.

Traditionally, CFD methods for studying flow are based on the Euler or Navier-Stokes equations. These equations assume that the fluid is being modeled as a continuum; however, at small scales this assumption begins to break down and conventional CFD approaches become inaccurate [3]. The limit to the regimes accurately captured by these models are flows with Knudsen numbers (Kn) between 0 and 0.1 [4]; where $Kn = \frac{\lambda}{L}$ with λ being the average distance traveled by a molecule between collisions (the mean free path), and L the macroscopic length scale within which flow occurs. Beyond this range, many of the continuum assumptions break down and corrections are necessary as the contributions from higher kinetic moments are no longer negligible [5].

Experiments have shown that the conventional methods may not produce accurate results for rarefied flows [6], [7], [8]. Alternative methods such as the direct simulation Monte Carlo [3], extensions to Navier-Stokes [9], [8], and use of the Burnett equations [10], have been investigated to address these situations. Due to its kinetic nature, the lattice

Boltzmann method (LBM) offers a promising alternative for simulating flows in which Kn falls outside the [0-0.1] interval. In this paper, we focus on recent advances to the lattice Boltzmann model (LBM) that extend its reach to accurately model flows for larger Kn ranges such as those described by Chan, Yuan and Chen [11]. These higher order methods impact both the communication bandwidth and computational complexity of the application. The goal of this paper is to assess the impact of the higher order models on computational performance and introduce ways to mitigate this cost. Our metric of success is defined as minimal wall clock time in seconds and maximal work units completed per second to enable the modeling of larger fluid systems in shorter physical time.

Our hypothesis is that the use of deep halo ghost cells alongside further enhancements such as optimized data handling and structures, loop reordering and separation, branch minimization, and communication tuning will enable us to significantly improve the code's exhibited performance. We first conduct experiments to measure the effect of the code quality and impacts of singe node optimizations. We determine the impact that the message aggregation has on the communication performance. We finally discuss experiments that address the challenges associated with scaling such as communication performance and threading with MPI/OpenMP. Having defined our optimization levels, we validate our methodology using both the IBM Blue Gene/P and IBM Blue Gene/Q architectures. This analysis not only provides an upper bound of the potential performance metrics for targeted supercomputing architectures, but also highlights the increasing performance restriction on the LBM due to the growing disparity between increases in bandwidth and flop rate on new architectures.

We demonstrate significant performance results: 83% of the predicted upper bound for Blue Gene/P and 79% on Blue Gene/Q. This correlated with a three-fold improvement on Blue Gene/P and almost an eight-fold improvement on Blue Gene/Q due to our optimizations for the extended models. We demonstrate that models of extreme fluid flows through the extended LBM can be efficiently simulated on large-scale supercomputing platforms and that the computational and memory burdens can be mitigated through careful tuning alongside the use of special features such as deep ghost cells and hybrid programming models.

II. The Lattice Boltzmann Method

The LBM has received considerable attention due to its advantageous handling of complex flow phenomena in irregular boundary conditions and ease of parallelization. It is an approach to computational fluid dynamics based on kinetic theory. Implicit in the method is the discretization of the velocity and space. A regular Cartesian grid is applied to the volume of a 3-dimensional mesh, and the fluid motion is derived by simultaneously solving a minimal form of the classical Boltzmann equation at each grid (or lattice) point [12]. The density of the grid spacing determines the resolution of the simulation.

The fundamental quantity of the LBM is the particle distribution function, $f(x,t)$, that describes the likelihood of finding a fictitious fluid particle at lattice point x, at time step t, moving at the discrete velocity c_i. The particles move only along discretized velocity paths defined by the lattice. The distribution is evolved according to Eq. (1) [12]:

$$f(x+c_i\Delta t, t+\Delta t) = f(x,t) - \omega\Delta t(f(x,t) - f^{eq}(x,t)) \quad (1)$$

There are two key components to the algorithm: collision and advection. The collision step is calculated through a relaxation towards local equilibrium, as shown in the right hand side of Eq. (1). In this work, we use the most common collision operator, the Bhatnagar-Gross-Krook (BGK), which relaxes to equilibrium on a single time scale [13]. The local equilibrium is defined as a truncated Hermite expansion of a local Maxwellian with density ρ and speed u [5]. The Navier-Stokes equation is recovered with a second order expansion:

$$f_i^{eq} = \omega_i\rho\left\{1 + \frac{\xi_i\cdot u}{c_s^2} + \frac{1}{2}\left(\frac{(\xi_i\cdot u)^2}{(c_s^2)^2} - \frac{u^2}{c_s}\right)\right\} \quad (2)$$

Higher-order expansions enable the physical effects beyond the continuum regime to be modeled. The third order accurate expansion is defined as the D3Q39 discrete velocity model, given by:

$$\begin{aligned} f_i^{eq} &= \omega_i\rho\left\{1 + \frac{\xi_i\cdot u}{c_s^2} + \left(\frac{(\xi_i\cdot u)^2}{2(c_s^2)^2} - \frac{u^2}{c_s}\right)\right. \\ &\quad \left. + \frac{\xi_i\cdot u}{6c_s^2}\left(\frac{(\xi_i\cdot u)^2}{c_s^2} - 3\frac{u^2}{c_s}\right)\right\} \end{aligned} \quad (3)$$

where ω defines the quadrature weight and c_s the speed of sound [14]. The added term in Eq. (3) is related to the velocity-dependent viscosity of the fluid. As discussed in [11], third-order truncation requires a discrete velocity model of sixth order isotropy as opposed to the fourth order needed for Eq. (2).

In this work, we focus on two velocity models. For continuum flow, we use the common 19-speed cubic D3Q19 lattice connecting each lattice point to its first and second neighbors [1]. The associated weights and discretized velocities are given in Table I. To study further regimes, we employ a model using the next-order kinetic moments, the 39-point Gauss-Hermite quadrature defined in [11].

The advection, or streaming step, involves propagating the fluid particles along the appropriate velocity trajectories. For the D3Q39 model, as opposed to the D3Q19 that focuses on up to second neighbors, particles can travel to lattice nodes that are as far away as the fifth nearest neighbor [14]. The velocities describe the 18 first and second neighbors or 38

Table I
PARAMETERS FOR THE TWO DISCRETE VELOCITY MODELS.

D3Q19 Lattice					D3Q39 Lattice				
c_s^2	ξ_i	ω_i	Neighbor Order	Distance	c_s^2	ξ_i	ω_i	Neighbor Order	Distance
$1/3$	$(0,0,0)$	$1/3$	0	0	$2/3$	$(0,0,0)$	$1/12$	0	0
$1/3$	$(\pm1,0,0)$	$1/18$	1	1	$2/3$	$(\pm1,0,0)$	$1/12$	1	1
$1/3$	$(\pm1,\pm1,0)$	$1/36$	2	$\sqrt{2}$	$2/3$	$(\pm1,\pm1,\pm1)$	$1/27$	2	$\sqrt{3}$
					$2/3$	$(\pm2,0,0)$	$2/135$	3	2
					$2/3$	$(\pm2,\pm2,0)$	$1/142$	4	$2\sqrt{2}$
					$2/3$	$(\pm3,0,0)$	$1/1620$	5	3

first, second, third, fourth, and fifth nearest neighbors and the 19th and 39th values are for the lattice point itself, these are represented in the first row of the tables.

III. SYSTEMS

A. Platform Overview

The two platforms used in this paper are the IBM Blue Gene/P and IBM Blue Gene/Q architectures. Both rely on a system-on-a-chip backbone. The Blue Gene/P has a 32-bit PowerPC 450 processor that runs at 850 MHz. Each node consists of 4 cores capable of executing SIMD instructions when data is 16-byte aligned resulting in a peak performance of 13.6 GFlop/s. There are 2 GB of memory per node and 1 thread per processor, allowing up to four threads per node. Point-to-point communication between nodes is handled via a 3D torus with a hardware (software) bandwidth per unidirectional link of 425 (375) MB/s [15].

Blue Gene/Q has a similar modular design but expands the options for threading, memory access, and speeds. It has a 64-bit PowerPC processor at 1.6 GHz. Each node consists of 16 cores with 4 potential threads per core. There is a 204.8 GFlop/s peak performance per node [16]. Memory per node is expanded to 16 GB and there is support for speculative execution and hardware assist to *sleep* threads while waiting for an event [17].

B. MFlup/s: A Performance Metric for the LBM

In order to determine the methods of optimization, it is first important to assess the bounds on performance expectations of our model for the platforms of focus.

When analyzing lattice Boltzmann performance, focusing on the flop/s is not the best metric as this can vary widely based on factors such as the implementation of the model, compilers, and hardware used. A more meaningful metric is the work done per unit time. For LBM, this means the number of lattice points updated per second. A standard measure for this is to measure *MFlup/s*, or million lattice point updates per second, which assesses the runtime of a production application depending only on domain size and number of time steps simulated. Equation 4 shows how the peak number of potential MFlup/s is calculated for a specific simulation. In this case, $T(s)$ refers to the execution time for s steps and N_{fl} defines the number of fluid cells [18].

$$P[MFlup/s] = \frac{s \cdot N_{fl}}{T(s) \cdot 10^6} \tag{4}$$

Based on the specific hardware details of each platform, we can calculate the maximum performance attainable of Eq. (4) and determine the performance limiting factors for our model in terms of bandwidth vs. computation. The application performance will either be limited by available memory bandwidth or peak performance. To calculate the attainable maximum performance P in MFlup/s, we use Wellein et al.'s model defined with Eq. (5) [18], in which B is the number of bytes per cell transferred to and from main memory and F is the number of floating point operations per cell.

$$P = \frac{Bm}{B} || \frac{Ppeak.}{F} \tag{5}$$

In this implementation there are two load operations and one store operation for every velocity mode. For the D3Q19 model, this results in $B = (19+19+19)*8 = 456$ bytes per lattice point while for the D3Q39 model, there are 936 bytes per lattice point. For the calculation of P in Table II, we use the main store bandwidth measurement for individual compute nodes to obtain the maximum attainable performance. Inherently, we will see production results below these values as those implementations span multiple nodes and require point-to-point communication over the torus.

Both systems are capable of performing a maximum of four double precision floating-point operations (two multiply and two add) per cycle. To get the high baseline, we assume maximal use of this functionality in this performance model. This is clearly an overstatement as the stream function consists primarily of load and store operations while the collide function has a high number of addition operations. For the D3Q19 model, our implementation has 178 core floating-point operations and for the D3Q39 model, it has 190 core floating-point operations. These rates do not depend on problem size.

The subsequent estimates for maximum achievable performance on these two platforms are shown in Table II, in

terms of peak MFlup/s given the main store bandwidth of the system and the peak given the flop/s for each processor. The calculated max MFlup/s are shown with the limiting factor for each system highlighted in red. Similar to the previous studied architectures, the bandwidth imposes the performance limit on each system.

Table II
TABLE OF THE MAXIMUM MFLUP/S ATTAINABLE ON THE IBM BLUE GENE/P AND IBM BLUE GENE/Q SYSTEMS FOR BOTH LATTICES WITH PERFORMANCE LIMITERS HIGHLIGHTED IN RED. IN ALL CASES, THE CODE IS EXTREMELY BANDWIDTH LIMITED. THE HARDWARE SYSTEM DATA FOR THE IBM BLUE GENE SYSTEMS COMES FROM [15], [16], AND [17].

D3Q19 Lattice				
System	**Bm**	**P(Bm)**	**Ppeak**	**P(Ppeak)**
BG/P	13.6 GB/s	29 MFlup/s	13.6 GFlop/s	76.4 MFlup/s
BG/Q	43 GB/s	94 MFlup/s	204.8 GFlop/s	1150 MFlup/s
D3Q39 Lattice				
System	**Bm**	**P(Bm)**	**Ppeak**	**P(Ppeak)**
BG/P	13.6 GB/s	14.5 MFlup/s	13.6 GFlop/s	71.5 MFlup/s
BG/Q	43 GB/s	45 MFlup/s	204.8 GFlop/s	1077 MFlup/s

C. What Does this Mean for LBM Simulations?

The fact that both models for both architectures are bandwidth limited indicates that the stream function is the limiting function as it consists of the bulk of the load/store operations in the movement of the particles. While the overall runtime can be reduced through arithmetic optimization of the collide function, scalability will be inherently limited by the memory bandwidth and therefore by the stream function. When extrapolating beyond the single node performance, the data is retrieved via point-to-point communication on the torus. Assuming all loads and stores occur at the torus bandwidth provides a lower bound for parallel performance. For D3Q19 this falls at 11.1 MFlup/s and 70 MFlup/s for BG/P and BG/Q respectively. As for D3Q39, the lower bounds are at 5.4 MFlup/s and 34 MFlup/s. Of course this is an overestimate and a real code will have a mix of various accesses to various cache levels as well as communication, but this provides a crude view of performance expectations that is surprisingly useful.

This goal of this analysis is simply to provide insight into the limits of potential performance tuning and therefore give greater context to the results of the previously discussed optimizations. The P(Fp) defines the number of MFlup/s that would be attained at the peak flop rate. The ratio of P(Bm) to P(Fp) provides the upper bound on potential hardware efficiency. For Blue Gene/P, the models have the potential of achieving 38% (D3Q19) and 20% (D3Q39) hardware efficiency. While some applications achieve greater than 60% efficiency, most production parallel codes only leverage at most 10% of the available flop/s. This makes LBM ripe for high efficiency on such platforms. It is worth noting that the off node memory accesses have a less steep drop off in bandwidth, so as the code is highly parallelized, there will be less of a performance impact.

This model is, of course, over simplified and contains many assumptions, however, it does provide strong ground for assessing the upper bound of potential performance on new architectures and targeting optimization efforts. In our case, it is worth noting that the ghost cell implementation will add computation cycles not accounted for in the flop/flup ratio.

IV. IMPLEMENTATION

In the work presented here, the goal is to assess the direct impacts on the computational performance due to algorithmic changes necessary to simulate fluid flow at finite Kn. To this end, all simulations in this work are of a cubic fluid system with periodic boundary conditions. This assumption allows the analysis to focus on the impact of the higher order terms and extended neighbors of the lattice instead of being dominated by boundary conditions. We further limit the study to a three-dimensional fluid system with one-dimensional domain decomposition. While this could restrict performance at the large-scaling limit, it again shifts focus to the algorithm and more specifically enables direct analysis of ghost cell depth impact. As the function of this code is to serve as the fluid component in a multiphysics coupling of red blood cell and blood plasma motion, the realistic domain decomposition will be irregular and rely on neighbor lists. Furthermore, it's been shown that cubic blocking can lead to overhead for long thin channels such as the geometries we would expect in artery and capillary blood flow models [18]. The code discussed in this paper is written in C and uses MPI and OpenMP for the parallelization.

As mentioned earlier, an LBM simulation consists of alternating steps completing the streaming and colliding of particle populations. A straightforward implementation of the method is shown in Fig. 2, in which a starting distribution of fluid particles is initialized, the stream function propagates the particles to adjacent lattice points and store these values in a temporary distribution function `distr_adv`. The collide function subsequently reads `distr_adv`, determines all resulting collisions, relaxes the population towards equilibrium, and updates the original array. In this way, it acts as a general stencil code using information from it's neighbors to update it's value and then pushing it's data to the neighbors, however, the data accessed in `distr_adv` is from another phase space.

Rüde, Pohl, and Wellein have extensively studied optimal data structures and cache blocking strategies for the BGK model (c.f. [18], [19], [20]). In our implementation, we use the *collision optimized* layout that they describe as optimal.

```
read initial distr
for (n< max_steps) {
  distr_adv=stream(distr);
  LBM_Exchange();
  distr = collide(distr_adv);
}
```

Figure 2. Naive implementation of the LBM.

```
for ix < xDim
for iy < yDim
for iz < zDim {
  for is < numVel {
    ixa=ix+icx[is]
    iya=iy+icy[is]
    iza=iz+icz[is]
    boundary_counditions();
    distr_adv[is][iza+iya*Lz+ixa*Lz*Ly]
        = distr[is][iz+iy*Lz+ix*Lz*Ly]
  }
}
```

Figure 3. Stream pseudocode. The icx, icy, and icz arrays define the velocity directions, ξ_i.

In order to maximize messaging performance and set the code up for an easy transition to the use of indirect addressing necessary for irregular domains, the distribution functions were stored in two dimensional arrays of $(NumVelocities, zDim \cdot yDim \cdot xDim)$ allocated in contiguous memory. [1].

Fig. 3 shows the details of the stream function. For each lattice point, all potential velocities are iterated over. The component of the velocity, ξ_i, is added to the correlating component of the lattice point coordinates. For example, particles with velocity $(1, 0, 0)$ at lattice point $(0, 0, 0)$ would stream to position $(1, 0, 0)$. The resulting distribution of the streaming step is stored in the temporary data structure of distr_adv.

The collide step is outlined in Fig. 4. For each lattice point and for each discrete velocity, macroscopic quantities of ρ and u are calculated locally. These values are then used to determine the relaxation towards equilibrium via the aforementioned BGK collision operator. Finally, the distribution array is updated. Note that the collision step relies on information from the neighboring processes stream function due to the fact that the stream function can result in particles displacing to lattice points contained on neighboring processors.

V. Optimizations

A precise simulation of fluid flow using either velocity model is demanding and requires well-optimized and scalable code. In this section, we present the sequential and parallel optimizations employed.

```
for ix < xDim
for iy < yDim
for iz < zDim {
  for is < numVel {
    calc_rho_and_vel()
    BGK=calc_BGK_op
    distr=update(distr_adv,BGK)
  }
}
```

Figure 4. Collide pseudocode.

A. Deep Halo Ghost Cells

As the update in the collide function requires data from the distr_adv array from all neighboring processors, this can lead to a communication bottleneck. A commonly employed tool to alleviate this contention at the boundaries is to add *ghost cells* or *halo cells*. The addition of this ghost layer increases the distribution array size by one in each direction of domain decomposition. At each time step, the neighboring processors exchange a copy of their border cells and receive the borders falling in their own ghost cell regions as shown in Fig. 5. Each processor adds an extra row to its domain of interest, as shown in blue, and populates this ghost cell row with the border data from its neighboring processor.

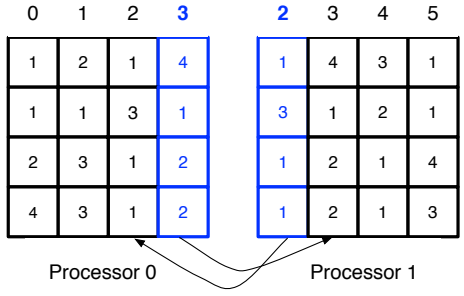


Figure 5. 2D example of ghost cells in x-dimension. Each processors receives a row from its neighboring processor to be used in the stencil calculation.

The use of an extra row of ghost cells is often found in large-scale models [21], [18]; however, by stopping at one row, potential for further tuning is being left unexplored. Kjolstad and Snir discussed implementation methods of the ghost cell pattern in [22] and suggested the investigation of *deep halos* as a potential method to trade off computation for communication. A deep halo refers to the use of ghost cell depth greater than one. Deep halos can be leveraged to further offset message latency by reducing the number of overall messages used in a simulation. While this requires extra computation to update the ghost cells, in some cases the benefit from message reduction and further

overlap of communication and computation can make this advantageous. By increasing the number of ghost cell width by a factor n, the data exchange can be minimized to only be required every n steps [22].

Note that for the D3Q39 lattice model, we must use a deep halo implementation simply for correctness. As mentioned previously, this model allows particles to move to neighboring grid points that are further away within one time step. The fundamental ghost cell depth must be set to include the number of neighbors that a particle could move within a time step (k). Discussions of ghost cell depth for the D3Q39 refer to the multiples of k included. For example, a ghost cell depth of 2 would include $2k$ additional cells at each side of a border exchange.

In a later section, we investigate the role that deep halo exchanges have on the performance of both velocity models investigated in this paper.

B. Data Handling (DH)

A thorough set of standard optimizations regarding the handing of the data was employed to improve performance. One of the overarching goals was to reduce the number of floating point operations in the two most intensive routines: stream() and collide(). Temporary variables were introduced to remove any redundant computation and arithmetic division was replaced with the multiplication of the reciprocal due to the heavy cycle count associated with division operations.

In this case, the largest impact came from optimal cache usage. Loops were restructured to both maximize cache reuse and reduce any recalculation. As mentioned earlier, the discrete velocities of the distribution function, $f(i)(zdim \cdot ydim \cdot zdim)$ are located contiguously in memory. To maximize cache reuse, we reorganized the loops such that all velocities are iterated over followed by the z-,y- and x-coordinates in memory order.

This was a moderate impact on performance on the Blue Gene/P architecture, 30%, but a very significant impact of an 75% increase in MFlup/s on Blue Gene/Q. This is due to the extensive cache hierarchy. In the original implementation, almost no loads during the collide function hit in the L2 cache while 3% hit in DDR. After the DH tuning, there was a .4% increase in L1 d-cache and L1P buffer hits and a 1.2% increase in L2 cache hits while DDR dropped to .01%. This resulted in a longer load latency in the original version. During the stream function, cache hits were now optimized to fall only in L1 d-cache and the L1P buffer. These measurements were taken with the IBM Hardware Performance Monitor [23].

C. Compiler Optimizations

We assessed the impact that various XL/C compiler optimizations had for our application compared to the default $O3$ optimization level which provided better loop scheduling and memory usage. We found that the most aggressive optimization level of $O5$ produced the strongest results while maintaining correct results, surpassing that of $O3$. While compilation took longer, the benefit gained was worthwhile. The most improvement, however, was gained through optimization of the intra-procedural analysis (IPA). By setting the qipa level to 2, we enabled whole-program alias analysis including the disambiguation of pointer dereferences and indirect function calls and whole program data reorganization. This resulted in significant performance gain while in our case, accuracy was maintained.

For the BG/Q implementation, we found that a lower optimization setting of $O3$ produced better results increased the produced MFlup/s by 2.5x. By investigating generated lst files, we found that the compilers were more successful in automating loop unrolling and optimizing the floating-point instructions for this architecture.

D. Loop Restructuring and Branching Reduction (LoBr)

To further minimize the over all runtime of the application, we restructured the loops and reduced any branching in the code. With the addition of ghost cells to the simulations, especially those of deep ghost cell levels, there are several distinct sections of the domain to be modeled on each processor. In the case of a 1D domain decomposition, there is the ghost cell region from the previous processor, the local domain of interest, and the ghost cell region covering data from the next processor. In the LBM, both the stream and collide functions must iterate through loops that cover all three regions. We found that by explicitly separating these into different for loop groupings, we were able to better take advantage of the cache and minimize index calculation.

More improvement was garnered through a branch reduction trick we developed to swap if statements with for loops. This is outlined in Fig. 6. We removed all if statements from the innermost loop and replaced them with a for-loop that is able to continue without stalling. The location of where a particle is displaced to determines which region the new index falls in. For storing data, there is an offset needed to store ghost cell data. We create an array of these new indices based on the x-index in the outermost loop. This array is then iterated over for 1, 2, or 3 passes depending on the number of regions being spanned.

E. Nonblocking Communication

In the naive implementation, blocking communication was used to exchange data between processors. This was switched to the use of non-blocking MPI_Irecv, MPI_Isend, and MPI_Waitall. Especially for the non-ghost cell case, there is no opportunity to allow an overlap of computation and communication as the collide function directly relies on the results of the stream function from it's neighbors. The MPI_Irecv is posted before the local stream calculation and the MPI_ISend posts at the completion of the local stream. This

```
for ix < xDim {
  ixa=ix+icx[is]
  count = 0;
  if (xmin<  ixa < xmax)
    index[count] = (ixa-my_xmin+GCS)*LyLz
    count++
  if (gc_min1<  ixa < gc_max1)
    index[count] =  (ixa-gc_min1)*LyLz
    count++
  if (gc_min2<  ixa < gc_max2)
    index[count] =   (my_Lx-GCS+ixa-gc_min2)*LyLz
    count++
  for iy < yDim {
    iya=iy+icy[is]
    for iz < zDim{
      iza=iz+icz[is]
      for is < numVel {
        boundary_counditions()
        a = iz+iy*Lz+(ix-my_xmin+GCS)*LyLz
        for (jj=0; jj < count; jj++)
          distr_adv[is][iza+iya*Lz+index[jj]] =
              distr[is][a]
      }
    }
  }
}
```

Figure 6. Stream pseudocode with branch optimization.

results in a small reduction in the communication overhead that will be shown in the Results section. In the ghost cell implementation, the data can be sent at the end of the time step and waited on before the next stream function commences.

F. Separate collide function for collide (GC-C)

When using ghost cells, and especially when using deeper halos, we can actually increase the computation/communication overload by a much further degree. We introduce a separate function to handle the collision phase of the ghost cell regions. As the data being sent to the processor's neighbor is the border region of the domain of interest on that processor, it can be calculated and sent before the ghost cell region collisions are computed. By separating out the handling of the ghost cells and the region of interest, we can hide the message latency by overlapping it with the ghost cell computation. This is outlined in Fig. 7.

G. SIMD Vectorization

Examining the compiler generated code for both BG/P and BG/Q, showed that we failed to have SIMD double hummer intrinsics leveraged, therefore cutting our potential hardware efficiency already in half. To maximize performance, we modified the code to explicitly generate double hummer intrinsics through direct calls to instructions like fpmadd. This required enforcing 16-byte alignment and the disjoint pragma. Alongside the intrinsics, we use XL/C

```
for (n < maxsteps) {
  read initial distr
  for (i < num_velocities) {
    for (z <z_dim)
    for (y < y_dim)
    for (x < x_dim) {
      if (n%GCL == 0) {
        MPI_Send
        MPI_Waitall
      }
      distr_adv = stream()
      distr = collide(distr_adv)
      if ( (n+1) % GCL ==0)  {
        MPI_Irecv(distr);
      }
      distr = gc_collide(distr_adv);
    }
  }
}
```

Figure 7. Separate handing of ghost cell collision.

pragmas to force loop unrolling of the innermost loops in the functions [23].

For Blue Gene/Q, we again tried several compiler options but tried hand coding the intrinsics functions. In this case, there were modifications to the compiler so this work needed to be re-implemented for BG/Q instead of BG/P. Also, BG/Q has the expanded ability to handle different data alignments than simply the 16-byte alignment required for BG/P. Specifically in the collide function, we were able to take advantage of the quad-word load, store, and arithmetic operations. We were able to take advantage of fused multiply-add instructions but were more limited. Without moving to vector doubles, we were not able to fully exploit QPX instructions [23].

VI. Results

To assess the impact of the various optimizations previously discussed, Fig. 8 shows the results of progressive tuning of the two velocity models on each hardware and their approach to the peak performance rate defined by our performance model. Performance is presented in terms of the previously discussed quantity, MFlup/s. For Blue Gene/P, the MFlup/s achieved for D3Q19 is 92% of the peak performance from our model. The slight discrepancy can be partially accounted for in that the model is actually targeting single node performance while Fig. 8 depicts results from multi-node runs. This difference was intentional in order to enable side-by-side comparison of the single node optimizations with the communication improvements. It does, however, introduce the previously discussed performance degradation from use of the torus for communication instead of all on node memory access. The torus has a lower bandwidth and will reduce the achieved MFlup/s. Moreover, the optimal runtime was achieved using the ghost

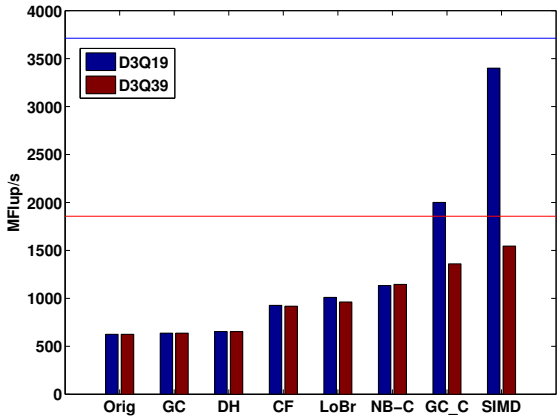
cell method, which adds lattice updates not accounted for in the performance model.

Additionally, the model shows a maximum hardware efficiency of 38% and with these optimizations, we achieve 31% of peak flop/s for the full simulation and 43% hardware efficiency in the compute heavy collide routine. This further confirms that the optimizations discussed have tuned the code almost to its maximum potential.

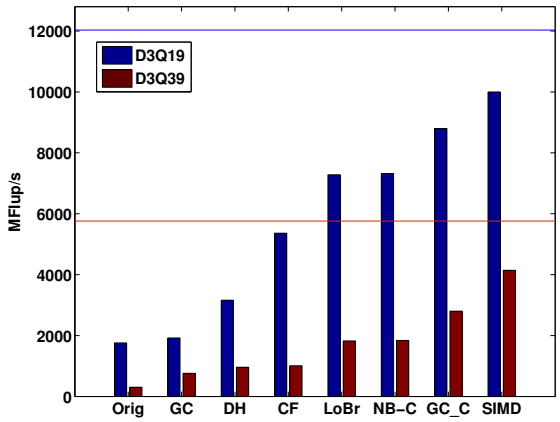
For D3Q39, it was slightly lower at 83% of the peak predicted performance value, likely due to the increased impact of the ghost cell implementation. In this case, 2 extra boundary rows are added around each processor boundary. The additional cost of these lattice updates are not accounted for and introduce a larger impact on the overall performance. The optimizations for this level with the largest impact were the compiler settings and the separate collide function for ghost cells. This is likely due to the extended number of ghost cells providing a more substantial option for communication/computation overlap. We will investigate impacts on communication overhead later in this section.

As for Blue Gene/Q, the largest impacts came from the compiler optimizations and the data handling. The intrinsics provided less of an impact, likely for two reasons. First, much of the performance gain was already achieved through the compiler and BG/Q gains a lot of its performance bump from the Quad Processing Extension (QPX) built-in functions. In the current implementation, we do not use the vector logical functions, leaving room for potentially further tuning. Optimal use of the cache and compiler optimizations proved the most fruitful optimizations. The max issue rate per core rose from 16.19% to 29.52%, meaning that each core is producing instructions at about 30% of the theoretical limit. This value is a good issue rate especially considering these results are from 128 nodes using 32 tasks per node with an unthreaded implementation. As shown in Fig. 8, the tuned version of the code approached the estimated performance maximum. For overall performance of the D3Q19 and D3Q39 models, we recovered 85% and 79% of the peak predicted performance. Again, these results are from a multi-node partition, introducing the degradation from intra-node communication.

Some of the optimizations encapsulated in Fig. 8 improved the load balance of the application and consequently the parallel nature of the application more than the performance as measured by MFlup/s of a single processor count. To gain insight into the impact of the communication tuning, we look at the time spent by the node spending the minimum, median, and maximum time in communication. This data, presented in Fig. 9, shows the communication balance of simply using non-blocking communication with solid lines. The blue lines refer to the D3Q19 model and the red to the D3Q39 model. The sharp slope of both lines indicates the strong load imbalance as one node spends as little as 4.8 seconds in communication while another spends



(a) Blue Gene/P Optimization Impacts.



(b) Blue Gene/Q Optimization Impacts.

Figure 8. MFlup/s achieved with each optimization enhancement on the two platforms in question. The horizontal lines represent the corresponding peak MFlup/s. In each case, 128 nodes were used.

40 seconds almost entirely in MPI_Waitall. The dash-dot lines represent the use of both non-blocking communication and ghost cells. The introduction of the ghost cells allows the data to be sent at the end of the time step instead of causing the collide function to wait for the results of the stream function of neighboring processors. While there is still limited overlap with computation, communication imbalance is reduced. Finally, the dashed lines show the improvement gained through the introduction of a separate collide function to calculate the ghost cell data. This function allows the sends to be posted before the ghost cell calculations. As the receives can be posted at the beginning of the time step, the latency of the message passing can be hidden by the time for computing the ghost cells. The communication imbalance is minimized to ranging from 3-5 seconds for the D3Q19

model for example, posing a significant improvement to the initial range of 4.8-40 seconds. Simulations conducted for a greater number of time steps and larger fluid system sizes saw roughly the same ratio to hold throughout.

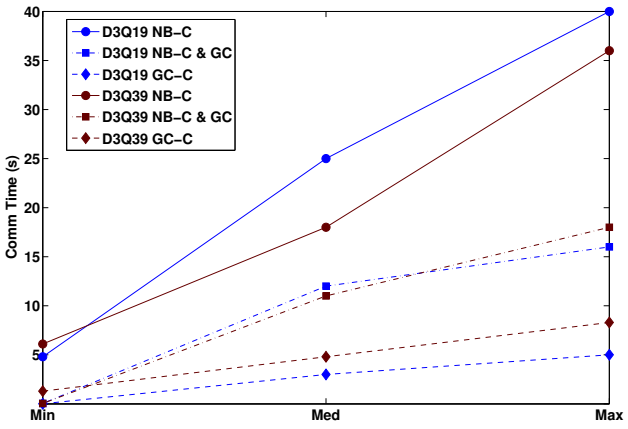
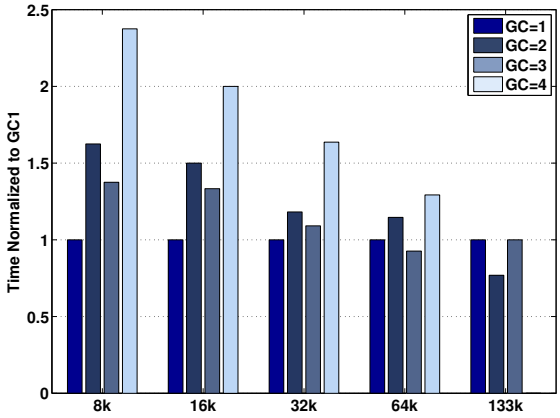


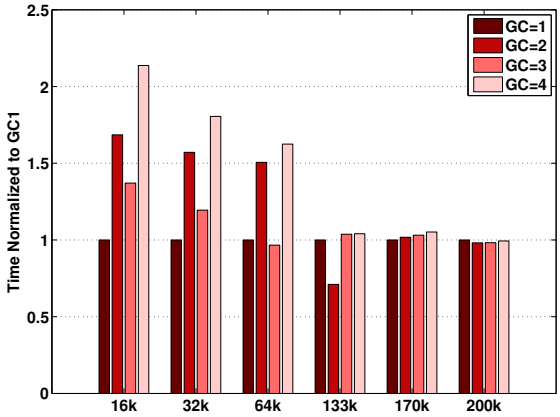
Figure 9. Time in seconds spent in communication for the processors that exhibited the minimum, median and maximum communication time at a range of optimization levels.

A. Deep Halo Ghost Cells

The use of deep halo ghost cells can further reduce message overhead by reducing the number of messages being sent. A greater number of ghost cells are retained on each processor, subsequently introducing a small computational cost, but messages are only exchanged every few time steps. The same amount of data is passed, but the reduction in number of messages allows for easier masking of the messaging latency. In order to assess the tradeoff between the communication gains and added computation, we simulated several different fluid system sizes for 300 time steps, enough so that the messaging tradeoff would have a visible impact on the runtime. For the D3Q19 model, 2048 processors on Blue Gene/P were used. The results given in Fig. 10 are normalized to the runtime for one ghost cell. GC refers to the ghost cell depth. Again, note that GC=1 for the D3Q39 model actually includes two extra lattice points in that direction as particles can move up to two points in a single time steps. The results highlight that at small population counts, the ghost cells have a higher impact on the surface/volume ratio and lead to typically longer runtimes. It is not until the larger sizes of 64,000 and 133,000 fluid nodes that a 2-ghost cell deep and 3 ghost cell deep implementation becomes optimal. The size indicates the size of the dimension being partitioned across processors. The other dimensions are held constant for the purpose of this study. For the 133,000 case, the individual nodes ran out of memory due to the addition of the fourth ghost cell and could not complete the simulation.



(a) D3Q19



(b) D3Q39

Figure 10. Results showing optimal ghost cell depth, GC, at a variety of fluid system sizes. The results for the D3Q19 model were obtained on 2048 processors of Blue Gene/P while the results for the D3Q39 were from 16 nodes on Blue Gene/Q run with 16 tasks and 1 thread per node. This difference was due to differences in memory constraints between the two models.

For the D3Q39 model, system sizes that fit into memory on BG/P were not large enough to overcome the added cost of computing an additional 2 lattice points in the direction of each neighbor for each ghost cell level, so 16 nodes on Blue Gene/Q were used with 16 tasks each and one thread. Fig. 10 shows the results for dimensions ranging from 16,000 to 200,000.

In both graphs of Fig. 10, deep levels of ghost cells are shown to be beneficial at various fluid sizes and can produce more efficient simulations. For example, with the D3Q19 model ghost cell=2 for 64k corresponds to a hardware efficiency of 27% and 43% efficient in the collide routine, achieving several percent higher than seen with other fluid

sizes.

For both Blue Gene/P and Blue Gene/Q, the number of ghost cells ideal for the D3Q19 model depends on the ratio of the dimension of the fluid system to the number of processors. The ideal ratios in Table III were consistent for both architectures, however, ratios beyond 66 per node were unable to be tested on either due to memory constraints. For systems with larger memory footprints, further lower bounds that require more ghost cells would likely be identified.

Table III
OPTIMAL GHOST CELL DEPTH FOR FLUID SIZE/PROCESSOR RATIOS IN THE D3Q19 LATTICE MODEL.

Lattice Points/Proc	Ghost Cell Depth
$R \leq 16$	1
$16 < R \leq 32$	3
$32 < R \leq 66$	2

On Blue Gene/P, the memory overhead associated with deep halo ghost cells of the D3Q39 model made it have no performance gain. On Blue Gene/Q, however, the deeper levels started to have an impact mimicking the results shown for the D3Q19 model. Again, the most efficient level did not simply increase linearly with the ratio as one might naively expect. Due to on-node memory restrictions, ratios beyond 800:1 were not able to be tested. At higher ratios, it is likely that even deeper ghost cell depths will be beneficial. At the maximum ratio tested here, the impact of 2 vs. 3 ghost cell layers was negligible.

Table IV
OPTIMAL GHOST CELL DEPTH FOR FLUID SIZE/PROCESSOR RATIOS IN THE D3Q39 LATTICE MODEL.

Lattice Points/Proc	Ghost Cell Depth
$R < 256$	1
$532 < R \leq 256$	3
$680 < R \leq 532$	2
$800 < R \leq 680$	2 or 3

B. Hybrid Implementation

We finally studied the role that a hybrid implementation could have for a LBM implementation that leverages a deep halo ghost cells pattern. In previous tuning studies conducted on Blue Gene/P, flat MPI and MPI/OpenMP programming models were shown to offer similar performance results for the LBM [21]. We found similar results as depicted in Fig. VI-B, however, the hybrid implementation allows us to both increase the size of the fluid system that can be simulated and reduce the number of ghost cells because it reduces the number of domains of interest that the problem is broken into, thus directly reducing the number of ghost cells used. Recall that for any ghost cell depth n, the number of ghost cells in a simulation is equal to the area of the cross sections of the number of domains multiplied by $2n$.

This tradeoff also provides the ability to model larger fluid systems on smaller processor counts. For the results presented here, we used the maximum ratio from the previous studies, fluid dimension of 66 lattice points per processor for the D3Q19 model and 800 lattice points per processor for the D3Q39 model.

The first set of test was conducted on 32 nodes of Blue Gene/P exploring the use of 1,2,3 and 4 threads compared to results modeling the same fluid system but maxing out the MPI rank count through use of virtual node mode or four MPI processes per node. The simulations were run for ghost cell ranges 1-4 with the smallest runtime at each level being displayed to show maximum performance. Second, 16 nodes on Blue Gene/Q were used with a range of task and thread combinations shown in Fig. VI-B.

As shown in Fig. 11, threading improves the performance of both models for both platforms. The minimal runtime for the D3Q19 model on Blue Gene/P is approximately the same for the hybrid model with 4-threads or the flat MPI model run in virtual node mode. This result is consistent with the previous group's results. The interesting point here, is that for the D3Q39 model, the hybrid model with 4-threads with two ghost cells actually outperforms the virtual node mode case. This improvement is due to the reduction in ghost cell overhead as the number of border cells is decreased. There is a much bigger impact on the D3Q39 model as it not only requires more memory bandwidth for the extended velocities but also has to take into account the two-speed nature of the model, resulting in twice as much overhead as seen in the D3Q19 model. In regards to Blue Gene/Q, the naive expectation was that the optimal setup would have at least one task per processor and between one and four threads per task. Due to the aforementioned benefit of ghost cell reduction through the shared memory optimization, the optimal pairing of tasks and threads for the higher order model is actually four tasks per node with 16 threads assigned, as shown in Fig. VI-B. This optimal pairing was true for both models.

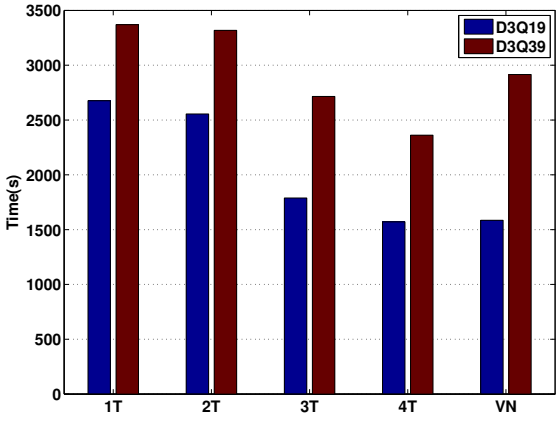
VII. CONCLUSION

Modeling fluid flows beyond the Navier-Stokes regime has been a long posed challenge. With the extension of the LBM to the D3Q39 discrete velocity model, flows at finite Kn are able to be accurately modeled and higher order kinetic moments recovered [5], allowing the accurate modeling of nanoscale flows such at those in the microvasculature or MEMS.

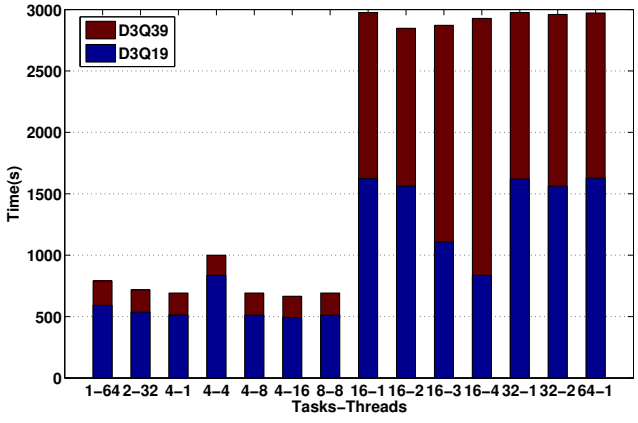
This extended model, however, introduces new computational challenges over previously studied LBM implementations. In this paper, we explored the performance impact of the extension and methods to reduce this impact. By

maximizing our data handling and streamlining the computation, we were able to produce results at 92% and 83% of our predicted upper bound on performance on Blue Gene/P for the two models, consistent with the 30% demonstrated hardware efficiency for D3Q19 and 21% of peak for D3Q39. For Blue Gene/Q, the production results were at 85% and 79% of the predicted performance maximums, confirming strong correlation with our simple performance projection. We exhibited an overall 3× improvement for Blue Gene/P and 7.5× improvement on Blue Gene/Q.

We showed that for both models, deeper levels of ghost cells proved beneficial as they minimized the overall number of messages being sent. The performance gain from deep level ghost cells, actually made the D3Q39 model with 4 threads outperform the single ghost cell implementation



(a) Blue Gene/P



(b) Blue Gene/Q

Figure 11. Impact of threading on both velocity model's performance. In each case here, the time of the minimal ghost cell implementation is shown.

maxing out flat-MPI ranks on Blue Gene/P. Similarly, on Blue Gene/Q, the use of deep level ghost cells alongside the hybrid programming model produced efficient simulations of the extreme condition fluid flows. We found that using a high level of threading per node resulted in maximal performance due to the ideal ratio of minimized communication due to ghost cell use and minimized added ghost cell overhead due to threading.

Finally, we demonstrated that the extended models are highly bandwidth limited, which poses limitations to the potential hardware efficiency when there is a greater imbalance between bandwidth and floating point capabilities. While we still achieve significant runtime reduction on the Blue Gene/Q architecture, investigation into methods to alter the algorithm as to reduce the memory accesses per lattice update could increase the potential hardware efficiency on such systems. The simulations of fluid at extreme conditions present a real world example of an application where focus needs to be on memory bandwidth improvement over increased flop rate. While the work presented in this paper offers demonstrated performance nearing the upper bound predicted for this platform, it simultaneously highlights the need for more methods to bridge the gap between bandwidth and floating-point performance limitations.

ACKNOWLEDGMENT

This work was supported in part by the Department of Energy's Computational Science Graduate Fellowship, grant number DOE CSGF DE-FG02-97ER25308, and in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy award DE-SC0004131. We would like to thank F. Mintzer and D. Singer for there help and support with use of the IBM Watson Blue Gene/P system and the LLNL Support Team for access to Blue Gene/Q. We also thank J.R. Hammond, T. Gamblin, A. Bhatele, M. Schulz, and T. Spelce for their assistance.

REFERENCES

[1] A. Peters, S. Melchionna, E. Kaxiras, J. Lätt, J. Sircar, M. Bernaschi, M. Bison, and S. Succi, "Multiscale simulation of cardiovascular flows on the IBM Blue Gene/P: Full heart-circulation system at red-blood cell resolution," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. IEEE Computer Society, 2010.

[2] A. Peters Randles, M. Baecher, H. Pfister, and E. Kaxiras, "A lattice Boltzmann simulation of hemodynamics in a patient-specific aortic coarctation model," in *Proceedings of Statistical Atlases and Computational Models of the Heart (STACOM) Computational Fluid Dynamics Challenge*, 2012.

[3] Q. Fan and H. Xue, "Compressible effects in microchannel flows [MEMS]," in *Electronics Packaging Technology Conference, 1998. Proceedings of 2nd*, dec 1998, pp. 224 –228.

[4] H. Chen, S. A. Orszag, and I. Staroselsky, "Macroscopic description of arbitrary Knudsen number flow using Boltzmann-BGK kinetic theory," *Journal of Fluid Mechanics*, vol. 574, pp. 495–505, 2007.

[5] R. Zhang, X. Shan, and H. Chen, "Efficient kinetic method for fluid simulation beyond the Navier-Stokes equation," *Physics Review E*, vol. 74, pp. 1–7, 2006.

[6] E. Arkilic, M. Schmidt, and K. Breuer, "Gaseous slip flow in long microchannels," *Microelectromechanical Systems, Journal of*, vol. 6, no. 2, pp. 167 –178, jun 1997.

[7] S. Colin, "Rarefaction and compressibility effects on steady and transient gas flows in microchannels," *Microfluidics and Nanofluidics*, vol. 1, pp. 268–279, 2005, 10.1007/s10404-004-0002-y. [Online]. Available: http://dx.doi.org/10.1007/s10404-004-0002-y

[8] E. J. Arlemark, S. K. Dadzie, and J. M. Reese, "An extension to the Navier–Stokes equations to incorporate gas molecular collisions with boundaries," *Journal of Heat Transfer*, vol. 132, no. 4, p. 041006, 2010. [Online]. Available: http://link.aip.org/link/?JHR/132/041006/1

[9] A. Sorger, M. Freitag, A. Shaporin, and J. Mehner, "CFD analysis of viscous losses in complex microsystems," in *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, march 2012, pp. 1 –4.

[10] K. Xu and Z. Li, "Microchannel flow in the slip regime: gas-kinetic BGK Burnett solutions," *Journal of Fluid Mechanics*, vol. 513, pp. 87–110, 2004.

[11] X. Shan, X. Yuan, and H. Chen, "Kinetic theory representation of hydrodynamics: a way beyond the Navier-Stokes equation," *Journal of Fluid Mechanics*, vol. 550, pp. 413–441, 2006.

[12] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001.

[13] P. Bhatnagar, E. Gross, and M. Krook, "A model for collision processes in gases," *Physics Review Letters*, vol. 94, p. 511, 1954.

[14] K. Suga, S. Takenaka, T. Kinjo, and S. Hyodo, "LBM and MD simulations of a flow in a nano-porous medium," in *Proceedings of the 2nd Asian Symposium on Computational Heat Transfer and Fluid Flow*, ser. ASCHT '09, 2009, pp. 112–117.

[15] IBM Blue Gene Team, "Overview of the IBM Blue Gene/P project." *IBM Journal of Research and Development*, vol. 52, pp. 1–2, 2008.

[16] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L.-T. Chiu, P. A. Boyle, N. H. Chist, and C. Kim, "The IBM Blue Gene/q compute chip," vol. 32. Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 48–60.

[17] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugaware, S. Kumar, V. Salapura, D. Satterfield, B. Burrow-Steinmacher, and J. Parker, "The ibm Blue Gene/q interconnection network and message unit," in *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. IEEE Computer Society, 2011.

[18] G. Wellein, T. Zeiser, G. Hager, and S. Donath, "On the single processor performance of simple lattice Boltzmann kernels," *Computers & Fluids*, vol. 35, no. 8-9, pp. 910 – 919, 2006, proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

[19] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, and T. Zeiser, "Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures," in *Proceedings of the 2004 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '04. IEEE Computer Society, 2004.

[20] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rde, "Optimization and profiling of the cache performance of parallel lattice Boltzmann codes in 2D and 3D," *Parallel Processing Letters*, vol. 13, p. 2003, 2003.

[21] S. Williams, L. Oliker, J. Carter, and J. Shalf, "Extracting ultra-scale lattice Boltzmann performance via hierarchical and distributed auto-tuning," in *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. IEEE Computer Society, 2011, pp. 1–10.

[22] F. Kjolstad and M. Snir, "Ghost Cell Pattern," in *2nd Annual Workshop on Parallel Programming Patterns, ParaPLoP'10*. Association for Computing Machinery (ACM), Mar. 2010.

[23] M. Gilge, "IBM system Blue Gene solution: Blue Gene/Q application development," http://www.redbooks.ibm.com/redpieces/pdfs/sg247948.pdf, 2012.