Moment Representation in the Lattice Boltzmann Method on Massively Parallel Hardware

Madhurima Vardhan
Department of Biomedical Engineering
Duke University
Durham, NC, USA
madhurima.vardhan@duke.edu

John Gounley
Computational Sciences and Engineering Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
gounleyjp@ornl.gov

Luiz Hegele
Department of Petroleum Engineering
Santa Catarina State University
Santa Catarina, Brazil
luiz.hegele@udesc.br

Erik W. Draeger
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, USA
draeger1@llnl.gov

Amanda Randles
Department of Biomedical Engineering
Duke University
Durham, NC, USA
amanda.randles@duke.edu

ABSTRACT

The widely-used lattice Boltzmann method (LBM) for computational fluid dynamics is highly scalable, but also significantly memory bandwidth-bound on current architectures. This paper presents a new regularized LBM implementation that reduces the memory footprint by only storing macroscopic, moment-based data. We show that the amount of data that must be stored in memory during a simulation is reduced by up to 47%. We also present a technique for cache-aware data re-utilization and show that optimizing cache utilization to limit data motion results in a similar improvement in time to solution. These new algorithms are implemented in the hemodynamics solver HARVEY and demonstrated using both idealized and realistic biological geometries. We develop a performance model for the moment representation algorithm and evaluate the performance on Summit.

KEYWORDS

lattice Boltzmann method, moment representation, memory, bandwidth

ACM Reference Format:
Madhurima Vardhan, John Gounley, Luiz Hegele, Erik W. Draeger, and Amanda Randles. 2019. Moment Representation in the Lattice Boltzmann Method on Massively Parallel Hardware. In *Proceedings of ACM Conference (SC'19).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SC'19, November, 2019, Denver, CO, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The lattice Boltzmann method has proven to be an efficient algorithm to computationally solve a variety of complex phenomena [30]. It is a popular alternative solver of the Navier-Stokes equations with a stencil-like structure that has proven highly amenable to large scale parallelization [4, 6, 23, 26, 35]. Unlike standard stencil operators, the data needed by LBM for each lattice point is different for each neighbor. Furthermore, relative to comparable approaches, LBM has large data storage costs and is highly memory-bound on current architectures [29].

The regularized lattice Boltzmann method was introduced by Latt in 2006 to improve the stability of LBM simulations [16, 17]. Regularization improves stability by retaining at each timestep only the distribution data required for a second-order approximation of the Navier-Stokes equations. By effectively truncating higher-order terms in LBM, unwanted oscillations and instabilities in the simulation can be suppressed. While the regularized LBM is necessarily less accurate at solving the lattice Boltzmann equation, LBM simulations for fluid dynamics are intended to solve the Navier-Stokes equations. For this latter purpose, regularization does not compromise accuracy and, indeed, has been demonstrated to improve accuracy in benchmark flows [16]. Interestingly, the second-order approximation of the Navier-Stokes equations only requires the first three sets of macroscopic moments (density, velocity, and the symmetric stress tensor), which in a 3-dimensional simulation corresponds to 10 double precision values per lattice site. This suggests remarkable potential for data reduction if one could store the macroscopic moments in main memory instead of the distribution data. In practice, the regularized LBM has thus far only been implemented by storing distribution data in the same manner as conventional lattice Boltzmann. To the best of our knowledge, the only attempt to store the moments instead of distribution data was by Ref. [1], but their work in fact predates the development of regularized LBM and suffers from compromised physics.

In this study, we introduce an implementation of the lattice Boltzmann method based on regularization that stores only moment-based data. Relative to current approaches, this reduces storage of simulation data by 47% relative to single distribution methods [2]. Further, we borrow ideas from nested loop optimization to introduce cache-awareness and minimize data motion [25]. This technique reduces the read and write cost per lattice site to 10 double precision values, which is also a 47% improvement on the state-of-the-art. Further, both reads and writes are well-coalesced, rendering architecture-tuned data layout schemes less important. Our approach is implemented in the hemodynamics code HARVEY [28], which is designed for simulations in complex vascular geometries. We provide details for both node-level performance and inter-node scalability of the proposed algorithm.

2 RELATED WORK

The large memory storage cost and bandwidth limitation of the LBM are primarily due to the need to store LBM particle distribution data for all stencil components at each lattice site. In a naive 3-dimensional simulation using a standard D3Q19 lattice (illustrated in Fig. 1), two copies of the particle distribution data are required, corresponding to 38 double precision values per lattice site. Over the past two decades, a series of improved LBM algorithms (AA, swap, shift, esoteric twist, etc.) have been developed to reduce this cost by 50%, to a single copy of the distribution data, with 19 values per lattice site [2, 11, 21, 25]. A similar story prevails for bandwidth: in naive implementations, the distribution data must be fully read from and written to main memory twice, with 38 double precision values being read and written per lattice site. By 'fusing' the kernels that comprise LBM, this cost can be reduced to a single read and write, with 19 double precision values per lattice site. However, bandwidth challenges are exacerbated due to the memory access pattern of LBM itself: either reads from or writes to main memory for the distribution function can be coalesced but not both.

In spite of these advances, optimization of LBM with respect to storage and bandwith remains an active area of research [29]. A series of blocking and tiling schemes have been developed to improve cache performance for LBM and similar stencil computations [8, 25, 33–35]. More recently, architecture-dependent modifications to data layout and addressing schemes have been used to successfully optimize data access costs [15, 38].

However, these current approaches do not directly address the fundamental algorithmic limitation regarding the amount of data to be stored and accessed. One avenue to reduce this cost is spatial and temporal wavefront blocking [10, 19, 32, 39], in which the data motion for multiple timesteps can be combined. These approaches offer strong performance but may be challenging to implement in coupled codes or complex geometries. Conversely, alternative LBM algorithms have been developed which store and access macroscopic fluid data (density and velocity) instead of the distribution data [20, 31]. This improvement comes by limiting the physics described by these methods, as they can only model laminar flows. As LBM's performance in the transition and turbulent regimes is among the method's advantages, these alternative methods do not represent a satisfactory general solution.

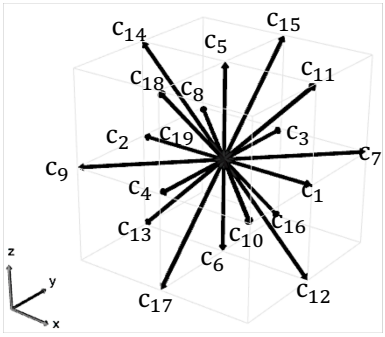

Figure 1: The D3Q19 lattice is a symmetric velocity discretization with the first 18 components $\mathbf{c}_i$ corresponding to nearest neighbor lattice points and component $\mathbf{c}_{19}$ for stationary (zero) velocity.

3 LATTICE BOLTZMANN METHODS

3.1 Distribution representation

The lattice Boltzmann method is characterized by a distribution function f of fictitious particles which move about a fixed Cartesian lattice [7]. Each component $f_i(\mathbf{x}, t)$ represents the population of particles located at lattice point $\mathbf{x}$ at time t with discrete velocity $\mathbf{c}_i$. For LBM simulations in three dimensions, the most common velocity discretization is the D3Q19 lattice represented in Figure 1. The macroscopic variables density ρ and momentum $\rho\mathbf{u}$ are computed as the first two moments of the distribution function:

$$\rho = \sum_{i=1}^{19} f_i(\mathbf{x}, t) \qquad \rho\mathbf{u} = \sum_{i=1}^{19} \mathbf{c}_i f_i(\mathbf{x}, t). \tag{1}$$

From these two moments, the equilibrium Maxwell-Boltzmann distribution $f^{eq}(\rho, \mathbf{u})$ is approximated as

$$f_i^{eq}(\rho, \mathbf{u}) = \omega_i \rho \Big(1 + \frac{\mathbf{c}_i \cdot \mathbf{u}}{c_s^2} + \frac{\mathbf{u}\mathbf{u} : \boldsymbol{Q}_i}{2c_s^4}\Big) \tag{2}$$

for D3Q19 lattice weights ω_i, lattice speed of sound $c_s^2 = \frac{1}{3}$, and the tensor $\boldsymbol{Q}_i = \mathbf{c}_i\mathbf{c}_i - c_s^2\mathbf{I}$. The evolution of the distribution function is expressed by the lattice Boltzmann equation,

$$f_i(\mathbf{x} + \mathbf{c}_i, t + 1) = (1 - \frac{1}{\tau}) f_i(\mathbf{x}, t) + \frac{1}{\tau} f_i^{eq}(\rho, \mathbf{u}), \tag{3}$$

with the Bhatnager-Gross-Krook (BGK) collision kernel [5] and relaxation time-scale τ.

Several different propogation patterns exist for the distribution representation of LBM. Most generally, the order of collision and streaming can be interchanged in the 'push' and 'pull' patterns, in order to optimize memory access for collision and streaming, respectively [38]. Nonetheless, we can broadly describe the spirit of the algorithm with fused collision and streaming kernels in the following pseudocode:

for $\mathbf{x} := 1$ to N do
  Read distribution $f(\mathbf{x}, t)$
  Compute ρ, $\mathbf{u}$
  for $i := 1$ to 19 do
    Compute $f_i^{eq}(\rho, \mathbf{u})$
    Collision: $f_i^* = f_i(\mathbf{x}, t) + \frac{1}{\tau}\left(f_i^{eq}(\rho, \mathbf{u}) - f_i(\mathbf{x}, t)\right)$
    Streaming: $f_i(\mathbf{x} + \mathbf{c}_i, t + 1) = f_i^*$

3.2 Moment representation

The regularized lattice Boltzmann method (RLBM) was introduced to reduce the number of degrees of freedom required for a second-order approximation of the Navier-Stokes equations [17]. The minimum number of degrees of freedom consist of the first three moments of the LBM distribution: density ρ, velocity $\mathbf{u}$, and the symmetric stress tensor $\mathbf{\Pi}$. Analogous to the first two quantities, the stress tensor is computed by the equation

$$\mathbf{\Pi} = \sum_{i=1}^{19} \boldsymbol{Q}_i f_i(\mathbf{x}, t). \tag{4}$$

The regularization procedure modifies the standard distribution-based LBM by producing a regularized distribution $\hat{f}$ prior to collision,

$$\hat{f} = f^{eq}(\rho, \mathbf{u}) + \frac{\omega_i}{2c_s^2}\boldsymbol{Q}_i : (\mathbf{\Pi} - \mathbf{\Pi}^{eq}) \tag{5}$$

based on the non-equilibrium component of the stress tensor $\mathbf{\Pi}$. From this regularized distribution, the collision and streaming operations proceed as in the standard lattice Boltzmann method. Without compromising accuracy, RLBM has proven to greatly enhance stability for both bulk flow [16] and boundary conditions [14, 18].

To the best of our knowledge, all existing implementations of RLBM simply add a regularization step to standard distribution representation. However, Latt observed that, since the full simulation state is defined by the first three sets of moments, an untapped potential exists to further reduce memory by storing the moments $\{\rho, \mathbf{u}, \mathbf{\Pi}\}$ instead of distribution function f [16]. The number of required moments varies with the simulation dimension d as $1+d+\frac{d(d+1)}{2}$; for D3Q19, each lattice site would have 10 moments instead of the 19 distribution components. As the streaming operation is only defined in terms of distribution components, a moment representation of RLBM must convert between moments and the corresponding distribution components. Such conversions occur in some LBM variants, such as the MRT collision kernel, which performs collision in moment space and maps back to distribution space for streaming [9]. The proposed moment representation of the LBM algorithm in this vein can be described by the following pseudocode:

for $\mathbf{x} := 1$ to N do
  Read moments $\rho(\mathbf{x}, t)$, $\mathbf{u}(\mathbf{x}, t)$, $\mathbf{\Pi}(\mathbf{x}, t)$
  Compute equilibrium moments $\mathbf{\Pi}^{eq}(\rho, \mathbf{u}, \mathbf{\Pi})$
  Collision $\mathbf{\Pi}^* = (1 - \frac{1}{\tau})\mathbf{\Pi} + \frac{1}{\tau}\mathbf{\Pi}^{eq}$
  for $i := 1$ to 19 do
    Compute distribution $f_i^* = \omega_i \rho\left(1 + c_s^2(\mathbf{u} \cdot \mathbf{c}_i) + c_s^4 \boldsymbol{Q}_i : \Pi\right)$
    Streaming $f_i(\mathbf{x} + \mathbf{c}_i, t + 1) = f_i^*$
  od
od
Compute moments via Equations 1, 2 and 4

This moment representation presents interesting differences from the distribution representation. The equilibrium computation and collision operation are performed on only 6 moments instead of 19 distribution components. This advantage helps balance the additional arithmetic that occurs when converting to and from moments and distributions. However, whether the moment representation reduces storage requirements or bandwidth limitations relative to the distribution representation depends on the propagation pattern in which the representation is implemented.

4 PROPAGATION PATTERNS

4.1 Distribution representations

In its simplest form, the distribution representation of LBM is implemented with two copies of the particle distribution data and with separate collision and streaming kernels. This configuration, which we denote 'AB2k', allows the two kernels to be optimized individually but effectively precludes any data reuse between them. One alternative to improve performance is to maintain the two copies of the particle distribution data, but to 'fuse' collision and streaming into a single kernel [25, 33]. However, while this alternative method ('AB1k') allows for data reuse between collision and streaming, it does not reduce overall storage costs. For a task with N fluid points, the storage requirement for AB2k and AB1k is

$$M = 38 * N \text{ doubles} + 19 * N \text{ integers} \tag{6}$$

for the two distribution copies and the indirect addressing adjacency list. Newer single kernel propagation patterns like AA, swap, shift, and esoteric twist retain only a single copy of the particle distribution data [2, 11, 21, 24]. In this study, we focus on the AA method, which reduces the memory storage for the distribution data by 50%:

$$M = 19 * N \text{ doubles} + 19 * N \text{ integers}. \tag{7}$$

Performance comparisons also demonstrate that single distribution schemes like AA can deliver superior performance on CPUs versus two distribution propagation patterns [12, 38]. The literature regarding optimization and scaling of distribution-based propagation patterns is too substantial to review here; we refer the reader to [13, 29, 37].

4.2 Layer scheme for moment representation

There are two closely related challenges to designing a propagation pattern for the moment representation of LBM. First, in order for the method to save memory with respect to a single distribution scheme, only a single copy of the moment data can be stored over the simulation domain. Second, moments at a lattice site cannot be recomputed until distribution components have streamed in from all adjacent lattice sites, necessitating the temporary storage of at least some distribution data during streaming. To address these two design requirements – a single copy of the moment data and the need for some temporary distribution data storage – we adapt the layer scheme from [1], which features a temporary buffer in which

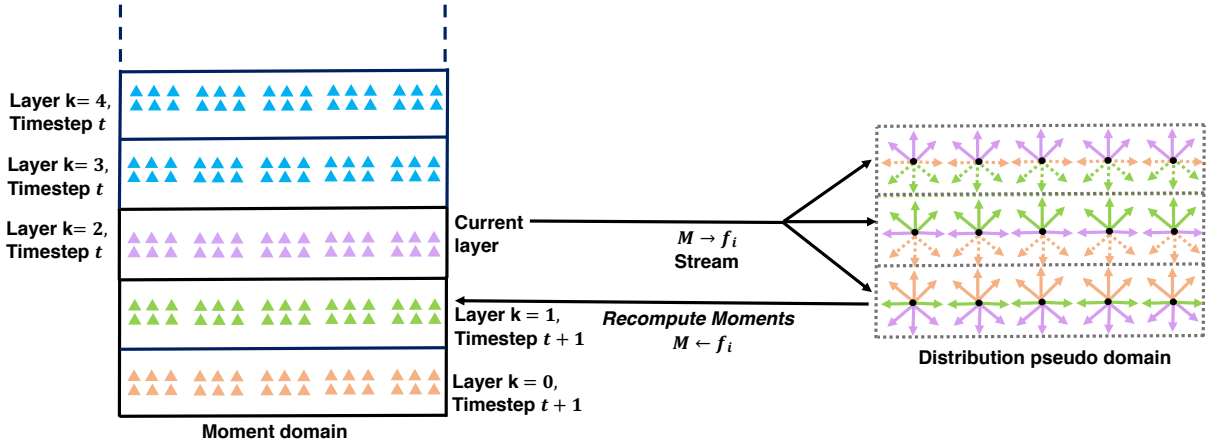

Figure 2: Moment representation algorithm: The moment domain and distribution pseudo domain are shown on the left and right sides of the figure, respectively. We illustrate the case of layer $k = 2$ at timestep t. The distributions computed from post-collision moments in layers 0 and 1 have been streamed to the pseudo domain. Dashed velocity lines contain information that was previously used to compute moments for layer $k = 0$ at timestep $t + 1$ and is no longer needed. Next, the post-collision moments in the current plane $k = 2$ are used to compute distributions ($M \rightarrow f_i$) and streamed to the 3 layers in the pseudo domain. Now, the bottom row in the pseudo domain now has the complete set of 19 distributions required to recompute new moments ($f_i \rightarrow M$). These recomputed moments are then written to layer $k = 1$ at the timestep $t + 1$. The cycle then continues, with collsion being performed on layer $k = 3$ at timestep t, converted to distributions, and streamed to the pseudodomain.

post-streaming distribution data can be stored until moments are evaluated.

We assume that the domain of each MPI task belongs to a single cuboid bounding box or, more generally, a series of such bounding boxes. Each bounding box is divided along one axis into a series of layers, which are a single lattice site high, as illustrated in figure 2. While it would be optimal to select this axis based on the dimensions of the bounding box, our current implementation is fixed to always use the z-axis. Starting from the bottom and looping over the fluid points in each layer of the moments domain, we perform the collision operation, convert the resulting moments into distributions, and stream the distribution components into the appropriate places in the distribution pseudo domain. After the layer k has been streamed, we apply boundary conditions in the distribution pseudo domain to layer $k - 1$, and convert the layer back to the moments domain. As the D3Q19 lattice only requires nearest neighbors, this layer-based propagation pattern can be implemented with only 3 layers in the distribution pseudo domain.

When considering sparse geometries, such as in human vasculature, layers may be sparsely populated by fluid points [26]. We take such cases into account by indirectly addressing the distribution pseudo domain. For a given task, the resulting propagation pattern for the layer scheme requires storage of

$$M = (10 * N + 3 * 19 * P) \text{ doubles} + 19 * N \text{ integers} \quad (8)$$

where P is the number of fluid points in the most populous layer. If $P \ll N$, the amount of storage for the moment representation is approximately 47% of a single copy of the distribution function. Moreover, required memory for the moment representation becomes similar in size to the adjacency list for indirect addressing.

4.3 Cache aware layer scheme

To improve data locality relative to single kernel propagation patterns for the distribution representation, it is necessary to minimize any access to main memory beyond the essential read and write of the moments array each timestep. In particular, this involves optimizing the cache locality and data reuse for the 3 temporary distribution-based layers. As observed by [1], a layer scheme immediately obtains this locality and data reuse for a sufficiently small problem size. In particular, we expect optimal cache performance when the entire distribution pseudo domain fits within the last level cache. In this section, we present a modification of the layer scheme for the moment representation in section 4.2 which maintains this advantage for general problem sizes per task via cache awareness.

Our strategy builds on a long history of optimization for LBM and related stencil computation for blocking and tiling schemes [8, 25, 33–35] and spatial and temporal wavefront blocking [32, 39]. We introduce a tiling scheme that divides the task bounding box into 'blocks', in a second dimension orthogonal to the previously defined layer division. Blocks are divided such that the number of fluid points in any layer of the block does not exceed a specified threshold, selected based on the size of the CPU L3 cache. As shown in figure 4, the layer algorithm from the previous section is performed successively on each block. Therefore, this limitation on the layer size breaks the full task bounding box into a series of smaller blocks such that the temporary layer size can be modulated based on the available space in cache.

Similar to the streaming and communication patterns in traditional LBM, we observe that a non-trivial interaction occurs at the boundaries between blocks. As distribution components stream to their nearest neighbors on the lattice, a distribution component can

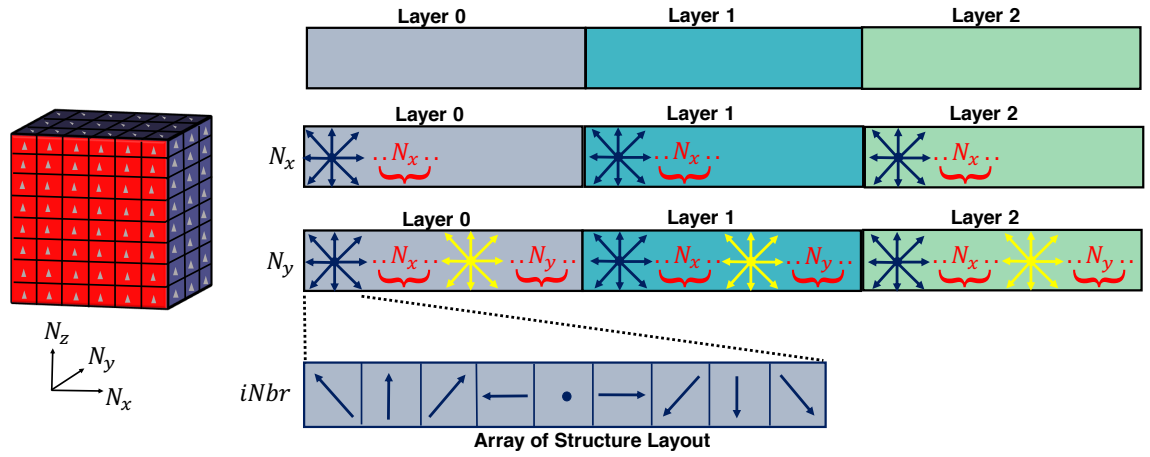

Figure 3: Pseudo domain setup: the three layers in the pseudo domain are populated with the distributions streamed from each moment layer in the array of structure format.

stream from one block to the next ('forward', block A to block B) and from a given block to the previous block ('backward', block B to block A).

For the lattice points in block B which have distribution components that will stream 'backwards' into the previous block A, we perform an additional collision operation on these lattice points when updating the previous block A. This process is illustrated in figure 4 (b). We then perform the streaming operation for only those distribution components which stream backwards. Consequently, some recomputation necessarily occurs for backwards streaming; this amount is proportional to the cross-sectional area of the block interface, which is generally small relative to the full bounding box volume.

A different solution is required for distribution components which stream 'forward', as the update to the moments in the previous block A precludes the possibility of recomputation when evaluating the subsequent block B. Instead, as depicted in figure 4 (b), we identify the distribution components in each layer of block A which would stream forward into block B and write them to a second temporary buffer. When the layer into which the distribution components are to stream is encountered in block B, we read the values from the second temporary buffer and write them to the appropriate locations in the temporary layer distribution array. As a result, additional data motion is required for this distribution components and, once again, the quantity of additional data motion is proportional to the block interface's cross-sectional area.

We observe that, in general, the previous division of a non-layer dimension of the geometry into blocks is not optimal from the standpoint of minimizing recomputation and additional data motion. Dividing the two non-layer dimensions into 'tiles' would allow for the optimal case, which could provide a significant improvement for approximately cubic bounding boxes. We opted for the single division into blocks for the sake of simplicity, as the 'tile' version would significantly increase the complexity of the secondary temporary buffers for the 'forward' streaming across block boundaries.

5 EXPERIMENTAL SETUP

5.1 HARVEY

The algorithms discussed in the previous sections for distribution and moment representations of LBM are implemented in HARVEY, a highly parallel hemodynamics solver [28]. Parallelized with MPI and OpenMP, HARVEY is designed for simulations in complex vascular geometries [27]. The sparsity of vascular geometries presents challenges for load balance, grid generation, and addressing. Our approach to the first two challenges does not differ between distribution-based and moments-based LBM schemes and implementation details are discussed in previous work [26]. For addressing, we use indirect addressing with an adjacency list; while the size of the adjacency list – 19 integers per fluid lattice point – does not vary for distribution-based and moments-based LBM schemes, the content of the adjacency list does change. For distribution representation schemes, we store position in the distribution array to which the distribution component would stream. However, for moment representation schemes, the adjacency list stores the position in the layer array to which the distribution component would stream.

A no-slip condition is enforced at the walls of the geometry using the halfway bounce-back method. This boundary condition can be embedded into the adjacency list, thereby avoiding a separate kernel. More complex boundary conditions, such as those required at inlets or outlets, are enforced in this implementation with a Zou-He condition for velocity or pressure [40]. We impose the boundary conditions while the points are still stored in the distribution pseudo domain, after collision and streaming but prior to recomputing moments.

Communication is performed over a halo region surrounding each task's bounding box. The complete set of 19 distribution components or 10 moments is communicated at each lattice point in the halo for distribution- and moment-based schemes, respectively. While halo-free schemes (e.g., [15]) can reduce communication for distribution-based schemes, having full knowledge of data on the

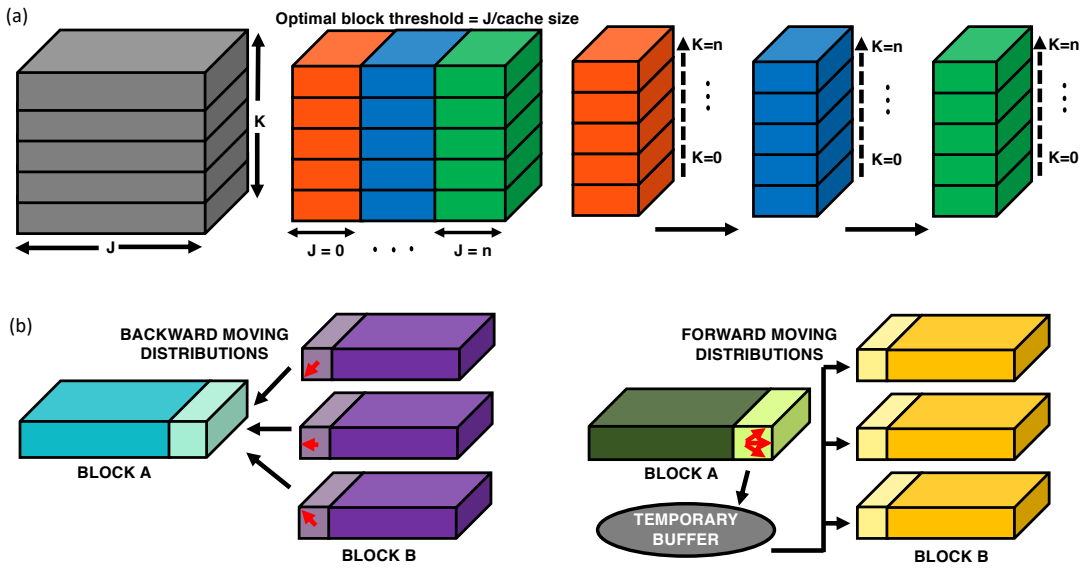

Figure 4: Diagram of the cache aware layer scheme. (a) Initial division into layers (far left, gray), the task bounding box is also divided into blocks uniformly across all layers (center left, colored by block). In the simulation, we update each block successively, with collision and streaming being performed layerwise before moving to the next block (right). (b) Backwards streaming from block B to block A is performed during the update to block A (left). Forward streaming from block A to block B is performed in two parts: writing the distribution components streaming forward to a temporary buffer while updating block A and reading these components from the buffer while updating block B.

halo facilitates finite-difference schemes which can be used to apply boundary conditions and compute time-averaged quantities like wall shear stress.

Local fluid points are sorted in two ways to improve performance. First, to facilitate communication, the distribution and moment arrays are sorted to group sent points and received points to contiguous array locations. Second, interior points which are not communicated are sorted into contiguous arrays for the distribution representation schemes and into contiguous layers of tiles for the moment representation schemes. As the moment representation scheme loops over all fluid points in the layer (or the portion of the layer within the block, for the cache-optimized version), we observe that additional lookup is required to identify the members of the layer for those points sorted for communication.

Validation and convergence tests for a standard set of benchmark flow problems were conducted to ensure accuracy. Results for simulations with the regularized LBM method matched for the distribution and moment representation schemes up to the order of round-off errors. Additional validation of the regularized LBM implementation in HARVEY was conducted in [14] with experimental results for lid-driven cavity flow.

5.2 Architectures

On-node performance and scaling assessments were performed on an Intel-based cluster, with dual-socket compute nodes connected with 56Gb/s Infiniband interconnect. Each node has two Intel Xeon E5-2695V4 'Broadwell' processors with 40 CPU cores exposed to the job scheduler and a shared 55 MB L3 cache. Our application was compiled with the 2018 versions of the Intel C++ compiler and MPI library.

Additional scaling tests were performed on the Summit supercomputer at the Oak Ridge Leadership Computing Facility. Summit has 4608 IBM Power System AC922 nodes connected by dual-rail EDR InfiniBand arranged in a non-blocking Fat Tree topology. Each dual-socket Summit node has two IBM POWER9 processors and 6 NVIDIA GPUs, although this study was conducted solely on the CPU component of the nodes. Each node has 42 CPU cores exposed to the job scheduler. Our application was compiled with version 16.01 of the IBM XL C++ compiler and Spectrum MPI library.

5.3 Test geometries

To demonstrate the scaling behaviour and real world application, we used two test geometries representing different degree of spatial complexity: a cylinder and a patient-derived coronary artery model. Representative of an idealized blood vessel, the cylinder geometry was oriented along the x and z axes to assess the effect of orientation on performance at a 5 μm on Intel Xeon Broadwell processors. The same geometry at 5 μm resolution was also used to investigate the performance comparison of distribution function vs. moment-based propagation patterns. To understand large scale parallel behavior, higher resolution simulations were conducted using the cylinder geometry at 5 μm, 2.5 μm and 1.25 μm. The coronary artery geometry was derived from CT angiographic data, segmented using commercial software and obtained from appropriate institutional

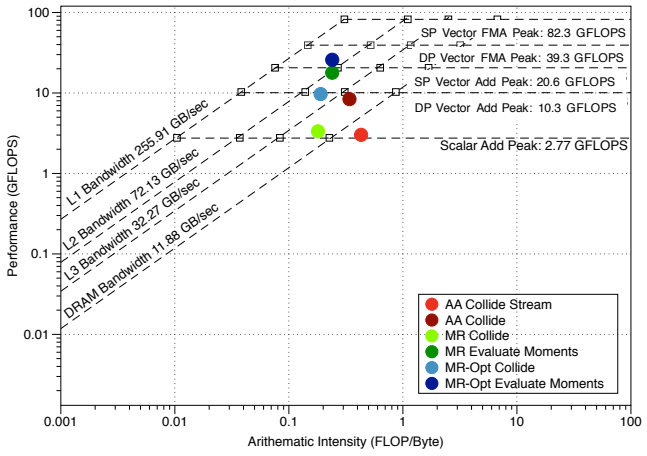

Figure 5: Roofline performance model for AA, moment representation (MR), and cache-optimized moment representation (MR-opt) schemes on the Intel 'Broadwell' architecture. For each scheme, we consider the two most significant kernels. For the AA scheme, these are the collision and streaming kernels for the even and odd timesteps ('Collide' and 'Collide Stream', respectively). For the moment representation schemes, these are the collision and streaming kernel and the moment evaluation ('Collide' and 'Evaluate moments', respectively).

review board approval. The arterial simulation was performed at 50 μm resolution to demonstrate physical hemodynamic blood flow.

5.4 Performance model

We assess node-level performance on the Intel 'Broadwell' architecture for the baseline and cache-optimized propagation patterns for the moment representation using a roofline performance model [36]. Given the well-documented bandwidth limitations of the algorithm, we judged that a roofline model would be the most comprehensive measure of kernel performance between LBM propogation patterns. Simulations were conducted using 40 MPI ranks per node and 2 OpenMP threads per rank. Constructed with Intel Advisor, the performance for a representative rank is shown in Figure 5. We focus on the two most important kernels in these propagation patterns: (1) collision and streaming and (2) evaluating moments.

In collision and streaming, we are reading 10 consecutively stored moments from memory at each lattice site, performing the collision operation, converting the resulting moments into distributions, and streaming the distributions into the distribution pseudo domain. For the final two components of this kernel, the loop over distribution components can be efficiently vectorized. Despite the conversion from moments to distributions, the arithmetic intensity is quite low and the performance for the 'MR Collide' kernel is only marginally better than the expected DRAM bandwidth. This is expected since the distribution pseudo domain does not fit within cache and strides between writes to the pseudo domain are highly variable due to the indirect addressing. Without modifying the vectorization, we observe that the arithmetic intensity remains the same for the cache optimized 'MR-Opt Collide' kernel but bandwidth improves to between L2 and L3 cache levels.

The kernel in which moments are evaluated is much simpler: read 19 contiguous memory locations from the distribution pseudo domain, convert to moments, and write moments back to main memory. The only computational component – the moment conversion – vectorizes well and the predictable memory strides improve bandwidth performance. We observe performance at the L2 bandwidth for the baseline moment representation scheme and the cache-optimized version slightly exceeds on this threshold.

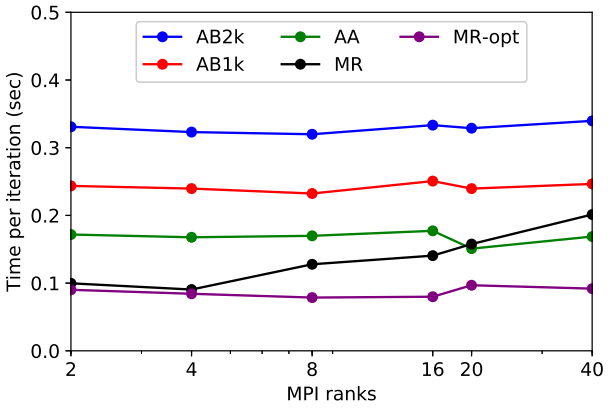

Figure 6: Performance comparison for MPI+OpenMP configurations on a Broadwell node. The 80 hardware thread are split evenly among the number of MPI ranks indicated.

To investigate the percent of peak performance achieved by the new method, we use the STREAM benchmark, a commonly used method for measuring memory bandwidth in HPC systems, to identify peak performance [22]. STREAM comprises of 4 basic functions: Copy, Scale, Add and Triad. Each of the functions perform 0, 1 or 2 arithmetic operations and access 1 or 2 input arrays and write one output array. STREAM provides a direct correlation to memory bandwidth and is therefore a commonly used tool to assess the performance of complex HPC workloads which are bandwidth limited [22]. We use the Copy function to assess the LBM collide and stream kernel because of the similar read and write operations. The peak performance with 80 threads for the Copy function of STREAM on the Intel 'Broadwell' is measured at 49.459 GB/s. Using this metric as the theoretical peak performance, we calculate the expected performance for the cache-optimized moment, AA collision and baseline moment kernels to be 8.9 GFLOPs, 16.8 GFLOPs and 8.9 GFLOPs, at the measured arithmetic intensity of 0.18 flops/byte, 0.34 flops/byte and 0.18 flops/byte respectively 5). The measured performance is higher at 9.7 GFLOPs for the cache-optimized collide kernel and lower at 8.39 GFLOPs for AA and 3.34 GFLOPs for baseline moment (Figure 5). This finding underscores that by fitting distribution pseudo domain on cache, the anticipated performance gains can be achieved with the cache optimized moment implementation.

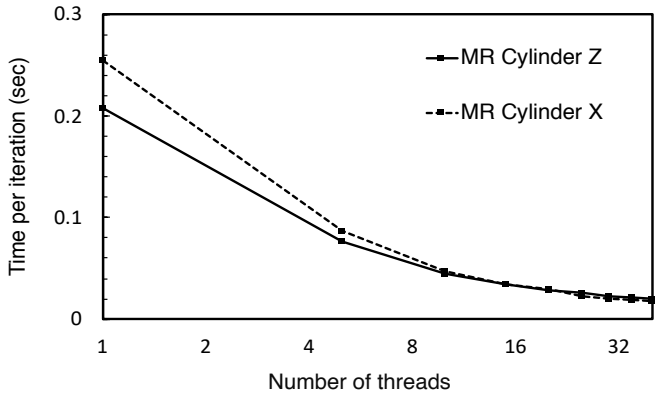

Figure 7: Effect of orientation for on-node performance with cylindrical geometries oriented the x- and z-axes for the baseline moment representation scheme for scaling from 1 to 40 OpenMP threads.

6 EXPERIMENTAL RESULTS

6.1 Single-node performance

An on-node performance comparison for the distribution- and moment-based schemes is shown in Figure 6. The time-per-iteration is computed from the maximum time spent by any task in the LBM time stepping loop. We observe that the AA scheme outperforms the two distribution AB1k and AB2k schemes and, more broadly, that performance is relatively consistent for the various MPI+OpenMP configurations. In contrast, the MR demonstrates improved performance for configurations with more threads per rank, at which it is significantly faster than AA. This difference in performance results from the relationship between the memory required for the temporary layers on each rank versus the available cache space. Due to the domain decomposition, the temporary layers easily fit within cache for fewer MPI ranks. However, for domain decompositions with more ranks, the size of the temporary layers exceeds the cache size and, accordingly, requires access to main memory. By performing its own decomposition of each task's bounding box, the cache-optimized version of the MR scheme allows for improved performance to be maintained for all MPI+OpenMP configurations tested.

6.2 Isotropy

Although the proposed layer scheme for the moment representation would ideally use the longest axis for the layer division, because our initial implementation decomposes along the z-axis, we take this opportunity to measure the impact of orientation on performance. While this dependence is largely ameliorated in the cache-optimized scheme or by domain decomposition for large MPI runs, it may be significant for pure OpenMP performance on node. To assess any potential anisotropic behavior of the baseline moment representation algorithm, we use the same cylindrical geometry and resolution from section 6.1 in two orientations: aligned with the axis along which layers are divided (z-axis) and orthogonal to this axis (x-axis). As can be noted in Figure 7, both orientations result in comparable performance for scaling from 1 to 40 OpenMP threads on a single rank. We do observe slight performance degradation for the cylinder oriented along x-axis, which was the sub-optimal direction since the implemented layer divisions were along the z-axis.

6.3 Inter-node performance

In this section, we extend the comparison between the distribution and moment representation schemes to inter-node performance. Simulations are conducted in a cylinder at 5μm resolution which is aligned with the z-axis. We assess strong scaling from 128 to 2048 MPI tasks, with up to 40 tasks per node. Relative performance for the distribution representation schemes is consistent with on-node performance. However, while AB1k and AB2k scale very well, AA performs poorly at large task counts due to increased communication costs. MPI communication for AA is challenging because different data is transfered on even and odd timesteps. The data layout in our implementation, which is the same for all three distribution representation schemes, does not fit well with the AA communication pattern on odd timesteps.

For the baseline moment representation scheme, we observe super-linear strong scaling. As the number of tasks increases, the size of the distribution pseudo domain becomes smaller relative to the available cache per core and performance improves accordingly. As a result, the baseline moment representation nearly matches the performance of an optimally-scaled AA scheme at 2048 tasks, despite a much slower runtime at 128 tasks. However, as the cache-optimized version of the moment representation controls for this effect, we observe that this scheme outperforms over the entire range of task counts studied. As the problem size per task becomes small, we observe the expected convergence of the cache-optimized version to the baseline moment representation. Nonetheless, scalability does not match that of AB1k or AB2k. We expect that the domain decomposition at these task counts produces non-optimal bounding boxes for our layer scheme and plan to address this in figure work.

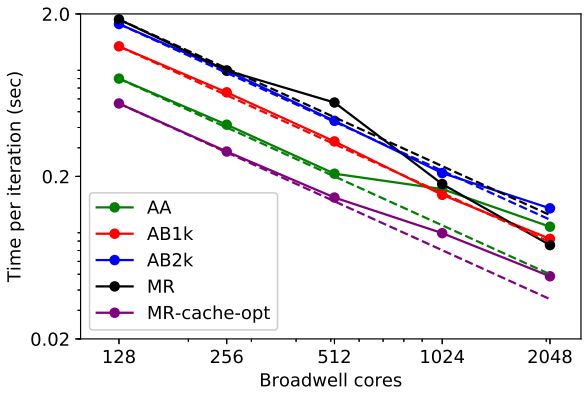

Figure 8: Inter-node performance for strong scaling in a cylindrical geometry on a cluster with Intel Broadwell CPUs.

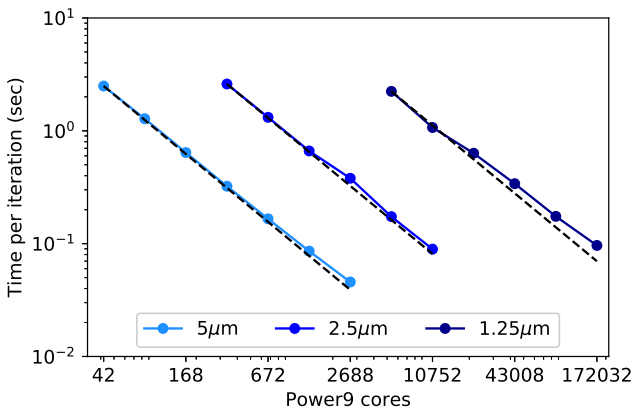

Figure 9: Strong scaling on Summit's Power9 CPUs for flow in cylindrical geometries at three resolutions.

6.4 Performance on large-scale architectures

While LBM implementations using a distribution representation typically have excellent scalability, it is necessary to confirm that this still holds for the moment representation scheme. In particular, the additional complexity and overhead of the cache-optimized moment representation scheme pose a particular challenge for strong scalabilty on large systems. To this end, large-scale strong and weak scaling tests were conducted on Summit for the cache-optimized MR scheme. Shown in Figure 9, we observe good strong scaling for runs in a series of cylindrical geometries. We observed parallel efficiencies of 90.8% for strong scaling from 1 to 32 and from 8 to 256 nodes (42 to 1344 and 336 to 10752 MPI ranks, respectively). Scaling performance degraded at larger node counts, with 72.5% parallel efficiency from 128 to 4096 nodes (5376 to 172032 MPI ranks), as load balance begins to degrade at these task counts.

Similarly, weak scaling results from Summit are shown in Figure 10. As our scheme does not substantially alter the normal LBM communication requirements, excellent weak scalability is to be expected. For three cylindrical geometries, we observe parallel efficiencies of 92-95% from 1 to 64 nodes (42 to 2688 MPI ranks). At higher node counts, we encounter substantial load imbalance between task which significantly reduce weak scaling performance. We are currently working to identify the cause but do not believe the cache-optimized moment representation to be the source.

6.5 Real world application - Patient-specific hemodynamic simulation

This performance results discussed above present the critical and necessary first step to demonstrate the efficient scaling behavior of our memory and cache optimized layer-based moment representation algorithm. In this section, we applied our moment representation algorithm to perform high resolution patient-specific hemodynamic simulation in complex arterial geometry. Memory savings are crucial for high-fidelity, hemodynamic simulations in complex geometries as the maximum achievable resolution of the simulations is limited by the data that can fit in memory on the system [4, 23, 27]. As such, these patient-specific simulations are gaining increasing importance and are FDA approved for diagnostic purposes [3]. While the layer-based moment algorithm is generalizable to any LBM application and fluid flow problem, in this work we present hemodynamic simulation as a representative application. Achieving physiological flows in large arterial regions has previously been shown to push the limits of available memory on current systems [26, 27]. Thus, we use the application of patient-specific hemodynamic simulation as a case study for our layer-based moment algorithm.

We use an image-derived coronary arterial geometry as input to the layer-based moment representation algorithm. Zou-He boundary conditions were implemented at the inlet and outlets [40]. Blood was modeled as an incompressible Newtonian fluid with a dynamic viscosity of 4 cP and density of 1.06 kg/m^3. The blood vessels were modeled as rigid walls with no-slip boundary condition. We conducted the simulation in a patient-specific arterial geometry. The results of the simulation are shown in Figure 11 as velocity streamlines along the complete arterial tree.

7 DISCUSSION AND FUTURE WORK

Our driving goal with this work is to reduce both memory storage and bandwidth costs for the LBM algorithm while preserving accuracy to facilitate higher-fidelity fluid simulations. This study has presented an alternative LBM propagation pattern based on the moment representation from the regularized lattice Boltzmann method. A single-node performance model was used to understand the data motion and performance of key kernels. Simulation results on both Intel Broadwell and IBM Power9 CPUs confirmed the expected performance increase enabled by reducing memory storage and data motion and by maximizing data-reuse with a cache-aware layer scheme. Further optimization is likely possible and will be explored in subsequent studies. Excellent parallel scalability of the moment representation was observed, consistent with other LBM implementations. Finally, we demonstrate the application of moment representation algorithm for patient-specific hemodynamic

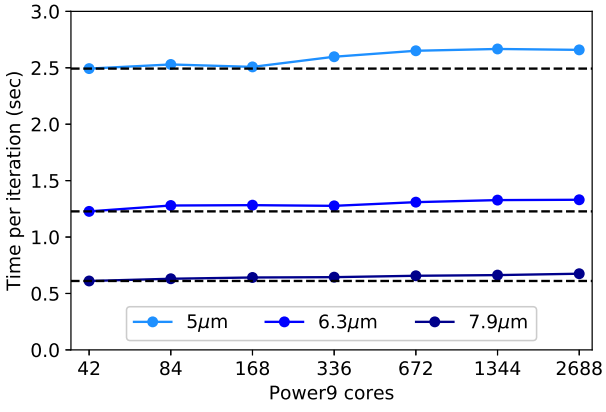

Figure 10: Weak scaling on Summit's Power9 CPUs for flow in a cylindrical geometries at three resolutions.

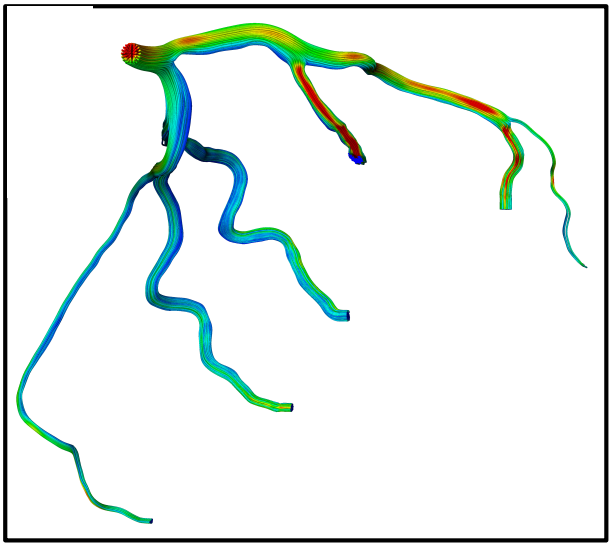

Figure 11: Velocity streamlines in a patient-derived left coronary arterial geometry.

simulations. As this work presents a new scheme for improving both time to solution and the scale of the problem that can fit in memory, we believe that it will set the stage for grand challenge-scale, high-fidelity hemodynamic simulations.

Looking forward to specialized hardware, such as GPU accelerators, we believe the method introduced here has significant potential. First, storage space continues to be a limitation on GPUs and our moment representation would be a significant step toward enabling larger problem sizes on these devices. To be sure, there is an inevitable trade-off between inter-thread parallelism and data reuse, especially on GPUs. We certainly expect that the CPU-focused cache optimization strategy presented here will need to adapted for GPUs. Specifically, the decomposition of the task bounding box into blocks will have to be closely modulated by warp sizes. Additionally, we expect that a structure of arrays or clustered structure of array format will improve memory coalescing for the distribution pseudo domain, instead of the array of structures implemented in this study [15]. Nonetheless, precisely because we will increase the amount of LBM data on the GPU, we anticipate that the layer-based propagation pattern presented here will expose sufficient parallelism for efficient GPU computation. Overall, we see the moment representation implemented in this study as a critical step in minimizing memory requirements and thereby maximizing problem sizes that can be tackled on future hardware.

8 ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This work was performed under the auspices of the U.S. Department of Energy by LLNL under Contract DE-AC52-07NA27344. Support was provided by the LLNL Laboratory Directed Research and Development (LDRD) program. Research reported in this publication was supported by the Office of the Director, National Institutes Of Health under Award Number DP5OD019876. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. Support was provided by the Hartwell Foundation and Duke Theo Pilkington Fellowship. We thank all the members of the Randlelab for their careful review and feedback on this work.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan(http://energy.gov/downloads/doe-public-access-plan).

REFERENCES

[1] R Argentini, AF Bakker, and CP Lowe. 2004. Efficiently using memory in lattice Boltzmann simulations. *Future Generation Computer Systems* 20, 6 (2004), 973–980.

[2] Peter Bailey, Joe Myre, Stuart DC Walsh, David J Lilja, and Martin O Saar. 2009. Accelerating lattice Boltzmann fluid flow simulations using graphics processors. In *Parallel Processing, 2009. ICPP'09. International Conference on*. IEEE, 550–557.

[3] Stewart M Benton, Christian Tesche, Carlo N De Cecco, Taylor M Duguay, U Joseph Schoepf, and Richard R Bayer. 2018. Noninvasive derivation of fractional flow reserve from coronary computed tomographic angiography. *Journal of Thoracic Imaging* 33, 2 (2018), 88–96.

[4] Massimo Bernaschi, Mauro Bisson, Toshio Endo, Satoshi Matsuoka, Massimiliano Fatica, and Simone Melchionna. 2011. Petaflop biofluidics simulations on a two million-core system. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 4.

[5] Prabhu Lal Bhatnagar, Eugene P Gross, and Max Krook. 1954. A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. *Physical review* 94, 3 (1954), 511.

[6] Jonathan Carter, Min Soe, Leonid Oliker, Yoshinori Tsuda, George Vahala, Linda Vahala, and Angus Macnab. 2005. Magnetohydrodynamic Turbulence Simulations on the Earth SimulatorUsing the Lattice Boltzmann Method. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM.

[7] Shiyi Chen and Gary D Doolen. 1998. Lattice Boltzmann method for fluid flows. *Ann Rev Fluid Mech* 30, 1 (1998), 329–364.

[8] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. 2008. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 4.

[9] Dominique d'Humières, Irina Ginzburg, Manfred Krafczyk, Pierre Lallemand, and Li-Shi Luo. 2002. Multiple–relaxation–time lattice Boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 360, 1792 (2002), 437–451.

[10] Yuankun Fu, Feng Li, Fengguang Song, and Luoding Zhu. 2018. Designing a parallel memory-aware lattice Boltzmann algorithm on manycore systems. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 97–106.

[11] Martin Geier and Martin Schoenherr. 2017. Esoteric twist: an efficient in-place streaming algorithmus for the lattice Boltzmann method on massively parallel hardware. *Computation* 5, 2 (2017), 19.

[12] Nicholas Geneva, Cheng Peng, Xiaoming Li, and Lian-Ping Wang. 2017. A scalable interface-resolved simulation of particle-laden flow using the lattice Boltzmann method. *Parallel Comput.* 67 (2017), 20–37.

[13] Christian Godenschwager, Florian Schornbaum, Martin Bauer, Harald Köstler, and Ulrich Rüde. 2013. A framework for hybrid parallel flow simulations with a trillion cells in complex geometries. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[14] LA Hegele Jr, A Scagliarini, M Sbragaglia, KK Mattila, PC Philippi, DF Puleri, J Gounley, and A Randles. 2018. High-Reynolds-number turbulent cavity flow

using the lattice Boltzmann method. *Physical Review E* 98, 4 (2018), 043302.

[15] Gregory Herschlag, Seyong Lee, Jeffrey S Vetter, and Amanda Randles. 2018. GPU Data Access on Complex Geometries for D3Q19 Lattice Boltzmann Method. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 825–834.

[16] Jonas Latt. 2007. *Hydrodynamic limit of lattice Boltzmann equations*. Ph.D. Dissertation. University of Geneva.

[17] Jonas Latt and Bastien Chopard. 2006. Lattice Boltzmann method with regularized pre-collision distribution functions. *Mathematics and Computers in Simulation* 72, 2-6 (2006), 165–168.

[18] Jonas Latt, Bastien Chopard, Orestis Malaspinas, Michel Deville, and Andreas Michler. 2008. Straight velocity boundaries in the lattice Boltzmann method. *Physical Review E* 77, 5 (2008), 056703.

[19] Song Liu, Nianjun Zou, Yuanzhen Cui, and Weiguo Wu. 2017. Accelerating the parallelization of lattice Boltzmann method by exploiting the temporal locality. In *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. IEEE, 1186–1193.

[20] Nicos S Martys and John G Hagedorn. 2002. Multiscale modeling of fluid transport in heterogeneous materials using discrete Boltzmann methods. *Materials and structures* 35, 10 (2002), 650–658.

[21] Keijo Mattila, Jari Hyväluoma, Tuomo Rossi, Mats Aspnäs, and Jan Westerholm. 2007. An efficient swap algorithm for the lattice Boltzmann method. *Computer Physics Communications* 176, 3 (2007), 200–210.

[22] John D McCalpin et al. 1995. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) newsletter* 1995 (1995), 19–25.

[23] Amanda Peters, Simone Melchionna, Efthimios Kaxiras, Jonas Lätt, Joy Sircar, Massimo Bernaschi, Mauro Bison, and Sauro Succi. 2010. Multiscale simulation of cardiovascular flows on the IBM Bluegene/P: Full heart-circulation system at red-blood cell resolution. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10.

[24] Thomas Pohl, Frank Deserno, Nils Thurey, Ulrich Rude, Peter Lammers, Gerhard Wellein, and Thomas Zeiser. 2004. Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures. In *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 conference*. IEEE, 21–21.

[25] Thomas Pohl, Markus Kowarschik, Jens Wilke, Klaus Iglberger, and Ulrich Rüde. 2003. Optimization and profiling of the cache performance of parallel lattice Boltzmann codes. *Parallel Processing Letters* 13, 04 (2003), 549–560.

[26] Amanda Randles, Erik W Draeger, Tomas Oppelstrup, Liam Krauss, and John A Gunnels. 2015. Massively parallel models of the human circulatory system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 1.

[27] Amanda Randles, Erik W Draeger, Tomas Oppelstrup, Liam Krauss, and John A Gunnels. 2015. Massively parallel models of the human circulatory system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 1.

[28] Amanda Peters Randles, Vivek Kale, Jeff Hammond, William Gropp, and Efthimios Kaxiras. 2013. Performance analysis of the lattice Boltzmann model beyond Navier-Stokes. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 1063–1074.

[29] S Succi, G Amati, M Bernaschi, G Falcucci, M Lauricella, and A Montessori. 2019. Towards Exascale Lattice Boltzmann computing. *Computers & Fluids* (2019).

[30] Pedro Valero-Lara. 2018. *Analysis and Applications of Lattice Boltzmann Simulations*. IGI Global.

[31] David Vidal, Robert Roy, and François Bertrand. 2010. A parallel workload balanced and memory efficient lattice-Boltzmann algorithm with single unit BGK relaxation time for laminar Newtonian flows. *Computers & Fluids* 39, 8 (2010), 1411–1423.

[32] Gerhard Wellein, Georg Hager, Thomas Zeiser, Markus Wittmann, and Holger Fehske. 2009. Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, Vol. 1. IEEE, 579–586.

[33] Gerhard Wellein, Thomas Zeiser, Georg Hager, and Stefan Donath. 2006. On the single processor performance of simple lattice Boltzmann kernels. *Computers & Fluids* 35, 8-9 (2006), 910–919.

[34] Samuel Williams, Jonathan Carter, Leonid Oliker, John Shalf, and Katherine Yelick. 2008. Lattice Boltzmann simulation optimization on leading multicore platforms. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 1–14.

[35] Samuel Williams, Leonid Oliker, Jonathan Carter, and John Shalf. 2011. Extracting ultra-scale lattice Boltzmann performance via hierarchical and distributed auto-tuning. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 55.

[36] Samuel Williams, Andrew Waterman, and David Patterson. 2009. *Roofline: An insightful visual performance model for floating-point programs and multicore architectures*. Technical Report. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[37] Markus Wittmann, Viktor Haag, Thomas Zeiser, Harald Köstler, and Gerhard Wellein. 2018. Lattice Boltzmann benchmark kernels as a testbed for performance analysis. *Computers & Fluids* 172 (2018), 582–592.

[38] Markus Wittmann, Thomas Zeiser, Georg Hager, and Gerhard Wellein. 2013. Comparison of different propagation steps for lattice Boltzmann methods. *Computers & Mathematics with Applications* 65, 6 (2013), 924–935.

[39] Thomas Zeiser, Gerhard Wellein, Aditya Nitsure, Klaus Iglberger, U Rude, and Georg Hager. 2008. Introducing a parallel cache oblivious blocking approach for the lattice Boltzmann method. *Progress in Computational Fluid Dynamics, an International Journal* 8, 1-4 (2008), 179–188.

[40] Qisu Zou and Xiaoyi He. 1997. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of fluids* 9, 6 (1997), 1591–1598.