Data structure and movement for lattice-based simulations

Aniruddha G. Shet,[1] Shahajhan H. Sorathiya,[2] Siddharth Krithivasan,[2] Anand M. Deshpande,[1] Bharat Kaul,[1] Sunil D. Sherlekar,[1] and Santosh Ansumali[2,*]

[1]*Parallel Computing Lab, Intel Labs, Bangalore 560103, India*

[2]*Engineering Mechanics Unit, Jawaharlal Nehru Centre for Advanced Scientific Research, Jakkur, Bangalore 560064, India*

(Received 12 November 2012; revised manuscript received 25 June 2013; published 31 July 2013)

We show that for the lattice Boltzmann model, the existing paradigm in computer science for the choice of the data structure is suboptimal. In this paper we use the requirements of physical symmetry necessary for recovering hydrodynamics in the lattice Boltzmann description to propose a hybrid data layout for the method. This hybrid data structure, which we call a structure of an array of structures, is shown to be optimal for the lattice Boltzmann model. Finally, the possible advantages of establishing a connection between group theoretic symmetry requirements and the construction of the data structure is discussed in the broader context of grid-based methods.

DOI: 10.1103/PhysRevE.88.013314 PACS number(s): 47.11.−j, 05.10.−a, 02.70.−c

I. INTRODUCTION

The use of lattice- or stencil-based simulation is pervasive in computational physics for solving partial differential equations. Two important examples are a finite -ifference time domain for Maxwell's equations [1–4] and the lattice Boltzmann (LB) method for hydrodynamics [5–12]. An algorithmic feature of these methods is that one updates array elements according to some predefined pattern and data are exchanged only between neighboring sites. These features create a highly parallel structure for these algorithms that is suitable for massive parallel computing [11,13–15]. However, an aspect not often discussed is that these algorithms are memory bound rather than CPU bound [16,17]. The existing wide gap between computing and data access speeds is only expected to widen over the foreseeable future. Thus the choice of data structure is not obvious *a priori* as various parts of the same algorithm may have conflicting data-structure requirements [16,18]. We highlight these issues for the LB method, an ideal example for grid-based computing, and show that a redesign of the data structure leads to substantial improvement in performance.

We recall the algorithmic aspects of the LB models, where the central quantity is the discrete population $f_i(\mathbf{x},t)$ at location $\mathbf{x}$ and time t with velocity c_i with $i = 1,2,\ldots,N$, where N is the the number of discrete velocities. Typically, an m-dimensional LB model with N discrete velocities is called a DmQN LB model. The evolution equation for the population is written as a two-step process. The first step is the local collision operation

$$f_i^{\star}(\mathbf{x},t) = f_i(\mathbf{x},t) + \Omega_i(f(\mathbf{x},t)), \tag{1}$$

where f denotes the set of populations and Ω_i is the collision operator that conserves mass, momentum and energy (in isothermal models, energy conservation is ignored). It depicts a relaxation process of the population towards its equilibrium value. The second step of advection mimics the free propagation of molecules and is

$$f_i(\mathbf{x}+\mathbf{c}_i\Delta t, t+\Delta t) = f_i^{\star}(\mathbf{x},t), \tag{2}$$

with Δt as the time step.

Since all the floating point calculations in Eq. (1) are local in space, using the locality principle, it is natural to place all the populations at a given point in neighboring locations in memory [19] (see Fig. 1). Such a data structure is called an array of structures (AOS) in the parlance of computer science and collision-optimized data layout in the LB method literature [13,20]. However, using the locality principle [19] in Eq. (2) suggests that all the f_i values for a particular direction of movement i should be stored sequentially (see Fig. 2). In other words, an advection code is best written using a structure of arrays (SOA), also called an advection-optimized layout in the LB method literature.

The conflicting data-structure requirements result in a sharp performance drop (as compared to the peak computer performance) for LB methods with larger stencils [16]. In this paper we address the question of whether there is an optimal data structure for the LB method that can match the compute efficiency of an AOS while achieving the memory bandwidth comparable to a SOA. We restrict our discussion to the case of large data sets, i.e., where the data cannot fit into the L1 cache and bandwidth issues are important.

The paper is organized as follows. Section II reviews the performance of existing methods for LB method implementation and suggests changes to improve performance. Section III reviews how discrete velocities, which dictate the connectivity on the grid, are chosen in the LB model. Section IV, based on the elementary breakup of the discrete velocity set in the LB model, introduces a different data structure, which we call structure of an array of structures. SectionV discusses a minor modification to the hybrid layout that helps in getting closer to the peak performance. Finally, Sec. VI discusses implications of this algorithm for grid-based computing in general.

II. PERFORMANCE MEASURE

In this work we are studying the efficiency of LB method implementations when an auxiliary array is not used and thus memory requirement is one-half of the other case (see [20] for

*ansumali@jncasr.ac.in

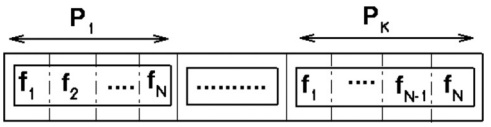

FIG. 1. Schematic diagram of an AOS data structure. Here P_K denotes the Kth spatial point characterized in D dimensions by coordinates x_j with $j = 1, \ldots, D$.

further details). Our concern is to raise the advection efficiency with the AOS data structure as this is best choice for achieving good efficiency in the compute stage of the LB method (see, for example, [16]). Before we do so, we need an efficiency measure for the implementations.

Similar to any grid-based code, the limiting factor affecting LB model simulations is often memory bandwidth rather than the floating point operations per second (FLOPS) efficiency of the machine, which implies that these codes are memory bound rather than compute bound. This makes peak FLOPS an uninformative performance measure as far as efficiency of the code is concerned. Thus the performance of a given LB simulation is measured often in terms of mega lattice-site updates per second (MLUPS) defined as [denoted as M_L in Eq. (3)] [13,20]

$$M_L = \frac{K_x K_y K_z}{10^6 T}, \tag{3}$$

where K_i denotes the number of grid points in the ith direction and T is the time taken for updating the grid. In this section we argue that this measure of MLUPS is less informative if we are interested in measuring performance as a function of the number of discrete velocities. In order to compare efficiency for higher-order lattice Boltzmann models, we introduce another performance measure, mega elementary updates per second (MEUPS), defined as [denoted as M_E in Eq. (4)]

$$M_E = \frac{K_x K_y K_z (N-1)}{10^6 T} \equiv (N-1) M_L, \tag{4}$$

where N is the number of discrete velocities ($N = 15$ for the D3Q15 model, $N = 19$ for the D3Q19 model, $N = 27$ for the D3Q27 model, and $N = 41$ for the D3Q41 model), and multiplication of $N - 1$ with MLUPS to normalize the number of memory operations in the advection step (rate-limiting step) for different discrete velocity models.

In Fig. 3 the MLUPS values of the advection step is plotted using the AOS data structure for various LB models [D3Q15, D3Q19, D3Q27 (see, for example, Ref. [10]), and D3Q41 (from Ref. [21])] where the SOA data structure for the D3Q15 model is also plotted as a benchmark to illustrate peak performance. Here advection is implemented using two of the commonly used algorithms: a loop-based algorithm with

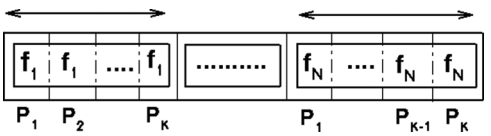

FIG. 2. Schematic diagram of a SOA data structure. Here P_K denotes Kth spatial point characterized in D dimensions by coordinates x_j with $j = 1, \ldots, D$.

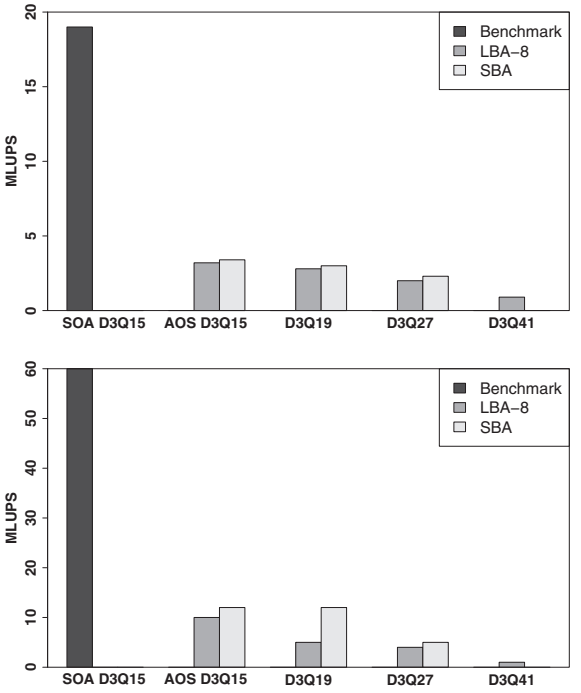

FIG. 3. The MLUPS for the SOA and AOS on a six-core AMD® Opteron™ 2439 SE processor (top) and Intel® Xeon™ E5-2670 processor (bottom).

eight loops (LBA-8) and a swap-based algorithm (SBA) (see Appendices A and B). One can see that memory bandwidth utilization of the AOS data structure is indeed poor along with a sharp performance drop for larger stencils. A part of

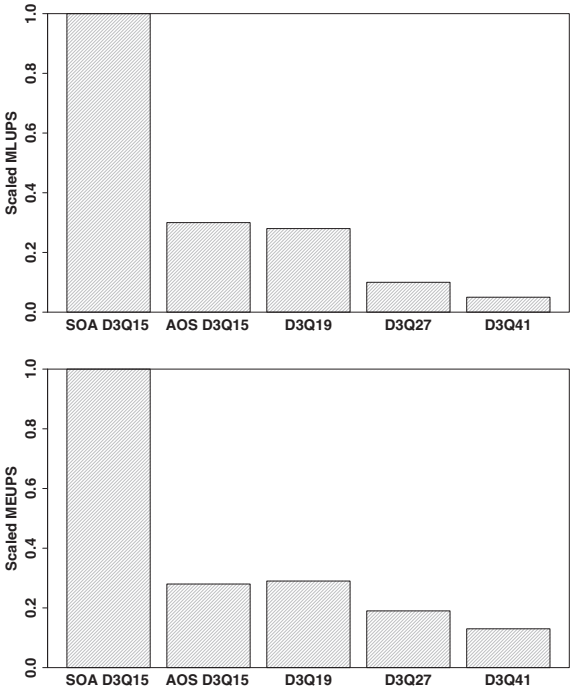

FIG. 4. Comparison of the ratio of MLUPS and MEUPS among different AOS implementations with respect to the SOA D3Q15 model on Intel® Xeon™ E5-2670 processor.

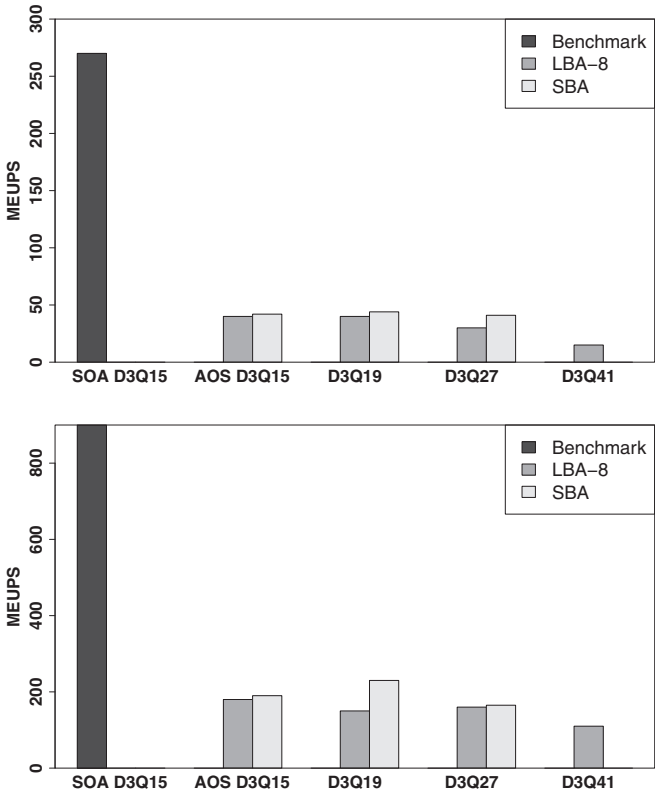

FIG. 5. The MEUPS for the SOA and AOS on a six-core AMD® Opteron™2439 SE processor (top) and Intel® Xeon™ E5-2670 processor (bottom).

this drop is expected since the number of memory shifts is directly proportional to the number of discrete velocities used. However, a large part of this drop is due to poor memory utilization.

The MLUPS as a performance measure is not able to distinguish between the two effects: drop due to the large number of discrete velocities and drop due to poor memory bandwidth utilizations. As MEUPS already accounts for the increase in number of discrete velocities, any visible drop is solely due to poor memory utilizations in higher-order LB models. To highlight this aspect, we have taken SOA implementation of the D3Q15 model as a benchmark and scaled the MLUPS and MEUPS values for different DmQN models with AOS implementation with respect to the corresponding

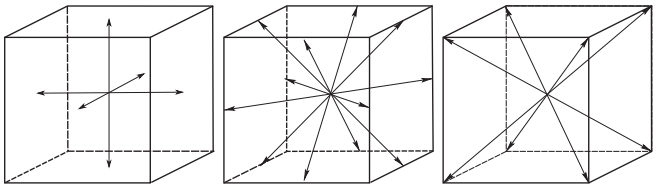

FIG. 7. The sc, fcc, and bcc shells as building blocks of the LB method.

values for the SOA D3Q15 model as shown in Fig. 4. It can be observed that the drop of MLUPS from the D3Q15 to the D3Q41 model is around 20 times. However, the drop in MEUPS from the D3Q15 to the D3Q41 model is around 7.5 times, which suggests that deterioration is a direct function of the number of discrete velocities at least over the limited range of discrete velocities explored here. Hence MEUPS can be a better measure for performance than MLUPS. Thus we have chosen to report advection performance in terms of MEUPS.

Figure 5 shows that none of the AOS implementations manage to use even 20% of the bandwidth, which suggests memory bandwidth utilization as a bottleneck for grid-based computing. Finally, we highlight a peculiarity in Figs. 3 and 5: The D3Q19 model performs a little better than the D3Q15 model. This is remarkable because the D3Q19 model has more data per grid point than the D3Q15 model. To understand this peculiarity, we recall that both the D3Q15 model and the D3Q19 model are subsets of the D3Q27 model. Whereas the D3Q27 model consists of three major building blocks, namely, the simple cubic (sc) shell, the face centered cubic (fcc) shell and the body centered cubic (bcc) shell, the D3Q15 and D3Q19 models are arrived at via pruning of fcc and bcc shells, respectively, from the D3Q27 model (see Fig. 7). Thus we can say that the bcc shell is more amenable to memory bandwidth usage than is the fcc shell. This suggests that the data structure and the choice of the discrete velocity model may follow the same physical considerations. We exploit this observation later to show that the basic building blocks of the discrete velocity model and data structure are the same.

Furthermore, we notice that the SBA performs better than the LBA-8, which is probably due to efficient utilization of prefetched data by compilers. However, a data flow analysis of

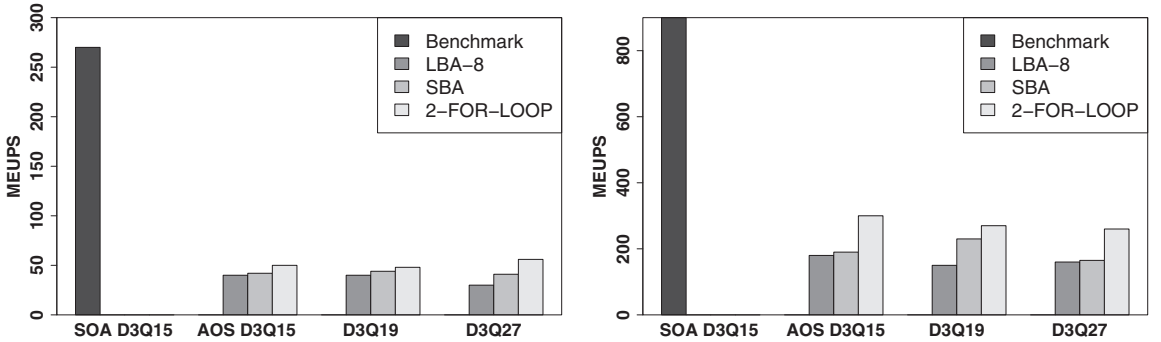

FIG. 6. The MEUPS after loop fusions that allow advection to happen in two *for* loops on a six-core AMD® Opteron™2439 SE processor (left) and Intel® Xeon™ E5-2670 processor (right).

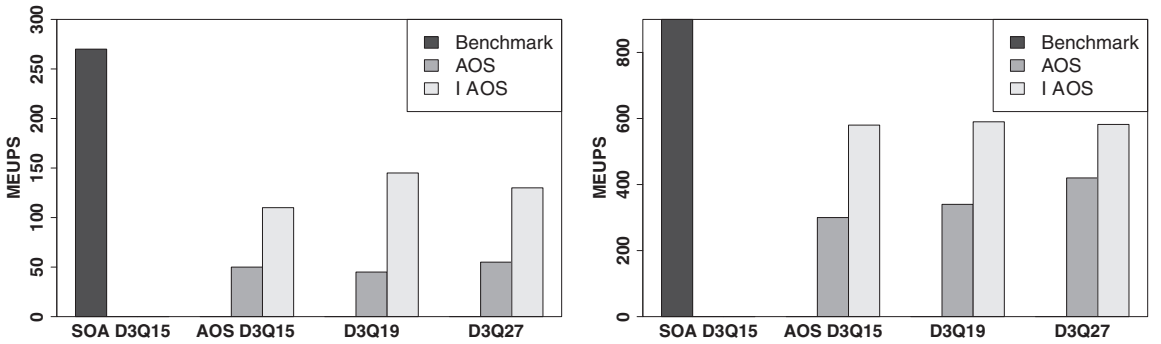

FIG. 8. The MEUPS after breaking the data structure into building blocks with two for loops on a six-core AMD® Opteron™2439 SE processor (left) and Intel® Xeon™ E5-2670 processor (right).

the LBA-8 reveals that the data dependence is one dimensional and only the direction of the outer loop matters (see the Appendices). Figure 6 shows that the performance of a simpler LBA-2, where redundant steps are removed, is much superior to both the LBA-8 and SBA due to aggressive optimizations by compilers.

III. DISCRETE VELOCITY SET FOR LB

Following [22] we describe how the choice of discrete velocity set in the LB method is dictated by symmetry and isotropy requirements. The following are the two requirements on the discrete velocity needed to get the correct hydrodynamic description.

(i) Closure under inversion. If a discrete velocity $\mathbf{c}_i \equiv (c_{ix},c_{iy},c_{iz})$ is an element of the set, so should be its inverse, i.e.,

$$\mathbf{c}_i \in \mathcal{C} \Rightarrow -\mathbf{c}_i \in \mathcal{C}. \tag{5}$$

(ii) Closure under reflection. If a discrete velocity $\mathbf{c}_i \equiv (c_{ix},c_{iy},c_{iz})$ is an element of the set, then so should be all possible reflections of it, i.e.,

$$\mathbf{c}_i \in \mathcal{C} \Rightarrow (\bar{c}_{ix},\bar{c}_{iy},\bar{c}_{iz}) \in \mathcal{C}, \tag{6}$$

where $\bar{c}_{ij}$ with $j = x,y,z$ denotes all possible permutations of $\mathbf{c}_i$ components.

These requirements show that the essential building blocks for the LB method are the sc, fcc, and bcc shells (Fig. 7),

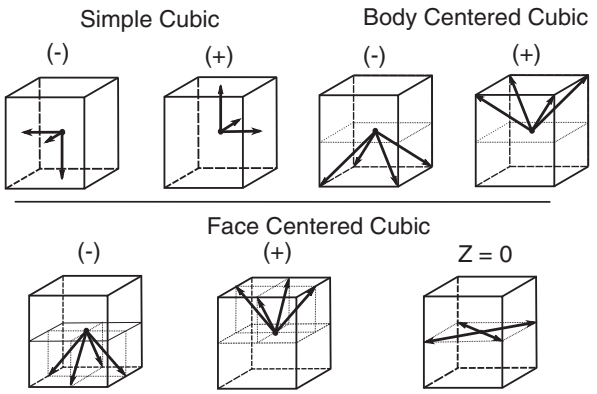

FIG. 9. The SOAOS data-structure construction from the AOS for the LB model.

which satisfy the closure conditions, and later we use them as the alternative data structure itself.

IV. HYBRID DATA STRUCTURE: STRUCTURE OF AN ARRAY OF STRUCTURES

Motivated by the construction of the LB method from building blocks [22], we build a hybrid data structure, termed a structure of an array of structures (SOAOS), where the AOS is constructed over individual building blocks rather than over an entire data set. As seen in Fig. 8, this intermediate data

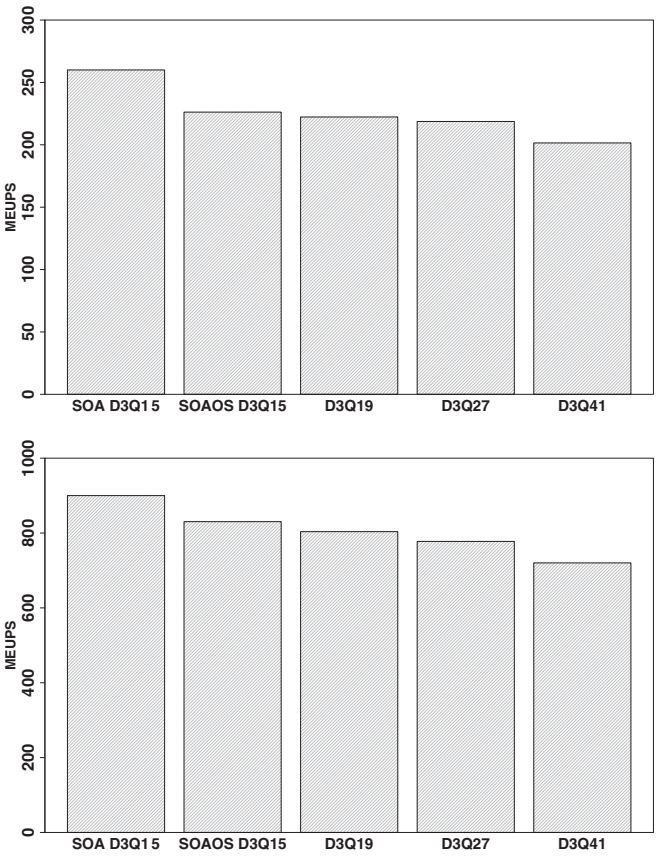

FIG. 10. The MEUPS for the SOAOS on a six-core AMD® Opteron™2439 SE processor (top) and Intel® Xeon™ E5-2670 processor (bottom).

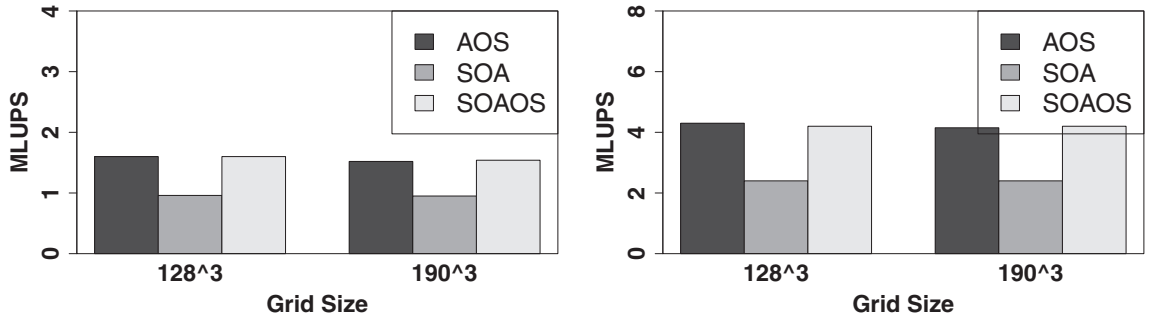

FIG. 11. Comparison of compute performance in terms of the MLUPS for the SOAOS with the AOS and SOA on a six-core AMD® Opteron™2439 SE processor (left) and Intel® Xeon™ E5-2670 processors (right). The maximum size we could run on the AMD system was limited due to the available RAM.

structure leads to better performance, which is about 50% of the peak value. We improve on this intermediate data structure by breaking it into more atomic building blocks. Here we recall that in the LBA-2 implementation, for any given building block (sc, fcc, and bcc) only half of them get updated in a single for loop. Thus it seems natural to break every block into two parts along the outer loop direction and not to insist on the condition of closure under inversion for individual blocks. Furthermore, instead of the earlier stated condition of reflection in D dimensions, we create a restrictive reflection condition in $D-1$ dimensions. This implies that if a discrete velocity $\mathbf{c}_i \equiv (c_{ix}, c_{iy}, c_{iz})$ is an element of the set, then all possible reflections of it in $D-1$ dimensions with z the outer loop direction are also members of the set, i.e.,

$$\mathbf{c}_i \in \mathcal{C} \Rightarrow (\bar{c}_{ix}, \bar{c}_{iy}, c_{iz}) \in \mathcal{C}, \tag{7}$$

where $\bar{c}_{ij} \in \{c_{ix}, c_{iy}\}$ with $j = x, y$. In Fig. 9 we present a SOAOS data structure based on these considerations. Figure 10 shows that this SOAOS structure is indeed able to almost saturate the memory bandwidth and hence is the best alternative to the SOA for advection.

Figure 11 shows that the present model is able to replicate the performance of the AOS for computation too. Here we work with multiple relaxation time collision models where floating point operations are always higher as compared to the Bhatnagar-Gross-Krook (BGK) approach (see, for example, [23]), which ensures that the quantitative benefit reported for the SOAOS approach remains conservative and provides only the lower bound of gain for the present approach. Though qualitative trends remain similar for different collision models, as a specific example we report data for the quasiequilibrium-type multirelaxation models proposed in Ref. [24]. With this choice of collision model, the overall application speedup with the hybrid data structure (SOAOS) is 1.3 times faster than the AOS and 1.55 times faster than the SOA when 256^3 grid points are assigned to each CPU core. Thus the hybrid model of the SOAOS, when coupled with the two-loop advection formulation, provides efficient and optimal LB codes without compromising on the simplicity of the algorithm.

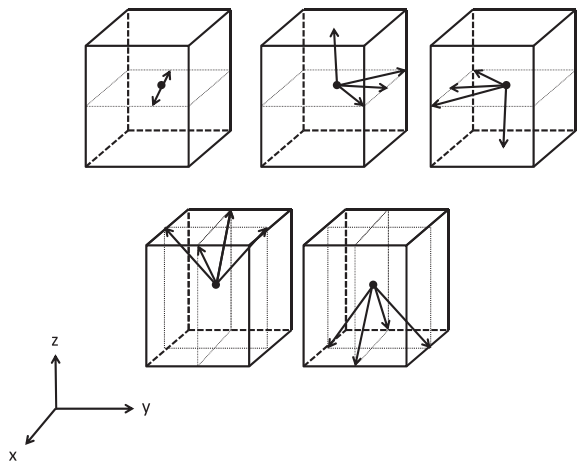

FIG. 12. Data-structure rearrangement for part of the fcc and sc structures.

V. OPTIMAL HYBRID MODEL

In this section we wish to point out that some fcc populations that lie in the x-y plane provide minor issues with the framework developed in preceding section. Ideally, this subblock should be again subdivided into two parts to take advantage of the loop structure. However, such an arrangement interferes with efficient optimization of the collision step. Hence, in this section we propose a minor rearrangement where the simple cubic structure and the $Z = 0$ structure in the face centered cubic structure are modified as shown in Fig. 12 and the distribution of velocities in different structures is shown in Fig. 13. The first structure has velocities only in the x direction. The next two structures are constructed such that the outer loop runs in one direction for all the updates. This modification improves overall performance of the advection step by another $\sim$10% as seen in Fig. 14.

VI. OUTLOOK

Before presenting the final performance number of the overall LB code on CPUs, we wish to highlight the vectorization friendliness of the algorithm, the capability of the method to scale well, and the efficient utilization of the available memory bandwidth or FLOPS (whichever is rate limiting),

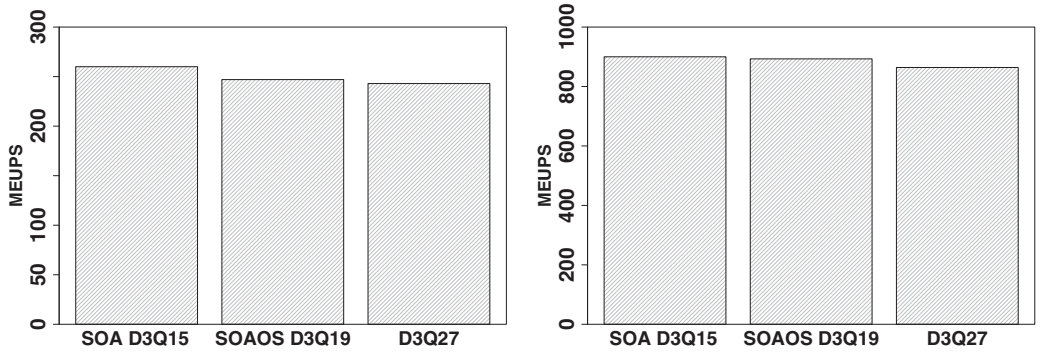

FIG. 13. Present data structure for the D3Q19 model.

which would be relevant to optimization on accelerators too. Hence we have extensively studied both of these aspects for CPUs. We have implemented vectorization for the D3Q19 and D3Q27 models with the present data layout and measured the total performance for a vectorized code (for collision). For vectorization Intel compiler was used for which more complete information can be obtained at [25]. We have used two other reference data for benchmark. The first one is the Palabos number (which provides the status of most of the code in the community) and the second one is ideal MLUPS. The ideal MLUPS for a DmQn model on a machine of maximum memory bandwidth B GB/s is (three read and write operations in the advection step and three read and write operations in the collide step)

$$\mathcal{T}_{\mathrm{MLUPS}} = \frac{B \times 10^9}{10^6 \times n \times 6 \times 8}, \tag{8}$$

where $\mathcal{T}_{\mathrm{MLUPS}}$ denotes the theoretical MLUPS and the factor of 8 bytes is for double precision.

In Fig. 15 we compare our LB code with the Ideal number and Palabos numbers for periodic geometry and the D3Q19 model with the BGK collision term. We see that our code is doing much better than the community benchmark and also is close to the Ideal numbers. For the purpose of benchmarking the code, we have considered a nontrivial example of a physical system with relevance to aerodynamics and suspension theory. Simulations were performed for a model of insect flight where an elliptical cylinder translates with constant velocity at an angle to the major axis. The lift coefficient was calculated for various angles of attack and was compared with a finite-difference study by Wang [26]. As expected, Fig. 16 shows that the results show a good comparison.

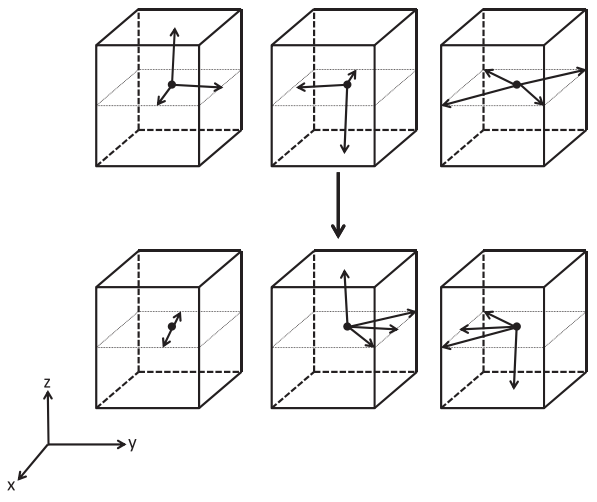

FIG. 14. The MEUPS for the SOAOS on a six-core AMD® Opteron™2439 SE processor (left) and Intel® Xeon™ E5-2670 processors (right) for the revised hybrid model.

We show that the simulation of such a nontrivial system benefits significantly from our present algorithm. To demonstrate this, we consider Palabos, a community-based effort, as a benchmark. The choice of Palabos as the benchmark is motivated by the fact that it is not only highly efficient in terms of computational speed via advanced optimization techniques (Hilbert curve tracing, swap, etc.), but also in terms of the efficiency of memory utilization (a major concern in practice). Similar to the present work, it uses single-memory implementation rather than double-memory implementation.

We chose a system size of $320 \times 320 \times 320$ distributed over many cores communicating via a message passing interface (MPI). As the LB model we considered the D3Q27 model lattice BGK since it is the simplest prototype to highlight the issue of memory bandwidth in higher-order LB codes. The collision part in the code was vectorized with the data structure as explained in Sec. V. The benchmark was performed on an Intel SandyBridge machine with a processor speed of 2.60 GHz. The Ideal number has been reported assuming that the boundary conditions do not change the performance. The comparison shown in Fig. 17 clearly indicates that the presented algorithm is superior to existing state-of-the-art ones even for complex simulations.

To conclude, the algorithm presented in the present work addresses the memory bandwidth issue in lattice-based simulations and thus allows for efficient implementations of the lattice Boltzmann model. This issue of memory bandwidth utilization is becoming increasingly more important as we migrate to improved semiconductor technologies, larger system sizes, and wider stencils for the discrete velocity model (see, for example [27], where the D3Q41 model is shown to be good for turbulence simulation). Finally, we add that our assessment is limited to standard CPUs and that no assumption should be made about the validity of the proposed approach on different platforms such as GPUs.

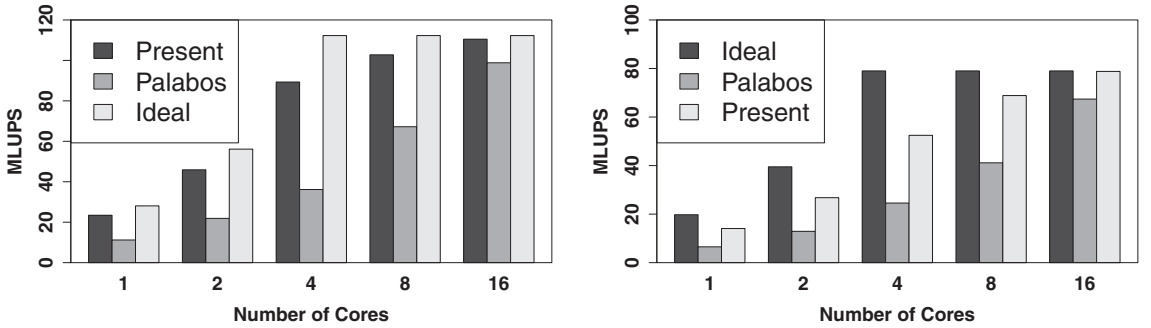

FIG. 15. Comparison of the MLUPS for the D3Q19 (left) and D3Q27 (right) models for a grid size of 320^3 between the Ideal number and the present data structure on Intel® Xeon™ E5-2670 processors. The maximum memory bandwidth per socket for this machine is 51.2 GB/s. Since the machine has four memory channels and each core can use up to two channels the maximum bandwidth does not change after four cores.

ACKNOWLEDGMENTS

S.A. is thankful to S. Melchionna, J. Latt, S. Succi, and C. Narayana for helpful discussions.

APPENDIX A: ONE-DIMENSIONAL IMPLEMENTATION OF ADVECTION

The basic algorithmic for the LB model can be understood in terms of advection of two scalar variables $\phi(x)$ and $\psi(x)$,

$$\phi(x) = \phi(x + \Delta x), \quad \psi(x) = \psi(x - \Delta x), \qquad \text{(A1)}$$

in one dimension, where x denotes the spatial location and Δx is the grid spacing. This set of equations can be implemented without wasting memory in two possible ways, which are typically called a loop-based algorithm and a swap-based algorithm. In the loop version of advection the data dependence in the update step is broken by a proper choice of the loop direction. In particular, the combination

$$\text{for } (i = 1 \text{ to } K) \quad \{\phi(i) = \phi(i + 1)\},$$
$$\text{for } (i = K \text{ to } 1) \quad \{\psi(i) = \psi(i - 1)\}$$

of forward and backward looping can be used to implement Eq. (A1).

The same equation [Eq. (A1)] can also be implemented using the duality between swap $\mathcal{A}_{\text{swap}}$ and shift as

$$\text{for}(i = 0 \text{ to } K)\ \{\mathcal{A}_{\text{swap}}(\phi(i), \psi(i + 1))\}.$$

After the end of this step of the swap-based implementation of Eq. (A1), the relative memory location for the two populations gets interchanged, which needs to be taken care of (see [20]).

APPENDIX B: THREE-DIMENSIONAL IMPLEMENTATION

All higher-dimensional algorithms that do not waste memory are derivatives of the one-dimensional alternatives of the LBA and SBA. In higher dimensions, while the swap algorithm remains the same conceptually, we show that the extension of the shift version to higher dimensions is nontrivial. We recall that in the shift version, the loop direction is chosen as opposite to the shift direction. This implies that we have eight possible loop orderings corresponding to the plus or minus shift in any direction. For example, for two scalar variables $\phi(x,y,z)$ and $\psi(x,y,z)$,

$$\phi(x,y,z) = \phi(x + \Delta, y + \Delta, z + \Delta),$$
$$\psi(x,y,z) = \psi(x - \Delta, y - \Delta, z + \Delta), \qquad \text{(B1)}$$

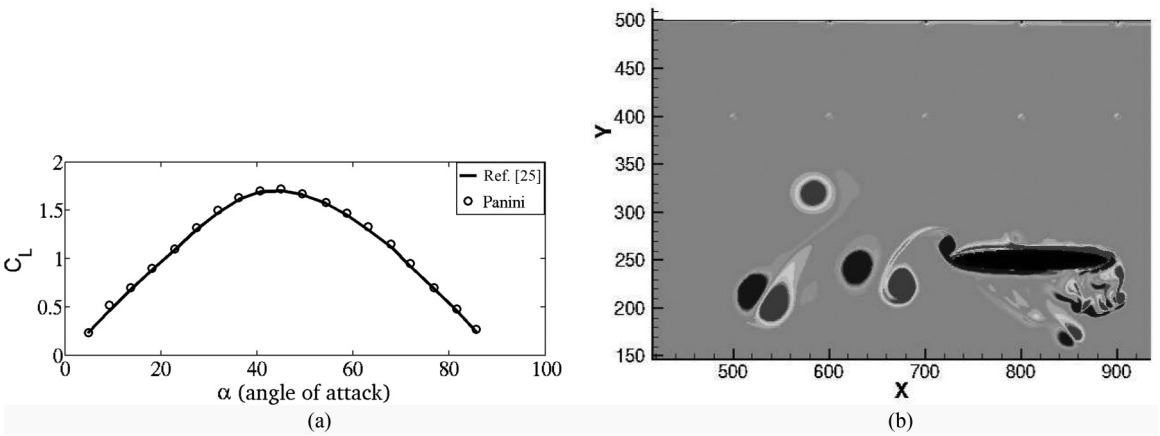

FIG. 16. (a) Comparison of the lift coefficient computed from the current code (named as Panini) with the result in [25]. (b) Vorticity contours zoomed close to the body of an elliptical cylinder flapping with $\text{St}_c = 2$ and $\text{Re} = 1000$ obtained from the current code.

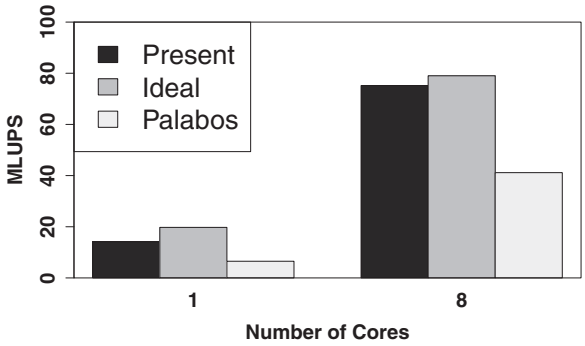

FIG. 17. Comparison of the MLUPS of the present code with Ideal and Palabos codes for a system size of 320^3 with one core and with eight cores on Intel® Xeon™ E5-2670 processor.

where Δ is the grid spacing, Eq. (B1) can be implemented as

$$\begin{aligned}&\text{for } (k = 1 \text{ to } K_z)\{\\ &\text{for } (j = 1 \text{ to } K_y)\{\\ &\text{for } (i = 1 \text{ to } K_x)\{\\ &\phi(i,j,k) = \phi(i+1,j+1,k+1)\}\};\end{aligned}$$

$$\begin{aligned}&\text{for } (j = K_y\text{to}1)\{\\ &\text{for } (i = K_x\text{to}1)\{\\ &\psi(i,j,k) = \psi(i-1,j-1,k+1)\}\}.\end{aligned}$$

This version is typically implemented in many available source codes, which we term eight-loop versions of the LBA, or LBA-8. However, this straightforward extension of one-dimensional logic in three dimensions is too conservative. Data flow analysis in the pseudocode presented above shows that the values of ϕ and ψ in the xy plane at $z = k$ are taken from the xy plane at $z = k + 1$. Thus the data dependence is purely one dimensional and only the direction of the outer loop matters. The pseudocode for such an implementation

$$\begin{aligned}&\text{for } (k = 1\text{to}K_z)\{\\ &\text{for } (j = 1\text{to}K_y)\{\\ &\text{for } (i = 1\text{to}K_x)\{\\ &\phi(i,j,k) = \phi(i+1,j+1,k+1);\\ &\psi(i,j,k) = \psi(i-1,j-1,k+1)\}\}\}\end{aligned}$$

is referred to in this paper as the two-loop version of the LBA, or LBA-2.

[1] K. Yee, Antenn. Propag. IEEE Trans. **14**, 302 (1966).
[2] G. Mur, Electromag. Compat. IEEE Trans. **EMC-23**, 377 (1981).
[3] J. Berenger, J. Comput. Phys. **114**, 185 (1994).
[4] A. Taflove and S. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 2nd ed. (Artech House, Norwood, MA, 2000).
[5] F. J. Higuera, S. Succi, and R. Benzi, Europhys. Lett. **9**, 345 (1989).
[6] H. Chen, S. Chen, and W. H. Matthaeus, Phys. Rev. A **45**, 5339 (1992).
[7] R. Benzi, S. Succi, and M. Vergassola, Phys. Rep. **222**, 145 (1992).
[8] X. Shan and H. Chen, Phys. Rev. E **47**, 1815 (1993).
[9] S. Chen, H. Chen, D. Martinez, and W. Matthaeus, Phys. Rev. Lett. **67**, 3776 (1991).
[10] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond* (Oxford University Press, New York, 2001).
[11] C. Aidun and J. Clausen, Annu. Rev. Fluid Mech. **42**, 439 (2010).
[12] S. Ansumali, I. Karlin, and H. Öttinger, Europhys. Lett. **63**, 798 (2007).
[13] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick, in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (IEEE, Piscataway, NJ, 2008), pp. 1–14.
[14] M. Bernaschi, M. Bisson, T. Endo, S. Matsuoka, M. Fatica, and S. Melchionna, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11 (ACM, New York, NY, USA, 2011), pp. 4:1–4:12.
[15] M. Bernaschi, M. Bisson, M. Fatica, S. Melchionna, and S. Succi, Comput. Phys. Commun. **184**, 329 (2013).
[16] G. Wellein, T. Zeiser, G. Hager, and S. Donath, Comput. Fluids **35**, 910 (2006).
[17] T. Ciamulski and M. Sypniewski, in *Antennas and Propagation Society International Symposium, 2007 IEEE* (IEEE, Piscataway, NJ, 2007), pp. 4897–4900.
[18] V. Heuveline, M. Krause, and J. Latt, Comput. Math. Appl. **58**, 1071 (2009).
[19] P. Denning, Commun. ACM **48**, 19 (2005).
[20] K. Mattila, J. Hyväluoma, T. Rossi, M. Aspnäs, and J. Westerholm, Comput. Phys. Commun. **176**, 200 (2007).
[21] S. S. Chikatamarla and I. V. Karlin, Phys. Rev. E **79**, 046701 (2009).
[22] W. P. Yudistiawan, S. K. Kwak, D. V. Patil, and S. Ansumali, Phys. Rev. E **82**, 046701 (2010).
[23] J. Latt and B. Chopard, Math. Comput. Simulat. **72**, 165 (2006).
[24] S. Ansumali, S. Arcidiacono, S. S. Chikatamarla, N. I. Prasianakis, A. N. Gorban, and I. V. Karlin, Eur. Phys. J. B **56**, 135 (2007).
[25] Intel. Optimization Notice. http://software.intel.com/en-us/articles/optimization-notice, 2012.
[26] Z. J. Wang, J. Fluid Mech. **410**, 323 (2000).
[27] S. S. Chikatamarla, C. E. Frouzakis, I. V. Karlin, A. G. Tomboulides, and K. B. Boulouchos, J. Fluid Mech. **656**, 298 (2010).