HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries

M. D. Mazzeo*and P. V. Coveney†

Centre for Computational Science, University College London,

20 Gordon St., London WC1H 0AJ, U.K.

June 27, 2007

Abstract

We describe a parallel lattice-Boltzmann code for efficient simulation of fluid flow in complex geometries. The lattice-Boltzmann model and the structure of the code are discussed. The fluid solver is highly optimized and the resulting computational core is very fast. Furthermore, communication is minimized and the novel topology-aware domain decomposition technique is shown to be very effective for large systems, allowing us to tune code execution in geographically distributed cross-site simulations. The benchmarks presented indicate that very high performances can be achieved.

Key words: Lattice-Boltzmann. Computer simulations. Parallel computing. Complex systems.

*E-mail: m.mazzeo@ucl.ac.uk

†E-mail: P.V.Coveney@ucl.ac.uk

1 Introduction

The study of fluid flow in confined and complex geometries is of considerable interest in several domains including porous media [2] and blood [3, 4]. Magnetic resonance imaging (MRI) [5] and advanced X-ray computed tomography (CT) [6] have been successfully applied to visualize and analyze flow in these situations. CT and magnetic resonance angiography (MRA) furnish non-invasive static and dynamical data acquisition [7]. In the clinical context, cerebral blood flow behaviour plays a crucial role in understanding, diagnosis and treatment of many conditions; indeed, some studies reveal relationships between specific flow patterns around vessel walls and cardiovascular diseases such as atherosclerosis [8]. However, experimental studies are often impractical owing to the difficulty of measuring transport properties in porous media and observing blood flow behaviour in humans.

Modelling and simulation undoubtedly have a crucial role to play in these domains. The aforementioned imaging modalities provide high resolution data that simulation tools can exploit to accurately reproduce flow fields through the reconstructed geometries. Moreover, the need to capture fluid flow effects at multiple space scales with sufficient accuracy inevitably requires us to deal with large and complex systems, while many relevant processes are characterized by substantial time scales [9]. This problem is compounded when dealing with larger systems since computational fluid solvers take more time to converge. The greater number of simulation time steps required to achieve the desired results further increases the computational demand.

Modelling and simulation offer the prospect of providing clinicians with virtual patient specific analysis and thus diagnosis; moreover, they raise the possibility of performing non-invasive virtual experiments to plan and study the effects of certain courses of surgical treatment with no danger to the patient [3]. However, immediate clinical impact demands very rapid turn around of results, in close to real time if the methods are to be of interactive value.

Plainly, therefore, the availability of computational models of sufficient complexity and power is crucial. The computational fluid "solver" must itself be numerically highly efficient. While conventional continuum fluid solvers, based on finite difference, finite volume and

finite element codes, certainly do exist [4, 10], they are beset with problems in three spatial dimensions due to the computational costs of mesh generation, the need to solve the auxiliary Poisson equation for the pressure field, and various unvoidable approximations associated with the calculation of the shear stress from the flow velocity field. For large systems, such as those we are concerned with here, it is essential to develop and utilise scalable, high performance parallel codes, which further complicates the use of continuum models. In addition, the intricate geometry of porous media and blood vessels, as well as the treatment of fluid boundary conditions at such walls, are difficult for continuum fluid dynamics models to handle.

The lattice-Boltzman method is a special discretization of the continuous Boltzmann equation and is thus a kinetic approach that, unlike conventional numerical schemes, does not discretize the macroscopic continuum equations with the attendant need to solve the Poisson equation. Therefore, the lattice-Boltzmann (LB) method [1] offers an attractive alternative: our fluid solver is based on this modelling and simulation method.

The paper is organized as follows. In Sec. 2, we review some related work. We describe the lattice-Boltzmann model in Sec. 3 and boundary condition methods used in our code in Sec. 4. We then present the computational core and some single processor benchmarks on regular and complex systems in Sec. 5. The parallel implementation, associated benchmark results and a discussion are given in Sec. 6 followed by conclusions in Sec. 7.

2 Overview of related work

The high computational cost and large memory requirement associated with fluid flow simulation of very large and complex systems require the use of a suitable physical model, very extensive computational resources, as well as applications capable of exploiting them effectively, the adoption of optimizations that ensure low memory consumption, and a minimal number of computational operations. Much work has been done in order to speed-up single-processor and parallel lattice-Boltzmann simulations in regular systems. However, relatively few studies have been performed in order to improve lattice-Boltzmann simulations of fluid flow confined in complex geometries. The parallel program presented in this work, called

"HemeLB", represents a contribution in this direction.

In the context of LB fluid flow simulations in non-regular systems, particularly porous media [36–38], a number of studies have been carried out using the strategy presented by Donath *et al.* [44] to reduce the memory requirements. This approach avoids the storage of data associated with obstacles: 1D arrays, one for each data type, store the information about the fluid sites. The identification of the neighbouring distribution functions, needed during the LB streaming stage, relies on an extra 1D connectivity buffer. We note that an expensive global look-up table for the connectivity buffer is not required: if the 3D configuration file stores the data about all lattice sites, it suffices to read this file only once in order count the total number of fluid lattice sites, to allocate the 1D buffers and to build the connectivity by means of a look-up table composed of just two slabs of the entire system wherein the lattice site types and indices in the 1-D buffers are stored. The slabs must iteratively scan the entire 3D domain and must be swapped at every iteration. Thus, when the connectivity –supplied by an index or a pointer– needs to be established between a 1D buffer element and a neighbouring (fluid) one, it is sufficient to find its data in the occupied slab[1].

High performance lattice-Boltzmann simulations of large systems[2] on cache-based microprocessors can be realized if there is sufficient spatial and/or temporal coherency since the poor bandwidth and latency of RAM memory cannot guarantee to keep the CPUs busy. This means that the data should be referenced frequently and/or with a sufficiently high level of sequentiality so that it is more likely to already reside in cache memory[3]. The high cost of short latency memory limits the adoption of large ones. Usually, caches are organized in a hierarchical fashion in which the largest one, characterized by the lowest price per mm^2, is the slowest and interfaces directly with RAM, while the smallest and fastest serves the CPU

[1] An approach that uses slabs to save memory is presented in detail in [33] in the context of the implementation of the lattice-Boltzmann method for regular systems.

[2] A system is considered to be large if it does not fit in any of the caches.

[3] While this is obvious for data accessed frequently, automatic prefetch of contiguous data guarantees their placement in cache before their processing. Some modern cache-based microprocessors prefetch non-contiguous data disposed in regular patterns and thus their workflow is optimized with respect to non-contiguous data arranged irregularly. The interested reader should consult Kowarschik and Weiß [25] for more detailed discussions on temporal and spatial coherency as well as organization of cache memories.

directly. This approach represents a good compromise in terms of cost and execution speed. Many cache-aware optimizations such as loop blocking, loop interchange, loop fusion and array merging [25] can substantially improve cache exploitation and thus reduce the execution time of many applications. Nevertheless, the presence of streaming and collision stages in lattice-Boltzmann simulations requires access to non-contiguous data, making it difficult to achieve high performance when simulating systems of even a modest size. Pohl *et al.* [26] and Schulz *et al.* [64] presented two different "compressed grid" approaches which reduce the total memory consumption by a factor of almost two by exploiting the deterministic data dependencies which occur in the propagation step. The three-slabs approach of Martys and Hagedorn [43] and Argentini *et al.* [33] is also useful to reduce the total memory consumption. However, these memory-saving strategies do not solve the issue of modest performance on large systems since many non-contiguous data needed during the propagation step still reside far from each other. 1-D and 3-D loop blocking techniques [26, 28, 29] show a similar performance improvement by augumenting space locality in small systems. The one technique which permits high performances to be attained on large systems seems to be the n-way blocking presented by Pohl *et al.* [26], Iglberger [29], and Velivelli and Bryden [30]. For 3D systems, 4-way blocking relies on the maximum exploitation of the first-neighbour LB kernel by combining 3D space blocking with a 1D time one. Many time steps are performed on 3D blocks of different shapes involving access to the whole domain at one time only. This strategy guarantees good spatial and temporal coherency and delivers high performance (see also Donath [31] and [32] for further benchmarks on cache-aware LB implementations). Relatively simple prefetch and preload techniques [27, 34] can lead to a significant increase in memory bandwidth and improve the overall performance substantially. However, it is difficult or impossible to apply the 4-way blocking approach to flow simulation in "sparse" (non regular) systems because the data workflow is completely irregular.

Probably the best known data representation which is suited to LB fluid flow simulations in sparse and complex cartesian geometries is that presented by Donath *et al.* [44] in the context of flow in porous media, where an approach based on regular 3D grids is abandoned in favour of 1D buffers which store the distribution functions of the fluid lattice sites and

the connectivity information required during the propagation step. This strategy avoids the need to store the data related to non-fluid lattice sites. Sparse data representations have been successfully used in recent studies concerning porous media [36, 37].

However, while domain decomposition strategies based on cubes, slabs or parallelepipeds [45–50] yield good computational and communication load balancing when applied to regular systems, they are not well-suited to complex sytems. Some domain decomposition strategies for lattice Boltzmann codes well-suited for non-regular lattices already exist, and we consider these briefly here. The state-of-the-art of domain decomposition, represented by multilevel $k-$way partitioning scheme [52, 53], can be used to partition non-regular systems. The need for users to explicitly implement these complex domain decomposition techniques can be avoided by using existing software such as the METIS library [54], which can be incorporated in parallel lattice-Boltzmann codes to obtain high quality domain decompositions and thus good execution performance on a large number of processors. However, even state-of-the-art parallel multi-level implementations require $O(N/10^6)$ seconds on $O(N)$ graph vertices [4] even when performed on $O(100)$ processors [53]. This leads to unreasonable elapsed times when tackling systems with $O(10^8)$ or $O(10^9)$ fluid lattice sites. Moreover, these multilevel partitioning algorithms require a substantial amount of memory consumption, which makes the use of a parallel approach compulsory as well as many processors even for systems which are not very large. Generally speaking, these multilevel methods use recursive bisection schemes for the initial partitioning. The interfaces between partitions are then recovered and load balancing is improved iteratively. As a consequence, these strategies have a large associated computational cost. Unfortunately, they do not guarantee optimal communication and computational load balancing.

The domain decomposition approach used in Pan *et al.* [63] is worthy of note. It combines a 1D data representation with connectivity [44], tested in Sec. 5.2, which is well-suited for sparse systems and an efficient parallelization strategy based on the orthogonal recursive bisection (ORB) algorithm [56]. In the ORB approach, the computational grid is decomposed into two partitions, such that the workload, due to fluid lattice sites only, is optimally

[4] In our case, a vertex in a graph corresponds to a fluid lattice site.

balanced. The same process is then applied recursively $k-1$ times to every partition, proceeding on the basis of orthogonal bisections. The total number of processors must be equal to 2^k. The amount of data required for an efficient implementation is proportional to the total number of fluid lattice sites N. Furthermore, the computational time scales as $O(Nlog(N))$; while the domain decomposition is straight forward to implement and assures good load balancing, it does not produce a satisfactory communication balance.

In the domain decomposition strategy employed by Wang *et al.* [68], as the system is read from file, N/P (P is the number of processors) fluid lattice sites are assigned to each processor consecutively. The number of fluid lattice sites per processor is either 0 or 1, depending on whether N/P is an integer or not. Thus, the resulting computational load balance is perfect. The recovery of the data to be communicated is straightforward since the fluid lattice sites responsible for the communications, called here "interface-dependent lattice sites", are stored in an orderly fashion [68]. Hence, the communication pattern is regular and no data flow discontinuities occur. The method is simple, assures perfect load balancing, does not require CPU time to optimize the decomposition and requires a memory consumption of $O(N/P)$ per processor, since global data are not necessary. Wang *et al.* [68] maintained their approach to be superior to that of the ORB technique. The domain decomposition approach itself is not new. In fact, it has been exploited in multilevel graph partitioning as an initial decomposition where it is referred as the *as-is distribution*. However, such domain decomposition can be afflicted by poor communication balance even for simple systems, as will be shown in Sec. 6.1. Finally, we note that space-filling curves [55] have been used to decompose sparse geometries in computational fluid dynamics (CFD) applications [65,67,67]. The quality and the speed of these partitioning strategies is sub-optimal, while the computational cost usuallly increases as $O(Nlog(N))$, making it too expensive for very large systems.

3 The HemeLB model

The lattice-Boltzmann model adopted in the HemeLB code is an incompressible version of the lattice Bhatnagar, Gross and Krook (BGK) D3Q15 one [11] (3-D model with 15 velocity directions) proposed by Zou *et al.* [12], dubbed D3Q15i. The approximation of the

continuum collision term is that suggested by Bhatnagar, Gross and Krook [13] for which the distribution function is assumed to evolve towards its local equilibrium value, at a rate controlled by a single relaxation parameter, τ:

$$\Omega \approx \frac{f^{(eq)} - f}{\tau}. \tag{1}$$

The resulting lattice BGK equation (LBGK) is

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) - f_i(\mathbf{x}, t) = -\frac{f(\mathbf{x}, t) - f^{(eq)}(\mathbf{x}, t)}{\tau} \tag{2}$$

and the equilibrium distribution functions are:

$$f_i^{(eq)} = w_i \Big(\rho + \frac{\mathbf{e}_i \cdot \mathbf{u}}{{c_s}^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2{c_s}^4} - \frac{\mathbf{u} \cdot \mathbf{u}}{2{c_s}^2} \Big), \tag{3}$$

where w_i is a weight coefficient, c_s is the speed of sound, $\mathbf{e}_i$ is the velocity of the particle along the direction i and the hydrodynamic density ρ and macroscopic velocity $\mathbf{u}$ are determined in terms of the distribution functions from

$$\rho = \sum_i f_i = \sum_i f_i^{(eq)}, \quad \mathbf{u} = \sum_i \mathbf{e}_i f_i = \sum_i \mathbf{e}_i f_i^{(eq)}. \tag{4}$$

The discrete velocities $\mathbf{e}_i$ and the weight coefficients w_i (see Zou *et al.* [12] for details) must be chosen in order to assure isotropic hydrodynamics.

4 Boundary conditions

Here, the boundary condition methods used in HemeLB are presented. Many boundary condition approaches have been investigated in the past, but few of them are well-suited to dealing with complex boundaries regardless of their orientation and shape. Probably the most studied and widely adopted wall boundary condition method is the bounce-back rule [14] for no-slip walls. Many boundary condition methods (e.g. [16, 19, 20]) are unable to handle the situation depicted in Fig. 4.

Instability issues and ambiguities, including those arising at boundaries where the normal and tangential components are not well defined, severely restrict the application of some

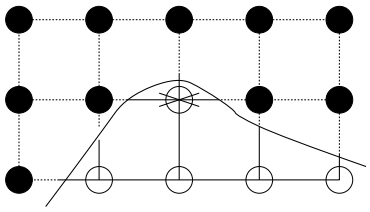

Figure 1: Schematic diagram of a smooth boundary for which both horizontal inward-pointing directions indicated by the arrows originate from solid sites. Lattice sites within the computational fluid domain are denoted by empty circles. Many boundary condition methods are unable to determine the unknown distribution functions at the central fluid lattice site.

boundary condition methods such as those used in [21] and [22]. The velocity boundary condition of Junk and Yang [23] handles the problem in Fig. 4, is second order accurate and thus attractive. However, it is very complex and computationally expensive, as it needs to store precise information related to the intersection between the boundary surface (which is not always available) and the lattice.

We have implemented and studied various boundary condition methods. The non-equilibrium extrapolation method of Zhao-Li *et al.* [15] handles any type of boundary shape efficiently and is simple, making it worth investigating. By contrast the method of Zou and He [24] has very restricted applicability, but we have nevertheless tested it for non-inclined axis-aligned boundaries where we can define the pressure (dubbed "pressure boundaries")[5].

4.1 New boundary condition method

We have also developed and tested a new and simple extrapolation method for fluid flows confined by pressure and no-slip boundaries, which is outlined here. Let F be a no-slip boundary fluid lattice site. After streaming of the known (outward-pointing) distributions

[5]Although this boundary condition method cannot be applied for complex boundaries, we decided to consider it because we found that its application for both the pressure boundaries and no-slip walls confining a rectangular duct yields results in accordance with 2D Poiseuille glow within machine accuracy, regardless of the Reynolds number, lattice resolution and viscosity. This excellent result is in contrast to that achieved by Zou and He [24] where such high accuracy was not reported.

$\{f_k(F, t+\Delta t)\}$ (the suffix k stands for "known"), the extrapolated density at the new time step is

$$\rho(F, t+\Delta t) = \sum_{f_k \in known} f_k(F, t+\Delta t) + \sum_{f_u \in unknown} f_u(F, t), \tag{5}$$

where $\{f_u(F,t)\}$ are the pre-collisional, inward-pointing, distribution functions (the suffix "u" stands for "unknown"). If F belongs to a pressure boundary, the extrapolated velocity is

$$\mathbf{u}(F, t+\Delta t) = \sum_{f_k \in known} \mathbf{e}_k f_k(F, t+\Delta t) + \sum_{f_u \in unknown} \mathbf{e}_u f_u(F, t), \tag{6}$$

and all the distribution functions at time $t+\Delta t$ are calculated from the following equations for no-slip walls and pressure boundaries respectively:

$$f_i(F, t+\Delta t) = {f_i}^{(eq)}\big(0, \mathbf{u}(F, t+\Delta t)\big), \tag{7}$$

$$f_i(F, t+\Delta t) = {f_i}^{(eq)}\big(\overline{\rho}, 0\big), \tag{8}$$

where $\overline{\rho}$ is the inlet or outlet pressure. When there are at least six unknown distribution functions at the inlet and outlet lattice sites, the method sets:

$$f_i(F, t+\Delta t) = {f_i}^{(eq)}\big(\overline{\rho}, 0\big). \tag{9}$$

The method is based on the following approximation:

$$f_u(F, t+\Delta t) \approx f_u(F, t), \tag{10}$$

which is reasonable if the flow is slow, that is the characteristic time scale is much larger than the spatial one (see [16] for details). The velocity and the pressure imposed by velocity and pressure boundaries respectively are constrained to be equal to the prescribed ones and thus, for example, slip velocity cannot occur at a zero-velocity no-slip wall.

We have tested our new boundary condition method by simulating the fluid flow in a 3D square duct system subjected to a pressure gradient provided by an inlet pressure which is

set to be greater than the outlet one. The analytical velocity profile is given by White [17]. The error analysis of the new extrapolation boundary condition method (NewE) was carried out using the D3Q15i lattice Boltzmann model (see Sec. 3 for details) and compared with the bounce back rule (BB) and the non-equilibrium extrapolation method (NEE) presented by Zhao-Li *et al.* [15]. The initial condition is the equilibrium distribution calculated with a unitary density and zero velocity, and the steady-state is considered to be reached if

$$\frac{\sum_i \sum_j |u_x(i,j,t+\delta) - u_x(i,j,t)| + |u_y(i,j,t+\delta) - u_y(i,j,t)|}{\sum_i \sum_j |u_x(i,j,t)| + |u_x(i,j,t)|} \leq \delta Tol, \tag{11}$$

where $1 \leq i \leq nx$ and $1 \leq j \leq ny$ define the site location in every section perpendicular to the square duct axis, $lx \times ly \times lz = (nx-1) \times (ny-1) \times (nz-1)$ is the system size in lattice unit and $Tol = 10^{-8}\delta$ with $\delta = 1/ly$. The maximum relative error is calculated to be

$$err_m = \frac{1}{u_0} max \sqrt{(u_x{}' - u_x)^2 + (u_y{}' - u_y)^2 + (u_z{}' - u_z)^2}, \tag{12}$$

where ($u_x{}'$, $u_y{}'$, $u_z{}'$) is the analytical velocity and the maximum error is over the entire lattice [18, 24]. The pressure at the inlet and outlet is regulated by means of the method of Zou and He [24] (dubbed "CoP"). In Table 1, we report the maximum relative error, the time steps required for convergence and the order of convergence, from a least-squares fit, for some Reynolds numbers and relaxation parameters (the last two parameters were chosen according to Zou and He [24]).

From Table 1, it emerges that our new boundary condition method is first-order accurate while its results are always more precise than the bounce back rule and the method of Zhao-Li *et al.* [15], except for the fastest flow in combination with the two smallest lattice sizes only, where the bounce back rule provides higher accuracy. From the method of Zhao-Li *et al.* the results are first-order accurate only (although they were asserted to be second order accurate in the original work [15]). Moreover, it is clear that our new method produces faster convergence rates than the others. For $\tau = 0.6$, the simulation with the new boundary condition method is stable at $Re = 120$, while with the non-equilibrium extrapolation method and bounce-back instabilities occur at $Re = 100$ and $Re = 105$ respectively. Furthermore, the new scheme is very simple to implement and very efficient since a subtlety, which will

Table 1: Maximum relative errors and orders of convergence, carried out with a least-squares fitting procedure, for 3D square duct flow with various boundary condition methods (see Sec. 4 for their acronyms and references). The symbol $(-n)$ represents 10^{-n} and the numbers in square brackets are the iterations required to reach convergence.

	$lx \times ly \times lz$	$8 \times 4 \times 4$	$16 \times 8 \times 8$	$32 \times 16 \times 16$	$64 \times 32 \times 32$	order
	CoP+NEE	3.963(-1)	5.234(-1)	2.404(-1)	1.171(-1)	0.6399
		[417]	[1959]	[6842]	[25834]	
Re = 10	CoP+BB	3.071(-1)	2.601(-1)	1.476(-1)	7.894(-2)	0.6697
		[697]	[2345]	[12140]	[60707]	
$\tau = 0.6$	CoP+newE	7.239(-1)	2.882(-1)	1.227(-1)	6.166(-2)	1.189
		[224]	[1266]	[5676]	[23154]	
	CoP+NEE	1.134(-0)	7.893(-1)	3.855(-1)	1.892(-1)	0.8784
		[239]	[778]	[2616]	[9446]	
Re = 5	CoP+BB	4.128(-1)	2.874(-1)	1.496(-1)	7.802(-2)	0.8153
		[325]	[1545]	[7553]	[35603]	
$\tau = 0.8$	CoP+newE	2.688(-1)	1.305(-1)	6.062(-2)	3.425(-2)	1.002
		[130]	[520]	[2132]	[8434]	
	CoP+NEE	1.460(+0)	9.384(-1)	5.284(-1)	2.805(-1)	0.7968
		[148]	[448]	[1450]	[5026]	
Re = 0.2	CoP+BB	8.535(-1)	3.840(-1)	1.724(-1)	8.320(-2)	1.123
		[348]	[1550]	[6932]	[30634]	
$\tau = 1.1$	CoP+newE	3.557(-1)	1.219(-1)	5.151(-2)	2.723(-2)	1.236
		[105]	[336]	[1210]	[4546]	

be presented in the next section, enables us to avoid calculating the equilibrium distribution functions twice at each boundary lattice site.

4.2 Efficient implementation of the new boundary condition method

In our new boundary condition method, the distribution functions at the velocity boundary are set equal to the equilibrium ones calculated on the basis of the assigned boundary velocity and the extrapolated density. For pressure boundaries, the density is instead assigned while the velocity is extrapolated. Rather than performing this calculus at the end of the streaming step, it can be done before the collision step and prior to writing the output results, without affecting the final results. At this point, it is evident that we need to calculate the equilibrium distribution functions at each boundary lattice site only once. Furthermore, at a no-slip wall the velocity is zero and many terms needed to calculate the equilibrium distribution functions at each lattice site then vanish. Thus, their computation requires a very small number of operations.

5 The computational core of HemeLB

In this section, we present a simple data structure which is adapted to sparse systems by avoiding retention of almost all information about the void component while achieving good data locality. The grid of the problem domain is split into a two-level representation: if any fluid lattice sites are present within a cell of the coarse grid, this is decomposed into a finer one. Multi-level and hierarchical grids have previously been applied to lattice-Boltzmann simulations [21, 39–42]. These approaches have been used in the context of grid refinement techniques to capture with sufficient accuracy small-scale hydrodynamic behaviour within one or more sub-domains of the simulation domain. However, in our case no data are exchanged between different grids belonging to the same level of resolution, but are passed instead to their parent coarser grids. In our approach, a two-level data structure is used to represent a domain with a single (unique) resolution. No data exchange takes place between the coarser (parent) grid and the finer one: the distribution functions are propagated within or between the latter only. This two-level structure saves memory with respect to

the "full matrix" representation when one or more parent cells are completely occupied by non-fluid lattice sites since corresponding finer grids cannot then be allocated. Furthermore, interpreting each finer grid as a sub-domain for which good cache use takes place, spatial locality is automatically achieved as long as data located at the interfaces accross different sub-domains, which we call "blocks", are needed infrequently. For one single loop over the lattice sites of a block with side b in lattice units, the ratio between those interfacing with neighbouring blocks and the interior ones scales as $1/b$. On the other hand, on increasing the block size cache utilization is likely to become worse, as occurs in medium and large regular systems.

Multi-level grids with more than two levels can be adopted. However, we found that a two-level grid with $b = 8$ represents a good tradeoff in terms of simplicity, cache use and adaptivity to sparse systems. With this choice, if the coordinate along the x axis of a lattice site is i, those of the corresponding block and its interior site are $bi = i >> 3$ and $si = i - (bi << 3)$ respectively [6]. However, although this mapping between the coordinates of a lattice site in the single level representation and those in the two-level one is easy to implement, it is computationally expensive and should not be used during the propagation step. We use instead a faster approach based on a look-up table of precomputed indices; this is described in the next section.

5.1 Benchmarks on regular systems

In the present sub-section, timing results of standard lattice-Boltzmann implementations for regular systems are compared to those obtained with the two-level data structure presented in Sec. 5. The benchmarks have been carried out by adopting the D3Q19 model [11] for cubic systems of N^3 lattice sites. The bounce back rule is applied to the no-slip walls surrounding the systems. 3D and 4D data are stored in 1D arrays and a pre-computed lattice grid of integers `is_solid[N*N*N]` keeps track of the presence of the boundary conditions for each velocity direction l and lattice site n:

```
is_solid[ n ] & (1 << l) =
```

[6] In C, << and >> shift the data to the left and right respectively.

a) = 1 if its l-th bit is 1 and the location corresponding to `(i, j, k) + dir(l)` occupies the solid part;

b) = 0 elsewhere.

This procedure allows us to investigate a simple problem while maintaining a certain level of generality related to our objective of addressing simulations of sparse systems. The implementations tested were not tuned to any specific hardware. Apart from the adoption of the 3D loop blocking technique in one implementation and an efficient way of referencing the lattice sites, no further optimizations have been implemented. Some data structures and optimizations have previously been investigated [26–29,31]. The code to handle the collision and propagation phases (in one step only) by means of the "collision-optimized" (or "array-of-structures" data layout) with the "push-scheme" [28] is illustrated in the following C lines (some declarations are omitted for brevity):

```
//sourse and destination buffers
//of distribution functions
double *old_f;
double *new_f;

old_f = (double *)malloc(sizeof(double) * N * N * N * 19);
...
// for each time step perform the following:
for (i = 0; i < N; i++)  {
  for (j = 0; j < N; j++)  {
    for (k = 0; k < N; k++)  {
      Feq (&f_old[ijk = (i * N + j) * N + k], f_eq);

      for (l = 0; l < 19; l++)  {
        ijkl = ijk * 19 + l;
        f_old[ijkl] += omega * (f_old[ijkl] - f_eq[ l ]);
      }
      for (l = 0; l < 19; l++)  {
        ijkl = ijk * 19 + l;
        if (!(is_outside[ijk] & (1 << l)))
        {
          f_new[(((i+e_x[l])*N+(j+e_y[l]))*N+(k+e_z[l]))*19+l] = f_old[ijkl];
        }
        else
        {
          f_new[ ijk * 19 + inv_dir[l] ] = f_old[ijkl];
        }
}}}}
temp = old_f;   // the buffers are swapped
old_f = new_f;
new_f = temp;
```

where `Feq ()` is the routine that calculates the local equilibrium distribution functions, `omega` is $-1/\tau$, `ex[l]`, `ey[l]`, and `ex[l]` are the three components of the lattice direction `l` and `inv[l]` indicates the opposite direction to `l`. We refer to this implementation as the "standard" one. We note that many operations can be avoided in the standard implementation. For example, $(((\texttt{i} + \texttt{e\_x}[\texttt{l}]) * \texttt{N} + \cdots) * \texttt{19} + \texttt{l}$ depends only on the reference lattice site[7], `N` and `l`, and has the form $\texttt{ijk} + \texttt{f\_inc}[\texttt{l}]$ where `f_inc[]` is a precalculated integer array of 19 elements. Exploiting this array and avoiding unnecessary additions and multiplications as far as possible, an optimized standard code is produced. This is called "Opt Standard" and has the following structure:

```
double *f_old_a, *f_old_b;
double *f_new_p;

int *is_out_p;
...
// for each time step perform the following:
f_old_a = f_old;
f_new_p = f_new;
is_out_p = is_outside;

for (i = 0; i < N*N*N; i++)  {
  f_old_b = f_old_a;
  lbmFeq (f_old_a, f_eq);

  for (l = 0; l < 19; l++)
    {
      *f_old_a += omega * (*f_old_a - f_eq[ l ]); ++f_old_a;
    }
  for (l = 0; l < 19; l++)
    {
      if (!(*is_out_p & (1 << l)))
        {
          *(f_new_p + f_inc[ l ]) = *f_old_b;
        }
      else
        {
          *(f_new_p + inv_dir[ l ]) = *f_old_b;
        }
      ++f_old_b;
    }
  ++is_out_p;
  f_new_p += 19;
}
temp = old_f;   // the buffers are swapped
old_f = new_f;
new_f = temp;
```

[7]The access to an nD array in C requires n additions. For example, `ex[l]` is equivalent to `*(ex + l)`.

Our new method for handling sparse geometries efficiently is based on a two-level data structure that can be represented by a 2D array instead of a 1D array, with one dimension for the block identifier and one to identify the lattice site and the velocity direction. For the purpose of comparison with respect to the optimized standard implementation, the arrays used are again chosen to be 1D. In this way, the collision and propagation steps are identical and the single factor that makes the difference in performance is the data reordering due to the two-level representation. However, a strategy is needed to identify efficiently the neighbouring lattice sites involved in propagation. We observe that each lattice site has a neighbouring one whose parent block coordinates differ from the reference one by -1, 0 or 1. Furthermore, the identifier of the neighbouring lattice site in its block always ranges between 0 and $8^3 - 1$.

At this point, a neighbour map can be constructed based on the foregoing considerations. Since the data are stored in 1D arrays, a 1D lookup table `f_map[]` of $8^3 \times 19$ elements can incorporate all the information regarding the increment in coordinates associated with the neighbouring block and the neighboring lattice identifier. The lookup table is constructed in the following manner:

```
// declarations
m = -1;
for (i = 0; i < 8; i++)  {
  for (j = 0; j < 8; j++)  {
    for (k = 0; k < 8; k++)  {
      for (l = 0; l < 19; l++)  {
        ni = i + e_x[l];
        nj = j + e_y[l];
        nk = k + e_z[l];
        si = ni - ((bi = ni >> 3) << 3);
        sj = nj - ((bj = nj >> 3) << 3);
        sk = nk - ((bk = nk >> 3) << 3);
        block_inc = (bi * (N>>3) + bj) * (N>>3) + bk;
        site_id  = (((si << 3) + sj) << 3) + sk;
        f_map[ ++m ] = (block_inc * (8*8*8) + site_id) * 19 + l;
}}}}
```

where (i, j, k) and (ni, nj, nk) are the coordinates of the reference lattice site and those of the neighbouring one in the single level representation respectively, (si, sj, sk) are the coordinates of the latter in the two-level representation and (bi, bj, bk) the increments in coordinates of the neighbouring block with respect to the reference one. The resulting implementation, called "newA (8x8x8)", is based on `f_map[]`. If the two-level representation is based on blocks of

2^3 lattice sites, the lattice site identifiers are reordered as in the Morton scheme [35]. The resulting implementation is denoted "newA (2x2x2)". The array `is_solid[]` is still exploited to handle no-slip boundaries, but it has a two level representation in this case.

We note that the array `is_solid[]` is not needed as long as the neighbouring lookup table is extended to identify the neighbouring lattice sites for all velocity directions, at the expense of significant memory consumption. The neighbouring lookup table is represented in two-level fashion and the resulting full connectivity implementations are dubbed "newB" and "newB+" respectively, according to whether the effect of the inclusion of the "IF" branch related to the bounce back rule is considered or not. In "newB" the indices are set equal to the key value "-1" whenever the corresponding neighbouring locations reside in the wall.

An implementation of the 3D blocking technique with loop blocking of 8, denoted "blocking (8x8x8)" has been applied to the optimized standard implementation in order to increase space coherence, cache reuse and thus performance for the simulation of large systems. All the implementations tested have undergone the same level of optimization. Usually, a halo region of lattice sites surrounding the system handles the bounce back rule more efficiently than the "IF" branch present in the propagation step. However, fusion of streaming and collision steps, fast referencing and an optimized implementation of the equilibrium distribution functions guarantee rapid executions here.

Apart from the possibility of tuning the codes presented in this section to the specific hardware used, the version which implements the 3D blocking technique represents a state-of-the-art LB implementation which is well-suited to regular systems when a predefined order involving the propagation step, as in the compressed-grid and 4-way blocking algorithms [26, 29], is not exploited. Furthermore, these methods cannot be applied to sparse systems where data blocking yields better performance than those achieved using mesh reordering schemes [44]. Our benchmarks for the LB implementations have been conducted using an Opteron 2 GHz with a L2 cache of 1024 KB, Linux kernel 2.6.19 and an IBM Power4 1.7 GHz with 1.9 MB L2 cache and IBM AIX OS. The compiler used on the Opteron is the Intel version 8.1 with flags $-$`O3` $-$ `unroll` $-$ `unroll` $-$ `loops` $-$ `IPF` $-$ `fma` $-$ `axW` $-$ `xW` $-$ `inline` while on the Power4 (RISC architecture) the executable was produced with `xlc` $-$`qstrict`

`-qstrict_induction -O5 -qipa=level=2 -qhot=vector -qcache=auto -lm -bmaxdata:0x70000000`. Double precision is used to calculate local equilibrium distribution functions and their moments. The performance as a function of the cube root of the total number of lattice sites is measured in millions of site updates per second (MSUPS) and is shown in Fig. 2 for the codes presented in this section.

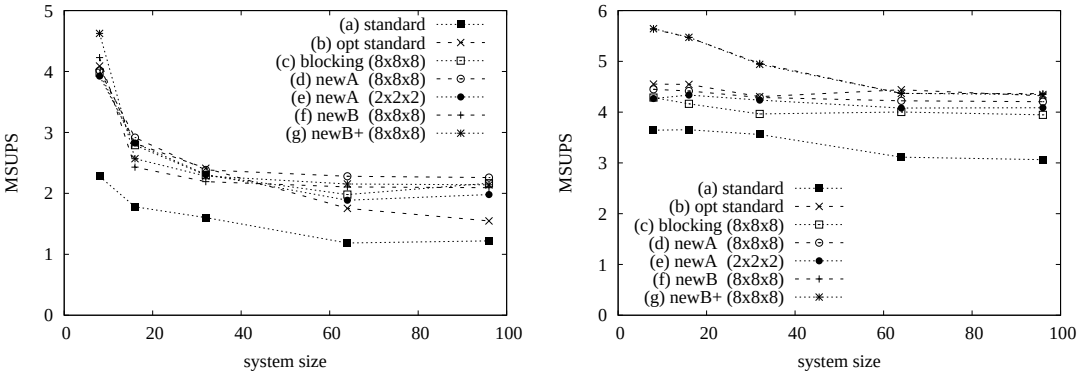

Figure 2: Performance measured in millions of lattice site updates per second (MSUPS) as a function of the cube root of the total number of lattice sites for (a) the standard ("standard"); (b) its optimized counterpart ("opt standard") and (c) its 3-D loop blocking version with blocking factor of 8 ("blocking (8x8x8)"); that using the two-level representation with (d) a look-up table of indices used in the propagation step with blocks of 8^3 ("newA (8x8x8)") and (e) 2^3 ("newA (2x2x2)") lattice sites respectively; (f) a two-level buffer of indices to handle the propagation step ("newB (8x8x8)"), and (g) as in (f) supplying the bounce back rule without the "IF" branch ("newB+ (8x8x8)"). The timing results have been conducted on an Opteron 2 GHz (left) and a Power4 1.7 GHz (right) (see Sec. 5.1 for further details).

The superiority of implementations which adopt the two-level data structure with respect to the standard one and its optimized version is evident on the Opteron machine. On the Power4, the large cache and extensive hardware support for prefetch, branch prediction, and both out-of-order and speculative execution of instructions optimize caches use and execution on large systems. Thus, the performance degradation for larger systems is low and

the simple optimized standard implementation performs very well. The implementations "newA (8x8x8)" and "newB (8x8x8)" exhibit similar performances on both processors and are often the fastest executions. The two-level implementation with full connectivity "newB+ (8x8x8)" is slightly faster than "newB (8x8x8)" due to the elimination of the expensive "IF" branch in the propagation step. The standard implementation is substantially inferior to its optimized counterpart. Therefore, we stress that redundant operations cannot always be avoided by modern compilers and should be eliminated by the programmer. On the Opteron, the 3D loop blocking implementation exhibits significantly higher performance than the non-blocking counterpart, indicative of good cache reuse. Two-level data representations deliver very high performance. It is interesting that the two-level data structure with 2^3 blocks (Morton) does not produce the same benefit as an implementation with larger blocks. Finally, we have demonstrated that data reordering techniques can produce higher performance than typical cache-aware optimizations such as loop blocking.

5.2 Benchmarks on sparse systems

Here, we compare the performance of the two-level data layout with that achieved by the standard approach normally used for LB simulations of fluid flow confined in sparse geometries as presented by Donath *et al.* [44]. In the standard implementation, the single level representation is constructed by scanning respectively over the x, y and z directions, while in our new strategy the two-level representation is constructed following the allocated fluid blocks directly.

We carried out the benchmarks by applying the incompressible D3Q15i model (see Sec. 3) to a set of bifurcations of different sizes. The simulation cells are composed of 16^3, 32^3, 64^3, 128^3 and 256^3 lattice sites with 225, 1601, 12449, 98211 and 780359 fluid lattice sites respectively. The volume rendering of the velocity field of the largest system (using $\tau = 0.6$ and characterized by a peak velocity of 0.2) is shown in Fig. 3. The collision and propagation steps are implemented as in the piece of C code labelled "newB" in Sec. 5.1. The boundary conditions implemented are those presented in Sec. 4.1. During streaming the "IF" branch is avoided at boundary lattice sites by allocating $N * 15 + 1$ values (N is the number of fluid

lattice sites and 15 the number of velocity directions) for the source and destination buffers: the last (ghost) value of the destination buffer `f_new[]` is referenced when a particle streams towards the wall.

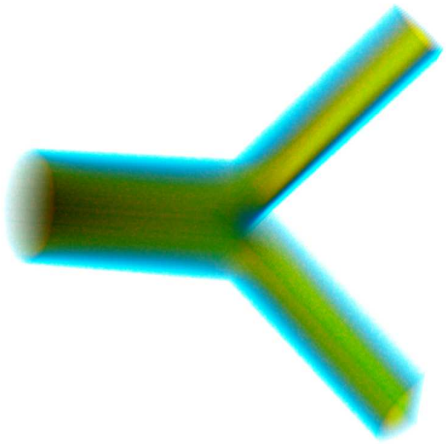

Figure 3: Volume rendering of the velocity flow field in a bifurcation of 780359 fluid lattice sites characterized by $\tau = 0.6$ and a peak velocity of 0.2 lattice units. The wall and pressure boundary conditions applied to the inlet and outlet lattice sites and no-slip walls respectively are discussed in Sec. 4.1. Yellow and blue colours represent maximum and minimum (zero) speeds respectively.

The processors, operating systems and compilers are those used in Sec. 5.1 for the benchmarks on cubic systems. The performance in MSUPS is shown in Fig. 4 for the standard and new approaches described in this section.

We first note the excellent performances attained with both kinds of processors. Our new approach exhibits higher performance than the standard technique on the two largest systems using the Opteron and very similar behaviour in the other cases. As anticipated, the two-level-grid-based approach yields better performance on large systems than the standard technique because data are likely to reside more closely in memory address space and thus in cache. The ratio between wall fluid sites and others is greater for smaller systems. Since the wall fluid sites are handled with less operations than the others (see Sec. 4.2 for details), the aftorementioned ratio as well as cache use affect the performance behaviour as a function

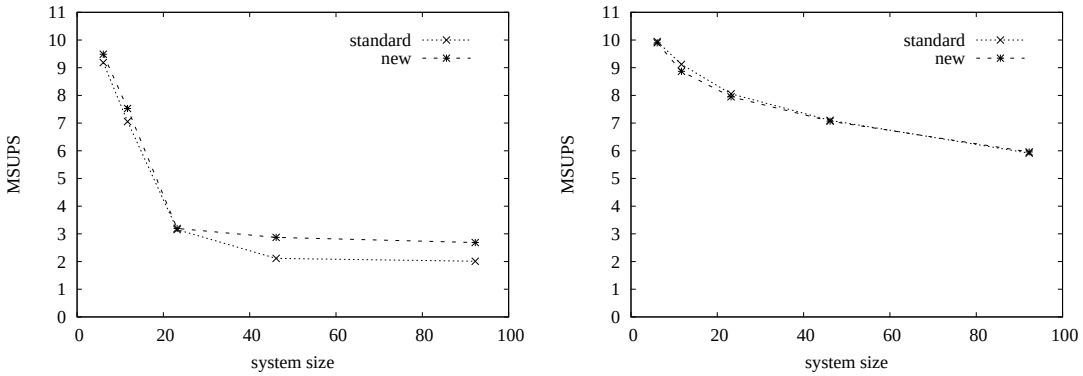

Figure 4: Performance measured in millions of site updates per second (MSUPS) as a function of the cube root of the fluid lattice sites for the standard ("standard") and the new appoach described in Sec. 5.2. The timing results were conducted on an Opteron 2 GHz (left) and a Power4 1.7 GHz (right) (see Sec. 5.2 for further details).

of system size. In future work we will establish the performance of this algorithm for more complex systems in which the corresponding pattern irregularity in address space is higher. This feature causes a decrease in performance in standard single level representations [44] and, consequently, our new approach is expected to have a relatively greater increase in performance since the irregularity affects only the effective size of blocks used in the finer component of the two-level grid.

6 The parallel implementation

In this section we first present the parallelization strategy adopted in HemeLB (Sec. 6.1) which is shown to be well-suited to simulate fluid flow in large, complex systems. Then, our new partitioning strategy is presented and compared with other ones in terms of advantages and disadvantages. The new domain decomposition approach is found to be very fast and of high quality (Sec. 6.1 and 6.3). The computational performance achieved with the adoption of the domain partitioning tecnique presented here is reported in Sec. 6.3.

6.1 The domain decomposition

Here, a domain decomposition approach based on a graph growing partitioning (GGP) algorithm is presented. GGP partitioning techniques have been presented and used in the past [57–59], but have never been adopted in LB simulations. The GGP scheme does not scan the system in a predefined order and, as a consequence, it is afflicted by one of the main limitations of many other domain decomposition strategies: the need to store global data for efficient execution. Moreover, while the computational cost is optimal with respect to the number of elements to distribute it remains $O(N/10^6)$ seconds on $O(N)$ elements. Nevertheless, the nature of the GGP is likely to yield good communication and computational load balance. In this section, the GGP algorithm is reviewed in the context of LB simulations of sparse systems and then a novel modification is presented which overcomes limitations related to the global data requirement and inherent speed. Finally, the important impact of the proposed GGP scheme on the output is outlined.

At the beginning of the GGP scheme, all the fluid lattice sites are marked as "non-visited". A fluid lattice site is picked and marked as "visited" and assigned to the processor with rank 0. An iterative search starts from this lattice site and grows a region around it until the desired number of fluid lattice sites N/P is reached, as explained below. The expansion proceeds by means of two "coordinate" buffers that are used to locate the fluid lattice sites at the current and next iteration respectively. For example, at the first iteration, the first coordinate buffer will point to the first selected fluid lattice site only, while the second one will contain the coordinates of the non-visited neighbouring fluid lattice sites. At the end of every iteration the two aforementioned buffers are swapped. The propagation from every fluid lattice site in the first coordinate buffer to the neighbouring non-visited ones is constrained to follow the velocity directions in the LB model used. When the number of fluid lattice sites reaches N/P, a set of $P-1$ similar iterative searches, one for every processor, starts from other non-visited fluid lattice sites until the total number of fluid lattice sites visited is N. Particular care must be taken if an iterative search cannot continue because the visited fluid lattice sites in the first coordinate buffer are completely surrounded by solid lattice sites, but the number of visited fluid lattice sites has not yet reached N/P. In this

case, another iterative search from a non-visited fluid lattice site is started but the processor count is not incremented.

We note that, in general, each fluid lattice site is checked more than once: the iterative search must start from a non-visited lattice site which is not known *a priori*; the expansion does not proceed in a predefined order and, inevitably, the iterative search checks many fluid lattice sites to see if they are available (not visited). A dynamic look-up table of the non-visited fluid lattice sites available at every iteration, and the velocity directions from which the visited fluid lattice sites originate, speeds up the iterative search by minimizing repeated accesses at the expense of an increase in memory consumption. However, the algorithm updates $O(10^6)$ fluid lattice sites per second on modern processors, and, consequently, the domain decomposition requires minutes for a system of hundreds of millions of fluid lattice sites. Furthermore, the memory consumption is $O(N)$ bytes.

Our improvement on the naive GGP algorithm relies on the two-level representation presented (Sec. 5). The input file is read a first time to store the number of fluid lattice sites for every block of the coarsest grid [8]. The resulting lookup table, dubbed `fluid_sites_per_block`[], is global but has a very low memory overhead [9]. The GGP algorithm is then applied to the coarse grid only. Here, the fluid lattice sites of a block are assigned to a processor. If the accumulated number of fluid sites for a processor exceeds $np = N/P$, the successive search starts with $(np - N/P)$ previously visited fluid lattice sites to prevent any fluid lattices sites being assigned to processors with larger rank, which would result in poor load balancing. The partition is stored in another global lookup table equivalent to `fluid_sites_per_block`[], `proc_id`[].

This coarse-level-based GGP approach plays a crucial role in managing the overall computational and memory cost as well as in the output of the results. First of all, the computational cost is reduced by three orders of magnitude if each block is composed of $O(1000)$ lattice sites [10]. This allows us to decompose a system of $O(10^9)$ fluid lattice sites in a sec-

[8] This step can be avoided if it is accomplished directly in a preprocessing stage.

[9] For a simulation cell of 1024^3 fluid sites the memory requirement is around 4 MB only, assuming that every block contains 8^3 lattice sites and 2 bytes are needed for every element of `fluid_sites_per_block`[].

[10] This was the choice we adopted since every block contains $8^3 = 512$ lattice sites.

ond. Moreover, each processor will need to store detailed information on the lattice sites contained only in the blocks marked with its identifier (rank) and their neighbouring ones. This is accomplished by reading the input file a second time and using `proc_id[]`. In this way, the memory consumption is kept of $O(N/P)$ plus a small amount due to the global lookup tables `fluid_sites_per_block[]` and `proc_id[]` [11]. 2-D representations of the result of the proposed domain decomposition method applied to a square and a bifurcation with 128^2 and 61451 fluid lattice sites respectively are provided on the right hand side images of Fig. 5, while Wang *et al.*'s method [68] produced the images on the left hand side. There are always 16 partitions and each block of the finer resolution of the two-level representation contains 8^2 lattice sites. The grey-scale helps one to recognize the identifier of each partition. The two methods always assign to each partition 1024 lattice sites in the square system and this results in perfect load balancing. However, the shape of the partitions influences the communication balance. With our new partitioning method the minimum and maximum number of interface-dependent fluid lattice sites are halved and thus, inter-processor communications are reduced substantially. In the bifurcation, the smallest and largest partitions produced with the new technique contain 3439 and 3792 fluid lattice sites respectively and, thus, the load balance would be sub-optimal. While Wang *et al.*'s method [68] assures perfect load balancing, its interface-dependent fluid lattice sites in the smallest and largest partitions are respectively 190/212 and 2370/876 of those attained with our new method [12].

The parallel LB scheme presented by Schulz *et al.* [64] can overlap the inter processor non-blocking communications with the computation of the interface-independent fluid lattice sites. This may result in great benefit since the communications may be partially or completely hidden. We found that the standard vendor MPI imlementation (vMPI) used to handle inter-processor communications does not overlap non-blocking communications with computation regardless of the amount of (perfect) computational balance imposed. As a consequence, achieving good computational and communication balance turns out to be crucial.

[11] Eventually, the memory due to `fluid_sites_per_block[]` can be freed before any local $O(N/P)$ memory allocation.

[12] The differences in the values are also due to the presence of system boundaries and hence, the inner partitions are more likely to share a larger number of distribution functions with the neighbouring ones.

Moreover, a faster computational core makes the effect of large and unbalanced communications even more critical [45, 68]. From Fig. 5, we can say that Wang *et al.*'s method [68] is a generalization of the slab-based partitioning technique for sparse systems and inherits the latter's communication imbalance even when P is very low with respect to N. In contrast, the success of GGP techniques, including that presented here, resides in the nearly-spherical propagation of the visited fluid sites at each iteration[13]. However, perfect load balancing of the sort that characterizes the original GGP approach is no longer guaranteed, since the search proceeds for arbitrary blocks of lattice sites, for which a sub-optimal load balancing is always achieved.

6.2 Optimizations

We now discuss a variety of optimizations incorporated in HemeLB. The resulting code is very fast yet hides the considerable complexity inherent in simulations of sparse systems, minimizing the amount of intra-machine communication with respect to a specific domain decomposition while optimizing inter-machine communication in cross-site runs in grid deployments during each LB time step, keeping low those communications needed for the output of flow fields.

6.2.1 Minimization of communications

Sparse and complex systems present various difficulties in terms of parallel implementations. Generally, following domain decomposition, the identifiers of the distribution functions to be communicated and the location of the corresponding fluid lattice sites must be determined since they are not known *a priori*, as would be the case for regular domain decompositions. One parallel approach which does not need this step is that of Wang *et al.* [68].

[13] In general, the precise shape of the advancing front involved in the GGP search depends on the graph to be partitioned, the presence of obstacles and of visited graph vertices. At each iteration of the GGP search, the visited fluid lattice sites involved at the previous step are arranged in a cube for the D3Q15, D3Q18, D3Q19 and D3Q27 LB models if the propagation has not encountered any obstacle.

Communicating the identifiers doubles[14] the size of inter-processing communications at each time step. In HemeLB this is avoided; instead, only the distribution function values must be communicated. After domain decomposition is achieved, each processor calculates its interface-dependent identifiers. Let us consider only two neighbouring processors A and B without loss of generality. A and B have precomputed a buffer, labelled `f_id`[] of identifiers [15]. In general, `f_id`[`m`] calculated by processor A is not equal to that of B for any m. This unmatching of identifiers is overcome performing one single point-to-point communication from A to B if A is smaller than B. In this way, the buffer `f_id`[] of B is overwritten with that of A. This procedure needs to be applied to every neighbouring processor in the pre-processing phase only. During each LB time step, only the values of the interface-dependent distribution functions need be communicated.

6.2.2 Transparency of implementation complexity

As we have seen in Sec. 5.1, removing redundant operations and complex buffer access can reduce the execution time considerably. The implementation complexity pertaining to sparse systems is a source of many possible optimizations. The optimization presented in Sec. 6.2.1 represents a first improvement in this direction because the use of extra buffers of indices and many associated operations are avoided[16]. In HemeLB, two-level representations (see Sec. 5) are represented by 1D arrays. Thus, access to an element of an array requires minimum computational cost. The complexity associated with sparse systems is hidden in extra connectivity buffers. One of them is responsible for the on-processor propagation of particles between fluid lattice sites. The streaming of particles between two different processors takes place in three steps: (a) copying of the distribution functions from the source buffer `f_old`[] to that to send to the neighbouring processor `f_to_send`[]; (b) copying of the latter buffer into the receiving one `f_to_recv`[] by means of communication; (c) copying of the

[14]Usually, each distribution function is stored in 8 bytes and its velocity direction and the location of the interface-dependent fluid lattice site cannot be stored in only 4 bytes for large systems.

[15]In the HemeLB program, each element contains 8 bytes, 6 for the global lattice location and 2 for the velocity direction involved in the communication.

[16]Access to a n-dimensional array involves n additions in C. The situation is worse in programming languages like Fortran where many additions and multiplications are required.

distribution functions from the receiving buffer `f_to_recv`[] to the destination one `f_new`[]. Steps (a) and (c) represent the difficulty that must be faced if one wants to avoid the use of extra buffers corresponding to multiple indices, because the interface-dependent fluid lattice sites are not arranged regularly and are not known *a priori*. The problem is solved using two extra buffers of indices with 4 bytes per element: `f_old_id`[] and `f_new_id`[] set during the preprocessing step following domain decomposition through the buffers `f_id`[] and a two-level local lookup table `f_map`[] that maps from the lattice locations to the source and destination buffer indices. In this way, the steps (a) and (c) are simply accomplished with the following lines of code

```
f_to_send[ : ] = f_old[ f_old_id[:] ];
...
f_new[ f_new_id[:] ] = f_to_recv[ : ];
```

where the symbol : means that all the elements of the arrays must be referenced.

At first, the streaming towards void or fluid lattice sites covered by neighbouring processors which must be handled by communication appears to be avoided only by including an "IF" branch indicating whether the neighbouring lattice site is contained on an adjacent processor or not. However, the "IF" branch may be eliminated by connecting the source buffer to a ghost value of the destination one. Thus, the physically meaningless streaming of distribution functions towards solid lattice sites and/or their allocation to adjacent processors only has the effect of modifing this ghost value. If the partition has N/P values, the ghost one is covered by the $[(N/P) * 15 + 1]$-th element. The source buffer must also be sized with $(N/P) * 15 + 1$ elements, since at the end of the LB time step they are swapped.

With the adoption of the features presented in this section, the resulting code becomes very simple as the programming complexity caused by sparse representations is then completely transparent. Moreover, the optimizations presented in this section and the previous one permit us to maintain a low overall memory consumption and reduce the execution time by more than 30%.

6.2.3 Optimization of inter-machine communication

The exploitation of multiple machines permits us to tackle "grand-challenge" problems including very-large-scale fluid flow simulations. Indeed, many problems in biological and physical sciences require computational resources beyond any single supercomputer's current capacity [69–72]. The advent of cross-site simulations is already taking place today by means of the various geographically distributed MPI interfaces, such as MPICH-G2 [73] and its new version MPIg [74]. A first investigation of the feasibility and scalability of cross-site HemeLB simulations using geographically distributed domain decomposition (GD^3) [72] is presented in Sec. 6.3. The communication between geometrically remote machines is characterized by a latency one or two orders of magnitude greater than that affecting intra-machine data exchanges while the inter-machine bandwidth is considerably lower than the intra-machine one. Thus, the optimization of inter-machine communications is essential for such an approach to be viable.

The GGP technique produces partitions of high quality when the ratio between the total number of fluid lattice sites and the number of processors used is large. When this ratio is small, the system can be firstly subdivided into m parts with m small ($O(10)$), then each part in a further m parts and so on. This approach is equivalent to the ORB method if $m = 2$ and the partitions are obtained proceeding from orthogonal bisection. Thus, if the number of machines used is m, the strategy adopted is to subdivide the system into m partitions. The size of each one is chosen to be proportional to the number of processors of the corresponding machine. Even though the topology is described in the Globus RSL (resource specification language) job description file, the application is unable to access this file to discover the topology. Another method is required to inform the application about the topology; two possible methods are (a) to create an extra input file to specify the topology or (b) to use MPIG routines to discover it. The latter is the preferred option as it does not require maintainance of two distinct files. Thus, we have implemented a topology discovery mechanism based on the MPI function `MPI_Attr_get` in HemeLB in order to detect the number of machines used and the number of processors for each machine. Furthermore, the inter-machine non-blocking communications are split out from the others, then hidden in the computation

of the other communications and the computation of interface-independent fluid lattice sites whenever possible, following the scheme of Schulz *et al.* [64]. Briefly, in this parallelization strategy, after the collision of the interface-dependent fluid lattice sites the propagation of their particles to the neighbouring processors is initiated by means of non-blocking communications. Completion of these communications is enforced only after the computation of interface-independent fluid lattice sites. The topology discovery implementation, the two-level GGP decomposition and the adoption of the Schulz *et al.* parallelization scheme [64] permit us to (a) optimize inter-machine communications so as to hide high latency and low memory bandwidth and (b) to tune cross-site simulations to specific heterogenous resource distributions. Unfortunately, between MPIg, MPICH-G2 and vMPI, only MPIg hides inter-machine communications via non-blocking communications within the supercomputing resources used in our benchmars. The current parallel implementation of HemeLB does not optimize the domain decomposition to take into account the different performances of the multiprocessor machines that may be present in a grid. However, this heterogeneity can be taken into account simply adopting the algorithmic scheme used to consider the different number of processors used in each machine.

6.2.4 Attainment of pattern regularity

Except for the partitioning scheme presented in Wang *et al.* [68], in which the interface-dependent fluid lattice sites are stored contiguously, more sophisticated domain decomposition techniques are prone to yield pattern irregularity since the data to be communicated are, in general, not located in close proximity within memory address space. This induces poor cache use when handling those data and engenders lower overall performance. The use of the extra connectivity buffer for the propagation step makes no difference to the precise order of the indices of the source and destination buffers. Hence, pattern regularity of the interface-dependent fluid lattice site data is achieved by locating them closely in the last part of the source and destination buffers. A second benefit stems from the use of direct memory addressing of the inner fluid lattice sites, as explained below. The adoption of the parallel scheme [64] relies on the subdivision of interface-dependent fluid lattice sites from

the other ones. In the original work, this classification was handled by additional expensive buffers and costly indirect addressing operations. Here, instead, inner fluid lattice sites are stored contiguously and thus the corresponding elements in the source buffer are accessed sequentially, resulting in significant memory and computational gains.

6.2.5 Output optimization

In this section, we outline the code optimization we have performed regarding simulation output. The output data are chosen to be the effective pressure, velocity and von Mises stress flow fields in single precision. Therefore, the memory consumption is $15 \times 8/(5 \times 4) = 6$ lower than that obtained by storing all the distribution functions for every lattice site[17].

In passing, we note that a fundamental requirement of petascale computing is checkpointing where the entire system state is dumped at every user-defined simulation period. This is essential (a) to avoid the need to restart the simulation from the beginning if it crashes and (b) to monitor the partial results and modify the configuration parameters during the course of the simulation (steering). Full checkpoint capabilities have been integrated in HemeLB and the portable binary format used, XDR (external data representation), allows different platforms to work with the same input and output data.

With the adoption of the GGP domain decomposition technique presented in Sec. 6.1, the time to write the flow fields on output file can be reduced substantially with respect to many other partition approaches if data must be presented in a specific order. In this case, assuming that only the processor 0 writes to the output file but cannot store the partial flow fields of all the other processors, $O(N)$ single point-to-point communications must be performed since, for example, the first fluid site may belong to processor 3 say, the second one to the processor 1 say, and so on. This problem is minimized in axis-aligned domain decomposition approaches where the partitions are regular; it is wholly absent in the method of Wang *et al.* [68] since the data are subdivided in an orderly fashion among the processors. The GGP domain decomposition technique presented in Sec. 6.1 proceeds for blocks of m^3 lattice sites. As a consequence, the single point-to-point communications drop from

[17] It is assumed that each fluid lattice site has 15 distribution functions stored in double precision.

$O(N)$ to $O(N/m^3)$, that is around three orders of magnitude less if $m = 8$. The resulting communication cost becomes reasonable since the effects due to high latency and limited bandwidth are kept low.

6.3 Performance on single and distributed multiprocessor architectures

In this section, we report some benchmarks for the parallel HemeLB code presented in Sec.6. Single-site parallel performance has been tested on HPCx, the IBM SP p690+ Power4 1.7 GHz housed at Daresbury Laboratory (UK) [75] and on the Cray XT3 MPP Bigben TeraGrid machine located at the Pittsburgh Supercomputing Center (USA) [76]. The IBM machine is composed of a number of nodes communicating via the IBM High Performance Federation Switch. Every node is comprised of 8 chips, each of 2 processors that share L2 and L3 caches of 1.9 MB and 36 MB respectively. The dual-core AMD Opteron 2.6 GHz CPUs run the Catamount Operative System (OS), while the front end processors run SuSe Linux. The interconnect and storage network are Cray Seastar and Infiniband respectively. The GNU compiler 4.0.2 was used with flag `-O3`.

Benchmarks were conducted on two square ducts of $256 \times 128 \times 128$ and $1024 \times 512 \times 512$ fluid lattice sites respectively and a bifurcation of 6253409 with the same shape as shown in Fig. 3 but confined in a bounding box of 512^3 lattice sites. The bounce back rule and the boundary condition method presented in Sec. 4.1 were used to tackle the regular systems and the bifurcation respectively. The minimum number of proccessors used for the system of $1024 \times 512 \times 512$ was 128. We compared the performance delivered using the new GGP method (see Sec. 6.1) to that achieved with two parallel versions in which (a) an ideal domain decomposition approach partitions the smallest system and (b) the Wang *et al.* method [68] is adopted when tackling the bifurcation. HPCx was exploited for this purpose. We applied the most favorable configuration to the latter partitioning method: the bifurcation is oriented along the horizontal x axis and the latter partitioning scheme was applied by scanning before the z and y directions; this produces partitions with fewer interface-dependent fluid lattice sites than that attained by cutting the bifurcation into horizontal subdomains.

We report HemeLB's performance on Bigben for the bifurcation. In Fig. 6 the performances in MSUPS are reported as a function of the processor count P using HPCx only (left) on the regular systems and both multiprocessor architectures on the bifurcation (right). All benchmarks were obtained without checking instability and applying convergence criteria. However, their impact is very modest since they involve only one "All-reduce" communication of a few bytes which is performed at every n time steps where n may be large ($O(100)$).

We note that our LB fluid solver achieves excellent performance, in general. The smallest regular system is executed at a rate of 175 time steps per second on 128 processors. Low performance has been observed on that system and the use of 512 processors only since our GGP technique assigned to the processor with rank 509 around three times the number of fluid lattice sites of those on the other processors, which causes poor load balance. The problem can be overcome by operating the domain partitioning at the finest level of the two-level data representation directly or at the coarsest grid with smaller blocks. For small and medium system the resulting time remains very low.

Except for the highest processor count, we note that HemeLB with our new partitioning technique produces similar performance to that achieved by adopting a regular domain decomposition. With $P = 1024$ the communication cost dominates over the computation. For example, the communication cost and the computational one per time step associated with interface-independent lattice sites calculated by the processor with rank 0 are 0.9515 and 0.3738 ms respectively. In this connection, we remark that the use of an MPI implementation which overlaps communication with computation is essential in order to attain very high performance, epecially when N/P is low. The GGP technique plays an important role in the outputting of results, as anticipated in Sec. 6.1 and 6.2.5. For example, the output results are written in 12.9 and 11.59 seconds using 1 and 1024 processors respectively. Thus, the point-to-point communication between the processor that writes to disk and the other ones has negligible impact. On the largest regular system, $MSUPS(P = 1024) = 2.12MSUPS(512)$ denoting the high quality of the partitioning carried out with our GGP technique.

The times required to accomplish the domain decomposition calculated by the processor with rank 0 and that required to manage the rest of the preprocessing phase without considering

input reading, –buffers management (BM)–, are 0.482 and 3.336 seconds on 128 processors and 0.4483 and 0.391 seconds 1024 processors respectively. As already mentioned, our GGP scheme can domain decompose very large systems and is exceptionally fast because it operates on a coarse grid. Furthermore, BM is quite fast and scales very well. Our new domain decomposition approach subdivides the bifurcation into sub-domains of higher quality than those obtained by the Wang *et al.* method [68]. For example, the latter technique produces sub-domains with no interface-independent fluid lattice sites when the processor number is 512 or 1024. This is due to an insufficient number of available vertical layers to decompose the system into partitions with thickness greater than one in lattice units[18]. The performance superiority of our domain decomposition approach over the other implementations is evident for any $P \geq 32$. When the processor count is 1024, communication of distribution functions between different sub-domains and computation associated with interface-independent lattice sites, calculated by the processor with rank 0, are 0.7559 and 0.7374 ms per time step respectively. In spite of this, we believe that the performance would be about twice as good again if we had available an MPI implementation which could overlap communication with computation[19]. On Bigben the performance is around four times lower than that attained at HPCx machine on 1024 processors but is characterized by an excellent scalability. For example, $MSUPS(P = 512) = 1.988 MSUPS(256)$.

Cross-site benchmarks have been performed on the IA64 Linux Itanium2 1.5 GHz TeraGrid machines located at the University of Chicago/Argonne National Laboratory (UC/ANL), San Diego Supercomputer Center (SDSC), and the National Center for Supercomputing Applications (NCSA). The interconnect, NFS and storage networks are Myrinet, Gigabit Ethernet and Fibre Channel respectively on all machines. The Intel 8.1 compiler with the `-O3` flag was used for each timing result.

The first benchmark concerns three runs on the $256 \times 128 \times 128$ fluid lattice site system: one single-site run involving two processors on two different nodes of the ANL machine and

[18]Each side of the minimum bounding box that contains the whole system is smaller than 512 in lattice units.

[19]In this case, in fact, only the communicational and interface-dependent computation costs would affect the simulations while the computation due to interface-independent lattice sites would be accomplished before completition of communication.

Table 2: HemeLB's performance results in MSUPS for (a) one single-site run on two processors of the ANL supercomputer and two cross-site ones involving one processor located at SDSC machine and the other processor at the NCSA supercomputer using MPICH-G2 and MPIg respectively and (b) one single-site run on 64 processors of the NCSA supercomputer and one cross-site run involving 48 processors located at NCSA and 16 at the ANL exploiting MPIg to handle inter-machine communications. See text for further details. vMPI stands for vendor MPI.

MPI implementation	topology	system size	MSUPS
vMPI	2 ANL	$256 \times 128 \times 128$	2.820
MPICH-G2	1 NCSA + 1 SDSC	$256 \times 128 \times 128$	1.611
MPIg	1 NCSA + 1 SDSC	$256 \times 128 \times 128$	2.913
vMPI	64 NCSA	$1024 \times 512 \times 512$	79.16
MPIg	48 NCSA + 16 ANL	$1024 \times 512 \times 512$	66.63

two cross-site ones using MPICH-G2 and MPIg in turn to handle communications beween a processor located at NCSA and one at SDSC. A cross-site run using 48 processors located at the NCSA machine and 16 processors at ANL is compared to a single-site one involving 64 processors at NCSA. Both processors of each dual-core node were exploited for every benchmark. The second cross-site benchmarks were carried out on the system of $1024 \times 512 \times 512$ fluid lattice sites. The performance of all the benchmarks accomplished at IA64 ANL, SDSC and NCSA supercomputers is reported in Table 2.

Remarkably, the cross-site run related to the smallest system, handled with MPIg, is faster than its single-site counterpart. The analysis of detailed timing results reveals that the inter-machine non-blocking communications handled by MPIg take place entirely during the computation at interface-independent fluid lattice sites. Therefore, the resulting execution time is less than that achieved in the single-site run where vendor MPI (vMPI) is not able to

interleave intra-machine non-blocking communications with computation. The performance degradation is enhanced in the cross-site run which used MPICH-G2 since inter-machine communications take place over longer times. For the largest system, the cross-site performance is somewhat slower than its single site counterpart albeit within 20% of the latter. The domain decomposition of the system with $1024 \times 512 \times 512$ fluid lattice sites into 64 partitions took place in 0.23 seconds only. The MSUPS value is not very high because each sub-domain comprises around two and four million fluid lattice sites when simulating the smaller and larger systems respectively; therefore, the performance is affected by poor cache use as in the single processor runs on the Opteron processor as documented in Sec. 5.1 and 5.2.

It is difficult to compare HemeLB's performance with that of other parallel lattice-Boltzmann codes developed for complex systems. Each code has been tested using different lattice-Boltzmann models, architectures and flow field systems. However, to our knowledge, the parallel implementation that yielded best performance results in terms of scalability and overall execution speed is that presented by Wang *et al.* [68]. We have directly demonstrated that the domain decomposition strategy adopted in [68] is inferior to that incorporated in HemeLB. Moreover, HemeLB exploits several novel optimizations that significantly speed up computation and the output of flow fields. Furthermore, we highlight that among the best parallel codes for regular systems (referenced in Sec. 2), only those presented in [50, 51] are comparable to HemeLB in terms of speed-up and fluid site updates per second for a particular processor count using non-vector machines. This is an excellent result since HemeLB has not been tuned to any specific architecture, while in [50, 51] specific hardware optimizations and regular domain decomposition approaches were applied.

7 Conclusions

We have described a very efficient parallel lattice-Boltzmann code which is well-suited for application to sparse fluid systems. The fast computational core relies on a two-level data representation. This methodology guarantees the minimization of memory consumption when tackling complex and sparse systems and to yield a faster execution than conventional

implementations due to better cache use. Several optimizations are described that reduce redundant operations, increase pattern regularity, simplify the computational core, reduce memory consumption and optimize intra-machine communications. The novel topology-aware two-level domain decomposition is very fast and guarantees high quality domain decomposition partitions. We have demonstrated its capability to effectively decompose a very large system in less than half a second, and its superior communication balance compared to other methods previously used in flow simulations in porous media, while ensuring good workload distribution. Consequently, the parallel scalability and performance of the HemeLB code are very good. The ability to overlap non-blocking communications with computation whenever possible through the use of an optimized MPI implementation is crucial if maximum computational performance is to be realized. Furthermore, the domain decomposition adopted has a direct and positive effect on the output of the flow fields, drastically reducing point-to-point communications. Moreover, a simple modification of the domain decomposition technique permits us to optimize inter-machine communications in grid-based cross-site runs. As a consequence, HemeLB can effectively tackle large and complex systems via distributed heterogeneous computational resources. We remark that HemeLB is also well-suited to efficiently simulate regular systems. In conclusion, HemeLB delivers very high performance on multiprocessor machines whether for regular or complex (sparse) fluid flow systems.

One major aim we have is to use HemeLB for the study of a variety of human cerebral blood flow scenarios, ranging from normal to neuropathological conditions, including aneurysms and arterio-venous malformations, as well as whole brain blood flow. The excellent multiprocessor performance attained using heterogeneous resources is particularly attractive for addressing patient-specific cerebral blood flows in clinically relevant wallclock times. The use of sufficient aggregated computational power should enable us to tackle large systems at an interactive rate.

We plan to develop HemeLB as part of a complete problem solving environment wherein the medical data from various imaging modalities are manipulated by graphical editing tools, embedded withing HemeLB, and rendered by state-of-the-art visualization software. We

have also integrated full checkpoint capabilities within HemeLB. The output data are written in a portable binary format. This allows simulation parameters to be changed during the course of a run and output data to be monitored through steering tools. The latter will be integrated in the near future in order to effectively investigate time-dependent simulation flows and to create a robust and direct connection between the application that manipulates the medical data, HemeLB, and the tool which visualizes the output flow fields.

8 HemeLB usage

In this section, some details are provided about setting up and using HemeLB. HemeLB resides in only five files totalling about 2500 C lines. "config.h" is the file which include the constants and the definition of the data structures used and a few mathematical functions. In "lb.cc" the functions to calculate the equilibrium distribution functions, the momenta, the effective von Mises stress and the boundary conditions are listed. The file "topology.cc" contains functions useful for performing the domain decomposition and to discover the topology used. It may happen that some MPI constants which are used by the function "netFindTopology" are not defined anywhere. If this is the case the commented function with the same name can be used in place of the other one; the presence of only one machine is assumed during the domain decomposition. The functions useful for reading the file of the parameters, and the input configuration, and to write the output flow fields are listed in "io.cc".

At the beginning of the simulation the application reads through the command line the ASCII input file which must list the following files:

a) the input configuration (XDR format);

b) that of the input parameters (see below);

c) the output flow field (XDR format);

c) the checkpoint file (XDR format);

The ASCII file listing the input parameters must contain on different lines:

a) a flag indicating the status of the checkpoint file: "0" if the initial distribution functions

are set to be equal to the equilibrium ones calculated with zero velocity and unitary density; "1" if the initial distribution functions are set accordingly to the flow fields specified in the checkpoint file;

b) the relaxation parameter;

c) the number of inlets;

c) the prescribed density at each inlet (on different lines);

d) the number of outlets;

e) the prescribed density at each outlet (on different lines);

f) the maximum number of time steps allowed;

g) the tolerance used for the convergence criteria;

h) the rate n at which the checkpoint file must be written: it will be output every n time steps;

i) the rate at which the convergence criteria must be applied;

The input and output files which decribe the configuration and the calculated flow field respectively can be processed by complex graphic tools which are not documented yet. However, the input configuration used to obtain the benchmarks related to the bifurcation of 6253409 fluid lattice sites (see Sec. 6.3) is provided in `bifurcation_512x512x512_vw.dat`, while the parameters listed in `bifurcation_tau06_512x512x512.asc` can be employed. Performance statistics and some characteristic simulation data are output on standard output.

Cross-site runs may employ the geographically distributed MPI interfaces MPICH-G2 [73] or its new version MPIg [74] and are launched by means of a Globus RSL job description file (see [73, 74] for details). Specifically, with MPIg the environment on each different platform employed must be set using a specific machine-dependent "soft" file and the application must be queued with the command `mpiexec --globus-rsl-file=<rsl-file-name>`. This platform-dependent setup is not needed with MPICH-G2 and the command `mpirun -globusrsl <rsl-file-name>` is sufficient to submit the job.

Some platform-dependent variables often need to be set in single-site runs. However, these features do not concern HemeLB; rather it is necessary to read platform-specific doc-

umentation. For example, the files related to the HemeLB application can be compiled at HPCx with the following line (the header file, "config.h", must be located in the same directory as the other source files)

```
mpcc_r -O5 -qstrict -qstrict_induction -qipa=level=2 -qhot=vector -qcache=auto \
 -lm -bmaxdata:0x70000000 lb.cc topology.cc io.cc main.cc -o hemelb.exe
```

and a job running on 128 processors with a wall clock time of one hour may be submitted with the following file

```
#@ shell = /bin/ksh
#
#@ job_name = hemelb
#
#@ job_type = parallel
#@ cpus = 128
#@ node_usage = not_shared
#
##@ network.MPI = csss,shared,US
#@ bulkxfer = yes
#
#@ wall_clock_limit = 01:00:00
#@ account_no = <my_account_number>
#
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error  = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue

# suggested environment settings:
export MP_EAGER_LIMIT=65536
export MP_SHARED_MEMORY=yes
export MEMORY_AFFINITY=MCM
export MP_TASK_AFFINITY=MCM

poe ./hemelb.exe <my_input_file> > <my_output_file>
```

On ANL, NCSA and SDSC machines HemeLB can be compiled as follows

```
mpcxx -O3 lb.cc topology.cc io.cc main.cc -o hemelb.exe
```

The Globus RSL job description file to use MPICH-G2 and submit a cross-site run on 48 processors located at SDSC and 16 at ANL with a wall clock time of 60 minutes and two processors per node is

```
+
( &(resourceManagerContact="tg-login1.sdsc.teragrid.org/jobmanager-pbs")
   (host_xcount=2)
   (xcount=24)
   (count=48)
   (host_types=ia64-compute)
```

```
   (jobtype=mpi)
   (project = <my_project_code>)
   (maxwalltime=60)
   (label="hemelb 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
       (GLOBUS_TCP_PORT_RANGE ""))
   (directory=<my_directory_path_at_SDSC>)
   (executable=<my_executable_at_SDSC>)
   (arguments=<my_input_file_at_SDSC>)
   (stdout=<my_output_file_at_SDSC>)
   (stderr=<my_error_file_at_SDSC>)
)

( &(resourceManagerContact="tg-grid1.uc.teragrid.org/jobmanager-pbs")
   (host_xcount=2)
   (xcount=8)
   (count=16)
   (host_xcount=2)
   (xcount=24)
   (count=48)
   (host_types=ia64-compute)
   (jobtype=mpi)
   (project = <my_project_code>)
   (maxwalltime=60)
   (label="hemelb 1")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
       (GLOBUS_TCP_PORT_RANGE ""))
   (directory=<my_directory_path_at_ANL>)
   (executable=<my_executable_at_ANL>)
   (arguments=<my_input_file_at_ANL>)
   (stdout=<my_output_file_at_ANL>)
   (stderr=<my_error_file_at_ANL>)
)
```

while the MPIg-based Globus RSL job description file counterpart is

```
+
( &(resourceManagerContact="tg-login1.sdsc.teragrid.org/jobmanager-pbs")
   (host_xcount=24)
   (xcount=2)
   (count=48)
   (host_types=ia64-compute)
   (jobtype=mpi)
   (project = <my_project_code>)
   (maxwalltime=60)
   (label="hemelb 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
   (directory=<my_directory_path_at_SDSC>)
   (executable=<my_executable_at_SDSC>)
   (arguments=<my_input_file>)
   (stdout=<my_output_file_at_SDSC>)
   (stderr=<my_error_file_at_SDSC>)
)

( &(resourceManagerContact="tg-grid1.uc.teragrid.org/jobmanager-pbs")
   (host_xcount=8)
   (xcount=2)
   (count=16)
```

```
   (host_xcount=2)
   (xcount=24)
   (count=48)
   (host_types=ia64-compute)
   (jobtype=mpi)
   (project = <my_project_code>)
   (maxwalltime=60)
   (label="hemelb 1")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1))
   (directory=<my_directory_path_at_ANL>)
   (executable=<my_executable_at_ANL>)
   (arguments=<my_input_file>)
   (stdout=<my_output_file_at_ANL>)
   (stderr=<my_error_file_at_ANL>)
)
```

9 Acknowledgments

The benchmarks on TeraGrid machines were obtained by means of NFS grants DMR070013 and DMR070014. MDM is grateful to EPSRC for funding his studentship in association with Reality Grid (GR/67699/02 and EP/C536451/01). We would like to thank Luis Fazendeiro of the Centre for Computational Science (CCS), University College London (UCL), Prof. Frank Smith and Dr. Nick Ovenden of the Mathematical Modeling Group (MMG) at UCL for helpful discussion on boundary conditions; also Daniel Scott, Stefan Zasada, Dr. Radhika Saksena, and Dr. Simon Clifford of the CCS for their precious general technical support. We are also very grateful to Dr. Brian Toonen and Prof. Nicholas Karonis of the Mathematics and Computer Science Division at Argonne National Laboratory as well as TeraGrid support for technical assistance and help in setting up cross-site runs.

References

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

[2] S. Chen and D. Doolen, *Lattice Boltzmann method for fluid flows*, Annu. Rev. Fluid Mech., **30**, 329 (1998).

[3] C. A. Taylor, M. T. Draney, J. P. Ku, D. Parker, B. N. Steele, K. C. Wang and C. K. Zarins, *Predictive medicine: Computational techniques in therapeutic decision-making*, Computer Aided Surgery **4**, 231 (1999).

[4] R. Botnar, G. Rappitsch, M. B. Sheidegger, D. Liepsch and K. Perktold and P. Boesiger, *Hemodynamics in the carotid artery bifurcation: a comparison between numerical simulations and in vitro MRI measurements*, J. Biomech., **33**, 137 (2000).

[5] M. D. Mantle; A. J. Sederman, L.F. Gladden, *Single- and two-phase flow in fixed-bed reactors: MRI flow visualisation and lattice-Boltzmann simulations*, Chem. Eng. Sci., **56**, 523 (2001).

[6] Y. Nakashima and Y. Watanabe, *Estimate of transport properties of porous media by microfocus X-ray computed tomography and random walk simulation*, Water Resour. Res., **38**, 1272 (2002).

[7] M. Goyen, M. E. Ladd, J. F. Debatin, J. Barkhausen, K. H. Truemmler, S. Bosk and S. G. Ruehm, *Dynamic 3D MR angiography of the pulmonary arteries in under four seconds*, J. Magn. Reson. Imag., **13**, 372 (2001).

[8] M. J. Thubrikar and F. Robicsek, *Pressure-induced arterial-wall stress and atherosclerosis*, Annals of Thoracic Surgery, **59**, 1594 (1995).

[9] G. Giupponi, J. Harting and P. V. Coveney, *Emergence of rheological properties in lattice Boltzmann simulations of gyroid mesophases*, Europhys. Lett., **73**, 533 (2006).

[10] K. Imaeda and F. Goodman, *Analysis of non-linear pulsatile flow in arteries*, J. Biomech., **13**, 1007 (1980).

[11] Y. Qian, D. d'Humières, and P. Lallemand, *Recovery of Navier-Stokes equations using a lattice-gas Boltzmann method*, Europhys. Lett., **17** (6), 479 (1992).

[12] Q. Zou, S. Hou, S. Chen and G.D. Doolen, *An improved incompressible lattice Boltzmann model for time independent flows*, J. Stat. Phys., **81**, 35 (1995).

[13] P. L. Bhatnagar, E. P. Gross and M. Krook, *A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems*, Phys. Rev., **94**, 511 (1954).

[14] D. d'Humières and P. Lallemand, *Numerical Simulations of Hydrodynamics with Lattice Gas Automata in Two Dimensions*, Complex Systems, **1**, 599 (1987).

[15] G.Zhao-Li, Z. Chu-Guang and S. Bao-Chang, *Non-equilibrium extrapolation method for velocity and pressure boundary conditions in the lattice Boltzmann method*, Chinese Physics, **11**, 366 (2002).

[16] R. Mei, Li-Shi Luo and W. Shyy, *An Accurate Curved Boundary Treatment in the Lattice Boltzmann Method*, Chinese Physics, **155**, 307 (1999).

[17] F. M. White, *Viscous Fluid Flow*, McGraw-Hill, New York 123 (2001).

[18] D.R. Noble, J. G. Georgiadis and R. O. Buckius, *Direct Assessment of lattice Boltzmann hydrodynamics and boundary conditions for recirculating flows*, J. Stat. Phys., **81**, 17 (1995).

[19] R. Mei, W. Shyy and D. Yu, *Lattice Boltzmann Method for 3-D Flows with Curved Boundary*, NASA/CR-2002-211657 ICASE Report No. 2002-17 (2002).

[20] Z. Guo, C. Zheng and B. Sci, *An extrapolation method for boundary conditions in lattice Boltzmann method*, Phys. Fluids, **14**, 2007 (2002).

[21] O. Filippova and D. Hänel, *Grid Refinement for Latttice-BGK Models*, J. Comput. Physics, **147**, 219 (1998).

[22] R.S. Maier, R.S. Bernard, D.W.Grunau, *Boundary conditions for the lattice Boltzmann method*, Phys. Fluids, **8**, 1788 (1996).

[23] M. Junk and Z. Yang, *Asymptotic Analysis of Lattice Boltzmann Boundary Conditions*, J. Stat. Phys., **121**, 3 (2005).

[24] Q. Zou and X. He, *On pressure and velocity boundary conditions for the lattice Boltzmann BGK model*, Phys. Fluids, **9**, 1591 (1997).

[25] M. Kowarschik and C. Weiß, *An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms*, In U. Meyer, P. Sanders, and J. Sibeyn, editors, Algorithms for Memory Hierarchies - Advanced Lectures, volume 2625 of Lecture Notes in Computer Science, pages 213-232. Springer, March 2003.

[26] T. Pohl, M. Kowarschik, J. Wike, K. Iglberger and U. Rude, *Optimization and Profiling of the Cache Performance of Parallel Lattice Boltzmann Codes in 2D and 3D*, J. Parallel Processing Letters, **13**, 549 (2003).

[27] S. Hausmann, *Optimization and Performance Analysis of the Lattice Boltzmann Method on x86-64 based Architectures*, Bachelor Thesis, Erlangen (2005).

[28] G. Wellein, T. Zeiser, G. Hager, S. Donath, *On the Single Processor Performance of Simple Lattice Boltzmann Kernels*, Computers & Fluids **35**, 910 (2006).

[29] K. Iglberger, *Cache Optimizations for the Lattice Boltzmann Method in 3D*, Bachelor Thesis.Lehrstuhl für Informatik 10 (Systemsimulation), Institut für Informatik, University of Erlangen-Nuremberg, Germany (2003).

[30] A. C. Velivelli, K. M. Bryden, *A cache-efficient implementation of the lattice Boltzmann method for the two-dimensional diffusion equation*, Concurrency Comput.: Pract. Exper., **16**, 1415, (2004).

[31] S. Donath, *On Optimized Implementations of the Lattice Boltzmann Method on Contemporary High Performance Architectures*, Lehrstuhl für Informatik 10, Institut für Informatik, University of Erlangen-Nuremberg, Bachelor thesis (2004).

[32] M. Kowarshik, *Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchical Memory Architectures*, PhD thesis, Lehrstuhl für Informatik 10, Institut für Informatik, Universität Erlangen-Nüremberg, ISBN 3-936150-39-7 (2004).

[33] R. Argentini, A. F. Bakker and C. P. Lowe, *Efficiently using memory in lattice Boltzmann simulations*, Future Gener. Comput. Syst., **20**, 973 (2004).

[34] A. C. Velivelli, K. M. Bryden, *Optimizing Performance on Modern HPC Systems: Learning From Simple Kernel Benchmarks*, In: Proceedings of the 2nd Russian-German Advanced Research Workshop on Computational Science and High Performance Computing, HLRS, Stuttgart (2005).

[35] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*, IBM Ltd, Ottawa, Ontario, (1966).

[36] X. X. Zhang, A. G. Bengough, J. W. Crawford, and I. M. Young, *A lattice BGK model for advection and anisotropic dispersion equation*, Adv. Water Resour., **25**, 1 (2002).

[37] X. X. Zhang, A. G. Bengough, L. K. Deeks, J. W. Crawford, and I. M. Young, *A novel three-dimensional lattice Boltzmann model for solute transport in variably saturated porous media*, Water Resour. Res., **38**, 1167 (2002).

[38] X. X. Zhang, L. K. Deeks, A. G. Bengough, J. W. Crawford, and L. M. Young, *Determination of soil hydraulic conductivity with the lattice Boltzmann method and soil thin-section technique*, J. Hydrology, **306**, 59 (2005).

[39] O. Filippova, B. Schwade and D. Hänel, *Multiscale lattice Boltzmann schemes for low Mach number flow*, Phil. Trans. R Soc. Lond. A, **360**, 467 (2002).

[40] S. Geller, M. Krafczyk, J. Tölke, S. Turek and J. Hron, *Benchmark computations based on Lattice Boltzmann, Finite Element and Finite Volume Methods for laminar flows*, Ergebnisberichte des Instituts für Angewandte Mathematik, Nummer 274 (2004).

[41] N. Thürey and U. Rude, *Optimized Free Surface Fluids on Adaptive Grids with the Lattice Boltzmann Method*, SIGGRAPH 2005 Poster.

[42] J. Tölke, S. Freudinger and M. Krafczyk, *An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations*, Computer & Fluids, **35**, 820 (2006).

[43] N. S.Martys, J. G.Hagedorn, *Multiscale Modeling of Fluid Transport in Heterogeneous Materials Using Discrete Boltzmann Methods*, Mater. Struct, **35**, 650 (2002).

[44] S. Donath, T. Zeiser, G. Hager, J. Habich and G. Wellein, *Optimizing Performance of the Lattice Boltzmann Method for Complex Structures on Cache-based Architectures*, in: F. Huelsemann, M. Kowarschik, U. Ruede (Eds.): Frontiers in Simulation: Simulation Techniques - 18th Symposium in Erlangen, September 2005 (ASIM), SCS Publishing House, Erlangen, 728 (2005).

[45] J. Ni, J. Zhang, C-L Lin and S. Wang, *Parallelization of a Lattice Boltzmann Method for Lid-Driven Cavity Flow*, UI research booth, Supercomputing (2003).

[46] J.-C. Desplat, I. Pagonabarraga, and P. Bladon, *LUDWIG: A parallel Lattice-Boltzmann code for complex fluids*, Comput. Phys. Commun. **134**, 273 (2001).

[47] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein and T. Zeiser, *Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures*, Proceedings of the ACM/IEEE SC2004 Conference, Supercomputing, 21 (2004).]

[48] P. Giovanni, F. Massaioli, and S. Succi, *High-resolution lattice Boltzmann computing on the IBM SP1 scalable parallel computer*, Comput. Phys. **8**, 705 (1994).

[49] G. Amati, S. Succi, and R. Piva, *Massively Parallel Lattice-Boltzmann Simulation of Turbulent Channel Flow*, Int. J. Mod. Phys. C, **8**, 869 (1997).

[50] G. Wellein, P. Lammers, G. Hager, S. Donath and T. Zeiser, *Towards optimal performance for lattice Boltzmann applications on terascale computers*, in: A. Deane *et al.* (eds), Parallel Computational Fluid Dynamics - Theory and Applications. Proceedings of the Parallel CFD 2005 Conference, College Park, MD, USA, May 24-27, 2005. Elsevier, ISBN 0-444-52206-9, 31 (2006).

[51] X. Wu, V. Taylor, S. Garrick, D. Yu and J. Richard, *Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System*, IEEE International Conference on Cluster Computing, September 25-28, 2006, Barcelona, Spain.

[52] G. Karypis and V. Kumar, *Multilevel k-way Partitioning Scheme for Irregular Graphs*, J. Parallel Distrib. Comput. **48**, 96 (1998).

[53] G. Karypis and V. Kumar, *Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs*, SIAM Review **41**, 279 (1999).

[54] G. Karypis, K. Schloegel, and V. Kumar, *Parallel graph partitioning and sparse matrix ordering library* (2003).
http://www-users.cs.umn.edu/ karypis/metis/parmetis/files/manual.pdf

[55] H. Sagan, *Space-Filling Curves*, Springer-Verlag (1994).

[56] V. Kumar, A. Grama, G. Karypis, *Parallel computing.* The Benjamin/Cummings Publisshing Company, Inc., California (2004).

[57] A. George and J. W.-H. Liu., *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey (1981).

[58] T. Goehring and Y. Saad, *Heuristic algorithms for automatic graph partitioning*, Technical Report UMSI- 94-29. University of Minnesota Supercomputing Institute (1994).

[59] Jr. P. Ciarlet and F. Lamour, *On the validity of a front-oriented approach to partitioning large sparse graphs with a connectivity constraint*, Technical Report 94-37, Computer Science Department, UCLA, Los Angeles, CA (1994).

[60] S. T. Barnard and H. D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, 711 (1993).

[61] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, International Conference on Parallel Processing, 113 (1995).

[62] D. Kandhai, A. Koponen, A. G. Hoekstra, M. Kataja, J. Timonen, and P. M. A. Sloot, *Lattice-Boltzmann hydrodynamics on parallel systems*, Comput. Phys. Commun. **111**, 14 (1998).

[63] C. Pan, J. F. Prins, and Cass T. Miller, *A high-performance lattice Boltzmann implementation to model flow in porous media*, Comput. Phys. Commun. **158**, 89 (2004).

[64] M. Schulz, M. Krafczyk, J. Tolke and E. Rank, *Parallelization strategies and efficiency of CFD computations in complex geometries using Lattice Boltzmann methods on high-performance computers*, in High-Performance Scientific and Engineering Computing, Proceedings of the 3rd International FORTWIHR Conference on HPSEC, Erlangen, 2001, edited by M. Breuer, F. Durst, and C. Zenger (Springer-Verlag), Berlin, 115 (2002).

[65] M. J. Aftosmis, M. J. Berger, G. Adomavicius, *A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries*, 38th AIAA Aerospace Sciences Meeting and Exhibit, Reno NV Jan. 2000.

[66] M. Griebel, G. Zumbusch, *Hash Based Adaptive Parallel Multilevel Methods with Space-Filling Curves*, in NIC Symposium 2001, Proceedings, Horst Rollnik, Dietrich Wolf

(Editors), John von Neumann Institute for Computing, Julich, NIC Series, **9**, ISBN 3-00-009055-X, 479 (2002), Universität Bonn, Germany.

[67] M. J. Aftosmis, M. J. Berger, S. M. Murman, *Applications of space-filling curves to Cartesian methods for CFD*, 42nd AIAA Aerospace Sciences Meeting and Exhibit, Jan. 2004.

[68] J. Wang, X. Zhang, A. G. Bengough and J. W. Crawford, *Domain-decomposition method for parallel lattice Boltzmann simulation of incompressible flow in porous media*, Phys. Rev. E **72**, 016706 (2005).

[69] S. Dong, G. E. Karniadakis, N. T. Karonis, *Cross-Site Computations on the TeraGrid*, Computing in Science and Engineering, **7**, 14 (2005).

[70] B. Boghosian, P. V. Coveney, S. Dong, L.I. Finn, S. Jha, G.E. Karniadakis and N.T. Karonis, *Nektar, SPICE and Vortonics: Using Federated Grids for Large Scale Scientific Applications*, Challenges of Large Applications in Distributed Environments, IEEE 34 (2006).

[71] S. Jha, P. V. Coveney and M. Harvey, *SPICE: Simulated Pore Interactive Computing Environment*, Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference.

[72] B. M. Boghosian, L. I. Finn and P. V. Coveney, *Moving the data to the computation: multi-site distributed parallel computation*, 2006.
http://www.realitygrid.org/publications.shtml

[73] http://www3.niu.edu/mpi/

[74] http://www-unix.mcs.anl.gov/ toonen/mpig/

[75] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le and Balaram Sinharoy, *IBM, POWER4 System Microarchitecture*, IBM Server Group, (2001).
http://www.hpcx.ac.uk/

[76] http://www.teragrid.org/userinfo/hardware/resources.php

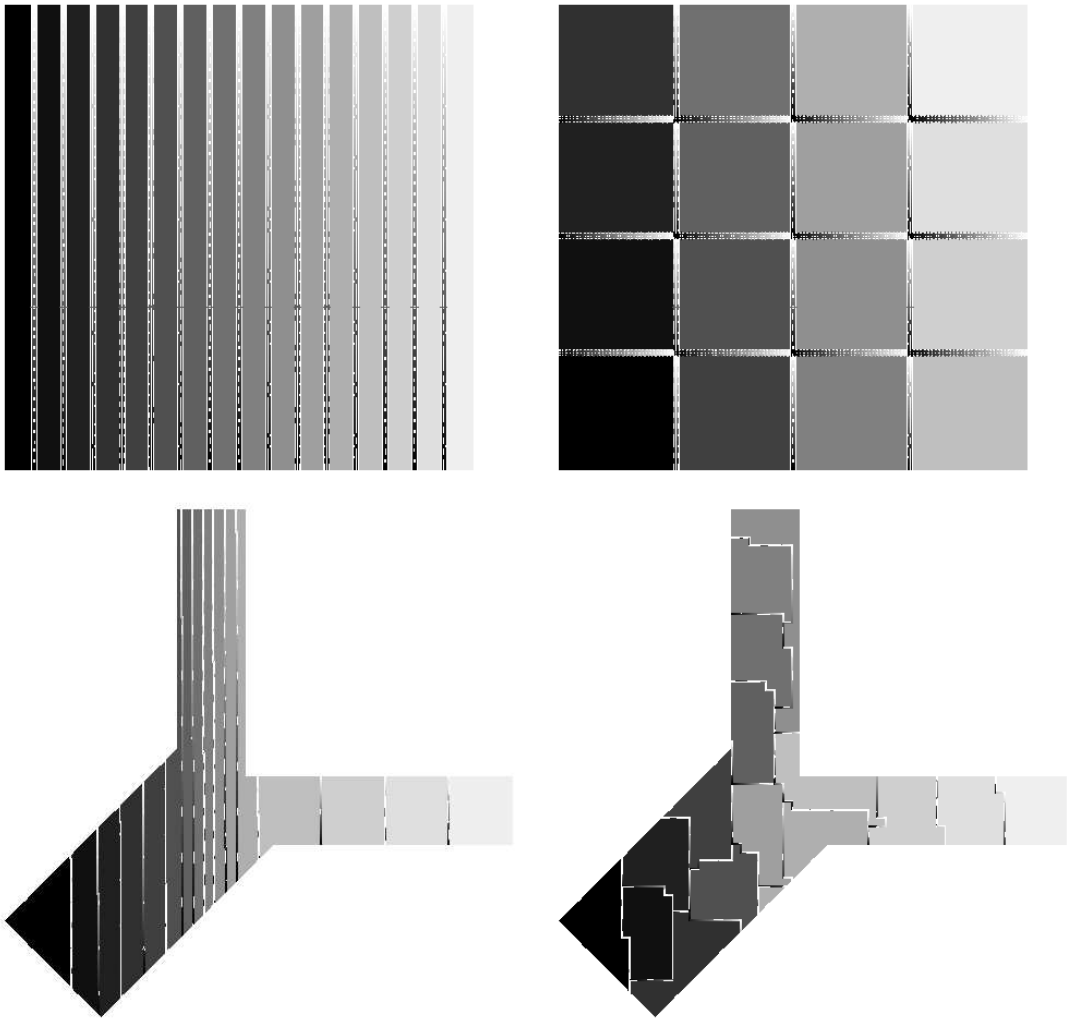

Figure 5: 2D partitions of a square and a bifurcation with 128^2 and 61451 fluid lattice sites respectively, as obtained with the domain decomposition method of Wang *et al.* [68] (left hand side images) and that presented in Sec. 6.1 with a block size of 8^2 for the finest resolution of the two-level representation (right hand side images); see Sec. 5 for details. In each image the number of partitions is 16 and their ranks are coloured according to a grey-scale map. The identifiers of the interface-dependent distribution functions between every pair of adjacent partitions are represented by tiny grey segments. The grey intensity is proportional to the identifier. Adjacent distribution functions have the same identifier and cannot be communicated during each LB time step.

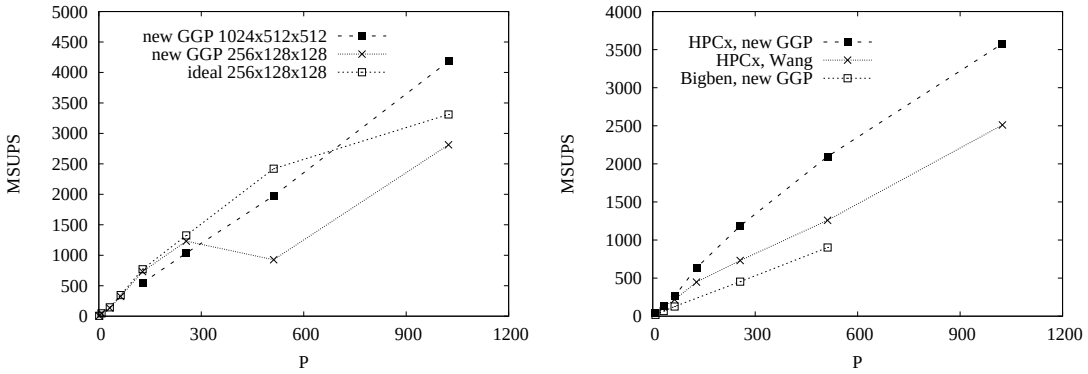

Figure 6: HemeLB performance measured in millions of lattice site updates per second (MSUPS) as a function of the number of the processors P used for the regular systems of $256 \times 128 \times 128$ and $1024 \times 512 \times 512$ fluid lattice sites (left) and a large bifurcation of 6253409 fluid lattice sites (right). The timing results obtained in testing the regular systems were collected on the IBM SP Power4 p690+ 1.7 GHz HPCx, located at Daresbury Laboratory (UK), while the Cray XT3 MPP Bigben at the Pittsburgh Supercomputing Center (USA) was used to study the bifurcation. Apart from the GGP method presented in Sec 6.1, denoted "new GGP", we used an ideal regular partitioning scheme and the domain decomposition approach presented by Wang *et al.* [68] in the parallel code to tackle the smallest regular system and the bifurcation respectively.