Analysing and modelling the performance of the HemeLB lattice-Boltzmann simulation environment

Derek Groen*, James Hetherington, Hywel B. Carver, Rupert W. Nash, Miguel O. Bernabeu, Peter V. Coveney

Centre for Computational Science, University College London, London, United Kingdom

ABSTRACT

We investigate the performance of the HemeLB lattice-Boltzmann simulator for cerebrovascular blood flow, aimed at providing timely and clinically relevant assistance to neurosurgeons. HemeLB is optimised for sparse geometries, supports interactive use, and scales well to 32,768 cores for problems with ~81 million lattice sites. We obtain a maximum performance of 29.5 billion site updates per second, with only an 11% slowdown for highly sparse problems (5% fluid fraction). We present steering and visualisation performance measurements and provide a model which allows users to predict the performance, thereby determining how to run simulations with maximum accuracy within time constraints.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Recent progress in imaging and computing technologies has resulted in an increased adoption of computational methods in the life sciences. Using modern imaging methods, we are now able to scan the geometry of individual vessels within patients and map out potential sites for vascular malformations such as intracranial aneurysms. Likewise, recent increases in computational capacity and algorithmic improvements in simulation environments allow us to simulate blood flow in great detail. The HemeLB lattice-Boltzmann application [1] aims to combine these two developments, thereby allowing medical scans to be used as input for blood flow simulations. It also enables clinicians to run such simulations in real-time, providing runtime visualisation feedback as well as the ability to steer the simulation and its visualisation [2]. One principal long-term goal for HemeLB is to act as a production toolkit that provides both timely and clinically relevant assistance to surgeons. To achieve this we must not only perform extensive validation and testing for accuracy, reliability, usability and performance, but also ensure that the legal environment and the medical and computational infrastructure are made ready for such use cases [4].

In this work we investigate the performance aspects of the HemeLB environment, taking into account the core lattice-Boltzmann (LB) simulation code and the visualisation and steering facilities. We present performance measurements from a large number of runs using both sparse and non-sparse geometries and the overheads introduced by visualisation and steering. Medical doctors treating patients with intracranial aneurysms are frequently confronted with very short time scales for decision-making. For HemeLB to be useful in such environments, it is therefore not only essential that the code simulates close to real-time, but also that the length of a simulation can be reliably predicted in advance. We demonstrate that it is possible to accurately characterise CPU and network performance at low core counts and integrate this information into a model that predicts performance for arbitrary problem sizes and core counts.

1.1. Overview of HemeLB

HemeLB is a massively parallel lattice-Boltzmann simulation framework that allows interactive use, eventually in a medical environment. Segmented angiographic data from patients can be read in by the HemeLB Setup Tool, which allows the user to indicate the geometric domain to be simulated using a graphical user interface. The geometry is then discretised into a regular grid, which is used to run HemeLB simulations. The core HemeLB code, written in C++, consists of a parallelised lattice-Boltzmann application which is optimised for sparse geometries such as vascular networks by use of indirect addressing. We precompute the addresses of neighbouring points within a single one-dimensional array instead of requiring that the points be stored in a dense, three-dimensional array. HemeLB also constructs a load-balanced domain decomposition

* Corresponding author.
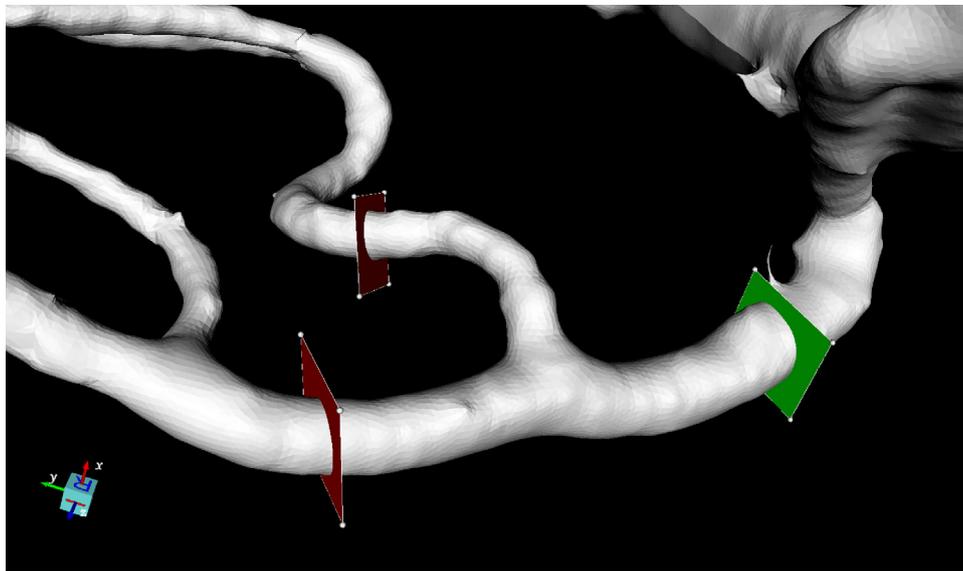 *E-mail addresses:* djgroennl@gmail.com, d.groen@ucl.ac.uk (D. Groen).

Fig. 1. Graphical overview of the bifurcation geometry in the HemeLB Setup Tool. We used this geometry to generate the Bifurcation and Large Bifurcation simulation domains. Inlets are shown by green planes, outlets by red planes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

at runtime, allowing the user to run simulations at varying core counts with the same simulation domain data. HemeLB is highly scalable due to a well-optimised communication strategy and the locality of interactions and communications in the parallelised lattice-Boltzmann algorithm. The File I/O operations are done in parallel using MPI-IO by a group of *reading processes*, which can be adjusted in size using a compile-time parameter.

The HemeLB Steering Client is a light-weight tool that allows users to connect remotely to their HemeLB simulation, receive real-time visual feedback and modify parameters of the simulation at runtime. Here, the visualisations are generated on-site within HemeLB, using a hand-written ray-tracing kernel [2]. In our work we run HemeLB with the steering server code enabled. As a result, one core is reserved for steering purposes, whether or not a client is connected, and is thereby excluded from the LB calculations.

HemeLB relies on ParMETIS version 4.0.2 [4] to perform its domain decomposition. It constructs an initial guess using a basic graph growing partitioning algorithm (see [1] for details), which it then passes to ParMETIS for optimisation using the `ParMETIS_V3_PartKway()` function. Constructing the initial guess requires less than a second of runtime in all cases, but the ParMETIS optimisation typically adds between 5 and 30 s to the initialisation time. We discuss several technical aspects and performance implications of our decomposition routine in Section 3.1.

HemeLB uses a coalesced asynchronous communication strategy to optimise its scalability [5]. This system bundles all communications for each iteration (e.g., exchanges required for the LB algorithm, steering and visualisations) into a single batch of non-blocking communication messages, one for each data exchange of non-zero size between a pair of processes in each direction. As a result, each iteration of HemeLB's core loop has only one `MPI_Wait` synchronisation point, minimising the latency overhead of HemeLB simulations. Communication of variable length data is spread over two iterations, the sizes being transferred during the first iteration while the actual exchange takes place during the second one.

The coalesced communication system is also used for the phased broadcast and reduce operations which are required for the visualisation and steering functionality. Here HemeLB arranges the processes into an *n*-tree and, for broadcasts, sends data from one level of the tree to the level below over successive iterations. For reductions, data is sent up one level of the tree over successive

iterations. Hence, both operations can take $O(\log(p))$ iterations, for p cores. In this approach HemeLB does require some additional memory for communication buffers. Additionally, the responsiveness of the steering is constrained, as data arriving in the top-most node takes $O(\log(p))$ iterations to be spread to all nodes.

1.2. Related work

A large number of researchers have investigated the performance aspects of various LB simulation codes over the past decade. These investigations have been done without real-time visualisation or steering enabled, and frequently use non-sparse geometries. We present a performance analysis of both sparse geometries and interactive usage modes in this work. Pohl et al. [6] compared the performance of LB codes across three supercomputer architectures, and concluded that the network and memory performance (bandwidth and latency) are dominant components in establishing a high LB calculation performance. Geller et al. [7] compared the performance of an LB code with that of several finite element and finite volume solvers, and deduced that LB offers superior efficiency in flow problems with small Mach numbers. Williams et al. [8] presented a hierarchical autotuning model for parallel lattice-Boltzmann, and report a performance increase of more than a factor 3 in their simulations. Several groups have considered the performance of LB solvers on general-purpose graphics processing unit (GPGPU) architectures. In these studies, they introduced a number of improvements, such as non-uniform grids [9], more efficient memory management strategies [10,11] and LB codes which run across multiple GPUs [12–14]. Other performance investigations include a comparison between different LB implementations [15], hybrid parallelisations for multi-core architectures in general [16,9,17] and performance analysis of LB codes on Cell processors [18–20].

A few studies within the physiological domain are of special relevance to this work. These include a performance analysis of a blood-flow LB solver using a range of sparse and non-sparse geometries [21] and a performance prediction model for lattice-Boltzmann solvers [22,23]. This performance prediction model can be applied largely to our HemeLB application, although HemeLB uses a different decomposition technique and performs real-time rendering and visualisation tasks during the LB simulations.
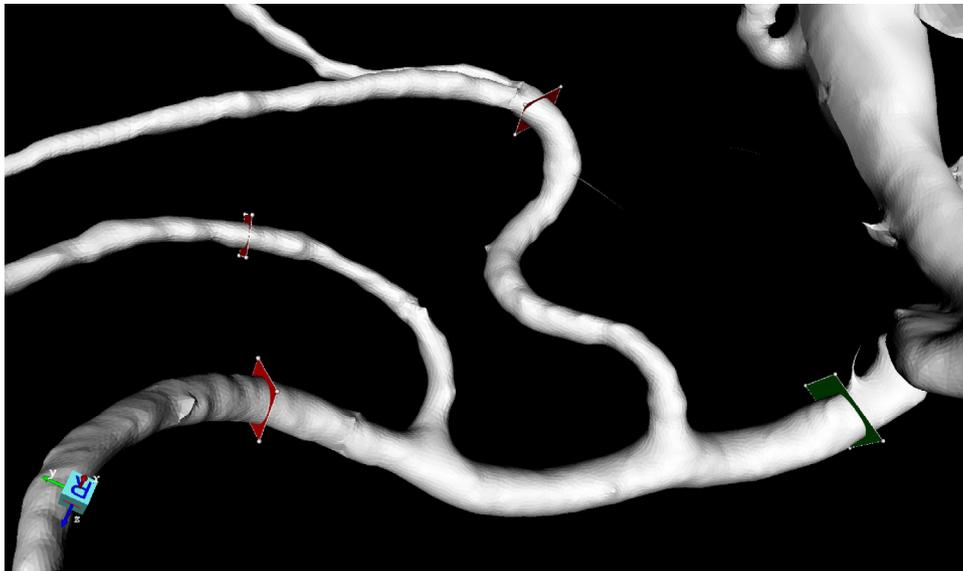
Fig. 2. Graphical overview of the network geometry in the HemeLB Setup Tool. We used this geometry to generate the Network, Large Network and Small Network simulation domains.

Mazzeo and Coveney [1] studied the scalability of an earlier version of HemeLB. However, the current performance characteristics of HemeLB are substantially enhanced due to numerous subsequent advances in the code, amongst others: an improved hierarchical, compressed file format; the use of ParMETIS to ensure good load-balance; the coalesced communication patterns to reduce the overhead of rendering; use of compile-time polymorphism to avoid virtual function calls in inner loops.

2. Performance analysis

We benchmarked HemeLB using simulation domains based on three distinct geometries, a vascular network (see Fig. 2, used to generate three simulation domains), a bifurcation of vessels (see Fig. 1, used to generate two simulation domains) and a cylinder. Both the network and the bifurcation geometries are sections of an intracranial vasculature model that has been constructed from multiple rotational angiography scans of a patient with an intracranial aneurysm treated at the U.K. National Hospital for Neurology and Neurosurgery. The third and least sparse geometry is an artificially created cylinder. We present an overview of the simulation domains we generated and use in our runs in Table 1. We also provide a brief description of the sparseness of each generated simulation domain. Our runs were impulsively started, applying a pressure gradient across the simulation domain, using Nash in-outlet conditions (Nash et al., in preparation).

2.1. Performance of LB computations

We have run blood flow simulations using the simulation domains listed in Table 1 using up to 32,768 cores on the HECToR

Phase 3 supercomputer at EPCC in Edinburgh, United Kingdom. The HECToR machine is a Cray XE6 with 90,112 cores (2.3 GHz AMD Opteron 6276), and has a peak performance of 9.2 GFLOP/s per core. Our simulations were done using a 15-directional lattice-Boltzmann kernel (D3Q15), the Lattice Bhatnagar–Gross–Krook [24] model with simple bounce-back boundary conditions and a fixed physical viscosity of 0.004 Pa s. We present the scalability results for all simulation domains in Fig. 3. We find that the small network simulation domain scales near-linearly up to 128 cores, despite consisting of only 77,182 lattice sites. All of the medium-sized simulation domains (Bifurcation, Cylinder and Network) scale linearly to 8192 cores. However, the communication overhead and load imbalance reduce the performance on higher core counts. The two largest simulation domains (Large Bifurcation and Large Network) show linear scaling from 512 cores up to 16,384 cores, and significant speedup to 32,768 cores, achieving a maximum performance of 29.5 billion site updates per second (SUPS). The performance obtained at 8192 cores for the medium-sized bifurcation corresponds to 419 timesteps per second, or 646 times slower than
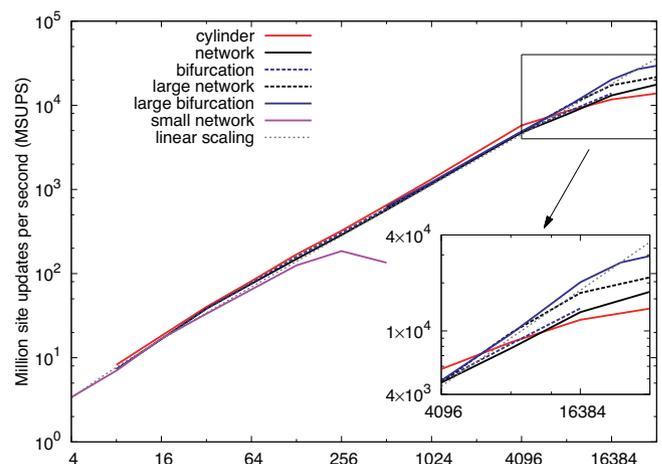
Table 1
Overview of the simulation domains used in our experiments. The percentage of the simulated box that consists of active fluid sites is given by the fluid fraction. Non-active fluid sites do not count towards the number of lattice sites in the simulation.

Name	# of lattice sites	Fluid fraction
Bifurcation	19,808,107	11%
Cylinder	15,607,040	65%
Network	18,836,545	5.1%
Large Bifurcation	81,132,544	11%
Large Network	44,650,496	5.1%
Small Network	77,182	5.1%

Fig. 3. Lattice site updates per second (SUPS) as a function of the number of cores used for simulations run on the HECToR Cray XE6 machine. We run simulations using each of the six simulation domains (Cylinder, Network, Bifurcation, Large Bifurcation, Large Network and Small Network).
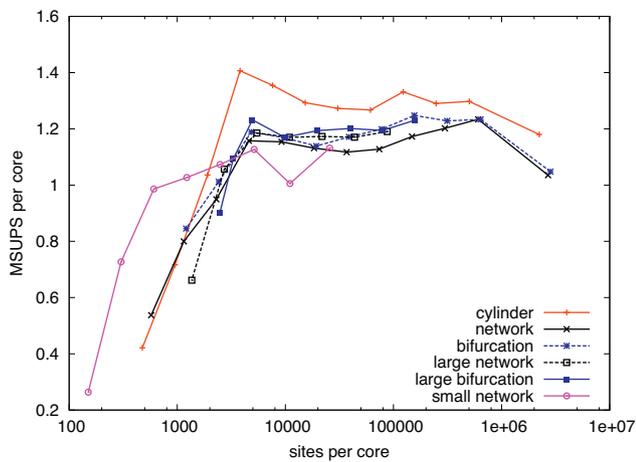
Fig. 4. Site updates per second (SUPS) per core averaged over all cores used in the simulation (excluding the one used for steering) as a function of the number of sites per core, for six LB problems.

real-time for a maximum timestep as limited by incompressibility constraints. The maximum timestep here is estimated by the need to keep the Mach number below 0.05, using a typical blood velocity for vessels of this size of 25 cm/s. At this rate, it takes HemeLB 553 s to simulate one heartbeat with a resolution of around 100 lattice points across a vessel diameter. We present the performance in SUPS per core as a function of the number of sites per core in Fig. 4, demonstrating that the SUPS per core is largely independent of other factors.

2.2. Visualisation performance

One of the features that sets HemeLB apart from many other LB codes is its ability to perform *in situ* rendering of the geometry at runtime [2], using a parallelised ray-tracing algorithm. The communication needs of the ray-tracing algorithm have been combined with those of the main simulation algorithm, through the coalesced communication strategy, massively improving the scaling when rendering frames. The images rendered by HemeLB can either be stored on disk for future reference or they can be forwarded as a streaming visualisation to the steering client. In this section we present several simulations where we assess the overhead introduced by rendering images, as well as that introduced by writing snapshots of the simulation data. These snapshots store the hydrodynamic variables at each lattice point, recording all information of physical relevance which is useful for visualisation and post-processing. File I/O operations are done in HemeLB using a subset of all processes, the *reading group*. Within this work, we adopted a reading group size of 32 processes, or the number of processes used by HemeLB, whichever was smaller. We have run four types of simulations using the Bifurcation simulation domain, one with snapshots and image-rendering disabled, one where we write snapshots to disk (10 snapshots per 1000 time steps, with each snapshot being 604 MB in size), one where we render and write images to disk (10 rendered images per 1000 time steps, with each image being 180 kB in size) and one with both snapshots and images enabled. We have carried out the tests using 256, 512, 1024 and 2048 cores. We present our results in Fig. 5. Here the overhead for rendering and writing images is marginal, and adds no more than a few percent to the execution time in most cases. Simulations which have snapshot writing enabled are both considerably slower and have more variable performance, due to the high disk activity involved with snapshot writing. When snapshot writing is enabled, the overhead caused by image rendering is difficult to observe, as the standard deviation bars of the performance
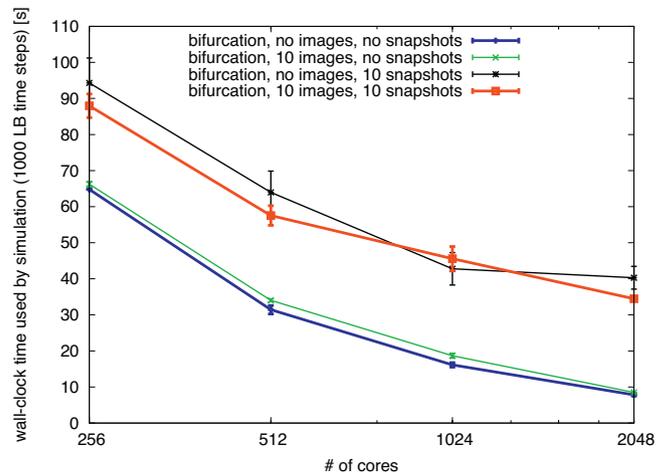

Fig. 5. Time spent to simulate 100 time steps as a function of the number of cores used for four settings: no snapshots and no images (blue), images only (green), snapshots only (black) and snapshots and images (red). We averaged the results from runs including any form of snapshot or image writing over three executions, and included a standard deviation error bar with each data point. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

measurements with and without images overlap. When the simulation writes 10 snapshots over 1000 LB steps, we observe an increase in the wall-clock time of ~24 s.

We have also run several simulations of 1000 LB steps where we render and write an image to disk every 5–200 LB steps. The results for these runs (which were done using 1024 and 2048 cores) are given in Fig. 6. Without rendering the simulations took 31.4, 16.1 and 7.81 s on 512, 1024 and 2048 cores respectively. We observe an overhead of less than 2 s per 1000 LB steps if we render and write no more than 10 images during that period. However, the performance deteriorates somewhat when we write more images, with a maximum measured overhead of ~6.5 s. We also again observe some jitter in our results, for example in the 1024 core simulation that rendered one image every 50 steps, which we attribute to fluctuations in the file system performance of the machine.
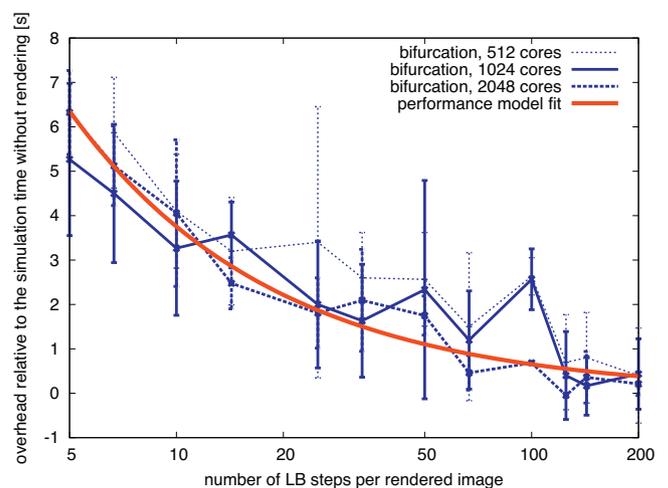

Fig. 6. Overhead in seconds relative to the simulation time without images rendered as a function of the number of LB steps per image rendered and written. The simulation with 0 images rendered took 31.4, 16.1 and 7.81 s on respectively 512, 1024 and 2048 cores. We averaged the measurement of the runs over three executions. Error bars are the resulting standard deviations. The prediction of our performance model, presented in Section 3.3, is given by the thick solid red curve. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

Table 2

Performance impact of running HemeLB with a connected steering client, simulating the Cylinder simulation domain using 1024 and 2048 cores. Here the mode is the method of running HemeLB, which can be without client (none), with the client used only for image streaming (images) or with the client used both for image streaming and steering the HemeLB simulation (both).

p	Mode	Frame-rate (1/s)		MSUPS per core	Mean LB steps per image
		Requested	Achieved		
1024	None	–	–	1.39	–
1024	Both	2.0	2.0	1.28	41.5
1024	Images	2.0	2.1	1.25	39.3
1024	Both	5.0	4.4	1.11	16.5
1024	Images	5.0	4.8	1.02	13.8
1024	Both	Max	5.9	0.84	11.3
1024	Images	Max	8.2	0.76	6.0
2048	None	–	–	1.46	–
2048	Images	2.0	2.1	1.26	77.2
2048	Both	2.0	2.2	1.32	78.6
2048	Both	5.0	4.6	1.15	32.2
2048	Images	5.0	4.8	0.99	26.9
2048	Images	Max	9.5	0.59	8.0
2048	Both	Max	10.6	0.66	8.1

Rendering one image per 5 LB steps using 2048 cores corresponds to a frame rate of about 13.6 frames/s, more than sufficient for smooth visualisations of the simulations in real time.

2.3. Steering performance

The previous subsection isolates the performance impact of the visualisation and rendering, with images written to disk. Here we study the performance impact of the HemeLB steering component, using the Cylinder simulation domain, where images are streamed over the network to a client. In this case, HemeLB produces images as described in Section 2.2, optionally limited by a maximum frame-rate per second. We also look at the performance impact of sending steering messages from the client to the HemeLB steering component. In order to obtain reproducible data, the steering client is set up with a scripted set of simulated user actions (orbiting the view point for image rendering). These results are presented in Table 2 and in Fig. 7 and were produced with the steering client running on the HECToR login node. For a frame-rate of 4.6 frames/s, which is usable for scientific steering, with bidirectional communication

Fig. 7. Performance impact of running HemeLB with a connected steering client. We show results for 1024 and 2048 cores without steering client (plotted at frame-rate zero), with the client used only for image streaming (images) and with the client used both for image streaming and steering the HemeLB simulation (both).

Table 3

Technical specifications of 12 LB simulations in our code comparison. We provide the name of the LB application used in the first column (including the source), followed by respectively the architecture used for the simulations and the number of cores used for the run.

Name	Architecture (peak GFLOPS/ core)	Cores
HemeLB (Cylinder)	AMD Opteron 6276 (9.2)	4096
HemeLB (Network)	AMD Opteron 6276 (9.2)	32
HemeLB (Large Network)	AMD Opteron 6276 (9.2)	512
LB3Dv7 (Shamardin p.c.)	AMD Opteron 6276 (9.2)	32
LB3Dv7-3phase (Shamardin p.c.)	AMD Opteron 6276 (9.2)	128
LBMHD [8]	AMD Opteron 1356 (9.2)	8192
LBMHD [8]	AMD Opteron 6172 (8.4)	49,152
LUDWIG [14]	AMD Opteron 6276 (9.2)	384
Palabos [27]	AMD Opteron 8356 (9.2)	4
HYPO4D (Groen p.c.)	BlueGene/P (3.4)	512
LBMHD [8]	BlueGene/P (3.4)	8196
Palabos [27]	BlueGene/P (3.4)	256
MUPHY [28]	BlueGene/L (2.8)	32
OpenLB [17]	Intel Xeon X5355 (10.64)	8
Palabos [27]	Intel Xeon X5550 (10.64)	4
HemeLB (Bifurcation, Sect 4.2)	Xeon E5-2680 (21.6)	128

between client and server, corresponding to 32 LB steps per rendered image, we observe an overhead of 28%.

2.4. Performance comparison with other codes

In this section we compare the performance of HemeLB with performance measurements of other LB codes as found in the literature. We gathered the number of million lattice site updates per second (MSUPS), the standard measure of LB performance, reported for other implementations. HemeLB is strongly optimised for efficiently handling sparse geometries while most codes are not, making like-for-like comparison difficult. The other applications may not be capable of simulating even moderate complexity domains, such as a cylinder, at all or only at the cost of allocating memory to non-fluid sites. Additionally, the directional resolution affects the number of calculations and memory accesses required per site update, as well as the presence of other special features, such as the additional presence of a D3Q15 magnetic field distribution model in LBMHD [8]. One particular example is LB3D [25] version 7, which calculates a number of additional forces, and is strongly optimised for multi-phase flow at the expense of single-phase flow performance. For LB3D we therefore included measurements for both single-phase and multi-phase flow performance.

We provide the LB performance configurations and results for several well-known LB codes in Tables 3 and 4. The MSUPS per core results here are obtained by dividing the total number of lattice site updates by the product of time spent on LB iterations and the number of cores. From each literature source, we picked the result from the run that showed the best MSUPS per core while running on at least one full processor. In the case of HemeLB we picked the best result from the non-sparse Cylinder, as well as from the very sparse Network and Large Network simulation domains, which are the only measurements in the tables using sparse geometries.

When we examine bulk flow only, the MSUPS per core performance of HemeLB is comparable with that achieved with LBMHD (although LBMHD calculates in 27 directions and HemeLB in 15), and about half of that achieved with Palabos on similar AMD Opteron architectures. The performance of HemeLB, however, is almost entirely preserved when using a very sparse simulation domain as HemeLB does not allocate memory or computational effort for non-active lattice sites, which are by definition common in sparse geometries. LBMHD has no known optimisations for sparse geometries while Palabos features a partial optimisation using the
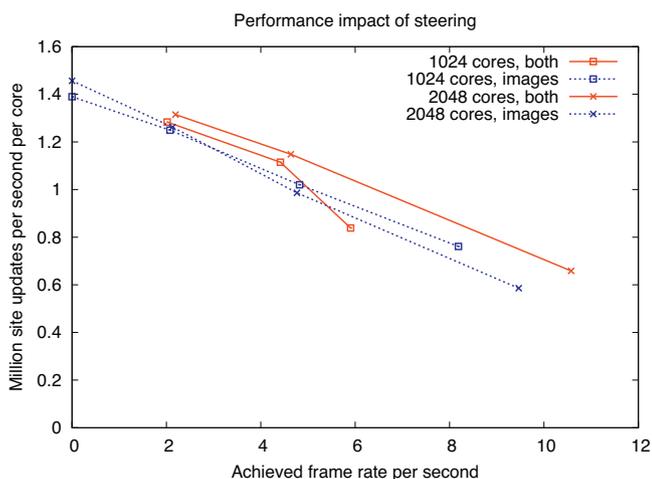
Table 4
Performance comparison of 12 LB simulations in our code comparison. We provide the name of the LB application used in the first column, followed by the number of lattice sites for each run, the directionality, and the obtained performance per core. We give the per core calculation performance in millions of site updates per second (MSUPS). In the case of LBMHD we assumed 1300 FLOPs per lattice operation, as mentioned in Williams et al. [19,8]. Runs that use a sparse simulation domain are marked with an asterisk. Three-phase flow runs requires considerably more FLOPs per site update than single-phase flow runs. Here, the OpenLB run used a data set with a fluid fraction of 0.145. The Palabos run on the Opteron relied on shared memory and multi-threading, and did not use MPI.

Name	# of lattice sites	Directional resolution	MSUPS per core
HemeLB (Cylinder)	15,607,040	D3Q15	1.41
HemeLB* (Network)	18,836,545	D3Q15	1.20
HemeLB* (Large Network)	44,650,496	D3Q15	1.19
LB3Dv7	16,777,216	D3Q19	0.30
LB3Dv7 (3-phase flow)	56,623,104	D3Q19	0.084
LBMHD (w/ magnetism)	6,115,295,232	D3Q27	~1.42
LBMHD (w/ magnetism)	28,311,552,000	D3Q27	~1.15
LUDWIG	339,738,624	D3Q19	~3.0
Palabos (shared memory)	64,481,201	D3Q19	2.55
HYPO4D	452,984,832	D3Q19	0.273
LBMHD	1,811,939,328	D3Q27	~0.5
Palabos	1,003,003,001	D3Q19	0.891
LUDWIG	16,777,214	D3Q19	0.087
MUPHY	262,144	D3Q19	0.529
OpenLB*	1,060,000	D3Q19	~0.4
Palabos	64,481,201	D3Q19	7.87
HemeLB* (Bifurcation)	19,808,107	D3Q15	3.49

multi-block method [26], of which we found no performance data using sparse geometries in the literature. The multi-block method is relatively inefficient because it allocates memory to some of the non-fluid sites and uses data structures that grow in complexity when off-lattice geometries are modelled more accurately. When a code is not designed for sparse geometries, additional optimisations (e.g., cache lookahead) are simpler to implement, hence the performance of a code which supports sparse geometries may not match that of codes which exploit such optimisations. Many of the benchmarks for other LB codes were performed on non-Opteron architectures, making it difficult if not impossible to do a one-on-one comparison. We nevertheless include these results for reference in the lower part of Table 3.

3. Modelling the performance of HemeLB

3.1. Parameter extraction

Before we are able to construct and apply the performance model, we need to extract a number of parameters specific to HemeLB. These parameters include the maximum neighbour count, the communication volume and the calculation and communication load imbalance.

3.1.1. Characterising maximum neighbour count

Each process within HemeLB (except for the steering process) models a subsection of the simulation domain, and exchanges information with its neighbours. Here we characterise the maximum neighbour count (k_{max}), which is an approximation of the maximum number of neighbours a process has in a given simulation.

To obtain the neighbour counts of each process, we have run the initialisation routine of HemeLB (without any simulation time steps) using 4–16,384 cores. The number of neighbours is dependent not only on core count but also on the geometry of the simulation domain, which makes it non-trivial to fully approximate it in the performance model. Instead, we choose to model close to a worst-case decomposition scenario, selecting the simulation domain with the highest neighbour count, and using the measured values there to determine k_{max} for any simulation domain. Because ParMETIS does not guarantee a reproducible decomposition, simulations may vary in neighbour counts for a given problem on a given number of cores. We therefore have repeated each measurement three times.

We present our measurements of the maximum neighbour count as a function of the core count in Fig. 8. We find that the maximum neighbour count for the network geometry ranges from 7 on 8 cores, up to as high as 94 on 16,384 cores. Based on this data, we created a logarithmic fit, approximating k_{max} as:

$$k_{max} = \frac{\log P}{\log 1.127}. \tag{1}$$

3.1.2. Characterising communication volume

To model the communication performance of HemeLB we also need information on the amount of data communicated between processes at each step. As the domain decomposition in HemeLB is done at runtime [1], we can only know the exact communication data volume after we have launched the simulation. To preserve the predictive power of the performance model, we have instead measured the communication volume for the three types of simulation domains across a range of core counts. After having performed the measurements, we fitted the data to a function of
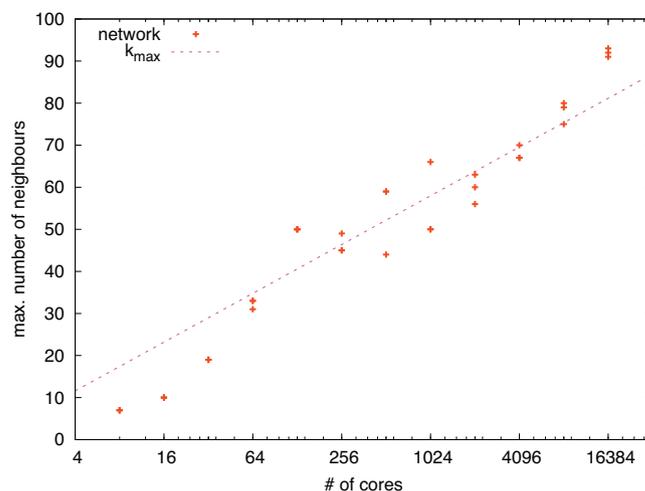

Fig. 8. Maximum number of neighbours as a function of the core count. Here we selected and fitted our model to the Network simulation domain, which has the highest neighbour count due to its sparseness. We ran 3 decomposition routines for each core count included in the figure, plotting the highest neighbour count separately for each instance. The neighbour count approximation used in our performance model is given by the dotted line.
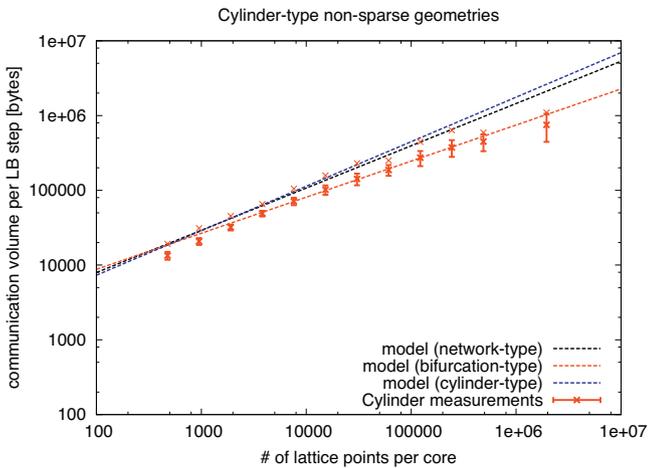
Fig. 9. Number of bytes sent per LB simulation step as a function of the number of lattice sites per core for Cylindrical geometries (measurements have been done using the Cylinder simulation domain). The fits we use for Cylindrical, Bifurcation and Network geometries in our performance model are given respectively by the red, blue and black dashed lines. Error bars show one standard deviation for the distribution across cores. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

the form ax^b to gain an approximate estimate while keeping the model relatively straightforward. We present our measurements of the communication volume and our fits for the cylindrical geometries in Fig. 9, for the bifurcation geometries in Fig. 10, and for the network geometries in Fig. 11. Here we find that the communication volume can differ by as much as a factor four between the domain types, making separate fits necessary for each type. We provide the exact formulation for each of the three fits in Table 6. Interestingly, these scale with less than $(N/P)^{2/3}$ as one would expect with, for example, a decomposition into cubes. At large N/P, i.e., few processes, the sparseness implies that large parts of the single-process volumes are bordered by boundary sites, rather than lattice sites residing on neighbouring processes. We therefore observe a scaling of less than $(N/P)^{2/3}$. In the limit of small (N/P), the measured communication volume does converge to the function $S = 250 \times (N/P)^{2/3}$ when the number of sites per process becomes lower, and the number of cores used higher in the simulations. Because the process-specific volumes are smaller here, the sparseness of the domain has a smaller effect on the measured (maximum) neighbour count.
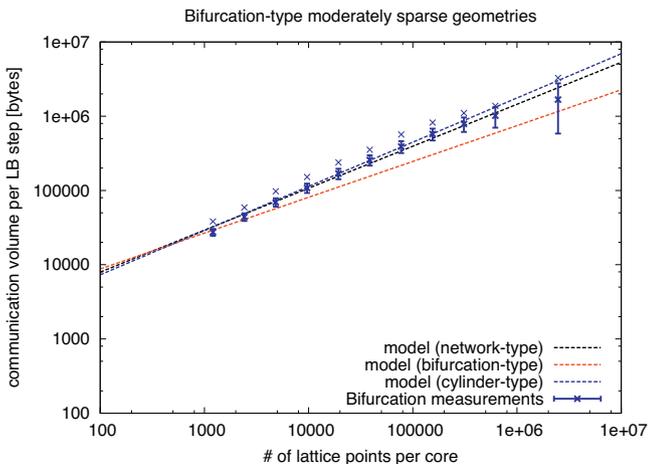

Fig. 10. As in Fig. 9 but for the Bifurcation geometry (using the Bifurcation simulation domain).
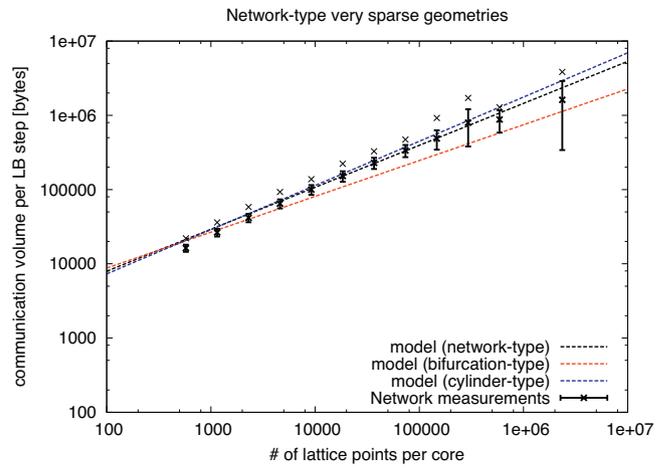

Fig. 11. As in Fig. 9 but for the network geometry (using the network simulation domain).

3.1.3. Characterising load imbalances

When using sparse geometries, individual processes within HemeLB contain subsets of the simulated system with heterogeneous shapes and sizes. These differences result in two types of load imbalance during the parallel LB calculation: a calculation load imbalance and a communication load imbalance. To obtain a platform-independent measure of the load imbalance in HemeLB, we choose not to include timing results in this procedure. Instead, we examine the number of lattice sites on each core to determine the calculation imbalance and the number of bytes sent by each process to determine the communication imbalance. Both metrics are reproducible on different platforms when using the same version of ParMETIS (4.0.2), although some variations may occur due to the stochastic nature of the ParMETIS decomposition technique.

In Fig. 12 we show the measured *calculation load imbalance* for three geometries as a function of the core count. We determine this calculation load imbalance by dividing the maximum number of lattice sites on any core within this run by the average number of sites over all cores in the same run. HemeLB is optimised for calculation load balance and we find an imbalance of less than 1.04 for most core counts. However, the calculation load imbalance is higher for both very low and very high core counts. This
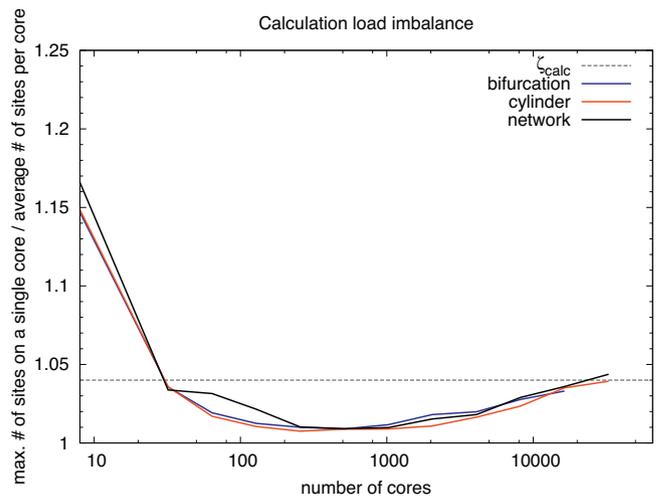

Fig. 12. Imbalance of the number of sites per core (i.e., our measure for calculation load imbalance) as a function of the number of cores for the three geometries. The value on the y-axis is the relative calculation overhead caused by load imbalance. These values are deterministic for a given core count and ParMETIS version.
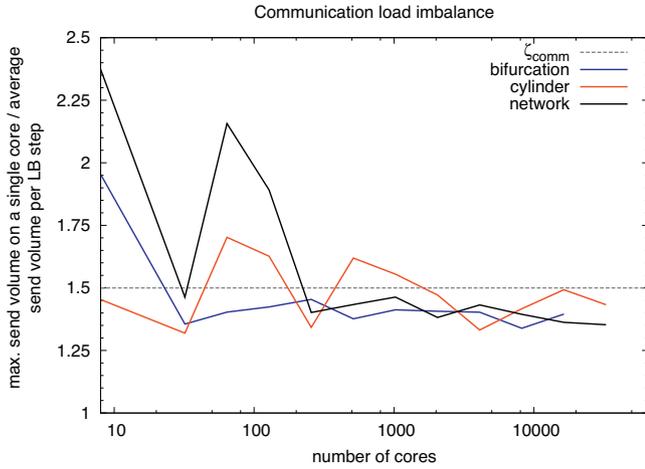
Fig. 13. Imbalance in the number of bytes sent (i.e., our measure for communication load imbalance) as a function of the number of cores for the three geometries. The value on the *y*-axis is the relative communication overhead caused by load imbalance. These values are deterministic for a given core count and ParMETIS version.

contributes in part to the superlinear scaling of HemeLB at lower core counts in some cases, and reduces scalability when there are less than 2000 lattice sites per core. Based on these measurements, we assume a calculation load imbalance (ζ_{calc}) of 1.04 in our performance model. In Fig. 13 we present the *communication load imbalance*, which we measure by dividing the maximum number of bytes sent by a single core in the run by the average number of bytes sent per core. All the communication measurements are given per step. We observe a large and erratic imbalance in the communication sizes. The ParMETIS domain distribution algorithm co-optimises for both calculation load balance and communication minimisation. However, these results suggest that it does not optimise for communication balance. This communication imbalance does not strongly diminish the code performance unless the performance is already dominated by communication. Within our model we take an approximate average of our measurements, and assume a communication load imbalance (ζ_{comm}) of 1.5.

3.2. LB calculations

To model the performance of the core LB simulator code we propose a time-complexity model which is loosely based on [22,23] but largely simplified. We use a range of parameters which we derived in Section 3.1. In this model we approximate the overall time spent to perform a single simulation step in HemeLB (T_{step}), using

$$T_{step} = \frac{\zeta_{calc} \times T_{calc} + \zeta_{comm} \times T_{comm}}{1.0 - O_{monitoring}}, \tag{2}$$

where T_{calc} is the average calculation time per core, (ζ_{calc}) is the calculation load imbalance constant, T_{comm} is the communication time per core, ζ_{comm} is the communication load imbalance constant and $O_{monitoring}$ is the fraction of time spent on monitoring overhead. Throughout our runs we found that ∼6% of the runtime is spent on monitoring, so we define $O_{monitoring} = 0.06$. The average calculation time per core is given by

$$T_{calc} = \frac{(N/P)}{\tau} \tag{3}$$

Here, the total number of lattice sites is given by *N* and the number of cores by *p*. We define the SUPS per core τ as a platform dependent constant for the HECToR machine in Table 5. We measured τ as an average from our HemeLB runs with 32 cores (1 node). The true SUPS capacity per core depends slightly on the number of sites per

Table 5
List of constant values used in our performance model. The λ value was measured using a ping test between nodes on HECToR. The σ value was taken by dividing the MPI point-to-point bandwidth specification on the HECToR website [29] (at least 5 GB/s) by the number of cores per node (32).

Constant name	Value
τ	1.57×10^6 SUPS per core (calc only)
λ	$2.5 \times 10^{-5} [s]$
σ	160 MB/s per core
ζ_{calc}	1.04
ζ_{comm}	1.5
$O_{monitoring}$	0.06

Table 6
List of fitting functions used in our performance model. Here the total number of lattice sites is given by *N* and the number of cores used by *p*.

Constant name	Value
$S_{cylinder}$	$1898 \times (N/P)^{0.482719}$ bytes per core per step
$S_{bifurcation}$	$942.0 \times (N/P)^{0.595517}$ bytes per core per step
$S_{network}$	$1176 \times (N/P)^{0.613449}$ bytes per core per step

core, but is in almost all cases within 20% of this average value. We model the time spent on communications, T_{comm}, using Table 6.

$$T_{comm} = \log_2(P) \times \lambda + \frac{S_{<x>}}{\sigma}, \tag{4}$$

where λ is the point-to-point latency of MPI communications between nodes in seconds, and σ the average throughput capacity per core in bytes. We assume that the number of messages exchanged per time step increases with the number of processes and we model this as $\log(P)$. The number of bytes sent out per core per step ($S_{<x>}$) is dependent on the geometry used as well as the number of sites per core. We have provided basic fits for three geometry layouts with different sparsity (network, bifurcation and cylinder) in Table 5. These fits are most accurate for simulations that have between 5000 and 200,000 sites per core.

3.3. Visualisation

When image rendering and writing is enabled in HemeLB, some overhead is introduced in the execution, and the new time per step (T_{step_vis}) becomes

$$T_{step\_vis} = T_{step} + T_{images}, \tag{5}$$

where T_{images} is the overhead for rendering and writing images. Because our overhead measurements show a large variability, we use a straightforward fit rather than a detailed sub-model to approximate this overhead. Based on our measurements on 2048 cores, we have derived an approximate fit of $T_{images} = 21.6k^{-0.76}$, with *k* being the number of LB steps per rendered image. We provide a graphical overview of the approximation in Fig. 6.

4. Model validation

4.1. Validation on HECToR

We have applied our performance model to calculate the theoretical execution times of the simulations we presented in Section 2.1. The predictions given by the model, as well as the measurements presented earlier, can be found in Fig. 14 for the Cylinder, Bifurcation and Large Bifurcation simulation domains and in Fig. 15 for the Network, Small Network and Large Network simulation domains. The predictions from our model are generally in agreement with our measurements, especially for the larger simulation domains. However, the model does not reproduce the superlinear speedup measured in the results. This is mainly because the model
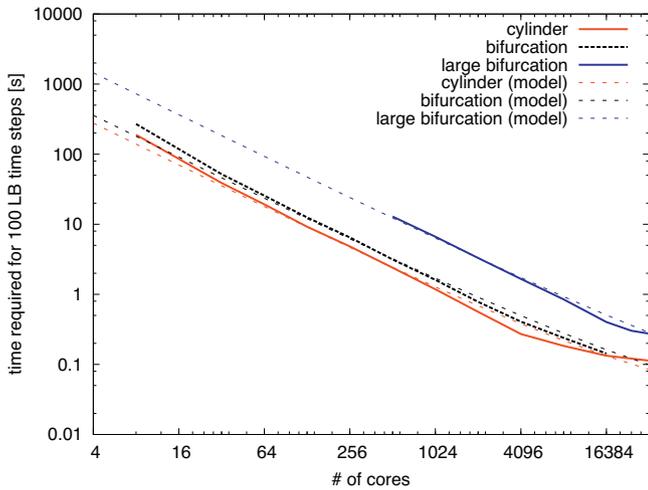
Fig. 14. Wall-clock time spent to simulate 100 time steps as a function of the number of cores used for the Cylinder, Bifurcation and Large Bifurcation simulation domains. These validation runs were done using the HECToR supercomputer. Predictions by our performance model are indicated by the dashed lines.

assumes a constant calculation and communication load imbalance, regardless of core count. In contract we measure relatively large calculation and communication load imbalances for runs on less than 32 cores (see Figs. 12 and 13). In this regime, the measured load imbalances are considerably higher than the ones assumed in our model, and the execution time is consequently slightly higher than in our model predictions.

4.2. Validation on SuperMUC

To test whether our performance model holds when applied to a different platform, we used a small part of an allocation arranged by MAPPER on the SuperMUC supercomputer at the Leibniz-Rechenzentrum in Garching, Germany. SuperMUC is an IBM System x iDataPlex machine with 147,456 compute cores and a total peak performance of 3.185 PFLOP/s (21.6 GFLOP/s per core). Each node has 16 cores, consists of two Intel Xeon E5-2680 CPUs, and is equipped with 32 GB of memory. The nodes are interconnected with an Infiniband FDR10 network, which divides the supercomputer into *islands*, each of which contains 8192 cores. We use this machine to run HemeLB simulations using the Bifurcation simulation domain, and to compare our measurements to our
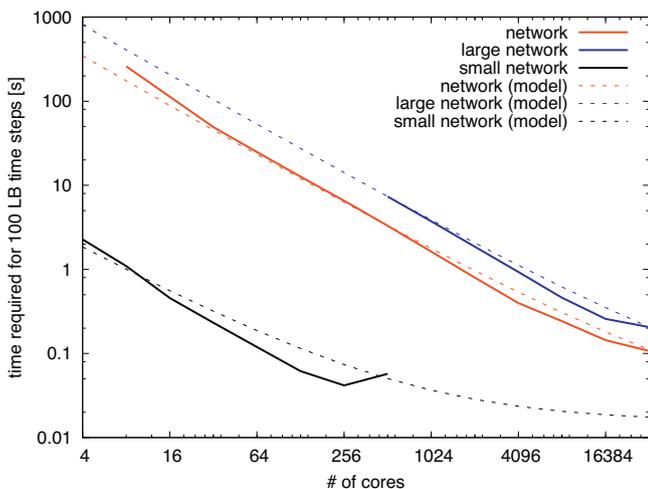
Table 7
List of constant values used in our performance model for SuperMUC. The λ value was measured using a `ping` test between nodes on SuperMUC. The σ value was taken by dividing the MPI point-to-point bandwidth specification on the SuperMUC website [29] (at least 5 GB/s) by the number of cores per node (16).

Constant name	Value
τ	4.2×10^6 SUPS per core (calc only)
λ	$1.83 \times 10^{-4} [s]$
σ	500 MB/s per core

model predictions. We provide the list of constant values in Table 7. The values of ζ_{calc}, ζ_{comm} and $O_{monitoring}$ are the same as those we used for HECToR, as these constants do not depend on the underlying architecture. We obtained a σ value of 500 MB/s per core through direct correspondence with LRZ, and measured a λ of 1.83×10^{-4} s by running a ping job between two nodes within the same island, and taking the average from 10 pings. As all small jobs on SuperMUC tend to get scheduled on the same island, it was unfortunately not possible to accurately measure the latency between islands. We obtained the value of τ by running a very short HemeLB simulation on one node and extracting the calculation rate per core, excluding any communications or other overhead (4.2×10^6 SUPS).

We present both our model predictions and our performance measurements in Fig. 16. Here we find that our simulation runs considerably faster on SuperMUC than on HECToR, achieving 3.49 MSUPS per core when using 128 cores, and 3.00 MSUPS per core when using 2048 cores. Our performance model accurately predicts the runtime for simulations up to 4096 cores, and matches the measured performance even more closely than in the HECToR validation tests. We have performed 2 runs using 8192 cores, one using one island, and on distributed over two islands. Our performance model predicts a time which is higher than the measured time for the single-island run. This may be because the `ping` test we used over the Infiniband has given us a somewhat higher λ than the actual point-to-point latency of communications in the MPI layer. Communications between islands experience much higher latency and lower bandwidth (at a 4:1 ratio). As a result, the run performed using two islands is an order of magnitude slower than the run using one island. Understanding the performance across islands would require us to assess the latency and bandwidth characteristics of the inter-island links (which would require a special access mode), and incorporate these in a separate "inter-island" set of the parameters λ and σ.

Fig. 15. As Fig. 14, but for the Network, Small Network and Large Network simulation domains. These validation runs were done using the HECToR supercomputer.
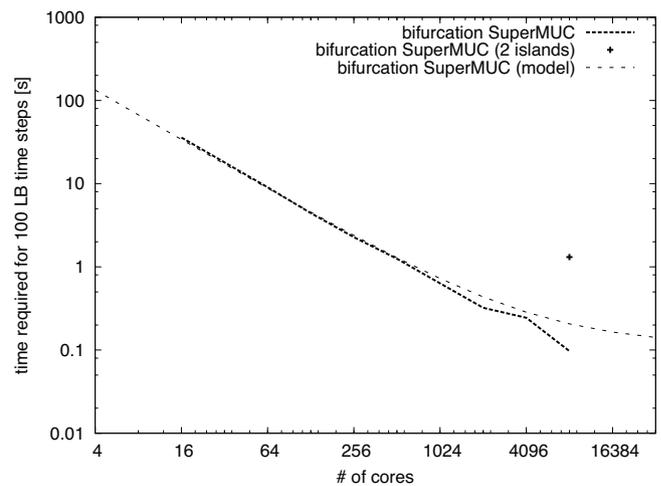

Fig. 16. Wall-clock time spent to simulate 100 time steps as a function of the number of cores used for the Bifurcation simulation domains, using the SuperMUC supercomputer. Predictions by our performance model are indicated by the dashed line.

5. Discussion

We have presented a range of performance measurements for HemeLB, covering the lattice Boltzmann simulation and the visualisation and steering functionalities. For the models studied here, HemeLB scales near-linearly up to 32,768 cores, even for highly sparse simulation domains such as vascular networks. The application achieves close to maximum efficiency when using between 5000 and 500,000 lattice sites per core. We have shown that HemeLB can render and write images once every 100 timesteps with an overhead of ∼10%, sharing streaming images and control with a steering client at 4.6 frames/s with a 28% overhead. We have demonstrated that it is possible to create a model which can estimate the run time of HemeLB simulations in advance. In our validation tests, we find that the predictions are between 70% and 140% of the actual runtime for simulations with at least 5000 lattice sites per core, and that our model remains largely accurate when applied to a different architecture (SuperMUC). We believe that accurate runtime predictions will be useful in the long term when HemeLB is used in a clinical setting, as doctors will be able to select the simulation with the highest accuracy that still meets the deadline for actual treatment.

To improve the accuracy of HemeLB simulations, as part of the MAPPER project [30], we have developed an intercommunication layer that allows the code to exchange boundary information with other simulation codes [31]. These couplings allow us to incorporate phenomena that are not resolved in HemeLB itself, such as the interaction between the blood flow in the intracranial vasculature and that in the rest of the human body. The boundary exchanges in these coupled simulations occur at high frequency and require rapid response times on both ends. The performance bottlenecks we have identified allow us to take the necessary steps to ensure an optimal performance for multiscale simulations using HemeLB.

The envisaged use-case for HemeLB, involving deployment within a clinical setting, is made more difficult by typical queuing and scheduling policies for supercomputers. One important benefit of supercomputing lies in enabling results to be produced in a timely fashion. With typical scheduling policies, however, many codes produce results only after a lengthy wait in a queuing system, significantly reducing the value-added of the supercomputing resource relative to a long-running simulation on a smaller machine. The value of supercomputing is particularly apparent when using interactive visualisation and steering [2], as this enables complex simulations to be investigated on timescales close to those of human engagement. However, this form of interaction is not possible without an advance reservation facility, enabling one to predict the time when one will be able to interact with the running simulation.

In particular, in the clinical context, patients and physicians already interact within a complex resource availability and scheduling environment. In this case, advance reservation will be necessary to make computing resources available concurrently with medical equipment, physicians, and patient needs. Furthermore, when HemeLB is used in a clinical context, rapid access to computing resources will become a safety-critical factor. This requires not just advance reservation, but support for urgent computing [4]. For the use cases we envisage for HemeLB, an urgent computing mechanism will need to be available on supercomputing resources.

Acknowledgements

References

[1] M.D. Mazzeo, P.V. Coveney, HemeLB: a high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries, Computer Physics Communications 178 (12) (2008) 894–914, http://dx.doi.org/10.1016/j.cpc.2008.02.013.

[2] M.D. Mazzeo, S. Manos, P.V. Coveney, In situ ray tracing and computational steering for interactive blood flow simulation, Computer Physics Communications 181 (2010) 355–370, http://dx.doi.org/10.1016/j.cpc.2009.10.013.

[4] S.K. Sadiq, M.D. Mazzeo, S.J. Zasada, S. Manos, I. Stoica, C.V. Gale, S.J. Watson, P. Kellam, S. Brew, P.V. Coveney, Patient-specific simulation as a basis for clinical decision-making, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 366 (1878) (2008) 3199–3219, http://dx.doi.org/10.1098/rsta.2008.0100.

[4] ParMETIS – http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview (2012).

[5] H.B. Carver, D. Groen, J. Hetherington, R.W. Nash, M.O. Bernabeu, P.V. Coveney, Coalesced communication: a design pattern for complex parallel scientific software, arXiv:1210.4400.

[6] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, T. Zeiser, Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures, in: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC'04, IEEE Computer Society, Washington, DC, USA, 2004, p. 21, http://dx.doi.org/10.1109/SC.2004.37.

[7] S. Geller, M. Krafczyk, J. Tölke, S. Turek, J. Hron, Benchmark computations based on lattice-Boltzmann finite element and finite volume methods for laminar flows, in: Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science, Computers and Fluids 35 (8–9) (2006) 888–897, http://dx.doi.org/10.1016/j.compfluid.2005.08.009.

[8] S. Williams, L. Oliker, J. Carter, J. Shalf, Extracting ultra-scale lattice Boltzmann performance via hierarchical and distributed auto-tuning, in: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2011, pp. 1–12.

[9] M. Schönherr, K. Kucher, M. Geier, M. Stiebler, S. Freudiger, M. Krafczyk, Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs, Computers & Mathematics with Applications 61 (12) (2011) 3730–3743, http://dx.doi.org/10.1016/j.camwa.2011.04.012.

[10] J. Habich, C. Feichtinger, H. Kostler, G. Hager, G. Wellein, Performance engineering for the lattice Boltzmann method on GPGPUs: architectural requirements and performance results, Computers & Fluids, http://dx.doi.org/10.1016/j.compfluid.2012.02.013, in press.

[11] C. Obrecht, F. Kuznik, B. Tourancheau, J.J. Roux, A new approach to the lattice Boltzmann method for graphics processing units, in: Proceedings of ICMMES-09, Mesoscopic Methods for Engineering and Science, Computers and Mathematics with Applications 61 (12) (2011) 3628–3638, http://dx.doi.org/10.1016/j.camwa.2010.01.054.

[12] C. Obrecht, F. Kuznik, B. Tourancheau, J. Roux, The TheLMA project: multi-GPU implementation of the lattice Boltzmann method, International Journal of High Performance Computing Applications 25 (3) (2011) 295–303, http://dx.doi.org/10.1177/1094342011414745.

[13] J. Myre, S.D.C. Walsh, D. Lilja, M.O. Saar, Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters, Concurrency and Computation: Practice and Experience 23 (4) (2011) 332–350, http://dx.doi.org/10.1002/cpe.1645.

[14] A. Gray, A. Hart, A. Richardson, K. Stratford, Lattice Boltzmann for large-scale GPU systems, in: Advances in Parallel Computing, vol. 22: Applications, Tools and Techniques on the Road to Exascale Computing, 2011, pp. 167–174.

[15] S. Donath, K. Iglberger, T. Zeiser, A. Nitsure, U. Rude, Performance comparison of different parallel lattice Boltzmann implementations on multi-core multi-socket systems, International Journal of Computational Science and Engineering 4 (1) (2008) 3–11.

[16] V. Heuveline, M. Krause, J. Lätt, Towards a hybrid parallelization of lattice Boltzmann methods, Computers and Mathematics with Applications 58 (5) (2009) 1071–1080.

[17] J. Fietz, M.J. Krause, C. Schulz, P. Sanders, V. Heuveline, Optimized hybrid parallel lattice Boltzmann fluid flow simulations on complex geometries, in: C. Kaklamanis, T. Papatheodorou, P. Spirakis (Eds.), Euro-Par 2012 Parallel Processing,

Vol. 7484 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2012, pp. 818–829, http://dx.doi.org/10.1007/978-3-642-32820-6_81.

[18] M.J. Harvey, G. de Fabritiis, G. Giupponi, Accuracy of the lattice-Boltzmann method using the cell processor, Physics Review E 78 (5) (2008) 056702.

[19] S. Williams, J. Carter, L. Oliker, J. Shalf, K. Yelick, Optimization of a lattice Boltzmann computation on state-of-the-art multicore platforms, Journal of Parallel and Distributed Computing 69 (9) (2009) 762–777.

[20] L. Biferale, F. Mantovani, M. Pivanti, M. Sbragaglia, A. Scagliarini, S.F. Schifano, F. Toschi, R. Tripiccione, Lattice Boltzmann fluid-dynamics on the QPACE supercomputer, Procedia CS 1 (1) (2010) 1075–1082.

[21] J. Bernsdorf, Simulation of complex flows and multi-physics with the lattice-Boltzmann method, Ph.D. thesis, University of Amsterdam (2008).

[22] L. Axner, High performance computational hemodynamics with the lattice Boltzmann method, Ph.D. thesis, University of Amsterdam (2007).

[23] L. Axner, J. Bernsdorf, T. Zeiser, P. Lammers, J. Linxweiler, A.G. Hoekstra, Performance evaluation of a parallel sparse lattice Boltzmann solver, Journal of Computational Physics 227 (10) (2008) 4895–4911.

[24] P.L. Bhatnagar, E.P. Gross, M. Krook, A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems, Physical Review 94 (1954) 511–525, http://dx.doi.org/10.1103/PhysRev.94.511.

[25] J. Harting, M. Venturoli, P.V. Coveney, Large-scale grid-enabled lattice Boltzmann simulations of complex fluid flow in porous media and under shear, Philosophical Transactions of the Royal Society A 362 (1821) (2004) 1703–1722, http://dx.doi.org/10.1098/rsta.2004.1402.

[26] B. Chopard, D. Lagrava, O. Malaspinas, R. Ouared, J. Latt, K.O. Lovblad, V. Pereira-Mendes, A lattice Boltzmann modeling of bloodflow in cerebral aneurysms, in: J.C.F. Pereira, A. Seguira (Eds.), V: European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2010, vol. 453, 2010, p. 12.

[27] Palabos LBM Wiki – http://wiki.palabos.org/ (2011).

[28] M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, E. Kaxiras, J.K. Sircar, MUPHY: a parallel MUlti PHYsics/scale code for high performance biofluidic simulations, Computer Physics Communications 180 (2009) 1495–1502, http://dx.doi.org/10.1016/j.cpc.2009.04.001.

[29] HECToR – http://www.hector.ac.uk (2012).

[30] MAPPER: Multiscale Applications on European e-Infrastructures – http://www.mapper-project.eu (2012).

[31] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R.W. Nash, S.J. Zasada, I. Saverchenko, M. Mamonski, K. Kurowski, M.O. Bernabeu, A.G. Hoekstra, P.V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus 3 (2) (2013), 20120087, http://dx.doi.org/10.1098/rsfs.2012.0087.

Hywel B. Carver received his MEng degree in Information Engineering from King's College, Cambridge, with Distinction in 2008. He is now a second-year PhD student, interested in using high-performance computing as a predictive tool in solving clinical problems. His current focus is the application of lattice-Boltzmann to intracranial haemodynamics, as a means of predicting rupture risks in neurovascular aneurysms.

Rupert W. Nash completed his doctoral studies in the School of Physics at the University of Edinburgh in 2010, studying simple models of swimming particles using lattice-Boltzmann methods. He has since been at the Centre for Computational Science at UCL, working on computational haemodynamics. His research interests are focussed on lattice-Boltzmann methods and high performance computing.

Miguel O. Bernabeu received his DPhil in computational biology from the University of Oxford, UK, in 2011 and his MSc and BEng from the Universidad Politécnica de Valencia, Spain, in 2005 and 2007. He is currently a 2020 Science Research Fellow at the Centre for Computational Science and CoMPLEX, University College London, UK. He has worked in various UK and EU projects developing HPC software for different aspects of cardiovascular modelling and simulation. His research interests include software engineering, parallel computing, and numerical methods with applications to cardiac electrophysiology and brain haemodynamics.

Prof. Peter V. Coveney holds a Chair in physical chemistry and is Director of the Centre for Computational Science, and an Honorary Professor in Computer Science. He is also professor adjunct within the Medical School at Yale University, and Director of the UCL Computational Life and Medical Sciences Network. Coveney is active in a broad area of interdisciplinary theoretical research including condensed matter physics and chemistry, materials science, life and medical sciences. He has published over 300 papers, books and edited works, including the acclaimed bestsellers The Arrow of Time and Frontiers of Complexity, both with Roger Highfield.

Derek Groen is a post-doctoral research associate in the CCS at University College London, specialised in multiscale simulation and parallel/distributed computing. He has worked with a wide range of applications, including those using lattice-Boltzmann, N-body and molecular dynamics methods, and participated in several EU-funded IT projects. He finished his PhD in 2010 at the University of Amsterdam, where he investigated the performance of N-body simulations run across geographically distributed (supercomputing) infrastructures. Derek currently works on constructing and testing multiscale simulation applications of cerebrovascular blood flow and clay–polymer nanocomposites.

James Hetherington is a research software engineer, combining computational science research experience with professional software development skills. He helps UCL to produce maintainable, usable, well-tested scientific software which with a lasting impact, bringing software engineering best practice into computational research. At AMEE UK Limited, he developed systems to make it easier for organisations to understand their environmental impacts, creating AMEE Explorer, winner of a Best of What's New award in Popular Science Magazine in 2010. He has also worked on software for model management at the MathWorks, systems biology at UCL CoMPLEx, and particle physics at the Cavendish Laboratory, Cambridge.