Journal of Computational Science 4 (2013) 412–422

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

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

ARTICLE INFO

Article history:
Received 29 August 2012
Received in revised form 22 January 2013
Accepted 17 March 2013
Available online 26 March 2013

Keywords:
Lattice-Boltzmann
Parallel computing
High-performance computing
Performance modelling

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.

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).

1877-7503/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jocs.2013.03.002

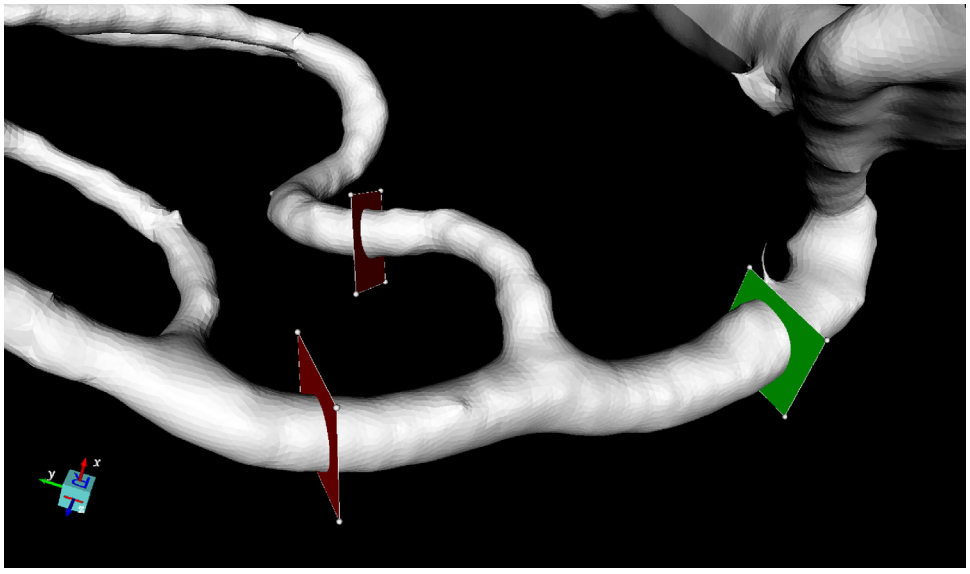

Fig. 1. Graphical overview of the bifurcation geometry in the HemeLB Setup Tool. We used this geometry to generate the Bifurcation and Large Bifurcation simulation domains. Inlets are shown by green planes, outlets by red planes. (For interpretation of the references to color in this figure legend, the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