Development and performance comparison of a HemeLB GPU code for human-scale blood flow simulation

Zacharoudiou, I.[1], McCullough, J.W.S.[1], Coveney, P.V.[1,2]

[1] Centre for Computational Science, Department of Chemistry, University College London, UK

[2] Institute of Informatics, University of Amsterdam, The Netherlands

1. Introduction

In the field of computational biomedicine, significant effort is being invested into the development of the virtual human - a digital replica of an individual's physiology. A full virtual human would allow clinicians, scientists and healthcare professionals to make use of patient-specific simulations and predictive models to optimise the care provided to an individual at all stages of life. Conducting simulations for the virtual human will demand codes that can both execute and scale efficiently on large-scale computing infrastructure.

The purpose of this paper is to provide another step forward in the efforts of making the virtual human a reality by taking advantage of the accelerators such as Graphics Processing Units (GPUs) that are becoming commonplace on supercomputers globally. We do this by developing a GPU version of HemeLB [1, 2], a high-performance lattice Boltzmann (LB) based fluid flow solver for simulating blood flow on patient specific images obtained from medical scans. HemeLB has been optimized for the sparse geometries characteristic of vascular trees and has demonstrated strong scaling on hundreds of thousands of CPU cores on various traditional machines including BlueWaters and SuperMUC-NG. Our vision for this work is to demonstrate HemeLB's capacity for execution on the largest and fastest current generation machines and prepare it for upcoming exascale machines. In both of these settings, a considerable part of the performance is sourced from the presence of accelerators on the nodes, typically GPUs. Here we present our implementation of a GPU version of HemeLB and provide scaling analysis up to tens of thousands of GPUs.

2. GPU version of HemeLB – Code development

Here we describe the steps taken to port the single component HemeLB CPU code to GPU architecture. The GPU version of HemeLB has been developed using the CUDA computing platform (CUDA C++) to run on NVIDIA's GPUs. Recent efforts were carried out to make the code platform agnostic, by porting the CUDA code to the HIP runtime API, making it able to run on AMD GPUs as well. We will restrict ourselves here on the CUDA version of the code.

Given the complex nature of the existing CPU version of HemeLB, our initial attempt for porting the HemeLB code on GPU architectures focused on exporting the compute intensive parts on the GPU (device), without making significant changes to the remaining structure of the code. These compute intensive parts involve mainly the collision-streaming steps of the LB algorithm. HemeLB distinguishes the following 6 types of collision–streaming, depending on the type of fluid sites and their corresponding streaming links: 1) Inner domain: only fluid sites

without any links to any type of boundaries (walls or inlets/outlets), 2) Walls: fluid sites with a link to a solid surface, 3) Inlet, 4) Outlet, 5) Inlet with Walls and 6) Outlet with Walls.

We implemented the above 6 different collision-streaming kernels with the following assumptions:

- Lattice Type: D3Q19
- Collision operator: Single relaxation time approximation, i.e. LBGK collision
- Wall boundary conditions: mid-link Bounce-Back
- Inlet/Outlet boundary conditions: Any type of inlet and outlet boundary conditions supported by HemeLB, i.e. Pressure or Velocity boundary conditions, to drive the blood flow.

2.1. Organising computations and MPI operations

The LB algorithm has demonstrated an outstanding amenability to parallel computing. Exchange of data between neighbouring MPI ranks is only required during the streaming step when the updated distribution functions stream in and out of the domain assigned to each MPI rank. Hence, a highly effective way of hiding the MPI communication overhead and enabling scaling of LB algorithms up to extreme scales is by performing the following at each LB iteration. First, perform collision-streaming at domain edges, i.e. sites which require communication with other MPI ranks during the streaming step. Then, issue the MPI exchange (MPI_Isend). Finally, perform collision-streaming at mid-domain (sites with links on the same MPI rank), while overlapping these computations with the MPI data exchange.

2.2. Optimisation strategies

The main ***optimisation strategies*** used during the GPU code development were:

A. Change of data organisation: The CPU version of HemeLB stores the fundamental flow data in an *Array of Structures* format. In the GPU version stores this has been altered in GPU global memory to a *Structure of Arrays* scheme. This option was shown to be more suitable for the GPU architecture [3].

B. Change in the sequence of steps: The GPU version of HemeLB has reordered the pattern of MPI exchanges of data. Effectively, all GPU collision-streaming kernels (domain-edges and mid-domain) are launched, with control returned to the host, before issuing the MPI exchange of data. This allows a better overlap of computations and MPI communication.

C. Use of different CUDA streams for all GPU operations: This allows overlapping the GPU kernels' execution and the memory copies (Device to Host and Host to Device).

D. Swap the distribution functions at the end of the LB iteration: Exchange the pointers for fundamental LB data, instead of explicitly swapping the data.

3. Large scale performance comparison – CPU and GPU versions of HemeLB

HemeLB has been specifically optimized to allow excellent strong scaling performance on the sparse and irregular geometries that are characteristic of vascular domains. In our comparisons of performance on CPU and GPU based architectures, we consider two particular domains of different resolutions. The first is a discretization of the full human venous tree consisting of 1,558,375,173 fluid sites, whilst the second represents the circle of Willis arterial structure

found in the brain possessing with 10,154,448,502 sites. Whilst both of these domains are of significant magnitude, we demonstrate that even these are not sufficient to adequately occupy current petascale machines to their full extent. These domains are illustrated in Fig. 1.

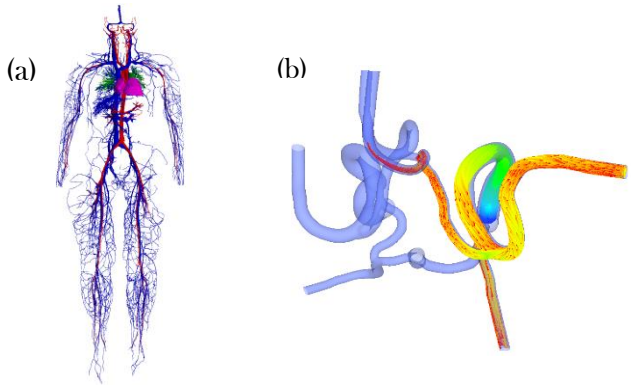

Figure 1 *Illustrations of the domains used to test the large-scale performance of HemeLB's GPU and CPU versions. (a) The blue veins represent the full human venous tree. (b) Circle of Willis.*

In this study, we are focussed on the strong scaling performance of HemeLB. A rigorous weak scaling analysis was not possible, due to the type of geometries considered. Through association with a number of research projects, HemeLB GPU has been able to be executed on a range of HPC machines and architectures including: a) Piz-Daint (CSCS - 5704 nodes, 12 CPU cores and 1 P100 GPU on each node), b) JUWELS-Cluster (JSC - 56 nodes, 40 CPU cores and 4 V100 GPUs on each node), JUWELS-Booster (JSC - 936 nodes, 48 CPU cores and 4 A100 GPUs on each node) and SUMMIT (ORNL – 4608 nodes, 42 CPU cores and 6 V100 GPUs on each node).

Fig. 2(a) shows a comparison of the CPU (on SuperMUC-NG) and GPU (on Summit) versions of HemeLB for simulations on the circle of Willis (~10 billion sites). A GPU speed-up of almost 2 orders of magnitude speed-up is observed (x85). Examining the GPU code performance and compared to a base measurement on 768 GPU cores, Fig. 2(a) demonstrates 90% perfect scaling performance on 6,144 GPUs and continued strong scaling to 18,432 GPUs (approximately 2/3 of Summit's capacity). The strong scaling efficiency drops to 72% at 12,288 GPUs and 60% at 18,432 GPUs. Increasing the simulation domain size (exaPipe geometry with 37,467,909,271 sites) increases the computation to communication ratio and consequently improves the strong scaling efficiency at 18,432 GPUs to 74%. This makes evident the improvement in performance through the use of a geometry that can better occupy the compute capacity of the machine. Given that the GPU code is still undergoing performance optimization and development, this behaviour represents an excellent outcome.

Some more normalised performance data for the CPU and GPU versions of HemeLB on a wider variety of machines are presented in Fig. 2(b). We report the performance of the simulations in MLUPS (million lattice site updates per second) on a node basis. This is informative as we have access to a broad range of high-end machines with different configurations of CPUs, GPUs, memory and bandwidth. For all machines, the drop in performance at higher node counts is due to the test domain not being sufficiently large to occupy the resources at that scale. Another interesting point is that here we can see how the construction of a node can impact performance. For example, Piz Daint, a machine with only a

single NVIDIA P100 GPU per node, delivers comparable performance to a single, CPU only, node on SuperMUC-NG. Whilst the nodes on Summit, containing 6 NVIDIA V100s, deliver a performance improvement of a factor of at least 10 on other reported machines.

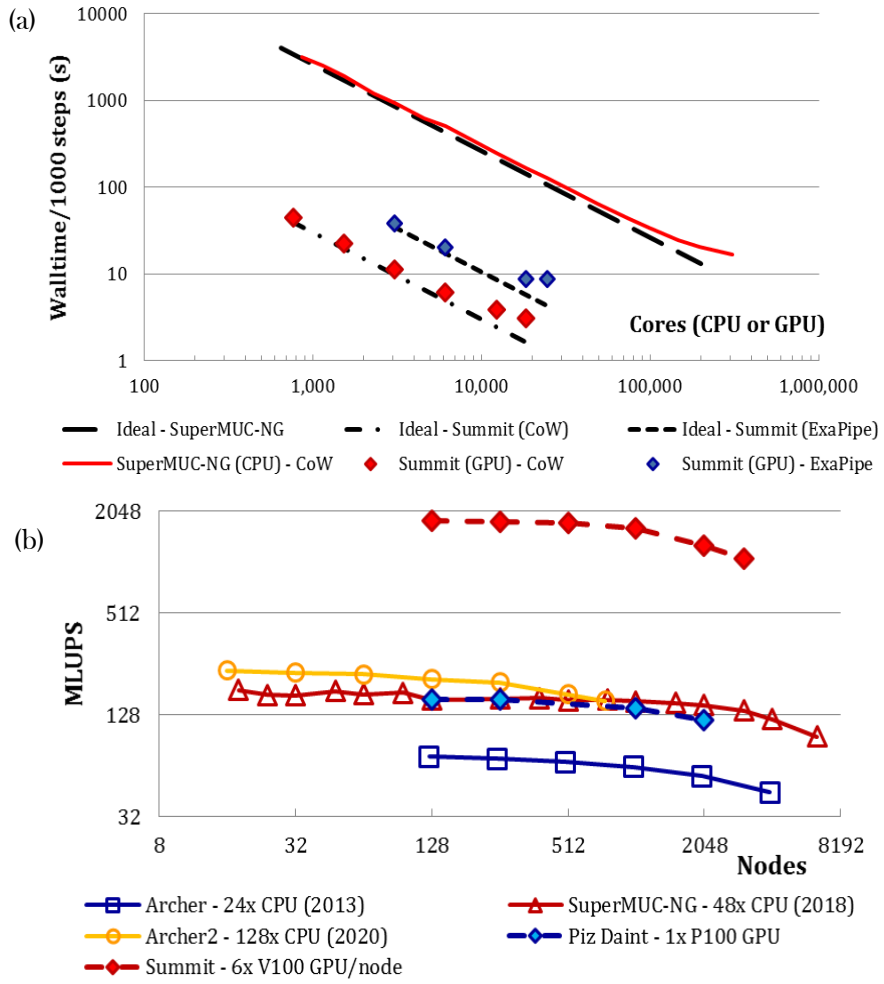

Figure 2: *Strong scaling performance:* ***(a)*** *Walltime comparison of strong scaling data of the CPU and GPU versions of HemeLB on SuperMUC-NG and Summit respectively. (b) The performance in MLUPS per node on a variety of HPC machines.*

4. Conclusions

We have successfully deployed HemeLB on GPUs by using the CUDA computing platform. We demonstrated that the GPU version of HemeLB continues to exhibit good scaling performance and therefore it is well positioned to be deployed on upcoming exascale machines.

References

[1] M.D. Mazzeo, P.V. Coveney (2008). HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. Computer Physics Communications.

[2] D. Groen, J. Hetherington, H. B. Carver, R.W. Nash, M.O. Bernabeu, P.V. Coveney (2013). Analysing and modelling the performance of the HemeLB lattice-Boltzmann simulation environment. Journal of Computational Science.

[3] N.P. Tran, M. Lee, and S. Hong (2017), Performance optimization of 3D lattice Boltzmann flow solver on a GPU. *Scientific Programming*.