# Heterogeneous System-on-Chip-Based Lattice-Boltzmann Visual Simulation System

Xiaojun Zhai, *Member, IEEE*, Minsi Chen, Sahar Soheilian Esfahani, Abbes Amira, *Senior Member, IEEE*, Faycal Bensaali, *Senior Member, IEEE*, Julien Abinahed, Sarada Dakua, Robin A. Richardson, and Peter V. Coveney

***Abstract*—Cerebral aneurysm is a cerebrovascular disorder caused by a weakness in the wall of an artery or vein, which causes a localized dilation or ballooning of the blood vessel. It is life-threatening; hence, an early and accurate diagnosis would be a great aid to medical professionals in making the correct choice of treatment. HemeLB is a massively parallel Lattice-Boltzmann simulation software, which is designed to provide the radiologist with estimates of flow rates, pressures, and shear stresses throughout the relevant vascular structures, intended to eventually permit greater precision in the choice of therapeutic intervention. However, in order to allow surgeries and doctors to view and visualize the results in real-time at medical environments, a cost-efficient, practical platform is needed. In this article, we have developed and evaluated a version of HemeLB on various heterogeneous system-on-chip platforms, allowing users to run HemeLB on a low-cost embedded platform and to visualize the simulation results in real-time. A comprehensive evaluation of implementation on the Zynq SoC and Jetson TX1 embedded graphic processing unit platforms are reported. The achieved results show that the proposed Jetson TX1 implementation outperforms the Zynq implementation by a factor of 19 in terms of site updates per second.**

***Index Terms*—Cerebral aneurysm, graphical progressing units (GPU), HemeLB, Lattice Boltzmann (LB), visualization, Zynq.**

## I. Introduction

MEDICINE and Physiology are being revolutionized through innovations made possible via the growth and application of information technology. The new technology allows the development of increasingly detailed computational models of the biological processes that sustain life, using data from both animals and humans. These models, in conjunction with the relevant experimental data, are helping researchers to gain insights into the physiology and pathology of a multitude of different biological systems, in many cases beyond what is possible through purely observational methods [1], [2].

Manuscript received February 19, 2019; revised September 20, 2019; accepted November 4, 2019. Date of publication December 2, 2019; date of current version June 3, 2020. This paper was supported by the National Priorities Research Program (NPRP) under Grant 5-792-2-328 from the Qatar National Research Fund (a member of Qatar Foundation). X. Zhai is partially supported by the UK Engineering and Physical Sciences Research Council under Grants EP/R02572X/1 and EP/P017487/1. *(Corresponding author: Xiaojun Zhai.)*

X. Zhai is with the School of Computer Science and Electronic Engineering, University of Essex, CO4 3SQ Colchester, U.K. (e-mail: xzhai@essex.ac.uk).

M. Chen is with the Department of Computer Science, University of Huddersfield, HD1 3DH Huddersfield, U.K. (e-mail: M.Chen@hud.ac.uk).

S. S. Esfahani and F. Bensaali are with the College of Engineering, Qatar University, Doha 2713, Qatar (e-mail: sahar.esfahani@qu.edu.qa; f.bensaali@qu.edu.qa).

A. Amira is with the Institute of Artificial Intelligence, De Montfort University, LE1 9BH Leicester, U.K. (e-mail: abbes.amira@dmu.ac.uk).

J. Abinahed and S. Dakua are with the Department of Surgery, Hamad Medical Corporation, Doha 3050, Qatar (e-mail: jabinahed@hamad.qa; sdakua@hamad.qa).

R. A. Richardson and P. V. Coveney are with the Centre for Computational Science, University College London, WC1E 6BT London, U.K. (e-mail: robin.richardson@ucl.ac.uk; p.v.coveney@ucl.ac.uk).

Digital Object Identifier 10.1109/JSYST.2019.2952459

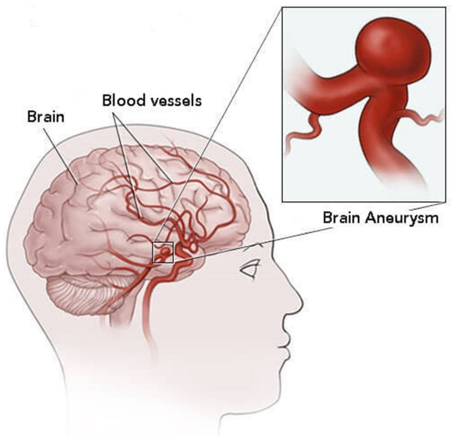

Fig. 1. Cerebral aneurysms [6].

Cerebrovascular disorders, such as the rupture of cerebral aneurysms during subarachnoid haemorrhages in Fig. 1, represent one of the most prevalent and devastating diseases of adults worldwide. There are noticeable ethnic and geographical differences in its incidence, prevalence, and outcomes. Although the incidence of such disorders tends to decrease in some developed countries, such as the U.K., where the total burden has been £0.5 billion annually [3], it is increasing in other countries and is assuming epidemic proportions in developing countries [4], [5].

Effective surgical treatments of intracranial aneurysms (ICAs) often make use of endovascular approaches, which are sometimes more effective and maintain low operative risk with short lengths of stay in hospital. These approaches are focused on using intra-aneurysmal coils and may fail in some cases due to incomplete occlusion of the defect. One solution is to use coils in combination with stents that reorganize the flow of blood into and around the aneurysm. A key factor in the successful deployment of such approach is the skill of the interventional radiologist in identifying the vascular geometry and estimating the consequent fluid flows from inspection of two-dimensional (2-D) and/or three-dimensional (3-D) images, in order to propose the best treatment for the patient. Currently, there are few methods to measure these flows and related pressures intraoperatively and the treatment is often based on the experience and intuition of the radiologist.

Computational haemodynamics techniques are widely used to estimate local fluid flow. The computational approach must not only model the native flows in the ICA but also the perturbations introduced by insertion of flow-diverting stents, as well as the effect on the pattern of clotting within coil-filled aneurysms. Most usefully, simulation input data should be specific to the patient in question, thus, taking into account the variability of vascular geometries, vessel wall mechanics, and flow-phase specific pulsatile changes in pressures and shear stresses across patients [7], or differences in physiological states for a given patient (such as heart rate or blood pressure) [8].

The use of a fluid mechanics model requires considerable computing resources for the simulation of what can often be large and complex systems, and the simulation software must efficiently deal with the (highly) sparse geometries that are common to vascular networks. Convention techniques for solving fluid flow are commonly based on finite element methods (FEM). Due to the irregularity in vascular morphology, this often implies a nontrivial task of generating meshes from complex vascular networks, and prescribing additional boundary conditions (BC) for resolving fluid–structure interaction.

To this end, we adopted a Lattice-Boltzmann (LB) solver for simulating haemodynamics and fluid–vessel interaction. Previous studies showed a comparable accuracy between LB and FEM [9], [10]. In this article, we used HemeLB, a massively parallel LB solver optimized for sparse and complex systems on large supercomputing resources [8], [11], [12]. It has been designed with the ultimate intention of allowing doctors to investigate cerebral blood flow behavior in the human body and demonstrated its potential in hospital environment [13], [14]. Additionally, HemeLB was designed to meet the requirements of both conventional scientific research software and professional software engineering techniques in an open-source way, which allows us to optimize its architecture and retarget to other hardware-accelerated platforms and evaluate the performance accordingly [15].

HemeLB has been successfully deployed on open academic high-performance computing (HPC) platforms such as HECToR, ARCHER, SuperMUC, and Blue Waters [8], [12], [13], among others. In order to allow such hemodynamic simulations to be run as part of a routine clinical practice, the workload will need to be deployed on a dedicated computational infrastructure. This is necessary to reduce the dependence of the final computational workflow on a distributed environment not fully under the control of the clinician.

A number of novel architectures: multicore, general purpose graphical progressing units (GPGPUs), and field programmable gate arrays have been developed recently, which have demonstrated great potential for accelerating computationally intensive tasks such as those found in machine learning and artificial intelligence applications [16], [17]. In this article, a solution for designing and implementing HemeLB on cost efficient embedded platforms such as to allow visualization of the simulation results in real-time and the execution of the code in the local environment of a hospital is presented, which is likely necessary for reasons of patient data protection. We first report the evaluation results of our HemeLB simulation software implementation on a cost- and energy-efficient multiprocessor system-on-chip (MPSoC) based Zynq SoC platform. We then present the results for the adaptation of the original HemeLB simulation software to a GPU implementation, which uses a Jetson TX1 embedded GPU platform. Additionally, a range of comprehensive tests using real patient input data have been carried out with this implementation, and the corresponding performance is reported. The results show that this platform could be the solution to the problem of in-hospital execution and real-time visualization, offering substantial processing performance on a number of local platforms.

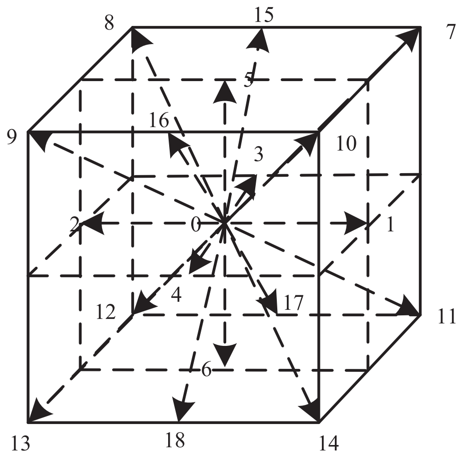

Fig. 2. Lattice node of D3Q19 model [18].

The remaining sections of this article are organized as follows. In Section II, HemeLB model is described. Section III presents the implementations of HemeLB. Experimental results and performance analysis are reported in Section IV followed by the conclusions in Section V.

## II. HemeLB Model

HemeLB uses the LB method to simulate the (continuum) fluid flow, modeled by a grid of particle density distribution functions representing ensembles of particles performing local propagation and collision processes over a discrete lattice mesh [11]. In the following system, we use a 3-D lattice with 19 discrete speeds, called the D3Q19 lattice shown in Fig. 2.

Fig. 2 illustrates one fluid site and the finite number of discrete lattice vectors $e_i$, via which fluid may travel to neighboring fluid sites in a D3Q19 model. The distribution function is assumed, in the simplest LB implementation, to evolve toward its local equilibrium value according to a single relaxation parameter $\tau$ as

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

where $f$ is the distribution function of the particles, and $\Omega$ is the Bhatnagar–Gross–Krook (BGK) collision operator. The lattice

**Algorithm 1:** Collision and Streaming Phases.

```
Input: $f_{old}$ is source buffers of particle distribution function.
Output: $f_{new}$ is destination buffers of particle distribution function
 1: procedure CollisionAndStream($N_x$, $N_y$, $N_z$, $d$)    ▷ $N_x, N_y$ and $N_z$ are the dimensions of grid; $d$ is the number of vectors.
 2:     $N \leftarrow N_x N_y N_z$                                   ▷ The number of fluid nodes.
 3:     $i \leftarrow 0$
 4:     while $i < N$ do
 5:         $f_i^{eq} \leftarrow w_i \left(p + \frac{\mathbf{e_i}\cdot\mathbf{u}}{c_s^2} + \frac{(\mathbf{e_i}\cdot\mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u}\cdot\mathbf{u}}{2c_s^2}\right)$    ▷ Calculate local equilibrium dist. func. $f_i^{eq}$
 6:         for all 19 directions do
 7:             if site is inside the geometry then              ▷ Streaming Operator
 8:                 $f_{new}(same\_direction) \leftarrow f_{old}$
 9:             else
10:                 $f_{new}(opposite\_direction) \leftarrow f_{old}$
11:             end if
12:         end for
13:         $i \leftarrow i + 1$
14:     end while
15: end procedure
```

BGK equation is then formulated as

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

In (2), the local equilibrium distribution function $f_i^{\text{eq}}$ is given as

$$f_i^{\text{eq}} = w_i \left(p + \frac{\mathbf{e_i}\cdot\mathbf{u}}{c_s^2} + \frac{(\mathbf{e_i}\cdot\mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u}\cdot\mathbf{u}}{2c_s^2}\right) \tag{3}$$

where $\tau$ is the relaxation time toward equilibrium for collisions, which is calculated separately from streaming. $w_i$ is a weight coefficient, $c_s$ is the speed of sound, $e_i$ is the particle's velocity in the direction *i*, and the hydrodynamic density *p* and macroscopic velocity *u* are determined by the distribution functions based on

$$p = \sum_{i=0}^{18} f_i = \sum_{i=0}^{18} f_i^{(\text{eq})} \tag{4}$$

$$\mathbf{u} = \sum_{i=0}^{18} \mathbf{e_i} f_i = \sum_{i=0}^{18} \mathbf{e_i} f_i^{(\text{eq})}. \tag{5}$$

The distribution functions are propagated along the lattice velocity $e_i$ to the adjacent sites. More specifically, the equilibrium distribution function $f_i^{\text{eq}}$ moves from the site at position (*x*, *y*, *z*) to the site at position (*x*, *y*, *z*) + $e_i$. The microscopic velocity in lattice nodes is given as

$$\overrightarrow{e_i} = \begin{cases} (0,0,0) & i = 0 \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm 1) & i = 1,2,3,4,5,6 \\ (\pm 1,\pm 1,0),(\pm 1,0,\pm 1),(0,\pm 1,\pm 1) & i = 7,8,9,\ldots,18. \end{cases} \tag{6}$$

The streaming calculation updates the particle distributions according to the 19 directions velocity $f_i$. Consequently, the density and velocity are calculated using (4) and (5) from $f_i$. In addition, the equilibrium distribution and the distribution function in the collision step are also calculated. Finally, the streaming and collision steps are repeated.

HemeLB is implemented with various BC, for example, velocity inlet BC: Ladd iolets [19], Bouzidi–Firdaouss–Lallemand (BFL): interpolated wall collision BC [20], and pressure iolets: mixed Dirichlet–Neumann BC [21].

## A. Computational Core of LB

A two-level data structure is used to represent a domain with a single resolution when doing early load decomposition, which includes a coarser and finer grid layers. The main benefits of this approach are that no data exchange takes place between the coarser grid and finer one, and the structure saves memory with respect to the full matrix representation. The LB algorithm has two main computation procedures: 1) collision phase; and 2) streaming phase, where the collision phase is used to compute the local velocity parameters in *n* directions (i.e., *n* = 19), and the streaming phase is used to update the velocities with the adjacent sites. Compared with the streaming phase, the collision phase contains the main arithmetic computations of the LB approach. This depends on the configuration of the model: for example, if the D3Q19 model is used, each site contains a $N^3$ cubic lattice with the 3-D data stored in 1-D arrays, and a lattice grid of integers $is\_solid[N \times N \times N]$ keeps the updates of the presence of the BC for each velocity direction *i*.

The procedure to handle the collision and streaming phases is illustrated in Algorithm 1. The run-time complexity of collision and streaming operation is $O(QN)$, where $Q$ is the number of neighboring fluid sites of a given node, and $N$ is the number of fluid site. Additionally, since there is no data dependence amongst the fluid nodes in this operation, the parallelization of the routine is relatively trivial.

## B. Parallelization Strategy in HemeLB

HemeLB simulation software uses domain decomposition as the parallelization strategy to ensure that it has fast and high quality of computational simulations. One of the main improvements made in HemeLB is to use two-level data representation, where the graph growing partitioning method is only applied to the coarser grid of the two-level data hierarchy [12]. By applying this, the memory overhead has been significantly reduced, and all the fluid lattice sites of a block are assigned to one of the processors, which would reduce the communication overhead for each fluid lattice site of a block. In addition to this, HemeLB also tries to achieve a good computational and communication balance in the distributed computing tasks, which ensures that all the parallel tasks are distributed accordingly on different processing cores. HemeLB has also been optimized to reduce inter-rank communication needed wherever possible, which significantly reduces the amount of intramachine communication in cross-site runs in grid deployments at each time step. In HemeLB, only the distribution function values are communicated, and each processor calculates its interface-dependent identifiers after domain decomposition is achieved. In addition, the two-level representations are represented by 1-D arrays, which further reduces the computational cost of accessing an element of an array, and the extra connectivity buffers help the on-processor propagation of particles between fluid lattice sites.

The output data from HemeLB are the effective pressure, velocity, and von Mises stress flow values for all (requested) fluid sites in single precision. In comparison to storing all the distribution functions for every lattice site, this approach requires six times lower memory consumption. In addition, HemeLB can be configured with a number of check points, which offers flexibility to store the intermediate simulation results to portable binary format files.

In summary, HemeLB reduces redundant operations, increases pattern regularity, simplifies the computational core, and optimizes intramachine communications. It also offers a topology-aware two-level domain decomposition to provide a high-quality domain decomposition, ensuring a good workload distribution, which is necessary for the parallel scalability of a LB simulation.

## III. Implementations of HemeLB

MPSoCs are widely deployed in HPC and application-specific embedded systems such as gaming and aerospace for real-time response. Moreover, they offer energy efficiency and performance advantages over uniprocessor architectures [22]–[24]. Thus, MPSoCs are becoming the computing engines of choice in embedded systems for real-time applications. A dramatic increase in their use is expected in the coming years and there will likely be hundreds of processors on a single chip [23]. In the following section, two novel implementations of HemeLB on the Zynq-7000 and Jetson TX1 development boards are discussed.

### A. Architecture of Zynq Implementation

The Zynq-7000 SoC was introduced to combine the software programmability of an ARM-based dual-core processor with the hardware programmability of an FPGA, which has a good potential to achieve a high performance-per-watt as well as the scalability to meet different application requirements [25]. In

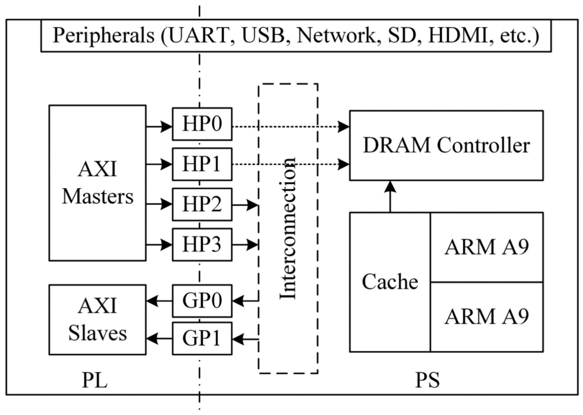

Fig. 3. Zynq Architecture.

this article, we use a Digilent Arty Z7-20 Zynq-7000 development board, equipped with a Zynq-7000 all programmable SoC, with 512 MB DDR3 memory and a 16 GB SD card. In addition, there is a 650 MHz dual-core Cortex-A9 processor together with programmable logic (PL) equivalent to Artix-7 FPGA [26]. The Arty Z7-20 has HDMI in/out ports, and audio out as well as a number of GPIO ports, directly connected to PL. In addition, there are various interfaces between processing system (PS) and PL for connecting different IP cores [27].

In this article, we adopt the Pynq framework to support our implementation, in which the PL are presented as hardware libraries that are used to support essential I/O and memory access [28]. An Ubuntu 16.04 Linux operating system is deployed on the PS, which is used to support the essential software building package for HemeLB implementation. The PS/PL interfaces of the proposed implementation are shown in Fig. 3, where PL and PS are interfaced by various AXI (advanced extensible interface) high performance (HP) and general purpose (GP) ports [16].

On the PS side, the ARM A9 processors load the PYNQ-Z1 v2.1 image from the SD card[28], which runs a 32-b Ubuntu 16.04 Linux operating system. This image provides high-level software implementation productivity for Zynq, which makes it easier to exploit the features of the HemeLB system.

### B. Architecture of Jetson Tx1 Implementation

Jetson TX1 module is NVIDIA's latest processor system-on-module for embedded applications, which has a low-cost Tegra X1 chip. Similar to the Zynq system, the Tegra X1 CPU subsystem consists of four ARM Cortex-A57 cores [29]. Additionally, the Tegra X1 also has 256 GPU cores based on the NVIDIA's Maxwell architecture. In this article, NVIDIA's CUDA technology has been used to program the GPU to handle 3-D graphics. Fig. 4 shows the architecture of the Nvidia Tegra X1 processor.

In this article, the Jetson Developer Kit is used to evaluate the performance of HemeLB implementation on the Tegra X1 processor, where JetPack 3.3 for Jetson TX1 has been installed on the kit that runs a 64-b Ubuntu 16.04 Linux operating system.

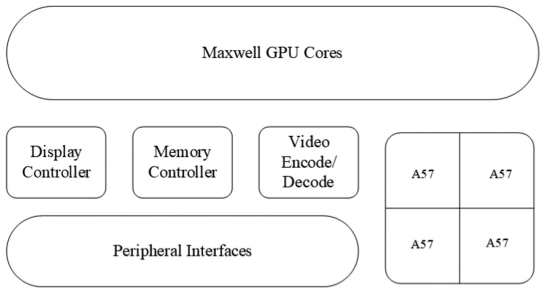

Fig. 4. Architecture of Nvidia Tegra X1.

TABLE I
HEMELB BOUNDARY SETTINGS

| cmake settings | Values |
|---|---|
| HEMELB_INLET_BOUNDARY | LADDIOLET |
| HEMELB_WALL_INLET_BOUNDARY | LADDIOLETBFL |
| HEMELB_USE_VELOCITY_WEIGHTS_FILE | ON |
| HEMELB_WALL_BOUNDARY | BFL |

TABLE II
HEMELB SIMULATION, GEOMETRY, AND INLETS/OUTLETS SETTINGS

| HemeLB settings | Values |
|---|---|
| Step length | $2 \times 10^{-5}$ s |
| Voxel size | $100 \times 10^{-6}$ |
| Inlet velocity | import from a file |
| Inlet radius | $1.9 \times 10^{-3}$ |
| Outlet amplitude | 0.0 mmHg |
| Outlet phase | 0.0 rad |
| Outlet period | 1.0 s |

## C. HemeLB Configurations

HemeLB package used in this article can be obtained from Github website [30]. In addition to this, an *openmpi* package is needed to support the compilation of HemeLB package. Once the entire package is built, we need to set a number of BC, which are listed in Table I.

In addition to the boundary settings specified by the abovementioned compilation flags, a list of settings on simulation, geometry and inlets outlets are listed in Table II.

## D. Cerebral Aneurysm Geometries

As cerebral aneurysms are a very patient specific study, clinicians from Hamad medical corporation (HMC) have carefully chosen images of three subjects of 3-D rotational angiography, which have been obtained from AXIOM-Artis in HMC. On average, each dataset consisted of 400 slices acquired along the long axes of the subjects. The data include 1 pixel per sample, average slice thickness of 0.3 mm, bit depth of 16, pixel spacing of 0.3 mm $\times$ 0.3 mm, and matrix size 512 $\times$ 512. Fig. 5 shows the three STereoLithograph (STL) files used in the experiments.

For all the STLs, the following velocity of blood flow applied in the inlet and used inflow conditions measured in the basilar artery of a patient, the time and velocity is presented in Fig. 6.

## E. Real-Time Visualization

In order to facilitate real-time interaction and simulation steering, the visualization of HemeLB is performed *in-situ* directly on dedicated GPUs with compute unified device architecture (CUDA) capability. The communication between the visualization client and the LB compute nodes was executed using the existing message passing implementation in HemeLB. The architecture of the proposed visualization framework is shown in Fig. 7, and the pseudocode for the CUDA ray casting kernel function is shown in Algorithm 2.

In Fig. 7, the compute nodes mainly handle the computations in HemeLB, and the lattice properties are calculated and cached in each node, then transferred to the master node. This node is also responsible for managing view steering and scheduling lattice data transfer from the compute nodes. The incoming lattice data are shared with CUDA-enabled GPU nodes, and the rendering are performed simultaneously.

In the current prototype, we use the direct volume rendering method for visualizing the lattice property fields of the lattice. It is worth noting that the number of voxels used for visualization does not necessarily match the size of the LB lattice. In common usage for real-time monitoring, the visualization volume can be 50% or 25% of the simulation resolution. This brought in an additional benefit of reducing the data transfer payload when requesting lattice data from the LB compute nodes.

The internal architecture of the visualization client is a three-tier system, as depicted in Fig. 8. The frontend is an OpenGL application. The middle tier is the host layer where application data including cached lattice properties and steering parameters are stored. The top tier is the rendering layer, which stores all the voxels for direct volume rendering. The CUDA cores in the top tier execute the volume rendering kernel based on the ray marching algorithm [31].

A number of memory transfers in this architecture are time-critical for real-time visualization. These include the passing of lattice properties (e.g., velocity, density, and pressure) to the visualization volume and a number of steering simulation. We deployed two-level memory access optimization that can increase throughput between the host and the GPU. First, the visualization volume was stored in a 3-D texture unit, which allowed the fast sampling of voxel values by the rendering kernel. It also allowed us to exploit the automatic trilinear interpolation when handling with the spatial resolution disparity between visualization and simulation.

Second, we used the GPU constant memory buffers for storing the viewing parameters and the lookup table for transfer function. This enabled direct memory access to be performed, thus, eliminating the need for additional bandwidth for memory copies. Furthermore, from a GPU thread's perspective, the efficiency of accessing constant memory is comparable to reading from its register.

The current implementation of the visualization client was tested on the NVIDIA Jetson TX1 development kit. Fig. 9 shows the pressure visualization results of the s10, s11, and s19 datasets.

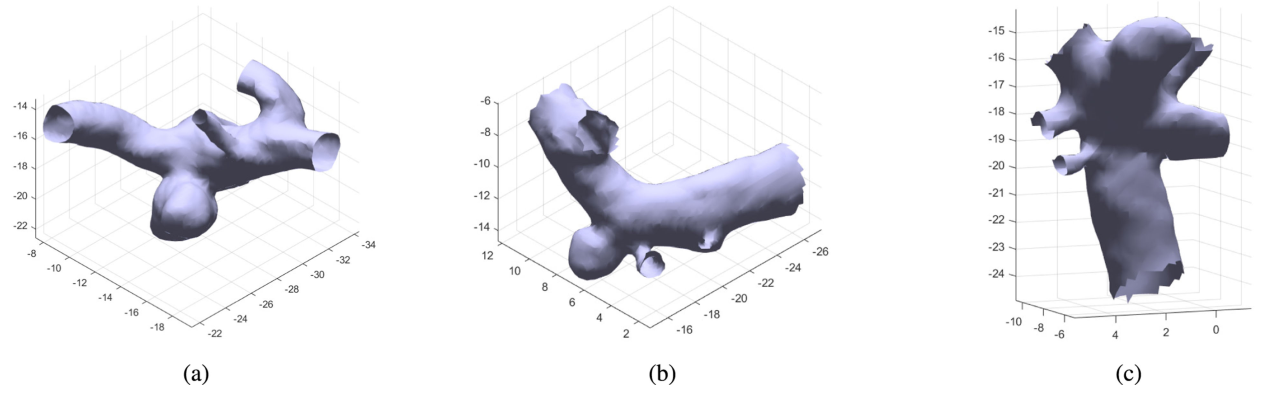

Fig. 5. Surface meshes used in our experiments. (a) Subject 10. (b) Subject 11. (c) Subject 19.

**Algorithm 2:** Pseudo Code for the CUDA Ray Casting Kernel Function.

```
Input: $VolumeData, TransferFunction, ImageWidth, ImageHeight$
Output: $OutputColour$
1: procedure VolumeRenderKernel($x, y, S, \delta s, \rho, \mathbf{M}$)    ▷ $x, y$ are pixel position; $S$ is ray marching step; $\delta s$ is ray step size; $\rho$ is voxel density; $\mathbf{M}$ is the view matrix
2:   $OutputColour \leftarrow 0$
3:   $u \leftarrow x/ImageWidth * 2 - 1$    ▷ $(u, v)$ are normalised screen coordinates
4:   $v \leftarrow y/ImageHeight * 2 - 1$
5:   $\mathbf{o} \leftarrow (0, 0, 0)$    ▷ Ray origin in canonical view space
6:   $\mathbf{d} \leftarrow (u, v, -2)$
7:   $\mathbf{d}' \leftarrow \mathbf{M}^{-1}\mathbf{d}$    ▷ $M$ is the view matrix
8:   $Ray \leftarrow (\mathbf{o}, \mathbf{d}')$    ▷ Ray defined in view space
9:   $t \leftarrow$ IntersectVolume($VolumeData, Ray$)
10:  if $0 \leq t < MAX$ then    ▷ Ray interests with the nearest voxel in Volume
11:    $\mathbf{p} \leftarrow \mathbf{o} + \mathbf{d}'t$    ▷ Position of the nearest voxel
12:    $\delta\mathbf{d} \leftarrow \mathbf{d}'\delta s$    ▷ Ray cast step vector
13:    $s \leftarrow 0$
14:    while $s < S$ do    ▷ March along the ray for $S$ steps
15:      $Value \leftarrow$ SampleVolume($p$)    ▷ Sample the value at ($\mathbf{p}$)
16:      $Colour \leftarrow$ TransferFunction($Value$)    ▷ Transfer $Value$ to RGBA
17:      $Colour \leftarrow Colour \cdot \rho$    ▷ Translucency is determined by $\rho$
18:      $OutputColour \leftarrow$ AlphaBlend($OutputColour, Colour$)    ▷ Integrate voxel colours
19:      $\mathbf{p} \leftarrow \mathbf{p} + \delta\mathbf{d}$    ▷ March along the ray by $\delta\mathbf{d}$
20:      $s \leftarrow s + 1$
21:    end while
22:  end if
23: end procedure
```

The current steerable parameters for visualization include model rotation, zooming, and adjusting the scaling and offset of the transfer functions.

## IV. Performance analysis

HemeLB has been benchmarked using simulation domains based on three distinct geometries from Fig. 5. An overview of the simulation domain used in the experiments is presented in Table III.

In Table III, the simulation were performed using a 19-directional LB kernel (D3Q19), the Lattice BKG model [32] with simple bounce-back boundary condition and a fixed physical viscosity of 0.004 Pa.s. In addition, the inlet velocities used in the simulations are from Fig. 6. Number of lattice sites for each simulation subjects are listed in Table III. In the Zynq platform simulation, the timing data are illustrated in Table IV, achieving a maximum performance of 215 751 site updates per second (SUPS) with 2 cores.

As it can be seen from Table IV, the timing performances for different subject samples are similar; however, for the large geometry subject, the SUPS is slightly higher than the small geometry subject. This means that the SUPS per core is largely independent of other factors.

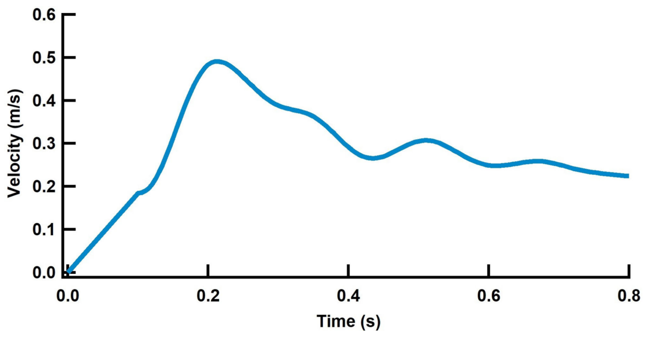

Fig. 6. Peak blood velocity profile used as input to HemeLB.

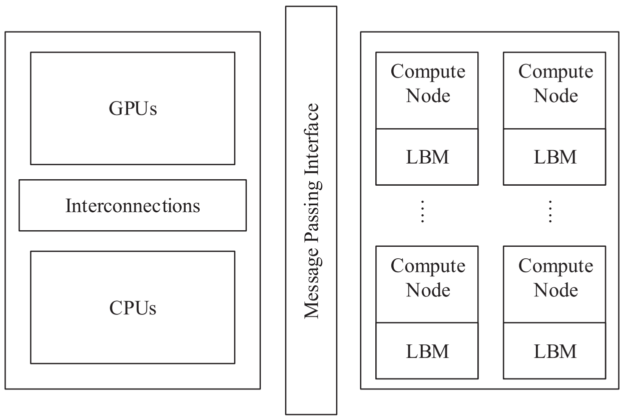

Fig. 7. Architecture of the proposed visualization framework.

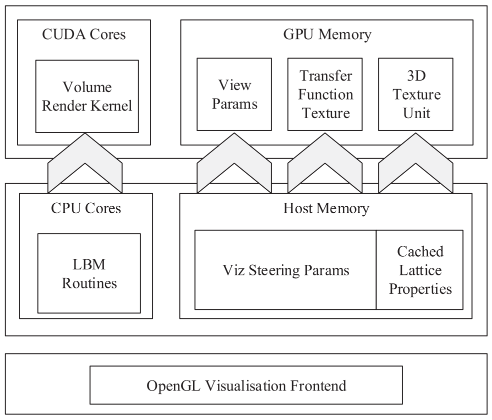

Fig. 8. Internal architecture of visualization client.

We present timing profiles for each subject using Zynq SoC in Fig. 10. As it can been seen from the comparisons in Fig. 10, the ratios from each time profile are similar, this is because the complexity of Algorithm 1 is linear, and the computational resources were constant; therefore, by increasing the lattice sites of the simulation, the proportions of each time profile were the same. The timing consumed for LB computation is slightly less than MPI communication, and both processes consume 97% of the overall processing time in the simulations.

TABLE III
OVERVIEW OF THE SIMULATION DOMAINS

| Name | Number of lattice sites | Number of blocks |
|---|---|---|
| Subject 10 | 186463 | 2880 |
| Subject 11 | 151700 | 2520 |
| Subject 19 | 81952 | 728 |

TABLE IV
TIMING DATA FOR SIMULATION ON ZYNQ PLATFORM

| Name | Total time (s) | Number of steps | SUPS |
|---|---|---|---|
| Subject 10 | 38,900 | 45,010 | 215,751 |
| Subject 11 | 36,900 | 52,232 | 214,732 |
| Subject 19 | 17,700 | 45,374 | 210,084 |

TABLE V
TIMING DATA FOR SIMULATION ON JETSON TX1 PLATFORM

| Name | Total time (s) | Number of steps | SUPS |
|---|---|---|---|
| Subject 10 | 2,020 | 45,010 | 4,154,802 |
| Subject 11 | 1,930 | 52,232 | 4,105,489 |
| Subject 19 | 907 | 45,374 | 4,099,768 |

Similar experiments have been conducted on the Jetson TX1 platform for performance evaluations. The timing data are illustrated in Table V, achieving a maximum performance of 4 154 802 SUPS with 4 cores. In comparison with the SUPS of the Zynq platform, the results from the Jetson TX1 are significantly improved, which is to say, about 19 times better than the Zynq implementation. This is mostly due to the capacities of the processors and memory on Jetson TX1 and PYNQ-Z1 boards, as the former uses a quad-core ARM Cortex-A57 processor at 2 GHz with 4 GB available memory compared to a Dual-core ARM Cortex A9 processor at 650 MHz with only 512 MB available memory. However, in terms of the SUPS results for different subjects, the Jetson TX1 platform has shown very similar performance compared to the Zynq platform, as the subjects with larger number of lattice sites achieved slightly higher SUPS compared to other subjects.

We also present timing profiles for each subject using Jetson TX1 in Fig. 11. As can been seen from the results in Fig. 11, the ratios from each time profile are similar, the timing consumed for LB computation is much higher than message passing interface (MPI) communication, this is different to the Zynq platform, which means that HemeLB works better on the Jetson TX1 platform. Overall, both LB and MPI communication processes consume 96% of the overall processing time in the simulations, which is close to the results from Zynq platform.

A list of memory throughput for each subject are reported in Fig. 12. In Fig. 12(a), we have compared the memory throughput of CUDA Host-to-Device (HtoD) for S10, S11, and S19, the peak throughput 176.06 MB/s is achieved in S19, and with an average throughput of 108.35 MB/s. The CUDA Memset (i.e., Fill block of memory in CUDA) for each subject are compared in Fig. 12(b), the throughput achieved in each subject are similar, where the peak throughput is $2.0 \times 10^4$ GB/s, and the average throughput is $1.477 \times 10^4$ GB/s, Fig. 12(c) shows the memory

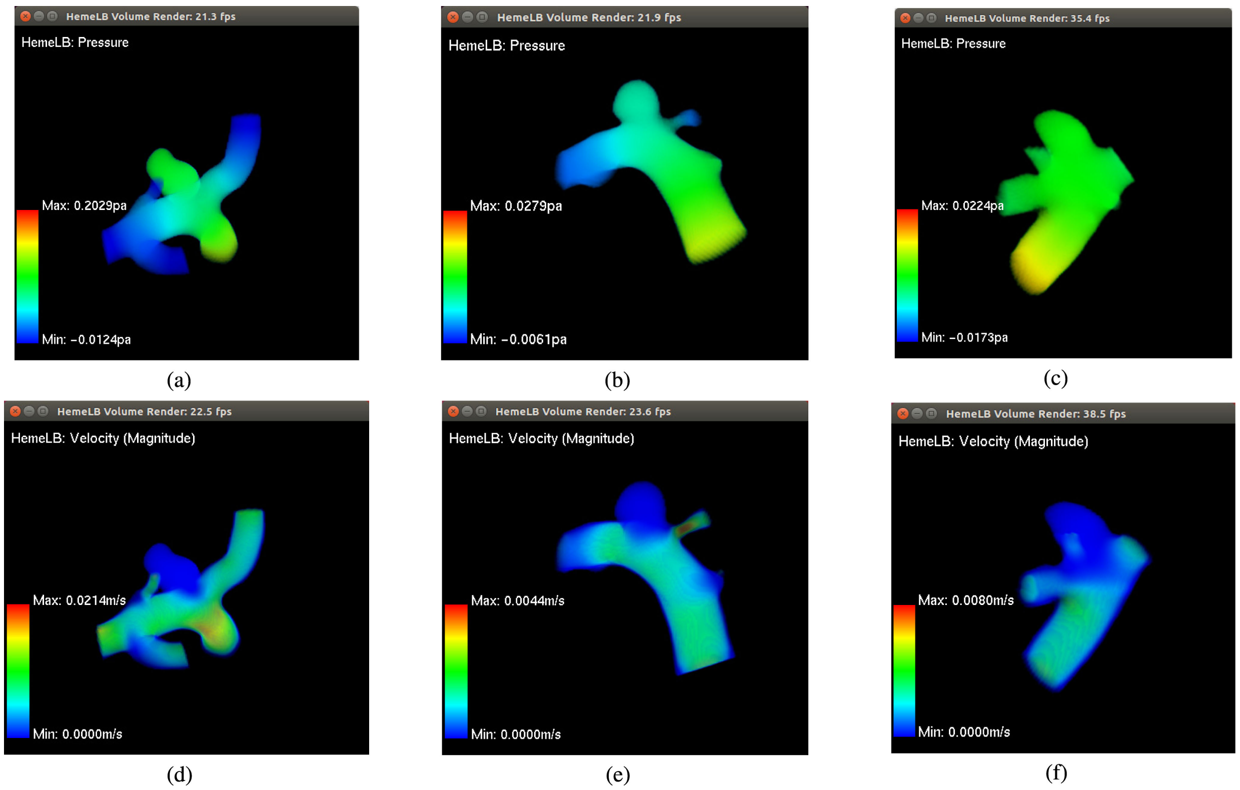
HemeLB Volume Render: 21.3 fps
HemeLB: Pressure
Max: 0.2029pa
Min: −0.0124pa
(a)
HemeLB Volume Render: 21.9 fps
HemeLB: Pressure
Max: 0.0279pa
Min: −0.0061pa
(b)
HemeLB Volume Render: 35.4 fps
HemeLB: Pressure
Max: 0.0224pa
Min: −0.0173pa
(c)
HemeLB Volume Render: 22.5 fps
HemeLB: Velocity (Magnitude)
Max: 0.0214m/s
Min: 0.0000m/s
(d)
HemeLB Volume Render: 23.6 fps
HemeLB: Velocity (Magnitude)
Max: 0.0044m/s
Min: 0.0000m/s
(e)
HemeLB Volume Render: 38.5 fps
HemeLB: Velocity (Magnitude)
Max: 0.0080m/s
Min: 0.0000m/s
(f)

Fig. 9. Visualization results from the visualization client. The pressures at each lattice site was shown using an interactively adjustable transfer function. (a) Subject 10: Pressure. (b) Subject 11: Pressure. (c) Subject 19: Pressure. (d) Subject 10: Velocity. (e) Subject 11: Velocity. (f) Subject 19: Velocity.

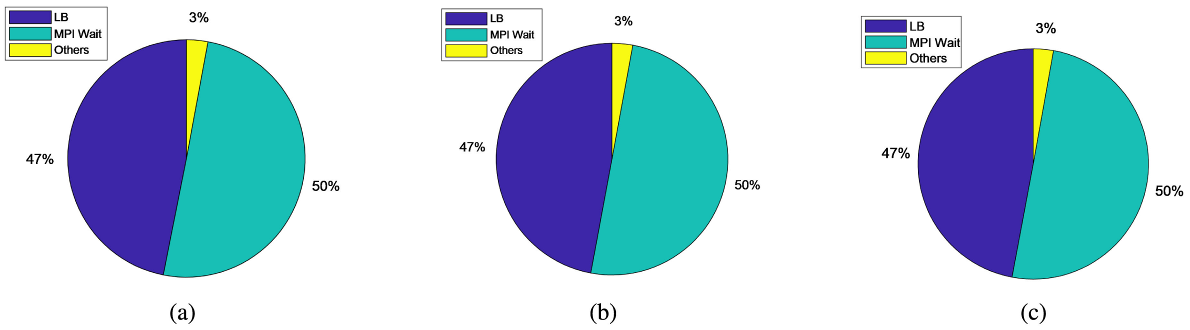

Fig. 10. Timing profile of each subject using Zynq SoC. (a) Subject 10. (b) Subject 11. (c) Subject 19.

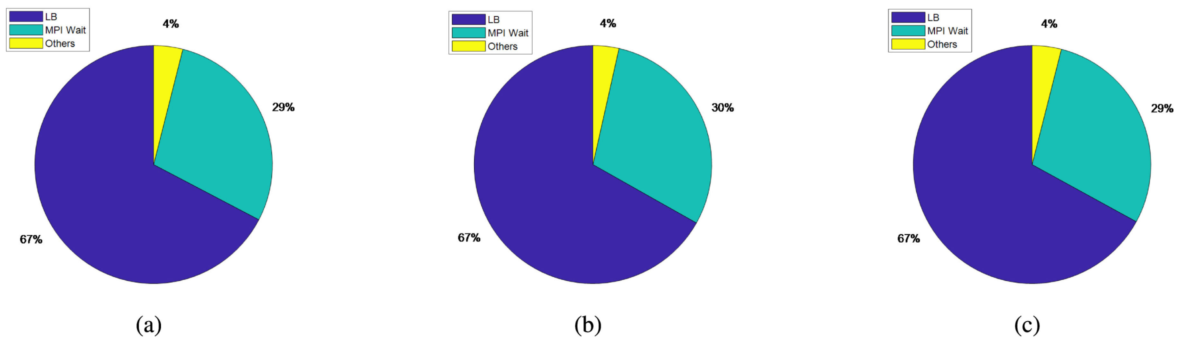

Fig. 11. Timing profile of each subject using Jetson Tx1. (a) Subject 10. (b) Subject 11. (c) Subject 19

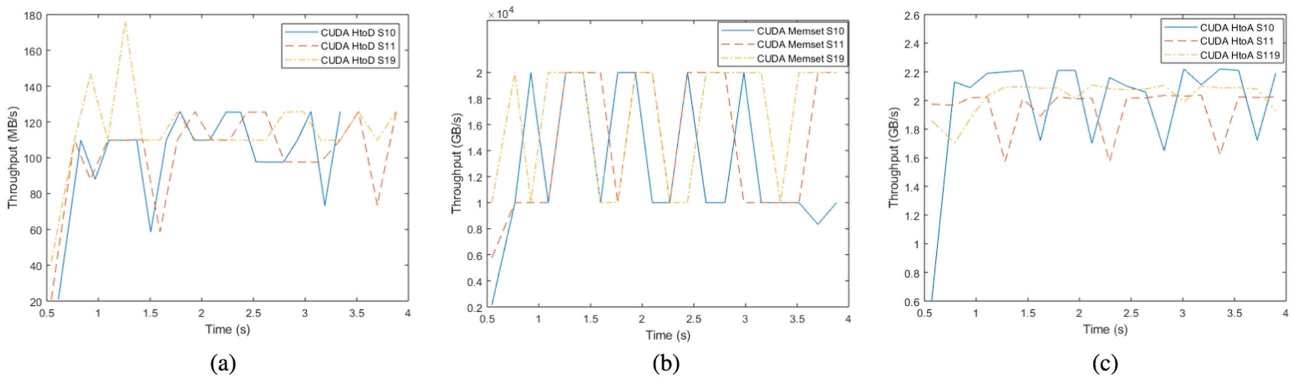

Fig. 12. Memory throughput for each subject using Jetson Tx1. (a) CUDA HtoD. (b) CUDA Memset. (c) CUDA HtoA.

throughput of CUDA Host-to-Array, the peak throughput is 2.22 GB/s, and average throughput is 1.99 GB/s.

## V. Conclusion

In this article, we presented the design and implementation of the HemeLB simulation software on low-cost MPSoC embedded platforms. The proposed solutions allow the HemeLB simulation software to be implemented in a local environment rather than a distributed environment, which is not fully under the clinicians control. A set of comprehensive tests using real patients data have been carried out on the implementations, and the corresponding evaluation of the performance of the implementation has been also reported. In addition, real-time visualization of the simulation results are also reported in this article. According to the evaluation results, Jetson-TX1-based SoC is more suitable for implementing HemeLB system than Zynq-based SoC, one of the reason is because the CUDA cores and multicore processor in Jetson TX1 offers better hardware infrastructure for accelerating real-time visualization applications.

In future, we will develop a user friendly graphic user interface to allow hospital users to manipulate the operations of the framework and steer the visualization, and to use the current system as a test platform in hospital environments.

## Acknowledgment

The statements made herein are solely the responsibility of the authors.

## References

[1] S. Okami and N. Kohtake, "Transitional complexity of health information system of systems: Managing by the engineering systems multiple-domain modeling approach," *IEEE Syst. J.*, vol. 13, no. 1, pp. 952–963, Mar. 2019.

[2] A. A. Abdellatif, A. Mohamed, and C. Chiasserini, "User-centric networks selection with adaptive data compression for smart health," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3618–3628, Dec. 2018.

[3] P. Scarborough, P. Bhatnagar, K. K. Wickramasinghe, S. Allender, C. Foster, and M. Rayner, "The economic burden of ill health due to diet, physical inactivity, smoking, alcohol and obesity in the UK: An update to 2006-07 NHS costs," *J. Public Health*, vol. 33, no. 4, pp. 527–535, 2011.

[4] N. Benkirane *et al.*, "Stroke risk factors in a Moroccan population: A multicentric prospective study," *J. Neurological Sci.*, vol. 357, 2015, Art. no. e367. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022510X15018109

[5] D. Lin, Y. Tang, and A. V. Vasilakos, "User-priority-based power control in d2d networks for mobile health," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3142–3150, Dec. 2018.

[6] Steps to Health, "Brain aneurysms: what are they and how to prevent them," 2018. [Online]. Available: https://steptohealth.com/brain-aneurysms-prevent/

[7] B. M. Jobst *et al.*, "Increased stability and breakdown of brain effective connectivity during slow-wave sleep: Mechanistic insights from whole-brain computational modelling," *Scientific Rep.*, vol. 7, no. 1, 2017, Art. no. 4634.

[8] A. Patronis, R. A. Richardson, S. Schmieschek, B. J. Wylie, R. W. Nash, and P. V. Coveney, "Modelling patient-specific magnetic drug targeting within the intracranial vasculature," *Frontiers Physiol.*, vol. 9, 2018, Art. no. 331.

[9] D. Kandhai, D. J. Vidal, A. G. Hoekstra, H. Hoefsloot, P. Iedema, and P. M. Sloot, "Lattice-Boltzmann and finite element simulations of fluid flow in a SMRX static mixer reactor," *Int. J. Numer. Methods Fluids*, vol. 31, pp. 1019–1033, 1999.

[10] L. Axner, A. G. Hoekstra, A. Jeays, P. Lawford, R. Hose, and P. M. Sloot, "Simulations of time harmonic blood flow in the Mesenteric artery: Comparing finite element and Lattice Boltzmann methods," *Biomed. Eng. Online*, vol. 8, pp. 23-1–23-8, 2009.

[11] M. D. Mazzeo and P. V. Coveney, "HemeLB: A high performance parallel Lattice-Boltzmann code for large scale fluid flow in complex geometries," *Comput. Phys. Commun.*, vol. 178, no. 12, pp. 894–914, 2008.

[12] D. Groen, J. Hetherington, H. B. Carver, R. W. Nash, M. O. Bernabeu, and P. V. Coveney, "Analysing and modelling the performance of the HemeLB Lattice-Boltzmann simulation environment," *J. Comput. Sci.*, vol. 4, no. 5, pp. 412–422, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jocs.2013.03.002

[13] D. Groen *et al.*, "Validation of patient-specific cerebral blood flow simulation using transcranial Doppler measurements," *Frontiers Physiol.*, vol. 9, pp. 721-1–721-13, 2018.

[14] H. Shi, J. Chen, W. Pan, K. Hwang, and Y. Cho, "Collision avoidance for redundant robots in position-based visual servoing," *IEEE Syst. J.*, vol. 13, no. 3, pp. 3479–3489, Sep. 2019.

[15] UCL, "GitHub - UCL/hemelb," 2019. [Online]. Available: https://github.com/UCL/hemelb

[16] X. Zhai *et al.*, "Real-time automated image segmentation technique for cerebral aneurysm on reconfigurable system-on-chip," *J. Comput. Sci.*, vol. 27, pp. 35–45, Jul. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877750317313005

[17] H. Djelouat, X. Zhai, M. A. Disi, A. Amira, and F. Bensaali, "System-on-chip solution for patients biometric: A compressive sensing-based approach," *IEEE Sensors J.*, vol. 18, no. 23, pp. 9629–9639, Dec. 1, 2018.

[18] X. Zhai *et al.*, "Zynq SoC based acceleration of the Lattice Boltzmann method," *Concurrency Comput., Pract. Experience*, vol. 31, no. 17, 2019, Art. no. e5184. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5184

[19] A. J. Ladd, “Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation,” *J. Fluid Mechanics*, vol. 271, pp. 285–309, 1994.

[20] M. Bouzidi, M. Firdaouss, and P. Lallemand, “Momentum transfer of a Boltzmann-lattice fluid with boundaries,” *Phys. Fluids*, vol. 13, no. 11, pp. 3452–3459, 2001.

[21] R. W. Nash *et al.*, “Choice of boundary condition for Lattice-Boltzmann simulation of moderate-Reynolds-number flow in complex domains,” *Physical Rev. E*, vol. 89, no. 2, 2014, Art. no. 023303.

[22] J.-J. Han, M. Lin, D. Zhu, and L. T. Yang, “Contention-aware energy management scheme for NoC-based multicore real-time systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 691–701, Mar. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6774493/

[23] T. Chantem, R. P. Dick, and X. S. Hu, “Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs,” in *Proc. Des., Autom. Test Eur.*, 2008, pp. 288–293. [Online]. Available: http://ieeexplore.ieee.org/document/4484694/

[24] X. Zhang, K. Huang, M. Yu, X. Jiang, and X. Yan, “BFCO: A BPSO-based fine-grained communication optimization method for MPSoC,” *IEEE Access*, vol. 6, pp. 18 771–18 785, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8307752/

[25] X. Zhai, A. A. S. Ali, A. Amira, and F. Bensaali, “MLP neural network based gas classification system on Zynq SoC,” *IEEE Access*, vol. 4, pp. 8138–8146, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7605493/

[26] L. H. L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. E. E. Stewart, *The Zynq Book: Embedded Processing With the ARM Cortex-A9 on the Xilinx Zynq-7000 all Programmable SoC*. Glasgow, U.K.: Strathclyde Academic Media. 2014. [Online]. Available: https://dl.acm.org/citation.cfm?id=2685817

[27] Digilent, *Arty Z7 Reference Manual [Reference.Digilentinc]*. 2018. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/arty-z7/reference-manual

[28] Pynq, *PYNQ - Python Productivity for Zynq - Home*. 2019. [Online]. Available: http://www.pynq.io/

[29] NVIDIA, “NVIDIA Tegra X1 NVIDIA’S New Mobile Superchip,” NVIDIA, Santa Clara, CA, USA, Tech. Rep., 2015. [Online]. Available: https://www.highperformancegraphics.org/wp-content/uploads/2015/Hot3D2/NVIDIA_X1_hpg2015_hot3d.pdf

[30] UCL, “Hemelb Github repository,” 2018. [Online]. Available: https://github.com/UCL/hemelb

[31] K. Zhou, Z. Ren, S. Lin, H. Bao, B. Guo, and H.-Y. Shum, “Real-time smoke rendering using compensated ray marching,” in *Proc. ACM Trans. Graph.*, vol. 27, no. 3, 2008, Art. no. 36.

[32] 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,” *Physical Rev.*, vol. 94, no. 3, pp. 511–525, May 1954. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.94.511