Journal of Parallel and Distributed Computing 152 (2021) 11–20

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Collaborative execution of fluid flow simulation using non-uniform decomposition on heterogeneous architectures

Gabriel Freytag [a,*], Matheus S. Serpa [a], João V.F. Lima [b], Paolo Rech [a], Philippe O.A. Navaux [a]

[a] Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
[b] Federal University of Santa Maria (UFSM), Santa Maria, Brazil

ARTICLE INFO

Article history:
Received 3 July 2020
Accepted 2 February 2021
Available online 15 February 2021

Keywords:
System-on-Chip
FPGA
GPU
Non-uniform partitioning
Lattice Boltzmann Method

ABSTRACT

The demand for computing power, along with the diversity of computational problems, culminated in a variety of heterogeneous architectures. Among them, hybrid architectures combine different specialized hardware into a single chip, comprising a System-on-Chip (SoC). Since these architectures usually have limited resources, efficiently splitting data and tasks between the different hardware is primal to improve performance. In this context, we explore the non-uniform decomposition of the data domain to improve fluid flow simulation performance on heterogeneous architectures. We evaluate two hybrid architectures: one comprised of a general-purpose x86 CPU and a graphics processing unit (GPU) integrated into a single chip (AMD Kaveri SoC), and another comprised by a general-purpose ARM CPU and a Field Programmable Gate Array (FPGA) integrated into the same chip (Intel Arria 10 SoC). We investigate the effects on performance and energy efficiency of data decomposition on each platform's devices on a collaborative execution. Our case study is the well-known Lattice Boltzmann Method (LBM), where we apply the technique and analyze the performance and energy efficiency of five kernels on both devices on each platform. Our experimental results show that non-uniform partitioning improves the performance of LBM kernels by up to 11.40% and 15.15% on AMD Kaveri and Intel Arria 10, respectively. While AMD's Kaveri platform's performance efficiency is of up to 10.809 MLUPS with an energy efficiency of 142.881 MLUPKJ, Intel's Arria 10 platform's is of up to 1.12 MLUPS and 82.272 MLUPKJ.

1. Introduction

The computational power currently available in high-performance computing (HPC) systems makes it possible to perform extremely complex tasks within hours, minutes, seconds, or even in real-time. Such computing power comes from the heterogeneity of interconnected devices built over a variety of architectures that deliver distinct levels of parallelism and computational power such as CPU, GPU, and FPGA devices [16,32]. To meet the growing demand for computing power, not only does the number of devices in these systems increase, but the devices themselves evolve and become even more powerful, consuming an increasing amount of energy.

However, some specialized architectures consume significantly less energy compared to conventional CPU and GPU ones. Devices built on ARM or FPGA architectures typically consume orders of magnitude less energy than devices with CPU or GPU architecture, respectively [24]. Nevertheless, reduced energy consumption implies in lower computational power.

Although heterogeneity of devices in HPC systems is no longer a novelty, some heterogeneous devices have emerged integrating multiple architectures into a single chip such as System-on-Chip (SoC). Through replacing external connections between different devices (usually PCIe connections) by significantly shorter and faster internal connections, it becomes feasible to share the same resources like memory space and, consequently, eliminate highly time-costly data transfers between different memory spaces. Moreover, by placing two architectures into a single chip, both architectures' computational power are limited, reducing the architecture's overall energy consumption [20,23].

Some examples of currently available heterogeneous architectures are: AMD Kaveri SoC [3] that integrates x86-64 CPU and Radeon R7 GPU processing units in the same chip; Intel Arria 10 SoC [1] that integrates ARM CPUs and an FPGA; and Xilinx Zynq SoC [7] that integrates ARM CPUs and an FPGA on the same chip. Although they are architectures with lower computational power, it is necessary to distribute the workload and to compute collaboratively in order to exploit the full potential of heterogeneous devices. However, one of the main challenges is the optimal workload distribution since they typically have different computing capabilities.

* Corresponding author.
E-mail address: gfreytag@inf.ufrgs.br (G. Freytag).

https://doi.org/10.1016/j.jpdc.2021.02.006
0743-7315/© 2021 Elsevier Inc. All rights reserved.

This article investigates performance and energy consumption impact of collaborative execution on two SoC devices based on CPU+GPU and CPU+FPGA architectures using non-uniform data partitioning. Our case of study is the five distinct kernels from the Lattice Boltzmann Method (LBM) to evaluate individually and collaborative execution performance and energy consumption of heterogeneous architectures. The main contributions of this paper are:

- We evaluated performance and energy consumption of two heterogeneous architectures: AMD Kaveri SoC and Intel Arria 10 SoC;
- We analyzed the performance and energy consumption of each device present in both architectures individually and in a collaborative way;
- We showed that non-uniform partitioning on collaborative execution improves performance with low impact in energy consumption of LBM kernels;
- We present an OpenMP + OpenCL D3Q19 Lattice Boltzmann implementation for heterogeneous architectures.

This article is an extension of our previous work [11], published in the SBAC-PAD 2019 conference. It comprises an energy efficiency analysis of the non-uniform data domain decomposition approach of both two CPU + GPU and CPU + FPGA System-on-Chip heterogeneous architectures. We increased the related work and discussed the trade-off between performance and energy efficiency.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the methodology and Section 4 describes the parallel implementation of Lattice-Boltzmann Method (LBM) for two low-power heterogeneous platforms. Our experimental results are presented in Section 5. Section 6 presents the discussion, and finally Section 7 presents conclusions and future work.

2. Related work

Many of the related works used GPUs or FPGAs as an offload target to accelerate applications. The authors in [15] evaluated performance and power consumption of kernel computations on an Arria 10 FPGA over an Intel Xeon Phi KNL and an NVIDIA Tesla K80. In [9] the authors evaluate the reliability behavior of AMD Kaveri aiming to find which configuration provides the lowest error rate or allows the computation of the highest amount of data before experiencing a failure. In [2,25,34] the authors accelerated deep learning networks using OpenCL and FPGAs. In [17] they designed computational intensive kernels of a tsunami simulator on FPGAs and GPUs using OpenCL. The authors in [19] evaluated the Rodinia benchmark over GPUs and FPGAs with OpenCL. In [36,37] they combined spatial and temporal blocking to evaluate performance and power efficiency of stencil computations on FPGAs. In [28] they optimized geophysics models on GPUs. Regarding LBM kernels, the authors in [18] optimized for GPUs and in [26] they studied optimization strategies for accelerators such as GPUs and Intel Xeon Phi KNL. In [12] a memory-aware 2D LBM was implemented for Intel Xeon manycore processors.

Collaborative processing on heterogeneous devices has been studied by several authors. Hetero-Mark [31] is a benchmark suite for collaborative processing for CPU–GPU architectures with support to OpenCL. They analyzed experimental results on an AMD A10-7850K APU. Chai [13] is also a collaborative processing benchmark suite for integrated devices. It compared task and data partitioning strategies and has support to FPGAs and GPUs. In [14], they evaluated two FPGA systems with Chai benchmark and analyzed the task and data partitioning. The authors in [21] designed binarised neural networks to take advantage of FPGA systems, and compared performance and energy consumption over an NVIDIA Titan X GPU. In [22] they optimized a Binarized Neural Network and evaluated its performance in an Intel Xeon E5-2699v3 CPU, an NVIDIA GTX Titan X GPU, an Intel Stratix V and an Intel Arria 10 FPGA and a custom ASIC platform without using OpenCL architecture. The performance analysis of collaborative computing in two heterogeneous integrated systems using OpenCL was presented by [5]. They evaluated an AMD A10-7850K platform with CPU cores and GPU computing units integrated into the same chip and an E3-1240 v3 CPU chip connected through PCIe to an Intel Stratix V GX FPGA on a Terasic DE5-Net board. The results showed that in both platforms, the use of the two available devices in a collaborative way led to better performance compared to use CPU only or GPU/FPGA only.

Moreover, some works investigate collaborative execution has a way to increase the energy efficiency of low energy architectures. In [35] the authors investigated energy conservation's problem for executing mobile applications by task offloading to the cloud. By collaboratively executing tasks on both mobile's hardware and in the cloud, it was possible to reduce the energy consumption on a mobile device significantly. In [33], they present an energy-optimal scheduling policy for collaborative execution in mobile cloud computing network. The experiment results show that the method can improve energy savings and extend mobile clients' battery lifetime. In [29], the authors proposed a run-time management approach that performs energy-efficient mapping and thread-partitioning between CPU and GPU devices on heterogeneous mobile SoCs. The experimental results show energy savings by the proposed adaptive approach over existing approaches.

Most of the works previously presented evaluate the performance of the different processing units individually. However, the use of these devices in collaborative processing can lead to significant performance improvements, as previously demonstrated by Chang [5]. Moreover, collaborative execution can also improve the energy efficiency of algorithms on low power architectures. In our work, we investigate the advantages of collaborative execution concerning performance efficiency employing non-uniform data domain decomposition. We also investigate the impacts of collaborative execution on energy-efficiency, despite not attempting to optimize energy consumption.

3. Methodology

This Section details the heterogeneous platforms, the methodology used in our experiments, and the Lattice Boltzmann Method.

3.1. Platforms and experimental design

We used the following two platforms. The first platform, called A10-7870K, is an AMD architecture that consists of 4 x86-64 CPU cores and 8 Radeon R7 GPU computing units integrated into the same chip. It has 6 GB of RAM, a Thermal Design Power (TDP) of 95 Watts, Ubuntu 14.04 (trusty) operating system, OpenCL Software Development Kit (SDK) version AMD APP SDK-3.0 and Clang C/C++ compiler version 7.0. The second platform, called A10SoCFPGA, is an Intel architecture that consists of 2 ARM Cortex-A9 CPU cores and an FPGA with 660,000 logic blocks integrated into a single chip. It has 1GB of RAM, a TDP of 30 Watts for the overall development kit, Angstrom 2014.10 operating system, Intel FPGA Runtime Environment (RTE) for OpenCL version 18.1, and ARM GCC C/C++ compiler version 4.7. From now, we will call the platforms AMD Kaveri and Intel Arria 10. Table 1 summarizes the environments.

Table 1
Configuration of evaluated platforms.

Platform	Parameter	Value
AMD Kaveri		
	CPU	AMD A10-7870K, 4 cores
	GPU	AMD Radeon R7, 8 cores
	Memory	6GB DDR3-2133
	TDP	95 Watts
Intel Arria 10		
	CPU	ARM Cortex-A9, 2 cores
	FPGA	660 000 logic blocks
	Memory	1GB
	TDP	30 Watts

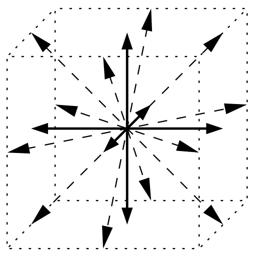


Fig. 1. D3Q19 lattice geometry.

Both architectures support the OpenMP [4] and OpenCL languages [30]. However, only AMD Kaveri supports OpenCL on its two devices. In Arria 10, only the FPGA supports OpenCL code, which is automatically converted into a binary using a High-Level Synthesis (HLS) compiler so that the FPGA can be programmed by the host (CPU) through the OpenCL language. Thus, in both Arria 10 and AMD Kaveri, we evaluate the method's performance in an OpenMP + OpenCL implementation (OpenMP in CPU and OpenCL in GPU/FPGA). The optimization flag used in both was *-O3*.

We performed experiments in both AMD Kaveri and Intel Arria 10 platforms, decomposing the method's data domain into two subdomains, one for each device. The proportion of data assigned to each device varied from 0 to 96. 0 means that the kernel is executed individually in one device (CPU, GPU, or FPGA). For the collaborative execution, we decomposed the domain in the z-axis non-uniformly using an inverse proportion for each subdomain. While CPU's subdomain proportion starts with a size of 16 and goes up to 80 increasing by 16, GPU's proportion decreases by 16 from 80 up to 16. For example, a subdomain of size $96 \times 96 \times 64$ for the CPU and of size $96 \times 96 \times 32$ for the FPGA corresponds to 66.7 and 33.3% of the original domain of size $96 \times 96 \times 96$, respectively.

In order to measure the method's performance in means of Million Lattice Updates Per Second (MLUPS), and energy efficiency in Million Lattice Updates Per KiloJoule (MLUPKJ), the following equation was used:

$$\frac{S_x \times S_y \times S_z \times N_{ts}}{10^6 \times U} \tag{1}$$

where S_x, S_y, and S_z stand for the domain size in x, y and z dimensions, N_{ts} is the number of time steps (which was equal to 200 in all our experiments), and U is the running time in seconds or the total energy consumption of the experiment for the MLUPS or MLUPKJ metric, respectively.

We measured the algorithm's running time and energy consumption on both platforms, using different workload proportions for each of its devices. For the performance, we measured the running time of all five kernels on each device individually, as well as the overall running time of the execution. Moreover, each experiment was executed at least ten times, with a confidence interval of 95% calculated with the Student's t-distribution.

To record the algorithm's energy consumption, we used two different tools on each platform: on AMD's Kaveri platform, we used CodeXL, and on Intel's Arria 10, we used LTPowerPlay and development kit's DC1613 A Linear Dongle. We measured the energy consumption using the tool's default frequency, which on AMD's platform was 100 ms, and on Intel's platform, it was approximately 500 microseconds. Therefore, the average power consumption values presented in this work for both devices on each platform is equal to the device's total power consumed during the entire experiment divided by the respective tool's measurement frequency. Moreover, each device's total energy is equal to the device's total running time multiplied by its average power consumption. In this way, we get an average energy consumption of each device on both platforms disregarding idle times, once the devices' running times on different workload proportions can be disparate.

3.2. *Lattice-Boltzmann method*

The Lattice Boltzmann Method (LBM) is a numerical method for fluid flow simulations, and fluid physics modeling originated from discrete particle kinetics called Lattice Gas Automaton (LGA). The Lattice Gas Automaton is constructed as a simplified, fictitious molecular dynamic in which space, time, and particle velocities are all discrete [6]. Thus, in LBM space, time and velocity are also discrete.

Lattice-Boltzmann Method is often adopted as an alternative technique for computational simulations of Fluid Dynamics instead of conventional numerical schemes based on discretizations of macroscopic continuum equations as discrete Navier–Stokes equation solvers [6]. In LBM, a lattice is formed by discrete points, each with a fixed number of discrete displacement directions on which particles perform spatial displacements at each iteration, enabling simple simulations of physical properties of fluid flows [12].

In this paper, we used a three-dimensional lattice structure with nineteen propagation directions, as shown in Fig. 1 and defined below [27]:

- A static point at coordinate $(0, 0, 0)$, where the particle has zero velocity. The value of ω_i in this case is $1/3$.
- Six nearest directions $(-1, 0, 0)$, $(+1, 0, 0)$, $(0, -1, 0)$, $(0, +1, 0)$, $(0, 0, -1)$ and $(0, 0, +1)$, with unity velocity and $\omega_i = 1/18$.
- Twelve diagonal line neighbors $(1, 1, 0)$, $(-1, 1, 0)$, $(1, -1, 0)$, $(-1, -1, 0)$, $(1, 0, 1)$, $(-1, 0, 1)$, $(1, 0, -1)$, $(-1, 0, -1)$, $(0, 1, 1)$, $(0, -1, 1)$, $(0, 1, -1)$ and $(0, -1, -1)$, with velocity $\sqrt{2}$ and $\omega_i = 1/36$.

To deal with the collisions against the boundaries of the structure we use a mechanism called Bounce-back which consists in the inversion of the speed vectors directions each time that a collision occurs against a static point preventing the forces leaving, returning them to the fluid [8]. For the experiments, we use a rectangular obstacle placed in the canal at the first third of the x-axis, as illustrated in Fig. 2.

4. Collaborative LBM implementation

In this Section, we describe the collaborative implementation of the Lattice-Boltzmann method developed for two low-power heterogeneous platforms: One is four CPU cores and eight GPU computing units integrated into a single chip, named *AMD Kaveri*; and two CPU cores and an FPGA with 660,000 logic blocks integrated in the same chip, named *Intel Arria 10*. In order to

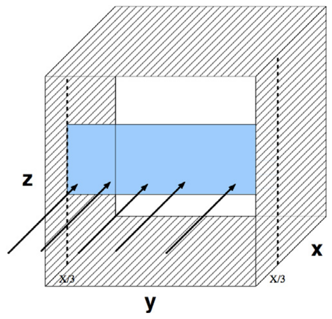


Fig. 2. Rectangular obstacle of the experiments.

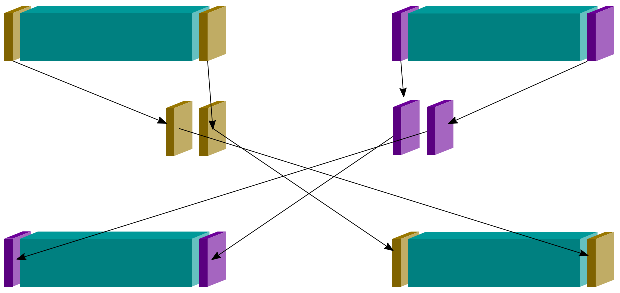


Fig. 3. LBM implementation with ghost zones.

use both sets of processing units available in each platform in a collaborative way, we decompose the method's data domain into two subdomains. Besides that, to be able to evaluate the performance of each kernel of the method on each platform and find the domain decomposition that optimizes the performance of the method, we implement a variable domain decomposition to enable the assignment of different subdomain sizes for each set of processing units on each platform.

As both platforms have support for the OpenCL architecture, we can use the OpenCL language to implement our Lattice-Boltzmann method and use their heterogeneous processing units. However, while the AMD Kaveri platform is fully OpenCL capable, meaning that both CPU and GPU are OpenCL devices and, in this way, can run OpenCL code, the Intel Arria 10 is only partially OpenCL capable and only the FPGA can run OpenCL code. In this way, we implement the parallel heterogeneous version of the Lattice-Boltzmann method using the OpenMP Application Programming Interface (API) to run the method's kernels in the CPU cores of both platforms and the OpenCL language to run the method's kernels in the GPU and FPGA processing units of the AMD Kaveri and Intel Arria 10 platforms, respectively.

4.1. Domain decomposition

To be able to use multiple devices simultaneously to compute the fluid dynamics of the Lattice-Boltzmann method in parallel, we divide its domain into subdomains by dividing the original three-dimensional domain in the *z* axis. In this way, each device can apply the kernels of the method in parallel on each subdomain. However, dividing the domain into multiple subdomains leads to inconsistencies in macroscopic values of the fluid due to dependencies on the nineteen propagation directions of the neighbor's particles of each particle of the fluid according to Fig. 1. Therefore, we use ghost zones to deal with the inconsistencies arising from the division of the domain solving the problem completely.

Since each particle of the fluid has nineteen directions of propagation of the forces in the model D3Q19, as shown in Fig. 1, as the fluid flows the forces of eighteen propagation directions of each particle are propagated through the neighboring particles in the fluid. With the division of the domain into subdomains, the existence of this data dependence makes it necessary, for particles located at the edges of each subdomain, to access data located in neighboring subdomains. However, the access to data located in other subdomains being manipulated by other devices can lead to concurrency in data access and consequently in inconsistencies. Therefore, to deal with these issues, we use ghost zones to keep a copy of the edges of the neighboring subdomains of each subdomain.

Beyond the division of the domain into subdomains to parallelize the routines of the method, the routines themselves were also divided into kernels. Each kernel has data dependencies which must be obeyed to ensure the consistency of the results of the method. In this manner, there is an order in which the kernels need to be executed by each device to simulate the flow of the fluid in its subdomain correctly. Besides that, when multiple devices are used collaboratively, the ghost zones of the neighbor subdomains of each device need to be updated. This ghost zone updates introduce a synchronization point in parallel execution of the method when using multiple devices.

Therefore, to deal with these issues, we use ghost zones to keep a copy of the edges of the neighboring subdomains of each subdomain. Fig. 3 shows an illustration of the procedures for updating ghost zones of two subdomains in two devices. In this work, as in [10], we decompose the domain of the Lattice-Boltzmann method only in one dimension, in this case in dimension *z*. Besides that, as we use at most two devices to compute the method and use a circular strategy to simulate the flow of fluids in the method, meaning that the particles leaving one side of the domain return on the opposite side in order to preserve the macroscopic values of the fluid, there are only four ghost zones that correspond to the four faces at the *z* dimension of the subdomains. These four faces are first copied to a temporary buffer by each device, and then each device copies the adjacent faces from the temporary buffers to its local copy of the *z* dimension faces of it neighboring subdomain. As can be seen in Fig. 3, the faces where the domain is divided are crossing copied, and the faces at the edges of the original domain are copied to the opposite side of the subdomains.

4.2. Lattice-Boltzmann kernels

The Lattice-Boltzmann method routines are composed of five well-defined kernels. The first kernel, called INITIALIZE, assigns an initial macroscopic value for each of the nineteen propagation directions of each particle of the fluid in the three-dimensional model used in our work. After initializing, the second kernel to be executed redistributes the forces of some of the propagation directions of each particle of the fluid and, therefore, is called *redistribute*. The third kernel, called PROPAGATE, is in charge of propagating the forces of the particles according to the flow of the fluid. After the propagation, the fourth kernel deals with the collision of the particles in the fluid with the barriers present in the domain, as described in Section 3.2, and is called BOUNCEBACK. The fifth and last kernel relaxes the forces of each of the propagation directions of each particle in the fluid, and it is called RELAXATION.

These five kernels must always be executed in the same sequence as described above. After the initialization of the macroscopic values in kernel INITIALIZE, the remaining four kernels need to be executed sequentially *t_max* times, as shown in Fig. 4. When two devices are used, after the execution of the *redistribute* kernel, each device copies its edge faces to the temporary buffers

```
1:  Initialize parameters
2:  for Each $N_x \times N_y \times N_z$ do
3:      INITIALIZE conditions
4:  end for
5:  for Each time step do
6:      for Each $N_x \times N_y \times N_z$ do
7:          REDISTRIBUTE
8:          PROPAGATE
9:          BOUNCEBACK
10:         RELAXATION
11:     end for
12: end for
```

Fig. 4. D3Q19 Lattice Boltzmann Method Algorithm.

```
__kernel void redistribute(...)
  x = get_global_id(0);
  y = get_global_id(1);
  z = get_global_id(2);
  /* redistribute computation */
```

Fig. 5. OpenCL kernel code snippet.

and wait for both devices to finish. Then, each device copies the neighboring subdomain edge faces from the temporary buffers to its local copies and execute the remaining three kernels, and this cycle repeats *t_max - 1* times.

4.3. OpenMP + OpenCL implementation

In OpenCL language, each of the Lattice-Boltzmann kernels become OpenCL kernels as in Fig. 5. OpenCL kernels are represented by the keyword *__kernel*. Function *get_global_id*(*dim*) returns the unique global work-item ID value for dimension identified by *dim*. Initially, to be able to execute kernels in the devices of the OpenCL platform, it is needed to create an OpenCL context using the IDs of the devices. After, these kernels can be queued in OpenCL queues with First In First Out (FIFO) type data structure from which they are then delivered for the devices associated with the queue by the OpenCL runtime. Each queue can be associated with one or more OpenCL devices, and in our OpenCL implementation of the Lattice-Boltzmann method, we use an OpenCL queue for GPU/FPGA device.

In the CPU, however, as it is not an OpenCL device, we use the OpenMP API to parallelize the kernels of the method. Fig. 6 shows a code snippet for the OpenMP implementation. As the kernels go through all particles in the three-dimensional domain, there are three nested for loops on which we use OpenMP for loop pragmas to parallelize the kernels in CPU. To achieve better performance, we collapse the three nested for loops using

```
void relaxation(...)
   #pragma omp parallel for collapse(3)
   {
   for(x = 0; x < lx; x++)
     for(y = 0; y < ly; y++)
       for(z = 0; z < lz; z++)
        /* relaxation computation */
   }
```

Fig. 6. OpenMP kernel code snippet.

the OpenMP pragma *#pragma omp parallel for collapse(3)* on all kernels.

In this implementation, a kernel is queued in the FPGA device queue, and then the corresponding kernel is executed in the CPU. Only after executing this kernel in the CPU, we can queue the next kernel in FPGA device queue and then execute the corresponding kernel in the CPU. Thus, after queuing the *redistribute* kernel and executing the corresponding kernel in the CPU, queues a copy of the edge faces of the FPGA subdomain and copies its subdomain edge faces to the temporary buffers. After copied, the CPU waits for the copy of the FPGA subdomain edge faces and then copies it from the temporary buffer to its local copy, and the remaining kernels and kernels are executed. This cycle repeats *t_max - 1* times.

5. Experimental results

In this section, we present the performance and energy efficiency results of our collaborative execution strategy over the AMD Kaveri and Intel Arria 10 SoC platforms. Our case of study was the computing kernels of the Lattice-Boltzmann method on a three-dimensional model with nineteen propagation directions of forces, namely D3Q19. We performed experiments using CPU only, GPU/FPGA only, and CPU + GPU/FPGA through a three-dimensional domain of size 96 × 96 × 96. The memory size of 1 GB of Intel's Arria 10 platform limited the size of the domain.

5.1. Performance

Table 2 presents the average running time of the five kernels of the method in the OpenMP + OpenCL version on the AMD Kaveri platform. The first column is the kernels name. The following columns are different decomposition sizes of the method domain, from CPU only to GPU only. The bold cells are the ones that performed better, and consequently, had the best partitioning for each kernel. INITIALIZE and BOUNCEBACK kernels performed better in GPU only execution. Collaborative execution did not improve the performance of these kernels because their running time was too short. The REDISTRIBUTE and PROPAGATE kernels performance improve by 12.09% and 12.15%, respectively. The RELAXATION kernel had a low-performance improvement of 2.39%. While the total running time of CPU only was 62.19 s, and using GPU only the total running time was 18.47 s. It means a performance efficiency of just 2.845 MLUPS for CPU only execution against 9.579 MLUPS for GPU only, as can be seen in Table 4. However, collaboratively using both CPU and GPU and dividing the domain into five distinct proportions (GPU only, 16 × 80, 48 × 48, GPU only and 16 × 80), the shortest running time achieved was 16.37 s using a non-uniform partitioning, giving an

Table 2
Average running time of all kernels on the CPU and GPU devices of the AMD Kaveri platform on each workload proportion using the OpenMP + OpenCL version with a domain of size 96 × 96 × 96.

		Initialize	Redistribute	Propagate	Bounceback	Relaxation
	CPU Only	0.059	17.588	18.212	1.544	24.785
80 × 16	**CPU**	0.049	8.469	9.749	0.925	13.659
	GPU	0.002	0.796	3.157	0.024	0.772
64 × 32	**CPU**	0.040	6.342	8.811	0.732	11.104
	GPU	0.004	1.616	6.307	0.048	1.518
48 × 48	**CPU**	0.030	4.827	**7.294**	0.593	8.603
	GPU	0.006	3.153	**8.65**	0.076	2.279
32 × 64	**CPU**	0.022	3.766	4.99	0.445	6.895
	GPU	0.010	4.012	9.908	0.102	2.884
16 × 80	**CPU**	0.013	**1.987**	1.934	0.227	**3.549**
	GPU	0.012	**3.989**	10.313	0.122	**3.105**
	GPU Only	**0.006**	4.811	9.846	**0.174**	3.636

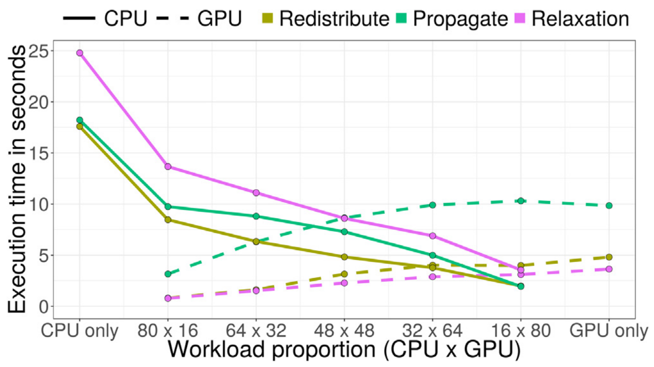


Fig. 7. Average running time of the three most representative kernels on the CPU and GPU devices of the AMD Kaveri platform on each workload proportion using the OpenMP + OpenCL version with a domain of size 96 × 96 × 96.

efficiency of 10.809 MLUPS. Thus, the non-uniform partitioning technique improves the entire method performance by 11.39%.

Moreover, looking more closely at Fig. 7, where we show each kernel's running times on the seven different workload proportions, it is possible to observe that each kernel achieves its lowest running time with different domain decomposition sizes. In this Figure, each solid line represents the CPU running time for a specific kernel. The dashed ones represent GPU running time. Each color is one of the three most representative kernels. The best running time for a specific kernel is at the intersection of two lines for the same kernel. For instance, we have the Relaxation kernel executed in 3.55 s in the CPU and 3.10 s in the GPU with a domain decomposition of one subdomain of size 96 × 96 × 16 for the CPU and another of size 96 × 96 × 80 for the GPU. Comparing the running time of the Propagate kernel using the same domain decomposition size that Relaxation kernel performed better to the running time of the Propagate kernel using the domain decomposition of 96 × 96 × 48 for the CPU and 96 × 96 × 48 for the GPU, which provided the kernel shortest running time, the running time was 16.12% worst using Relaxation domain decomposition size. The same happens with the other kernels. For the Initialize kernel, it achieved the shortest running time with a GPU only execution. For the Redistribute kernel the shortest running time was achieved with subdomains of size 96 × 96 × 16 and 96 × 96 × 80 for the CPU and GPU, respectively, and for the Propagate kernel with subdomains of size 96 × 96 × 48 and 96 × 96 × 48 for the CPU and GPU, respectively. For the Bounceback kernel, GPU only execution achieved the shortest running time.

Table 3 presents the average running time of the five kernels in the OpenMP + OpenCL version on the Intel Arria 10 platform. As in Table 2, the first column is the kernel's name, and the following columns are different decomposition sizes of the method domain, from CPU only to GPU only. The bold cells are the ones that perform better, and consequently, had the best partitioning for each kernel. Initialize kernel performed better in FPGA only execution. Thus, collaborative execution did not improve its performance because the running time is short. Bounceback was better using a decomposition of 96 × 96 × 16 for the CPU and 96 × 96 × 80 for the FPGA. It represents a performance improvement of 19.16%. The best distribution for Redistribute and Relaxation was 96 × 96 × 16 for the CPU and 96 × 96 × 80 for the FPGA. For these kernels, most of the work is executed in the FPGA. The number of cores of Intel Arria CPU explains it, which is only two. The Propagate kernel performance was improved by 14.29% using a distribution of 96 × 96 × 32 for the CPU and 96 × 96 × 64 for the FPGA. In this case, the CPU handles more data. While CPU only execution running time was 449.99 s, FPGA only running time was 186.24 s. It means a performance efficiency of just 0.393 MLUPS for CPU only and 0.95 MLUPS for FPGA only execution. However, collaboratively using both CPU and FPGA and dividing the domain into five distinct proportions (FPGA only, 16 × 80, 32 × 64, 16 × 80, and 16 × 80), the shortest running time achieved was 158.02 s, giving a performance efficiency of up to 1.12 MLUPS. In the end, the LBM performance improves by 15.15%.

Fig. 8 presents the average running time of the three most representative kernels of the method in the OpenMP + OpenCL version on the platform Intel Arria 10. The kernels are Redistribute, Propagate, and Relaxation. In axis x, we have seven different workload proportions from CPU only to FPGA only. The solid lines represent the CPU running time and the dashed ones, GPU running time. The colors represent each kernel. The best running time is at the intersection of two lines for the same kernel. With a three-dimensional domain of size 96 × 96 × 96, and using only the CPU device (leftmost in the Figure) the running time of the kernels were 83.21, 146.68 and 205.07 s and using only the FPGA device (rightmost in the Figure), the running times were 29.31, 83.02 and 72.08 s.

Nevertheless, collaboratively using both CPU and FPGA, the kernel's performance improves by up to 19.16%. For instance, Redistribute and Relaxation performance was better with a distribution of 96 × 96 × 16 for the CPU and 96 × 96 × 80 for the FPGA while the Propagate performance was better with 96 × 96 × 32 for the CPU and 96 × 96 × 64 for the FPGA. In the same way as for AMD Kaveri, in Intel Arria 10, the shorter running times of each kernel were not necessarily obtained with the same domain decomposition.

Table 3
Average running time of the all kernels on the CPU and FPGA devices of the Intel Arria 10 platform on each workload proportion using the OpenMP + OpenCL version with a domain of size 96 × 96 × 96.

		Initialize	Redistribute	Propagate	Bounceback	Relaxation
	CPU only	0.277	83.207	146.685	14.754	205.069
80 × 16	**CPU**	0.247	73.151	117.726	12.636	175.321
	FPGA	0.025	6.303	19.007	0.379	13.223
64 × 32	**CPU**	0.209	62.525	93.066	10.385	142.31
	FPGA	0.052	13.262	38.473	0.629	27.726
48 × 48	**CPU**	0.177	47.550	73.195	6.06	107.32
	FPGA	0.081	20.329	57.700	0.9	42.587
32 × 64	**CPU**	0.155	25.767	**47.511**	2.028	67.172
	FPGA	0.110	19.680	**71.158**	1.222	48.677
16 × 80	**CPU**	0.140	**14.409**	22.480	**0.954**	**34.67**
	FPGA	0.127	**24.666**	84.810	**1.367**	**60.688**
	FPGA only	**0.139**	29.312	83.020	1.691	72.078

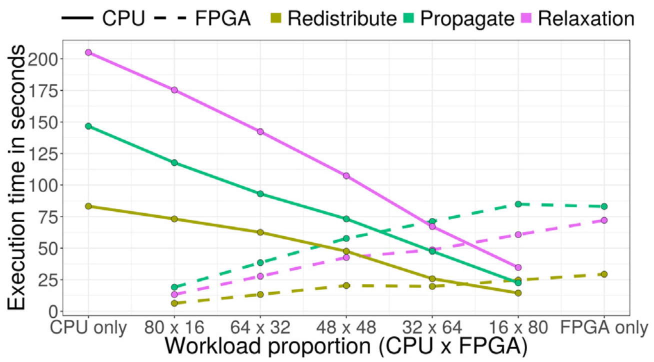


Fig. 8. Average running time of the three most representative kernels on the CPU and FPGA devices of the Intel Arria 10 platform on each workload proportion using the OpenMP + OpenCL version with a domain of size 96 × 96 × 96.

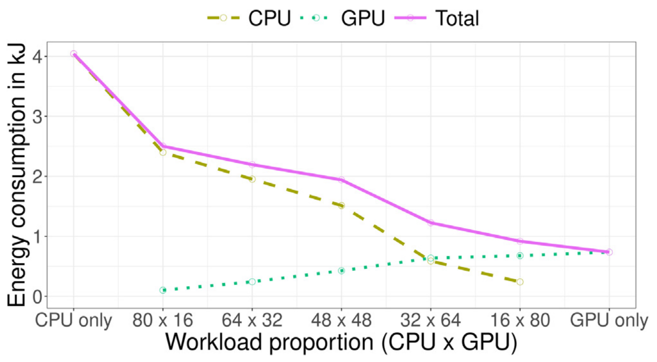


Fig. 9. CPU, GPU, and total energy consumption on AMD's Kaveri platform for each workload proportion using OpenMP + OpenCL implementation and domain size 96 × 96 × 96.

5.2. Energy efficiency

Alongside with the running time and performance of the LBM on the AMD Kaveri platform, in Table 4, we also present the total energy consumption, the average wattage of both CPU and GPU devices, and the energy efficiency for all seven domain decompositions. Total energy consumption corresponds to the sum of CPU and GPU energy consumption relative to the running time of the method on each device. Moreover, exceptionally in the case of CPU and GPU only experiments, once GPU and CPU devices, respectively, were not in use, the average wattages of these devices correspond to the average wattage of idle time and are not included in the reported total energy consumption.

If we look at the CPU and GPU only measures in Table 4, the disparity in energy consumption between these two devices is evident. As can be seen, while performing all the method's computations using only the CPU device (96 × 0) takes 62.188 s with an average wattage of 62.786 Watts (W), GPU only (0 × 96) takes just 18.473 s and has an average wattage of 39.378 W. CPU only consumes a total of 3.905 kilojoules (kJ) resulting in the energy efficiency of 45.318 Million Lattice Updates Per KiloJoule (MLUPKJ). In contrast, GPU only consumes merely 0.727 (kJ) of total energy and, therefore, has an energy efficiency of more than 243 MLUPKJ. It corresponds to a reduction of 3.37 times in running time and 5.37 times in total energy consumption.

By collaboratively executing all kernels using the same decomposition for each one, it is possible to observe that the total energy consumption reduces progressively as the GPU workload proportion increases. For example, while an 80 × 16 decomposition of workload between CPU and GPU consumes about 2.414 kJ of energy, a fifty by fifty decomposition (48 × 48) consumes 1.955 kJ, and a 16 × 80 consumes just 0.93 kJ of energy. It results in an energy efficiency of 73.313, 90.509, and 190.224 MLUPKJ, respectively. Since the GPU energy consumption is significantly lower and has a higher throughput than the CPU, the relative energy consumption increase of its workload proportion increment is smaller than the CPU's energy consumption decrease, as shown in Fig. 9.

As seen in Section 5.1, by collaboratively performing the method's kernels using both CPU and GPU devices and non-uniformly decomposing the domain, both the running time and the performance throughput of the method are significantly enhanced in comparison to GPU only. By dividing the domain into five distinct proportions (GPU only, 16 × 80, 48 × 48, GPU only and 16 × 80), one for each kernel, respectively, the running time in comparison to GPU only is improved from 18.47 to 16.37 s, and the total energy consumption increased from 0.736 to 1.248 kJ. It means a reduction of 1.13 times in running time and an increase in total energy consumption of 1.72 times, thus, reducing the method's energy efficiency from 243.246 to 141.786 MLUPKJ. Although total energy consumption increases compared to GPU only, compared to CPU only, it still means a striking reduction of 3.8 times in running time and a notable reduction of 3.13 times in total energy consumption.

Table 5, in its turn, presents the total energy consumption, the average wattage of both CPU and FPGA devices, and the energy efficiency, besides running time and performance, for all seven domain decompositions on Intel's Arria 10 platform. As well as in AMD's Kavari platform, performing all the method's computations using only the FPGA device consumes the lowest amount of total energy. While CPU only (96 × 0) takes 449.992 s

Table 4
Running time, performance, total energy consumption, average wattage of both CPU and GPU devices, and energy efficiency on AMD's Kaveri platform for all seven workload proportions using the OpenMP + OpenCL implementation with a domain of size 96 × 96 × 96.

Partition	Time [s]	Perf. [MLUPS]	Total energy [kJ]	Avg. CPU/GPU power [W]	Energy efficiency [MLUPKJ]
96 × 0	62.188	2.845	3.905	62.786/8.66	45.318
80 × 16	32.85	5.387	2.414	70.119/23.19	73.313
64 × 32	27.029	6.547	2.168	70.407/27.929	81.614
48 × 48	22.703	7.794	1.955	69.65/33.058	90.509
32 × 64	21.282	8.315	1.178	31.201/39.913	150.202
16 × 80	18.091	9.781	0.93	29.272/40.166	190.224
0 × 96	18.473	9.579	0.727	11.53/39.378	243.246

Table 5
Running time, performance, total energy consumption, average wattage of both CPU and GPU devices, and energy efficiency on Intel's Arria 10 platform for all seven workload proportions using the OpenMP + OpenCL implementation with a domain of size 96 × 96 × 96.

Partition	Time [s]	Perf. [MLUPS]	Total energy [kJ]	Avg. CPU/FPGA power [W]	Energy efficiency [MLUPKJ]
96 × 0	449.992	0.393	5.402	12.005/6.25	32.755
80 × 16	379.082	0.467	4.774	11.956/6.218	37.062
64 × 32	308.496	0.574	4.219	12.048/6.265	41.943
48 × 48	234.302	0.755	3.602	12.106/6.298	49.121
32 × 64	166.28	1.064	2.58	11.813/6.352	68.594
16 × 80	171.67	1.031	1.943	11.635/6.394	91.074
0 × 96	186.24	0.95	1.193	11.174/6.406	148.326

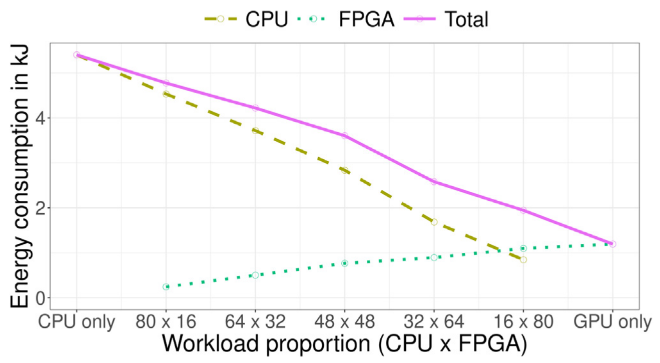


Fig. 10. CPU, FPGA, and total energy consumption on Intel's Arria 10 platform for each workload proportion using OpenMP + OpenCL implementation and domain size 96 × 96 × 96.

with an average wattage of 12.005 W, FPGA only (0 × 96) takes just 186.24 s and has an average wattage of 6.406 W. As a result, CPU only consumes 5.402 kJ and FPGA only consumes 1.193 kJ of total energy, giving an energy efficiency of 32.755 and 148.326 MLUPKJ, respectively. It corresponds to a reduction of 2.42 times in running time and 4.53 times in total energy consumption.

By collaboratively performing the computations, although not reducing the total energy consumption compared to FPGA only, as more workload moves from CPU to FPGA, the total energy consumption reduces gradually in comparison to CPU only. Taking decompositions 80 × 16, 48 × 48, and 16 × 80 as examples, while in the first decomposition a total of 4.774 kJ of energy is consumed, a fifty by fifty decomposition (the second one) consumes 3.602 kJ, and the third and last consumes just 1.943 kJ of energy. If we look at the energy efficiency of these decompositions, it means an increase from just 37.062 to 49.121, and up to 91.074 MLUPKJ, respectively. As well as in AMD's Kaveri platform, since FPGA has a lower energy consumption and a higher throughput compared to CPU, more data in FPGA reduces the total energy consumption, as can be seen in Fig. 10.

Moreover, by dividing the domain into five distinct proportions (GPU only, 16 × 80, 48 × 48, GPU only and 16 × 80), one for each of the five kernels, respectively, the method's running time is significantly improved as could be seen in Section 5.1. While FPGA only takes 186.24 s and consumes a total of 1.193 kJ of energy, the non-uniform domain decomposition takes 158.018 s, consuming a total of 2.151 kJ of energy. As a result, running time reduces 1.18 times, and total energy consumption increases 1.8 times, reducing the energy efficiency of the method goes from 148.326 to 82.272 MLUPKJ. However, compared to CPU only, it means a significant reduction of 2.85 times in running time and 2.51 times in total energy consumption.

6. Discussion

Our experimental results provide evidence that collaborative execution using non-uniform partitioning improves heterogeneous architectures performance. As a case of study, we show that LBM performance was improved by 11.39% and 15.15% in AMD Kaveri and Intel Arria 10, respectively.

Two points were essential in our approach to achieve that performance improvement. First, collaborative execution in heterogeneous architectures is possible due to the tight integration of the CPUs and the GPUs or FPGAs in these devices. It allows both devices working concurrently on the same workload, improving the overall system resources by employing both CPU threads and GPU or FPGA concurrency, thereby achieving higher performance. Second, non-uniform data partitioning is essential, which is a strategy that disjoint devices perform the same task on different subsets of the data.

From our experimental results, we make two major observations. First, as expected, it appears that each device is suitable or specialized for a specific kind of workload. That is, the performance of each computational kernel over a specific device depends on its workload. If the kernel is memory-bound, performance may be better if more workload is assigned to the CPU. On the other hand, if the kernel is CPU-bound assigning more workload to the GPU or FPGA may improve the overall performance. Second, choosing the optimal partitioning is one of the main challenges. Partitioning can be static, which a fixed fraction of workload is assigned to each device before execution, and dynamic that workload partitioning is defined at runtime.

Table 6
Performance gain using best non-uniform partitioning for collaborative execution of the all kernels on AMD Kaveri and Intel arria 10 platforms with a domain of size 96 × 96 × 96.

	AMD Kaveri		Intel Arria 10	
Kernel	Partitioning	Perf. gain (%)	Partitioning	Perf. gain (%)
Initialize	0 × 80	0.00	0 × 80	0.00
Redistribute	16 × 80	17.09	16 × 80	15.85
Propagate	48 × 48	12.15	32 × 64	14.29
Bounceback	0 × 80	0.00	16 × 80	19.16
Relaxation	16 × 80	2.39	16 × 80	15.80

Table 6 summarizes the performance improvement of each LBM kernel on both SoC devices. We calculate the performance gains over the best individually performance which was GPU and FPGA for AMD Kaveri and Intel Arria 10, respectively. The Initialize and Bounceback performed better in GPU only executions. The Redistribute and Relaxation performed better with a data partitioning of 16 × 80 in both CPU–GPU and CPU-FPGA. These kernels are more suitable to GPU and FPGA devices than CPU with a performance improvement of 17.09% in CPU–GPU and 15.85% in CPU-FPGA. Propagate kernel, nonetheless, performed better for 48 × 48 and 32 × 64 data partitioning. It means that this kernel is suitable to both devices, having almost the same performance is both CPU–GPU and CPU-FPGA.

Finally, according to the energy consumption results in Tables 4 and 5, GPU and FPGA's higher throughput and lower energy consumption are evident. Considering that the GPU is a significantly more parallel and efficient device than the CPU, it is already expected that not only is its running time shorter but also that its total energy consumption is lower, particularly on highly parallel stencil methods like LBM. Moreover, by collaboratively performing the method's computations using both CPU and GPU/FPGA devices, it is presumable that the total energy consumption increases in comparison to the use of a single device, as seen in our experiments. However, with an increase of 1.72 and 1.8 times in total energy consumption compared to the most energy-efficient decomposition (GPU only on AMD's Kaveri and FPGA only on Intel's Arria 10 platforms), it was possible to increase the overall performance of the method from 9.579 and 0.95 MLUPS to 10.809 and 1.12 MLUPS using a non-uniform domain decomposition on both platforms, respectively.

7. Conclusion

In this work, we analyzed the performance impact of collaborative execution on two low-power heterogeneous architectures: an AMD Kaveri SoC with CPU x86-64 and Radeon R7 GPU devices integrated into a single chip, and an Intel Arria 10 SoC with ARM CPU devices and an integrated FPGA on a single chip. Our case of study was the D3Q19 Lattice Boltzmann Method application with five distinct kernels. We performed experiments with individual executions, CPU only, and GPU/FPGA only, and in a collaborative way through data decomposition of the data domain. Our experimental results suggest that collaborative execution reduces running times and that non-uniform domain decomposition improves the kernel's performance by 11.40% and 15.15% with an increase in total energy consumption compared to the most energy-efficient on of just 1.72 and 1.8 times on AMD Kaveri and Intel Arria 10, respectively. While AMD's Kaveri platform achieved a performance efficiency of up to 10.809 MLUPS and an energy efficiency of 141.786 MLUPKJ, Intel's Arria 10 platform achieved 1.12 MLUPS and 82.272 MLUPKJ.

Future works include the design of experiments using kernels from different applications and the impact of a dynamic partitioning strategy on heterogeneous architectures.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research received funding from the Petrobras project, grant n. 2016/00133-9, by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil, Finance Code 001 and by the project "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9), from FAPERGS, Brazil, and CNPq, Brazil, program PRONEX 12/2014. The authors would like to thank Eduardo Roloff and Francis B. Moreira, from UFRGS, Brazil, for their valuables feedbacks about the paper. We also thank Claudio Schepke from UNIPAMPA, Brazil, for his Lattice Boltzmann method figures.

References

[1] A. Akagic, E. Buza, R. Turcinhodzic, H. Haseljic, N. Hiroyuki, H. Amano, Superpixel accelerator for computer vision applications on arria 10 soc, in: 2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS, IEEE, 2018, pp. 55–60.

[2] U. Aydonat, S. O'Connell, D. Capalija, A.C. Ling, G.R. Chiu, An OpenCL(TM) deep learning accelerator on arria 10, in: International Symposium on Field-Programmable Gate Arrays, 2017, pp. 55–64, arXiv:1701.03534.

[3] D. Bouvier, B. Sander, Applying AMD's Kaveri APU for heterogeneous computing, in: Hot Chips Symposium, 2014, pp. 1–42.

[4] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, J. McDonald, Parallel Programming in OpenMP, Morgan kaufmann, 2001.

[5] L.-w. Chang, J. Gómez-Luna, I. El Hajj, S. Huang, D. Chen, W.-m. Hwu, Collaborative computing for heterogeneous integrated systems, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17, 2017, pp. 385–388.

[6] S. Chen, G.D. Doolen, Lattice Boltzmann method for fluid flows, Annu. Rev. Fluid Mech. 30 (1) (1998) 329–364, arXiv:arXiv:1409.5645v1.

[7] L.H. Crockett, R.A. Elliot, M.A. Enderwitz, R.W. Stewart, The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc, Strathclyde Academic Media, 2014.

[8] S. Cui, N. Hong, B. Shi, Z. Chai, Discrete effect on the halfway bounce-back boundary condition of multiple-relaxation-time Lattice Boltzmann model for convection-diffusion equations, Phys. Rev. E 93 (4) (2016) 043311.

[9] G.P. Dávila, D. Oliveira, P. Navaux, P. Rech, Identifying the most reliable collaborative workload distribution in heterogeneous devices, in: 2019 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2019, pp. 1325–1330.

[10] G. Freytag, P.O.A. Navaux, J.V.F. Lima, L.H.S. Mello, Schnorr, P. Rech, Non-uniform domain decomposition for heterogeneous accelerated processing units, in: International Meeting on High Performance Computing for Computational Science, VECPAR, vol. 13, 2018.

[11] G. Freytag, M.S. Serpa, J.V.F. Lima, P. Rech, P.O. Navaux, Non-uniform partitioning for collaborative execution on heterogeneous architectures, in: 2019 31st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD, IEEE, 2019, pp. 128–135.

[12] Y. Fu, F. Li, F. Song, L. Zhu, Designing a parallel memory-aware Lattice Boltzmann algorithm on manycore systems, in: 2018 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD, 2018, pp. 97–106, http://dx.doi.org/10.1109/CAHPC.2018.8645909.

[13] J. Gómez-Luna, I.E. Hajj, L. Chang, V. García-Floreszx, S.G. de Gonzalo, T.B. Jablin, A.J. Peña, W. Hwu, Chai: Collaborative heterogeneous applications for integrated-architectures, in: 2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2017, pp. 43–54, http://dx.doi.org/10.1109/ISPASS.2017.7975269.

[14] S. Huang, L.-W. Chang, I. El Hajj, S. Garcia de Gonzalo, J. Gómez-Luna, S.R. Chalamalasetti, M. El-Hadedy, D. Milojicic, O. Mutlu, D. Chen, W.-m. Hwu, Analysis and modeling of collaborative execution strategies for heterogeneous CPU-FPGA architectures, in: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, ICPE '19, ACM, New York, NY, USA, 2019, pp. 79–90, http://dx.doi.org/10.1145/3297663.3310305, http://doi.acm.org/10.1145/3297663.3310305.

[15] Z. Jin, H. Finkel, Power and performance tradeoff of a floating-point intensive Kernel on OpenCL FPGA platform, in: Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018, IEEE, 2018, pp. 716–720.

[16] D.B. Kirk, W.-m. Hwu, Programming Massively Parallel Processors: A Hands-On Approach, Morgan kaufmann, 2016.

[17] F. Kono, N. Nakasato, K. Hayashi, A. Vazhenin, S.G. Sedukhin, Performance evaluation of tsunami simulation using OpenCL on GPU and FPGA, in: Proceedings - IEEE 11th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2017, vol. 2018-Janua, 2018, pp. 106–113.

[18] J. Kraus, M. Pivanti, S.F. Schifano, R. Tripiccione, M. Zanella, Benchmarking GPUs with a parallel Lattice-Boltzmann code, in: 2013 25th International Symposium on Computer Architecture and High Performance Computing, IEEE, 2013, pp. 160–167.

[19] S. Matsuoka, A. Smith, M. Matsuda, H.R. Zohouri, N. Maruyamay, Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs, in:, International Conference for High Performance Computing, Networking, Storage and Analysis, SC, vol. 2016, no. November, 2016, p. 35.

[20] S. Mittal, J.S. Vetter, A survey of CPU-GPU heterogeneous computing techniques, ACM Comput. Surv. 47 (4) (2015) 69.

[21] D.J.M. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, P.H.W. Leong, High performance binary neural networks on the Xeon+FPGA platform, in: International Conference on Field Programmable Logic and Applications, FPL, vol. 27, 2017.

[22] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, D. Marr, Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC, in: Proceedings of the 2016 International Conference on Field-Programmable Technology, FPT 2016, no. c, 2017, pp. 77–84.

[23] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, R. Sakellariou, A survey of power and energy predictive models in HPC systems and applications, ACM Comput. Surv. 50 (3) (2017) 37.

[24] K. O'Neal, P. Brisk, Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey, in: 2018 IEEE Computer Society Annual Symposium on VLSI, ISVLSI, IEEE, 2018, pp. 763–768.

[25] K. Ovtcharov, O. Ruwase, J.-y. Kim, J. Fowers, K. Strauss, E.S. Chung, Accelerating deep convolutional neural networks using specialized hardware, Microsoft Res. (2015) 3–6, arXiv:arXiv:1011.1669v3.

[26] C. Schepke, J.V. Lima, M.S. Serpa, Challenges on porting Lattice Boltzmann method on accelerators: NVIDIA graphic processing units and intel xeon phi, in: Analysis and Applications of Lattice Boltzmann Simulations, IGI Global, 2018, pp. 30–53.

[27] C. Schepke, N. Maillard, P.O.A. Navaux, Parallel Lattice Boltzmann method with blocked partitioning, Int. J. Parallel Program. 37 (6) (2009) 593–611.

[28] M.S. Serpa, E.H. Cruz, M. Diener, A.M. Krause, P.O. Navaux, J. Panetta, A. Farrés, C. Rosas, M. Hanzich, Optimization strategies for geophysics models on manycore systems, Int. J. High Perform. Comput. Appl. (2019) 1094342018824150.

[29] A.K. Singh, K.R. Basireddy, A. Prakash, G.V. Merrett, B.M. Al-Hashimi, Collaborative adaptation for energy-efficient heterogeneous mobile socs, IEEE Trans. Comput. 69 (2) (2019) 185–197.

[30] J.E. Stone, D. Gohara, G. Shi, OpenCL: A parallel programming standard for heterogeneous computing systems, Comput. Sci. Eng. 12 (3) (2010) 66.

[31] Y. Sun, X. Gong, A.K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. Mccardwell, A. Villegas, D. Kaeli, Hetero-mark, a benchmark suite for CPU-gpu collaborative computing, in: 2016 IEEE International Symposium on Workload Characterization, IISWC, 2016, pp. 1–10, http://dx.doi.org/10.1109/IISWC.2016.7581262.

[32] K. Vipin, S.A. Fahmy, FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications, ACM Comput. Surv. 51 (4) (2018) 72.

[33] J. Xie, L. Dan, L. Yin, Z. Sun, Y. Xiao, An energy-optimal scheduling for collaborative execution in mobile cloud computing, in: 2015 International Conference and Workshop on Computing and Communication, IEMCON, IEEE, 2015, pp. 1–6.

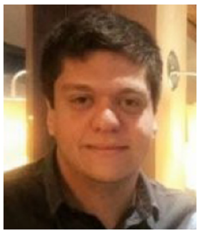
[34] J. Zhang, J. Li, Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network, in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17, 2017, pp. 25–34.

[35] W. Zhang, Y. Wen, D.O. Wu, Energy-efficient scheduling policy for collaborative execution in mobile cloud computing, in: 2013 Proceedings IEEE Infocom, IEEE, 2013, pp. 190–194.

[36] H.R. Zohouri, A. Podobas, S. Matsuoka, Combined spatial and temporal blocking for high-performance stencil computation on FPGAs using OpenCL, Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (2018) 153–162, arXiv:1802.00438.

[37] H.R. Zohouri, A. Podobas, S. Matsuoka, High-performance high-order stencil computation on FPGAs using OpenCL, Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018 (2018) 123–130.

Gabriel Freytag graduated in Computer Science at the Northwest Regional University of the Rio Grande do Sul State (UNIJUI), Brazil, and received his master's degree at the Federal University of Santa Maria (UFSM), Brazil. Currently, he is a Ph.D. student at the Federal University of Rio Grande do Sul (UFRGS). Its research focuses on performance and energy optimizations on heterogeneous architectures.



Matheus S Serpa graduated in computer science at the Federal University of Pampa (UNIPAMPA), Brazil, and received his master's degree at the Federal University of Rio Grande do Sul (UFRGS), Brazil, where he is currently a Ph.D. student. His research focuses on mitigating the contention on SMT processors execution units.

João Vicente Ferreira Lima received a joint Ph.D. degree in computer science by the Federal University of Rio Grande do Sul (UFRGS), Brazil, and the MSTII Doctoral School at the Grenoble University, France. He received a B.Sc. degree in Computer Science in 2007 by the Federal University of Santa Maria (UFSM), Brazil, and a M.Sc. degree in Computer Science in 2009 by the Federal University of Rio Grande do Sul (UFRGS), Brazil. He is associate professor at the Federal University of Santa Maria (UFSM), Brazil, since 2014. His research interests are high performance computing, runtime systems for HPC, parallel programming for accelerators, and distributed computing.

Paolo Rech received his master and Ph.D. from Padova University, Italy, in 2006 and 2009, respectively. He was then a Post-Doc at LIRMM in Montpellier, France. Since 2012 Paolo is an associate professor at UFRGS, Brazil. He is the 2019 Rosen Scholar Fellow at the Los Alamos National Laboratory and he is actively collaborating with major research centers as Jet Propulsion Laboratory and Rutherford Appleton Laboratory and industries as NVIDIA, AMD, and ARM. His main research interests include the evaluation and mitigation of radiation-induced effects in large-scale HPC centers and in autonomous vehicles for automotive applications and space explorations.

Philippe Olivier Alexandre Navaux, Professor Informatics Institute, University UFRGS. Graduated Electronic Engineering, UFRGS, Brazil, 1970, Master Applied Physics, UFRGS, Brazil, 1973, Ph.D. Computer Science, INPG, Grenoble, France, 1979. Professor graduate and undergraduate courses Computer Architecture and High Performance Computing. Leader GPPD, Parallel and Distributed Processing Group. Projects financed by government agencies H2020, Finep, RNP, CNPq, Capes. International Cooperation with France, Germany, Spain and USA. Industrial projects with Microsoft, Intel, HP, DELL. Has oriented near 100 Master and Ph.D. students, has published near 400 papers in journals and conferences. Member SBC, SBPC, ACM, IEEE. Consultant funding organizations DoE (USA), ANR (FR), FINEP, CNPq, CAPES, FAPERJ, FAPESP, FAPERGS, FAPEMIG, and others.