Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2022-0055

Vol. **33**, No. 1, pp. 214-244
January 2023

Heterogeneous LBM Simulation Code with LRnLA Algorithms

Vadim Levchenko[1,*] and Anastasia Perepelkina

Keldysh Institute of Applied Mathematics RAS.

Received 19 February 2022; Accepted (in revised version) 24 August 2022

Abstract. A design of a new heterogeneous code for LBM simulations is proposed. By heterogeneous computing we mean a collaborative computation on CPU and GPU, which is characterized by the following features: the data is distributed between CPU and GPU memory spaces taking advantage of both parallel hierarchies; the capabilities of both SIMT GPU and SIMD GPU parallelization are used for calculations; the algorithms in use efficiently conceal the CPU-GPU data exchange; the subdivision of the computing task is performed with an account for the strong points of both processing units: high performance of GPU, low latency, and advanced memory hierarchy of CPU. This code is a continuation of our work in the development of LRnLA codes for LBM. Previous LRnLA codes had good efficiency both for CPU and GPU computing, and allowed GPU simulation performed on data stored in CPU RAM without performance loss on CPU-GPU data transfer. In the new code, we use methods and instruments that can be flexibly adapted to GPU and CPU instruction sets. We present the theoretical study of the performance of the proposed code and suggest implementation techniques. The bottlenecks are identified. As a result, we conclude that larger problems can be simulated with higher efficiency in the heterogeneous system.

AMS subject classifications: 65Y05, 65Y20, 65-04, 76-10

Key words: LBM, Roofline, memory-bound, GPU, LRnLA.

1 Introduction

In the field of high-performance computational fluid dynamics (CFD), the Lattice Boltzmann Method (LBM) [1, 2] is an extremely popular method. It was proposed as a development of Lattice Gas Automata methods [3], and achieved worldwide recognition after several classic publications [4, 5]. From its early days, researchers predicted the power of the method to simulate extreme-scale problems. This power is in the simplicity of formulation of the method.

*Corresponding author. *Email addresses:* `lev@keldysh.ru` (V. Levchenko), `mogmi@narod.ru` (A. Perepelkina)

LBM has been implemented on many cutting-edge supercomputers [6–9]. There exist impressive applied simulations, such as wind modeling [10], flow simulation around the skeletal structure of a depth sponge [11], simulation of cerebrovascular blood flow [12], automotive simulation [13].

1.1 Heterogeneous computing model

In practice, usefulness of a method depends on the performance of its code implementation. Higher performance requires efficient use of computer hardware. Modern computers, as a rule, contain several computing devices with different architectures for hardware and for software (Instruction Set Architecture — ISA). Let us discuss the problem of developing an LBM code that uses all available computing power with the most efficiency. For such complex hybrid systems, one has to motivate the choice of and adequate (1) computing model, (2) algorithms, (3) implementation tools.

In systems with several processing devices, one of them is the main processor (CPU, host), and at least one is considered to be a coprocessor, which is dedicated to accelerating specific tasks. In this work, GPUs are taken as coprocessors. Considering the computing model, one may treat coprocessor as an extension of the processor. The coding tools in this environment treat the hardware uniformly, hiding the heterogeneity from the user with a common API (Application Programming Interface), such as OneAPI.

This method had its success in the previous generation. In 1980s, coprocessors such as Intel 8087, Weitek X167 were used to speed up floating point operations. Their architectures were extensions of ISA x86. Later, such coprocessors became integrated into mainstream CPU, and their ISAs was integrated into CPU ISA. Now, SIMD extensions for handling vector data (SSE/AVX2/AVX512) and matrix data (AMX) are also integrated in CPU. At the same time, these extensions remain optional. They are used for acceleration of some specific computing tasks. From this perspective, state-of-the art SIMD tools can be seen as a stage of coprocessor evolution. Their use is available through specific methods, compiled into libraries, and the interface can be obscured with compiler options or programming language extensions. The common tools for SIMD extensions of ISA x86+ include intrinsic methods, vectorized data types and vectorized operations implemented as special objects, extensions of compilers and programming language, loop auto-vectorization with OpenMP or optimizing compilers.

In our research of CPU/GPU heterogeneous computing models, we choose to avoid this evolution path. Its advantage is the fact that software code can be easily made universal for systems with and without coprocessors. Among LBM implementations, this path was followed in [14]. As a consequence, the software that was developed before introduction of coprocessors can be effortlessly adapted to new architectures. This is promoted in the Intel OneAPI standard [15]. On the other hand, GPU devices have their own address space and memory hierarchy, and memory bus with CPU RAM often presents itself as a performance bottleneck. If the fact is hidden from the programmer and ignored, computation efficiency often decreases. With other computing models, it can be taken