Commun. Comput. Phys.
doi: 10.4208/cicp.351011.260112s

Vol. **13**, No. 3, pp. 867-879
March 2013

Developing Extensible Lattice-Boltzmann Simulators for General-Purpose Graphics-Processing Units

Stuart D. C. Walsh[1,*] and Martin O. Saar[2]

[1] *Lawrence Livermore National Laboratory, Livermore, California, USA.*[†]
[2] *Department of Earth Sciences, University of Minnesota, Minneapolis, Minnesota, USA.*

Received 31 October 2011; Accepted (in revised version) 26 January 2012

Available online 29 August 2012

Abstract. Lattice-Boltzmann methods are versatile numerical modeling techniques capable of reproducing a wide variety of fluid-mechanical behavior. These methods are well suited to parallel implementation, particularly on the single-instruction multiple data (SIMD) parallel processing environments found in computer graphics processing units (GPUs).

Although recent programming tools dramatically improve the ease with which GPU-based applications can be written, the programming environment still lacks the flexibility available to more traditional CPU programs. In particular, it may be difficult to develop modular and extensible programs that require variable on-device functionality with current GPU architectures.

This paper describes a process of automatic code generation that overcomes these difficulties for lattice-Boltzmann simulations. It details the development of GPU-based modules for an extensible lattice-Boltzmann simulation package – LBHydra. The performance of the automatically generated code is compared to equivalent purpose written codes for both single-phase, multiphase, and multicomponent flows. The flexibility of the new method is demonstrated by simulating a rising, dissolving droplet moving through a porous medium with user generated lattice-Boltzmann models and subroutines.

PACS: 47.11.-j, 07.05.Bx

Key words: Lattice-Boltzmann methods, graphics processing units, computational fluid dynamics.

1 Introduction

Lattice-Boltzmann simulations are a "bottom-up" numerical method capable of modeling a variety of complex fluid mechanical problems (for example, complex boundary condi-

*Corresponding author. *Email addresses:* walsh24@llnl.gov (S. D. C. Walsh), saar@umn.edu (M. O. Saar)
†This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

tions, immiscible fluids, and heat and solute transport) that are difficult or impossible to handle with other modeling methods [1–3][‡]. Their versatility and relative ease of implementation makes lattice-Boltzmann methods particularly attractive for a wide range of applications in both science and engineering [2–4].

In addition, lattice-Boltzmann methods are readily parallelizable and are particularly suited to implementation on single-instruction multiple data (SIMD) parallel processing environments. In recent years, substantial performance increases have been achieved with lattice-Boltzmann methods by exploiting the SIMD environment in modern computer graphics processing units (GPUs) [5–8]. Possibly the first such model proposed by Li *et al*. [9] in the early 2000's achieved an impressive $50\times$ speedup over single core implementations at the time with 9.87 million lattice-node updates per second (MLUPs). Early on, significant drawbacks in the GPU programming model (reduced precision and the requirement that the algorithm be cast in terms of graphics operations), hindered the programmer's ability to develop more complex lattice-Boltzmann models, such as multiphase and multicomponent fluid flow simulations, and presented a significant barrier to widespread use of GPU-based programs [10]. In the years since, however, these barriers have been largely removed with the release of several general purpose GPU-based programming tools, such as BrookGPU [11], the ATI CTM platform [12], the Compute Unified Device Architecture (CUDA) programming model released by NVIDIA [13], and the closely related cross-platform OpenCL standard [14]. In this paper we use NVIDIA's CUDA – a C-like language for general purpose graphics card programming [13]. CUDA provides new functionality that distinguishes it from the early GPU programming models (*e.g.* random access byte-addressable memory and support for coordination and communication among processes through thread synchronization and shared memory), thereby allowing more efficient processing of complex data dependencies. CUDA also supports single and double precision, and IEEE-compliant arithmetic [13]. In addition, higher-level libraries have been created to simplify CUDA code development, such as the Thrust library, a collection of parallel algorithms modeled on the C++ Standard Template Library [15]. These advances have extended the applicability of GPU computation to a much broader range of computational problems in science and engineering [8].

Nevertheless, while these new GPU programming tools dramatically improve on earlier generations, GPU implementations continue to lack some of the flexibility of CPU based programs. In part this lack of flexibility arises from differences in GPU and CPU architectures. Increased GPU performance is achieved by distributing computational tasks across several multiprocessors. Together these multiprocessors are able to achieve substantial processing throughput, however, the individual GPU processing threads lack the performance, independence and resources (*e.g.* registers) found in their CPU counterparts. These hardware differences restrict the complexity of the operations available to

[‡]In contrast with traditional "top-down" numerical methods where the material behavior is modeled directly, lattice-Boltzmann methods may be viewed as "bottom-up" numerical methods in which the behavior emerges from underlying smaller-scale processes – namely the interactions of discrete analogues to the single-body particle distribution functions described by the classical Boltzmann equation.