Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 55 (2008) 1514–1524
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Comparison of implementations of the lattice-Boltzmann method

Keijo Mattila[a,b,*], Jari Hyväluoma[b], Jussi Timonen[b], Tuomo Rossi[a]

[a] *Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, FI-40014 Jyväskylä, Finland*
[b] *Department of Physics, University of Jyväskylä, P.O. Box 35, FI-40014 Jyväskylä, Finland*

Abstract

Simplicity of coding is usually an appealing feature of the lattice-Boltzmann method (LBM). Conventional implementations of LBM are often based on the two-lattice or the two-step algorithm, which however suffer from high memory consumption and poor computational performance, respectively. The aim of this work was to identify implementations of LBM that would achieve high computational performance with low memory consumption. Effects of memory addressing schemes were investigated in particular. Data layouts for velocity distribution values were also considered, and they were found to be related to computational performance. A novel bundle data layout was therefore introduced. Addressing schemes and data layouts were implemented for the Lagrangian, compressed-grid (shift), swap, two-lattice, and two-step algorithms. Implementations were compared for a wide range of fluid volume fractions. Simulation results indicated that indirect addressing implementations yield high computational performance. However, they achieved low memory consumption only for very low fluid volume fractions. Semi-direct addressing implementations could also provide high computational performance. The bundle data layout was found to be competitive, sometimes by a wide margin, in all the cases considered.

Keywords: Lattice-Boltzmann method; Computational fluid mechanics; High-performance computing; Memory addressing schemes

1. Introduction

While the lattice-Boltzmann method (LBM) has gained popularity in computational fluid dynamics [1], it has also been recognized that it is both computationally demanding and memory intensive. Depending on the specific application, simulations may be hampered by an excessive computing time or by high memory requirements. Here, we consider applications that require very large simulation domains – as measured by the number of lattice nodes – in which a significant volume fraction is occupied by solid structures. A typical example of such an application is flow in porous media such as paper and sandstone. Furthermore, we only consider here single-phase flows. Notice also that the geometry of the simulation domain is assumed to be immobile. This enables particular implementation techniques.

* Corresponding author at: Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, FI-40014 Jyväskylä, Finland.
E-mail address: kemattil@cc.jyu.fi (K. Mattila).

0898-1221/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2007.08.001

As a result of extensive research efforts, there already exist means to enhance the efficiency of LBM (see for example Ref. [2] and references therein). It is the aim of this work to analyze various implementations of LBM, so as to compare their computational performance and memory consumptions. More precisely, we consider explicit time-marching implementations of LBM with direct, semi-direct, and indirect addressing. Also, we analyze the effect of data layout on computational performance.

Time evolution in explicit time-marching implementations results from the alternation of two distinct steps, streaming and collision steps. A streaming step gives rise to coupling of data of adjacent lattice nodes. Differences between basic algorithms are related to their treatment of this data dependence. We have identified five basic algorithms for the implementation of LBM: the Lagrangian, compressed grid (shift), swap, two-lattice, and two-step algorithm [3–5]. Each of these algorithms has its own characteristic features, which are manifested in the efficiency of implementation. We implemented four of the algorithms with both semi-direct and indirect addressing. The Lagrangian algorithm was implemented only with direct addressing. We compare here these implementations, and analyze in particular the influence of the addressing scheme and data layout in each case.

Previously, in Ref. [6], Schulz et al. compared the two-lattice and two-step algorithms with indirect addressing. They also compared the memory consumptions of their implementations for direct and indirect addressing. Pan et al. [7] analyzed the two-lattice algorithm with direct and indirect addressing. A Lagrangian algorithm was introduced and compared with the two-step algorithm in Ref. [3]. To overcome the high memory consumption of the two-lattice algorithm, Pohl et al. introduced the compressed grid (shift) algorithm, and gave some benchmarking results including comparison with the two-lattice algorithm [4]. Recently, the compressed-grid algorithm was investigated further, and the name shift algorithm was proposed [5].

Recently a candidate for the implementation of LBM was developed and named the swap algorithm [5]. Moreover, in Ref. [5], the computational performance and memory consumption of implementations of the shift, swap, two-lattice, and two-step algorithms with a collision-optimized data layout and semi-direct addressing were compared. Here, this comparison is supplemented by analyzing the addressing schemes and data layouts. Collision-optimized as well as propagation-optimized data layouts and their effect on the computational performance of the two-lattice algorithm were also analyzed in Ref. [8].

This paper is organized as follows. Section 2 includes a very short exposition of LBM, while Section 3 describes the data storage model, data layouts, and addressing schemes utilized in our implementations of LBM. The basic algorithms considered are introduced in Section 4, and results of our numerical experiments are presented in Section 5. In the numerical experiments, we apply the D3Q19 model with the BGK collision operator and the halfway-bounce-back boundary condition. Finally, some conclusions are drawn in Section 6.

2. Lattice-Boltzmann method

In 1986, Frisch et al. introduced cellular automata that obey conservation laws [9]. It turned out to be an important idea, since their automata, lattice-gas cellular automata (LGCA), were capable of simulating real fluid flows. This led to a rapid progress from which LBM soon emerged [10]. Although there is a historical coupling of LGCA and LBM, it has already been shown that LBM can be derived directly from the Boltzmann equation [11–13]. In LBM, the state of the system is defined by single-particle distribution functions $f_i(\vec{r}, t)$ for the probability of finding a (fictitious) fluid particle at site $\vec{r}$ at time t with velocity $\vec{c}_i$. Here and hereafter, $\vec{r}$, t, and $\vec{c}_i$ are expressed in lattice units. The dynamics of the system is governed by the lattice-Boltzmann equation (LBE)

$$f_i(\vec{r} + \vec{c}_i, t + 1) = f_i(\vec{r}, t) + \Omega_i(\vec{f}(\vec{r}, t)), \quad i = 0, \ldots, b - 1, \tag{1}$$

where Ω_i is a collision operator and b is the number of possible velocities for the fictitious particles. With $\Omega_i(\vec{f}(\vec{r}, t))$, we emphasize the dependence of the collision operator on all b distribution functions associated with site $\vec{r}$ at time t. Eq. (1) describes the time evolution of distribution values $f_i(\vec{r}, t)$.

LBE can be split into collision and streaming steps:

$$\begin{array}{lrcl} \text{collision:} & f_i(\vec{r}, t^\star) & = & f_i(\vec{r}, t) + \Omega_i(\vec{f}(\vec{r}, t)) \\ \text{streaming:} & f_i(\vec{r} + \vec{c}_i, t + 1) & = & f_i(\vec{r}, t^\star). \end{array}$$

This partition of LBE provides a basis – at least conceptually – for the implementation of LBM. A characteristic of LBM is the simplicity of coding, local interactions that allow for parallel computing, and easy implementation of