華中科技大學

毕业设计 [论文]

题目： 基于 GPU 的复杂边界格子 Boltzmann 模拟

院　　系：能源与动力工程学院

专　　业：热能与动力工程

姓　　名：朱 炼 华

指导教师：郭 照 立

2013 年 6 月 9 日

目　录

摘　要

具有复杂流固边界的流动现象广泛出现在化石能源开采、新能源技术开发、化工过程、环境保护等领域，这类流动现象通常还耦合多组分/多相组分、化学反应等复杂现象。格子 Boltzmann 方法（LBM）由于能够高效处理复杂流固边界、容易耦合多相模型模型而成为研究这类问题的有力工具。LBM 在计算上具有天然并行性，非常适合于在近几年新出现的通用图形处理器（GPGPU）上并行实现。

现代图形处理器（GPU）的计算能力和存储带宽远远超过了目前主流的 CPU，其性能/价格比和性能/能耗比相对于传统的基于 CPU 的计算机集群或多核计算机具有很大优势。

本文利用 CUDA GPU 的编程技术，编制了 LBM 的 GPU 计算程序，模拟了多孔介质内（复杂边界）单组分单相和多组分多相流动，重点研究了复杂流固边界与多组分 LBM 模型在 GPU 上的高效处理方法，并对程序做了性能测试和分析。

关键词：　格子 Boltzmann 方法，通用图形处理器，GPGPU，多相流，渗流

Abstract

Numerical simulations of flows with complex solid-fluid boundary are of great importance in industrial areas such as Oil&Gas recovery, renewabel energy development, chemical process and environment engineering. Lattice Boltzmann method(LBM) is a powerful tool for simulating of these complex flow phenomenon due to its easy processing of solid-fuid boudary and ability of modeling of multiphase/multicomponent flow.

From the compuatational standpoint, LBM is well suited for parallel implementation. On the emerging massively parallel graphics processing units(GPU) platform, LBM exhabits an extrodinary performance at a rather low cost. The combining of the GPU and LBM is promising technique to simulate complex flow in complex geometries.

In this work, parallel LBM simulators were developed within the CUDA GPU programming framework. Using the simulators, we carried out simulation of single phase flow and mulitcompnet-multiphase flow in porous media. We also investigate performance optimization techniques to deal with the complex geometries and interacation of components. Performance evalution and detailed analysis are conducted for the optimization techniques.

Key Words: lattice Boltzmann method, graphics processing units, high performance computing, multiphase flow, porous media

目　录

一 绪论

1. 选题背景及意义

复杂边界流动现象广泛存在于石油天然气开采、化工过程、新能源技术、医学、环境保护等领域。这类现象可以抽象为流体在多孔介质中的流动。由于流固边界高度复杂，通常只能用实验方法和数值方法研究这类现象。目前用实验方法研究真实多空介质流动时，还缺乏有效的实验设备观测细微孔隙结构中流动现象，并且实验方法还具有费用昂贵、实验周期长、参数控制不易、流场数据获取不够全面等缺点。伴随着计算机技术的发展，研究这类现象的计算流体动力学方法发展越来越成熟，并且没有实验研究的诸多限制和缺点，已经成为一个重要的研究手段。其中格子 Boltzmann 方法（LBM）作为一种介观方法，相对于传统 CFD 方法，无须显式进行空间网格划分，处理复杂的流固边界简单，因此特别适合于在孔隙尺度上模拟多孔介质流动。另外，由于它在微观上基于分子动力学，继承了分子动力学刻画分子间相互作用简单的特点，而宏观现象如组分间和相间相互作用的微观本质就是分子间相互作用，因此用格子 Boltzmann 方法模拟多组分和多相流动也具有很大优势，比如它能自动捕捉复杂的相界面 [1]。但由于 LBM 是一种显式的时间推进方法，并且空间离散点较多，所以计算量通常非常大, 基于 LBM 的实际研究或工程应用一般都要利用并行计算加速 [2,3]。标准的 LBM 算法中粒子碰撞迁移都具有局部性，因此进行大规模并行计算通常都能获得良好的并行效率和可扩展性。早期的 LBM 并行程序大多数是基于消息传递函数接口协议（MPI），运行在通过高速网络互连的商业计算机集群上面。而最近几年刚刚兴起的通用计算图形处理器（GPGPU）则为 LBM 并行计算提供一条更高效、更廉价、更节能的途径。

图形处理器最开始作为 CPU 的协处理器，用于加速图形、图像处理等计算密集性任务。早期的 GPU 被用作固定函数的流水线，而经过多年的发展，GPU 的可变编程性逐渐增强，这使得将其用于非图形学领域的通用科学计算成为可能。早在 2000 年，就有计算机科学家及电磁学研究领域的研究者用 GPU 来加速他们的科学计算程序，这就是后来兴起的 GPGPU 的开端。GPU 相对于 CPU 提供的处理能力和存储带宽要大得多，目前市场上主流的 GPU 浮点数处理能力是同期 CPU 的 10 倍左右，而存储带宽则是 CPU 的 5 倍左右 [4]。并且受到市场需求的推动，GPU 的这一优势还在扩大。在价格和能耗上 GPU 相对于 CPU 也具有较大优势。如一台配置了较新型号消费级 GPU 的普通 PC 机，其单精度峰值计算能力可以达到 Tflops 级别 [5]。

GPU 生产商 NVIDIA 公司于 2007 年推出了 CUDA(统一计算构架), 利用 CUDA 技术进行 GPGPU 的编程难度大为降低。在 CUDA 推出之前，要想将 GPU 用于非图形学的科学计算，则必须将这种计算以图形学的语言来描述，如将普通的计算（数据操作）映射为光栅化和帧缓冲等图形操作，将普通数据映射为纹理等图形数据。这种面向图形应用领域硬件层面的编程方式，对于非图形学领域的工作者而言，入门门槛较高。CUDA 技术使用的是一种经过扩展的 C 语言，它在标准的 C 语言中加入了能调用 GPU 计算功能的并行机制，并提供了包含编译器驱动、常用并行函数库等在内的一整套完整的开发环境。熟悉 C 语言的用户可以用一种很自然的方式来编制 GPU 并行程序。CUDA 推出之后，并很快在科学计算领域流行起来，目前有越来越多的国内外研究单位开始利用该技术加速计算，国内外一些大学亦开始开设了 GPGPU 高性能计算的课程。NVIDIA 公司同时也推出了专用于加速科学计算应用的高端 GPU 系列产品 ——Tesla GPU。在 2012 年 11 月公布的全球超级计算机性能排行榜 TOP500 中，位列第一名的 Titan 超级计算机就使用了 261632 个 Kepler 构架的 K20GPU 作为加速器，其理论峰值计算能力达到了 27PFlops（2.7×10^7GFlops）。目前已经有分子动力学、计算流体力学、计算结构力学、计算化学、医学成像、计算金融等一系列科学计算应用被被移植到 GPU [6]，并获得了相对于 CPU 数倍甚至一个量级以上的的性能提升。

相对于其它的计算流体动力学方法，LBM 在 GPU 上往往能获得更高的加速比，甚至在单块 GPU 上就能获得相对于 CPU 两个量级的加速比。究其原因，一是因为在 LBM 计算过程中的访存具有空间局部性，每个节点的碰撞、迁移过程只涉及到存取临近格点的数据。能充分发挥 GPU 高带宽的优势；二是因为在 LBM 计算与图形、图像处理的计算、访存模式十分相近 —— 对大量同类数据元素进行某种简单的操作，这种模式非常适合于细粒度的并行化处理。

利用 GPU 加速，能大幅缩短 LBM 模拟的计算时间，在三维高分辨率模拟时，这一优势尤为明显。如原本需要一天时间运行的串行程序，利用 GPU 加速，在十几分钟内并可以算完。在孔隙尺度进行 LBM 模拟时，通常分辨率要求非常高，用常规 CPU 串行程序模拟时特别耗时。本文的目的就是利用 GPU 实现三维多空介质中单相和多相流动 LBM 模拟，并研究相应的性能优化方法。

2. 本课题国内外研究情况

早在 2003 年，Li 等人最先在 GPGPU 上实现了 LBM 并行计算 [7]，其做法是将速度分布函数映射为二维纹理数据，并将格子 Boltzmann 方程完全映射为光栅化和帧缓冲操作。他们按这种方式实现的 GPU 程序相比 CPU 版本的程序，计算速度提

速了十多倍。随后又更多的人在 GPU 上实现 LBM 并行计算。如文献 [8] 中, 作者模拟了墨水在吸水性纸张上扩散的过程。Fan 等人搭建了有 30 个 GPU 节点的计算机集群，并用 LBM 在上面模拟了纽约时代广场上的空气污染物的扩散过程 [9]。文献 [10] 和 [11] 分别模拟了多相环境的熔化和肥皂泡及羽毛的运动。值得指出的是，这些工作都是在 CUDA 技术推出之前进行的，所以都是用计算机图形学的编程方法实现的。

自 NVIDIA 公司于 2007 年推出 CUDA 技术后，Jonas Tölke 利用该技术在 NVIDIA 8800Ultra 显卡上实现了二维的 LBM 并行计算，并获得了相对于标准 CPU 一个量级以上的性能提升 [12] 。在该文中，作者首次提出利用 GPU 的共享内存辅助完成 LB 计算过程中的迁移步，以达到合并访问进而最大限度利用 GPU 提供的带宽。Jonas Tölke 随后又将该方法用于三维 LBM 模拟，结合一种特殊的 D3Q13 格子模型，该方法在单个 GPU 上达到了两个量级的加速比 [5]。

由于单块 GPU 上搭载的显存大小有限，不能满足大规模模拟需求，而如果将 MPI 的分布式存储技术与 GPU 结合，在通过高速网络互连的各个计算节点上搭载 GPU，构造多 GPU 集群，则可以有效解决这个问题。在这种构架中，GPU 和 GPU 之间的通信以 CPU 作为中间层，利用当今流行的 MPI 标准实现跨节点数据传输。上一节提到的 Titan 超级计算机利用的就是这种构架。在 LBM 的多 GPU 并行化方面，Obrecht 等人 [13,14] 以及国内中科院过程所李博 [15,16] 等人做了不少相关工作。关于 LBM 的 GPU 并行程序性能分析及优化方面的工作可见文献 [17–20]。

文献 [21, 21, 22] 中专门研究了多孔介质孔隙尺度 LBM 模拟的 GPU 并行化。考虑更复杂的流动如多相、多组分 LBM 模拟的 GPU 并行实现，目前文献中并不多见。Myre 等人 [20] 运用控制变量法详细测试并分析了单组分、多组分 LBM 的多 GPU 程序的性能，并得出了影响 GPU 程序性能的各个因素的相对重要性。Rosales 等人 [23] 用多 GPU 实现的多相 LBM 程序相对于单核 CPU 提速了 40 倍。Redapangu 等人 [24] 利用 GPU 实现的多相 LB 模型研究了浮力驱动的非混溶液体流动，相对 CPU 加速了 25 倍。这些文献中的流场多为简单流场。通过文献调研我们发现，针对多孔介质多相渗流的 GPU 实现，目前只有 Tölke 等人进行了相关研究 [25,26]。他们使用的多相模型是颜色 LBE 模型，这是一种早期的多相流模型 [27]，较后来提出的的 Shan-Chen 模型及其改进模型存在一些缺点 [28]。

3. 本文的主要研究内容和组织结构

（1） 研究内容

本论文主要的研究内容包括在 GPU 平台利用 CUDA 技术实现单组分单相渗流、多组分多相渗流格子 Boltzmann 数值模拟的并行计算，开发基于 GPU*的多相渗流高性能计算程序。具体研究内容包括如下方面：

1. 实现目前文献中常见的针对简单流动问题的 LBM 模拟 GPU 程序；
2. 分析多空介质孔隙尺度单组份单相流动 LB 模拟的 GPU 程序性能优化方法；
3. 高效实现多空介质孔隙尺度多组分多相流动 LB 模拟的 GPU 程序，使用的多相模型为一种基于 Shan-Chen 模型的改进模型。

（2） 章节安排

本文的组织结构如下：

第一章是绪论，介绍本文研究课题的相关背景、意义以及国内外关于本课题的研究情况。

第二章介绍格子 Boltzmann 方法的基本原理、边界处理及本文使用的多相 LBE 作用力模型。

第三章介绍现代 GPU 构架和 CUDA 技术以及 GPU 程序设计和优化的基本原则。

第四章介绍了目前文献上常见的 LBM 在 GPU 上实现方法，并利用我们实现的 GPU 程序模拟了二维方腔流和外力驱动的三维方截面直管道流动问题，验证了程序正确性并进行了程序性能测试。

第五章介绍了进行多孔介质孔隙尺度模拟时的减少计算量和存储空间的稀疏存储算法，并提出了一种结合位操作和逻辑运算的指令优化技术。随后分别针对这两种优化方法编制了三维 GPU 程序，模拟了 BCC 多孔介质结构中的单组分单相渗流，测试了其绝对渗透率验证了，并详细分析了影响程序性能的各种因素。

第六章针对多组分 LB 模拟的特点对第 5 章提出的算法进行了重新设计，并开发了相应的的 CUDA 程序。先用两个标准算例验证了程序的正确性，随后模拟了真实多孔介质中的多相渗流。最后分析了程序的性能。

第七章总结全文并展望未来的研究工作。

*目前是单 GPU 环境

二 格子 Boltzmann 方法基本原理

1. LBM 基本模型

格子 Boltzmann 方法（LBM）是一种求解 Navier-Stokes 方程的方法。在历史上由格子气自动机发展而来的，但其核心方程 ——格子 Boltzmann 方程（LBE）也可以由连续 Boltzmann 方程推导出来。它可以看做是连续 Boltzmann 方程在速度空间和几何位置空间的离散化形式。LBE 描述的是离散几何空间中粒子速度分布函数的演化规则：

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\delta t, t+\delta t) - f_i(\boldsymbol{x}, t) = \Omega_i\left(f(\boldsymbol{x}, t)\right) \tag{2.1}$$

其中 $f_i(\boldsymbol{x}, t)$ 是速度分布函数，表示在时刻 t, 占据空间位置 $\boldsymbol{x}$ 处的粒子具有离散速度 $\boldsymbol{c}_i$ 的概率，δt 是相邻演化步时间间隔。等式右边 Ω_i 是碰撞算子，表示演化过程相邻粒子之间的相互作用对速度分布函数的影响。格点上的宏观量如密度 ρ、速度 $\boldsymbol{u}$ 由速度分布函数求得：

$$\rho = \sum_i f_i, \quad \rho\boldsymbol{u} = \sum_i f_i\boldsymbol{c}_i \tag{2.2}$$

LBE 方程在计算上可以分为两个步骤，即碰撞步

$$f_i'(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) + \Omega_i\left(f(\boldsymbol{x}, t)\right) \tag{2.3}$$

和迁移步

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\delta t, t+\delta t) = f_i(\boldsymbol{x}, t) \tag{2.4}$$

粒子之间的碰撞要求满足质量守恒，动量守恒，即要求碰撞算子满足：

$$\sum_i \Omega_i(f) = 0, \quad \sum_i \Omega_i(f)\boldsymbol{c}_i = 0 \tag{2.5}$$

最简单碰撞算子 Ω_i 模型是由 Qian 等人提出的 DnQb LBGK 模型 [29]，相对于后来提出的多松弛时间模型，它也被称作单松弛时间模型。在 LBGK 模型中，Ω_i 的形式为：

$$\Omega_i(f) = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right) \tag{2.6}$$

其中 τ 为无量纲松弛时间，与流体动力学粘性关联

$$\nu = c_s^2(\tau - \frac{1}{2})\delta t \tag{2.7}$$

f_i^{eq} 为平衡态分布函数，由格点的宏观量决定：

$$f_i^{eq} = \omega_i \rho \left[1 + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{c}_i \cdot \boldsymbol{u})^2}{2c_s^4} - \frac{u^2}{2c_s^2}\right] \tag{2.8}$$

其中 ω_i 为权系数，$c_s = \sqrt{RT}$ 与声速有关的常数。本文使用的主要的三维格子为 D3Q19 模型，其离散速度为：

$$\boldsymbol{c} = \begin{bmatrix} 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} c$$

其中 c 为格子速度，定义为格子长度与时间步长之比，即 $c = \frac{\delta x}{\delta t}$ 。c_s 和 ω_i 分别为：

$$c_s = \frac{c}{\sqrt{3}}, \quad \omega_i = \begin{cases} 1/3 & c_i^2 = 0 \\ 1/18 & c_i^2 = c^2 \\ 1/36 & c_i^2 = 2c^2 \end{cases}$$

图2.1为 D3Q19 格子模型离散速度示意图。

2. 多松弛时间模型

在 LBGK 模型中，各个方向的速度分布函数的松弛时间都是 τ，所以也被称作单松弛时间模型。另一种常用的碰撞算子模型是多松弛时间模型（MRT），Lallemannd 和 Luo 曾对这一模型做过细致的理论分析，证明该模型比 LBGK 模型在参数选择自由度和数值稳定性方面具有更大优势，并且物理原理更清晰。值得指出在 LBGK 在模拟多孔介质流动时，会得到绝对渗透率与流体粘性相关联的非物理结果，而 MRT 模型则可以大大改善这一问题，Pan 等人对这一问题做了详细算例验证和分析 [30]。MRT 模型与 LBGK 模型不同之处在于它的碰撞过程是在矩空间中完成的。

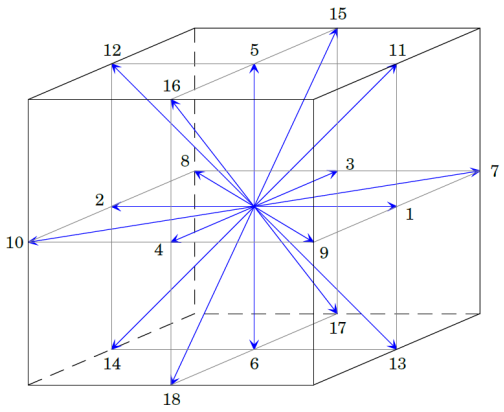


图 2.1: D3Q19 模型离散速度

碰撞前速度分布函数被一个正交变换矩阵 $\boldsymbol{M}$ 映射到矩空间：

$$m = \boldsymbol{M} \cdot \boldsymbol{f} \tag{2.9}$$

其中 m 为矩向量，对于 D3Q19 模型，$\boldsymbol{M}$ 为

$$\boldsymbol{M}=\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
-30 & -11 & -11 & -11 & -11 & -11 & -11 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8\\
12 & -4 & -4 & -4 & -4 & -4 & -4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1\\
0 & -4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1\\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1\\
0 & 0 & 0 & -4 & 4 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1\\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & -4 & 4 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0\\
0 & 2 & 2 & -1 & -1 & -1 & -1 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
0 & -4 & -4 & 2 & 2 & 2 & 2 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
0 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1\\
0 & 0 & 0 & -2 & -2 & 2 & 2 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0
\end{pmatrix}$$

相应的矩向量为

$$m=(\rho,e,\epsilon,j_x,q_x,j_y,q_y,j_z,q_z,3p_{xx},3\pi_{xx},p_{ww},\pi_{ww},p_{xy},p_{yz},p_{zx},m_x,m_y,m_z)$$

其中各分量的意义及其对应的平衡态形式见参考文献 [30]，此处不再详细给出。

MRT 模型中碰撞时先将 $\boldsymbol{f}$ 转换到矩空间得到矩向量 $\boldsymbol{m}$，然后在矩空间完成碰撞，最后把碰撞更新后的 m 转换回速度分布函数 $\boldsymbol{f}$，整个过程可以表示为

$$\Omega_i=\boldsymbol{M}^{-1}\boldsymbol{S}(\boldsymbol{m}-\boldsymbol{m}^{eq}) \tag{2.10}$$

其中 $\boldsymbol{S}$ 为对角矩阵，其对角线上的各个元素为矩向量各分量的无量纲松弛时间，其

形式为

$$\boldsymbol{S} = diag(0, s_1, s_2, 0, s_4, 0, s_4, 0, s_4, s_9, s_1 0, s_9, s_{10}, s_{13}, s_{13}, s_{13}, s_{16}, s_{16}, s_{16})$$

通常取 $s_9 = s_{13} = s_\nu$, s_ν 与流体动力学粘性相关

$$\nu = c_s^2(\frac{1}{s_\nu} - \frac{1}{2})\delta t \tag{2.11}$$

其它松弛时间可自由调节以获得更高的稳定性 [31]。

3. 边界处理

在迁移过程中，处于流场边界上的格点需要特殊处理，因为边界以外没有粒子迁移到这些格点，LBM 的边界处理的目标就是根据给定的宏观边界条件构造出这些未知的粒子分布函数。在这一节中笔者主要讨论本文用到的三种边界处理格式，分别为周期性边界条件、反弹格式及郭照立提出的非平衡外推格式 [32]。

周期性边界条件 某些问题流场本身就有空间周期性，这时可以只求解一个周期单元。这种边界处理在 LB 里面处理非常容易，当前时间步从一侧流入流场的粒子就是上一时刻从另一侧流入流场的粒子。

反弹格式 反弹格式是来处理无滑移速度壁面最简单的，并且物理背景清晰。其基本想法是认为靠经壁面的格点上的粒子在当前时间步流动到壁面并与壁面反弹后由其反方向弹回，在下一时间步正好又回到这个格点。这种格式自动满足质量守恒和动量守恒。需要注意的是，标准如果反弹格式中靠近壁面执行反弹的格点本身不执行碰撞更新操作，则称该格式为标准反弹格式，否则为修正反弹格式，前者仅具有一阶精度，而后者具有二阶精度。另一种具有二阶精度的反弹格式为 Half-Way 反弹格式，这种格式假设壁面正好位于格点连线之间。

非平衡外推格式 非平衡外推的基本思想是将边界格点上未知的分布函数分为平衡态和非平衡态部分，其中非平衡部分，由临近的流场内部格点的非平衡部分一阶外推得到，而平衡态部分则根据给定的边界宏观量来构造，构造过程缺少的宏观量也由临近的流场内部格点上的相应宏观量外推得到。这种格式整体精度是二阶的 [28]。

4. 多相/多组分 LBE 模型

宏观的相间相互作用本质上是组分中微观分子间的相互作用，因此如果能准确描述这种分子间相互作用，则宏观多相/多组分复杂的流动现象会自然体现出来。LBM 的微观本质和介观特点决定了 LBM 也可以通过构造粒子间相互作用模型来模拟多相/多组分流动。目前在 LBM 中，描述粒子间相互作用的方式有颜色模型、伪势模型、自由能模型和动力学模型。其中伪势模型又称 Shan-Chen 模型，是由 Shan 和 Chen 于 1993 年提出的 [33]。该模型假设流体粒子之间存在一种非局部的相互作用，并通过引入一种势函数来描述这种相互作用，作用力的大小就是势函数的梯度。该模型提出后，针对其存在的问题，又有多种改进模型被提出。本文所采用的多相多组分模型 [34] 就是针对对 Shan-Chen 模型的一种改进。该模型主要解决了 Shan-Chen 模型中平衡态密度和粘度相关的问题，并且能过模拟大粘度比的组分。下面将以该模型为背景简要介绍运用 LBM 模拟多组分/多相问题的要点。

(1) 组分间相互作用

在原始的 Shan-Chen 模型中，组分间的相互作用力通过一个势函数来描述，并且这个相互作用力对碰撞过程的影响通过校正有效平衡态速度来体现 [33]。相互作用力的表达式为

$$\boldsymbol{F}_k(\boldsymbol{x}) = -\psi_k(\boldsymbol{x}) \sum_{\bar{k}} g_{k\bar{k}} \sum_i \psi_{\bar{k}}(\boldsymbol{x} + \boldsymbol{c}_i \delta t) \boldsymbol{c}_i \tag{2.12}$$

其中 ψ_k 是势函数，对于多组分模型通常就取为组分的密度即，$\psi_k = \rho_k$。$g_{k\bar{k}}$ 是反映组分粒子间相互作用的系数。在本文使用的这个模型中，每个组分所受到的合外力显式（explict force ）的加到了其格子 Boltzmann 方程中 [34]，后文中称这种模型为 EF 模型。每个组分的 LBE 为

$$f_i^k(\boldsymbol{x} + \boldsymbol{c}_i \delta t) - f_i^k(\boldsymbol{x}, t) = \Omega_i^k(f(\boldsymbol{x}, t)) + \frac{\Delta t}{2} \left[f_i^{F,k}(\boldsymbol{x} + \delta t, t + \delta t) + f_i^{F,k}(\boldsymbol{x}, t) \right] \tag{2.13}$$

其中的 $f_i^{F,k}$ 是粒子所受合外力引起的速度分布函数的改变，碰撞算子 Ω_i 与单组分 LBE 形式相同。$f_i^{F,k}$ 定义为

$$f_i^{F,k} = \frac{\boldsymbol{F}_k \cdot (\boldsymbol{c}_i - \boldsymbol{u}_{eq})}{\rho_k c_s^2} f_i^{eq,k} \tag{2.14}$$

其中 $\boldsymbol{F}_k$ 为组分 k 的粒子所受外力，包括组分间作用力。$\boldsymbol{u}^{eq}$ 是有效速度，定义为

$$\boldsymbol{u}^{eq} = \sum_k \frac{\rho_k \boldsymbol{u}_k}{\tau_k} / \sum_k \frac{\rho_k}{\tau_k} \tag{2.15}$$

方程(2.13)右端包含新时间步的变量，所以是显式形式，通过做代换 $\bar{f}_i^k = f_i^k - \frac{\delta t}{2} f_i^{F,k}$，该方程可变为显式形式

$$\bar{f}_i^k(\boldsymbol{x} + \boldsymbol{c}_i \delta t) - \bar{f}_i^k(\boldsymbol{x}, t) = \frac{1}{\tau_k}\left[f_i^{eq,k}(\boldsymbol{x}, t) - \bar{f}_i^k(\boldsymbol{x}, t) - \frac{\delta t}{2} f_i^{F,k}\right] + \delta t f_i^{F,k} \tag{2.16}$$

每个组分的宏观量定义为

$$\rho_k = \sum_i \bar{f}_i^k, \quad \rho_k \boldsymbol{u}_k = \sum_i \bar{f}_i^k \boldsymbol{c}_i + \frac{\delta t}{2} \boldsymbol{F}_k \tag{2.17}$$

混合流体的真实速度为

$$\boldsymbol{u} = \frac{\sum_k \rho_k \boldsymbol{u}_k}{\sum_k \rho_k} \tag{2.18}$$

流体状态方程为

$$p = c_s^2 \sum_k \rho_k + \frac{c_0}{2} \sum_{k\bar{k}} g_{k\bar{k}} \psi_k \psi_{\bar{k}} \tag{2.19}$$

值得指出的是在原文献 [34] 中，为能模拟大粘度比组分，式(2.12)表达为

$$\boldsymbol{F}_k(\boldsymbol{x}) = -c_0 \psi_k(\boldsymbol{x}) \sum_{\bar{k}} g_{k\bar{k}} \nabla \psi_{\bar{k}}(\boldsymbol{x}) \tag{2.20}$$

式中 c_0 是个常速，对于 D3Q19 格子模型，$c_0 = 6$，式中的梯度算子用具有较好各向同性的的高阶差分算法替代。但考虑到 GPU 上对显存访问的特殊要求（见4.节中的解释），我们在后文中实现的多组分 GPU 程序使用的都是一阶中心差分格式。

（2） 流固界面相互作用

流固边界处的流体格点的某些方向上的相邻格点可能是固体格点，为反映固体边界对各组分的作用，可以将固体格点等效为另一种假想的组分，其密度为常数 ρ_s，它与流体组分间的相互作用系数用 g_{ks} 替代，g_{ks} 大小反应组分 k 对该固体表面的润湿性，通过调整不同组分的 g_{ks} 控制相界面接触角的大小 [35]。

5. 小节

本章介绍了 LBM 基本原理。在第一节中介绍了常用的 D2Q9 和 D3Q19 格子模型，以及单松弛时间（LBGK）模型，第二节介绍了后文将要使用的多松弛时间（MRT）模型，第三节介绍了常用的 LB 边界处理方法。第四节详细介绍了本文所使用多相 LB 模型。

三 GPU 构架及 CUDA 编程技术

1. GPU: 大规模并行处理器

GPU 最开始是为图形渲染而设计的，它负责处理计算机系统的图形渲染任务。图形渲染操作具有高度数据并行性，一般要对大量图形元素执行相同的操作，因此非常适合在 SIMD（单指令多数据）型并行构架上并行实现。其另一个特点是对每个元素执行的操作较为简单因此适合于细粒度并行。GPU 的设计正体现了图形处理的这些特点，现代 GPU 包含大量功能较为简单的处理单元和存储控制单元，通过在这些处理单元同时执行大量线程可以达到掩藏访存延迟的目的，这与 CPU 设计思想不同，CPU 核心只有一个具有复杂控制功能的处理单元，并用大缓存来掩藏访存延迟。图3.1所示为 CPU 和 GPU 的晶体管使用情况。由于图形渲染任务的并

图 3.1: GPU 和 CPU 的晶体管结构 来源:NVIDIA

行性，设计 GPU 时可以简单的通过增加晶体管数量来扩充其计算性能和存储带宽。图3.2和3.3所示为 CPU 和 GPU 计算速度和存储发展情况比较。可以看出，目前主流的 GPU 比同期 CPU 计算速度和存储带宽都高出很多倍。

2. GPU 构架

图形显卡在硬件上主要由 GPU 芯片、显存以及 PCI 总线组成。目前市场上主流的显卡主要由两家公司设计 ——NVIDIA 和 AMD，二者在结构和功能部件名称上存在一些差别。由于本文研究的是 NVIDIA 公司的 CUDA 技术构架下的 GPGPU 编程，所以后文对 GPU 结构和功能的介绍均以 NVIDIA 公司的显卡为例，具体的，我们将使用 NVIDIA 公司的高端显卡 ——Tesla C1060。

GPU 芯片上集成了大量处理单元，是显卡的计算功能部件。Tesla C1060 显卡使用的是一个 GT200 GPU 核心。核心中的处理单元按两层结构组织 ——多处理器（SM）和流处理器（SP），其中一个 SM 包含多个 SP。GT200 核心上有 30 个 SM，

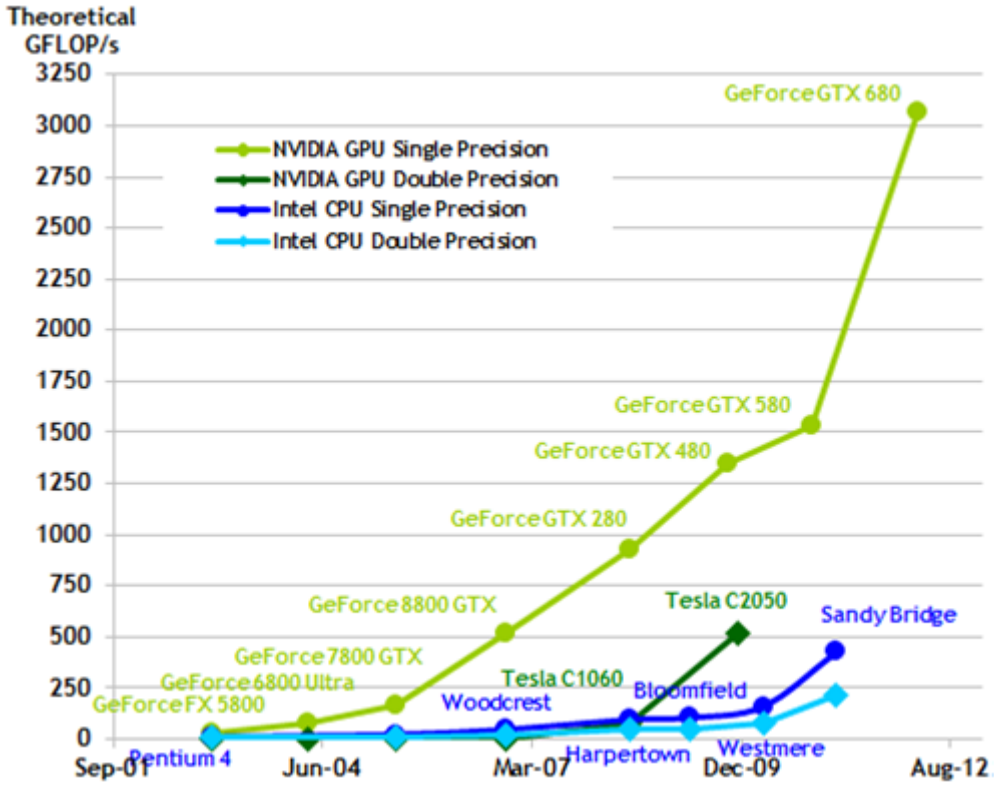


图 3.2: GPU 和 CPU 的每秒浮点运算次数 来源:NVIDIA

每个 SM 包含有 8 个 SP，因此一共是 240 个 SP。每个 SP 有一个单精度计算单元，每个 SM 拥有一个双精度计算单元。SP 工作频率为 1296MHz，SP 每个时钟周期能执行 3 次单精度浮点数操作，每个双精度计算单元每个时钟周期能执行 2 次双精度浮点数操作。因此 GT200 核心理论单精度计算能力为 $240 \times 1296(\text{MHz}) \times 3(\text{flops}) \approx 933\text{Gflops}$, 双精度计算能力为 $30 \times 1296(\text{MHz}) \times 2(\text{flops}) \approx 78\text{Gflops}$。GT200 核心的双精度计算能力是单精度的 1/12，因此其计算性能的优势主要体现在单精度浮点数计算上。在 NVIDIA 公司后来推出的 Fermi 构架和最新推出的 Kepler 构架的 GPU 上，双精度浮点数计算速度大为改善，可以达单精度的 1/2。值得指出的是，虽然 SP 是 GPU 核心实际的计算单元, 但由于 SP 没有独立的寄存器和取指、调度单元，所以 SP 并不是一个独立的类似 CPU 的执行单元。

显存作用类似于 CPU 构架中的内存，是 GPU 主要存储器。Tesla C1060 显存大小为 4GB。多处理器通过 8 个存储控制器与显存相连，每个存储控制器位宽为 64bit，所以显存总位宽是 $8 \times 64\text{bit} = 512\text{bit}$。存储单元工作频率为 2214MHz，理论外部存储带宽为 $512 \times 2214\text{Mhz} \approx 142\text{GB/s}$。和 CPU 相同，由于存在较大的访存延迟，GPU 的外部存储带宽往往是程序性能的一个限制因素。

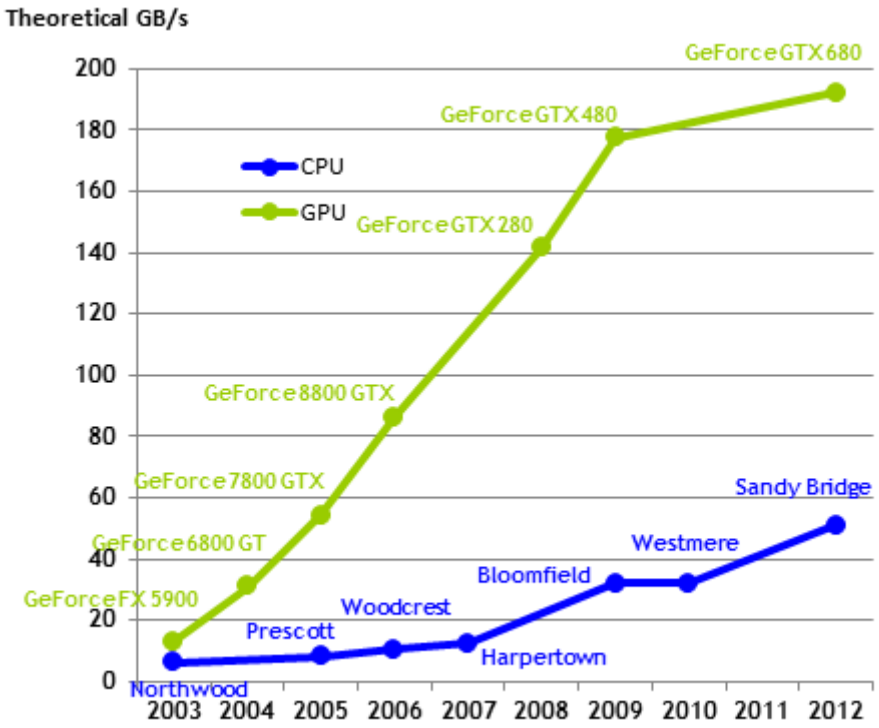


图 3.3: GPU 和 CPU 的的存储带宽 来源:NVIDIA

PCI-E 总线是显存与主板芯片组的数据通道，总线带宽根据所采用 PCI-E 规范版本及通道数量不同而有所差异。通常能提供上下各数 GB/s 的带宽。总线带宽比显存外部带宽带宽低得多，容易成为程序性能的瓶颈，所通常要尽量减少 CPU 主存与 GPU 显存之间的频繁地数据传输。

3. CUDA 编程模型

CUDA 技术的提出，大大简化了通用 GPU 计算程序的开发。在 CUDA 技术提出之前，要想利用 GPU 做通用计算，编程人员必须掌握计算机图形 API 和图形硬件细节，并将自己的应用程序按照计算机图形学里面的概念来描述和编码，这种方式繁琐而不自然。2007 年，NVIDIA 推出了 CUDA 技术，与传统 GPGPU 不同，CUDA 将图形学 API 隐藏起来，直接 C 语言基础上扩展，它允许编程者自己编写一种类似普通 C 函数的 *Kernel* 函数在 SP 上执行，从而利用 GPU 的大规模并行计算能力。另外它还提供线程执行控制、存储器管理等功能并以运行时库函数的方式供 CPU 上执行的普通函数调用。下面介绍 CUDA 技术中的几个关键概念。

Host 与 Device 一个完整的 CUDA 程序包含在 CPU 执行的部分和在 GPU 上执

行的部分。CPU 端被称作 *Host* 端，GPU 端被称为 *Device* 端。Host 端运行的代码是标准的 C 语言（或 Fortran、Python 等其它语言），而 Device 端运行的代码是由扩展的 C 语言编写的，并被封装在一个在声明时由`__global__` 修饰的函数中，这种函数被称作 *Kernel*。当 Host 端程序运行到要调用 Kernel 时，编译好的 Kernel 代码被装载到 Device 端，开始由 GPU 核心执行。图3.4示例为 CUDA 程序的执行过程。

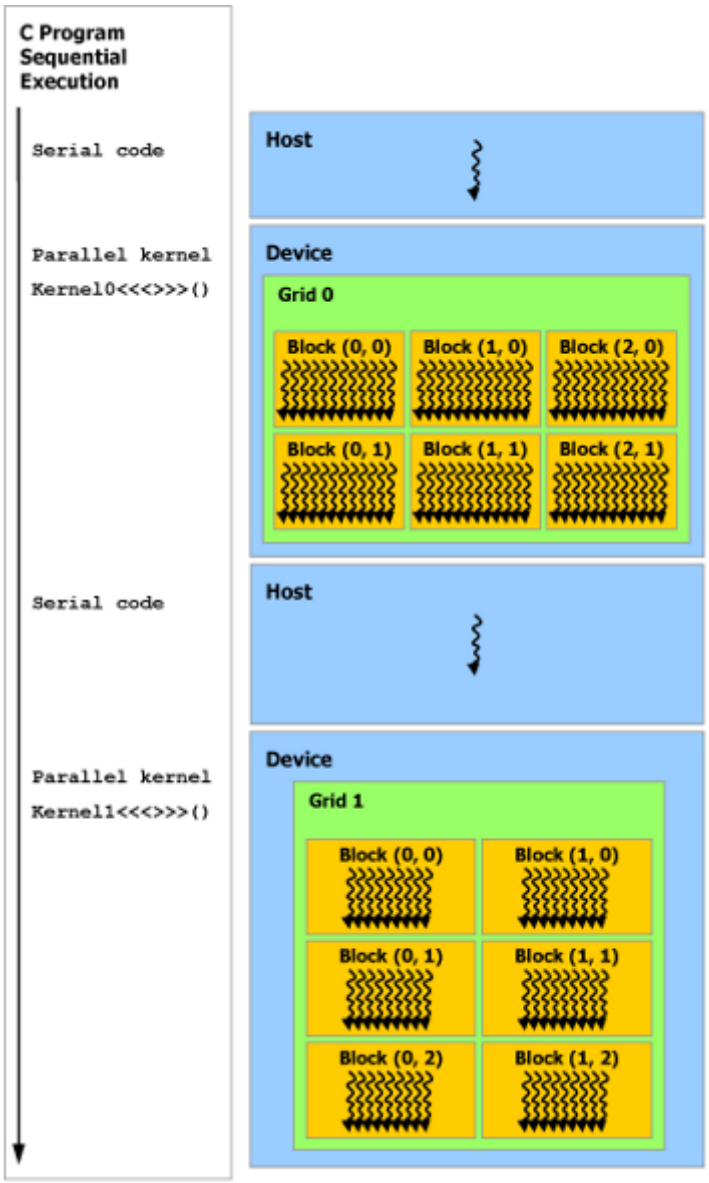


图 3.4: CUDA 程序执行流程 来源:NVIDIA

线程 线程（thred）是 CUDA 的核心概念，它是与流处理器 SP 相对应的逻辑上的

概念，可以理解为 GPU 代码的一条逻辑执行主线。在已经充分掩藏了访存延迟的情况下，GPU 核心里面的每个 SP 都同时运行着一个线程。这种线程的粒度相对于 CPU 上的线程而言非常小，并且通常没有复杂的逻辑判断分支执行路径。

Block 与 Grid 运行 Kernel 的线程按层次被组织为线程块 **Block** 与线程块网格 **Grid**。通常整个求解域被映射到一个 Grid 上，并被划分为细小的单元，Grid 中的每个线程负责处理一个单元。Grid 中的线程又按划分为 Block，划分方式可以是一维、二维或者三维，相应的 Block 也可以是一维、二维或三维的。Grid 和 Block 的大小（即 Grid 各维度上 Block 的数量和 Block 各维度上线程数量）分别由`dim3` 类型的参数指定。这种变量有三个整数分量，分别指定 Grid 或 Block 在各维度上元素的个数（二维可以看做特殊的三维情况，及第三个分量为 1）。在调用 Kernel 时指定这些参数，这些参数并以内建全局变量的形式出现在 Kernel 函数中，分别为`gridDim` 和`blockDim`。在 Kernel 中还有两个`uint3` 类型的局部变量 —— `blockIdx` 和`threadIdx`，它们也有三个整数分量。`blockIdx` 记录当前线程所属 Block 在 Grid 中的位置，`threadIdx` 记录当前线程在其 Block 中的位置。每个线程可以根据这些变量确定其在 grid 中的位置，进而确定其所处理的数据在显存中的位置。位于同一个 Block 中的线程共享部分高速存储器 ——共享内存。CUDA 还提供`__syncthreads()` 函数来实现块内线程同步。通过共享内存和线程同步，CUDA 提供了一种简单的线程间通信能力。Block 中的线程又隐式的被划分为若干个 *wrap*，wrap 的概念在 CUDA 源程序中没有体现，但线程执行和访存时是按 wrap 为单位进行的。

代码 1 提供一个完整的 CUDA 程序实例 ——矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的求和运算，上述各种概念在其中已有体现。

和 Host 端类似，在 Device 端也存在各种不同的存储器类型。它们的大小、访问开销各不相同。合理的使用用它们是提高 CUDA 程序性能的关键，这里我们介绍常用的几种存储器类型。

全局存储器 全局存储器（global Memroy）位于显存并占据了显存的绝大多数空间，功能类似于 Host 段的主存，用来存贮主要的计算数据，如 LBM 中的分布函数、宏观量等。全局存储器通过调用 `cudaMalloc` 来分配，通过调用 `cudaMemcpy` 实现全局内存与 Host 内存之间的数据传输。全局内存的访问开销相对其他类型的存储器非常大，通常有几百个时钟周期延迟。同一个 wrap 中的线程访问全局存储器时，若访存地址满足对齐要求并且相邻线程访问的地址连续，会被合并为一次访问请求，

只需要进行一次数据传输，否则会及进行多次数据传输，显存带宽会大幅下降，在早期的 GPU 上，这一效应尤为明显。

寄存器 寄存器 (register) 是 GPU 片上的一种高速存储器，Kernel 函数声明的临时变量通常就是这种类型。在 GT200 核心中，每个 SM 寄存器大小为 16KB，SM 同时为多个线程维护一份寄存器，所以每个线程分配到的寄存器数量非常有限。若 Kernel 中寄存器用量超过其上限，数据就会被存储到局部存储器，后者访存开销相当大。

共享内存 共享存储器（shaared memory）是位于 GPU 芯片上的一种高速存储器，几乎没有访存延迟，它可以被同一个 Block 中的所有线程访问。共享内存数量较少，是一种稀缺资源，如 GT200 核心中每个 SM 的的共享内存数量为 16KB。在 Kernel 函数中通过 `__shared__` 修饰声明。

常数存储器 常数存储器（constant memory）是一种只读的存储器，通常用来存储少量计算参数。常数存储器也较少，GT200 核心只有 64KB 大小。

局部存储器 局部存储器位于显存，当寄存器用完时会自动被使用，访问开销与全局存储器相同。

4. CUDA 程序优化的基本原则

为充分利用 GPU 提供的大量处理器和存储带宽，设计和优化程序时要遵守一些基本原则，否则 GPU 的实际性能会大为降低，下面列出一些常见的要求

- Block 大小为应该是 32 的整数倍；
- 控制 Kernel 汇总共享内存和寄存器用量，保证 SM 上至少能容纳 2 个活动的 Block 和 6 个活动的 wrap，否则不能充分掩藏全局存储器访问延迟；
- 合理选择 Block 尺寸，此要求通常与上一条要求相互矛盾，实际设计时要根据计算实验来确定最优 Block 尺寸；
- 尽量保证访存满足地址对齐和连续的要求。

5. 小结

本章前半部分介绍了 GPU 的基本构架和相应结构的功能，比较了其与 CPU 构架的差异，并揭示了 GPU 之所以具有大规模并行能力的底层原因。本章后半部分

主要主要介绍了 CUDA 技术的基本概念，涉及到了 CUDA 技术的线程层次结构和存储器类型，理解这些基本概念以及各种存储器的差异和使用要求是设计出高效率 CUDA 程序的前提。最后我们列出了 CUDA 程序优化的基本原则，我们将在后面章节根据这些原则的指导设计和优化 LBM 模拟的 CUDA 程序。

```
#define N 1024
#define BX 16
#define BY 16
float A_h[N][N], B_h[N][N],  C_h[N][N];//Host端存数组
float *A_d, *B_d, *C_d;//Device 端数据指针
//Kernel函数定义
__global__ void
 matrixAdd(float A[N][N], float B[N][N], float C[N][N])
{
        int x = blockId.x*blockDim.x + threadIdx.x;
        int y = blockId.y*blockDim.y + threadIdx.y;
        C[y][x] = A[y][x] + B[y][x];
}
int main()
{
        ...
        //Host端初始化A_h, B_h
        ...
        //Device分配显存存储 Device 端数据
        cudaMalloc（(void）**A_d, sizeof(float)*N*N);
        cudaMalloc（(void）**B_d, sizeof(float)*N*N);
        cudaMalloc（(void）**C_d, sizeof(float)*N*N);
        //拷贝 Host 端数据到 Device 端
        cudaMemcpy(A_d, A_h, cudaMemcpyHostToDevice);
        cudaMemcpy(B_d, B_h, cudaMemcpyHostToDevice);
        dim3 GRID(BX, BY);//定义 Grid 尺寸
        dim3 BLOCK(N/BX, N/NY);定义 Block 尺寸
        //调用Kernel
        matrixAdd<<<GRID, BLOCK>>>(A_d, B_d, C_d);
        //将结算结果C拷回 Host 端
        cudaMemcpy(C_h, C_d, cudaMemcpyDeviceToHost);
        ...
        //Host端的后续操作
        ...
        return 0;
}
```

代码 1: 一个实现矩阵相加 (C=A+B) 的 CUDA 程序

四 LBM 在 GPU 上实现的基本方法

本章介绍了目前文献中常见的 LBM 在 GPU 上的实现方法，着重讨论了影响程序性能的因素，并指出了其基本优化策略。随后我们分别编制了模拟顶盖驱动二维方腔流和外力驱动三维方截面直管道流的 GPU 程序，在验证了程序正确性后我们对其运行速度做了测试和分析。

1. LBM 在 GPU 上实现的基本方法

由于 GPU 和 CPU 的构架存在巨大差异，很多适用于 CPU 的优化技术在 GPU 上可能根本就行不通，甚至适得其反。例如 CPU 的 LB 程序中通常要避免数组的第一个维度大小为 2 的幂次，这是考虑到 CPU 缓存效率的缘故 [36]，而在 GPU 上情况却恰恰相反。下面将从 LBM 算法的特点结合 GPU 构架特点来分析在 GPU 上实现高效 LBM 计算应该采取的策略。

在 LBM 计算过程中，每个格点速度分布函数方向很多, 但其碰撞步执行的计算却很简单，在迁移步甚至没有计算而只是简单的数据加载与存储，因此 LBM 的计算访存比较大，在已经充分优化的情况下，程序的性能瓶颈还是在访存带宽，并且在 CPU 上和 GPU 上都是如此 [12]。考虑到这一特点,LBM 在 GPU 上的实现及优化目标为最大限度发挥 GPU 的带宽优势，即尽量提高访存效率。下面将从数据布局和存储器使用方面介绍 LBM 在 GPU 上实现的关键之处。

(1) 优化数据布局

在 CPU 上实现 LBM 时，速度分布函数（PDF）在内存中有两种最为常见的布局方式 ——结构数组（Arrays of Structure, AoS）和数组结构（Structure of Arrays，SoA）[36]。前者是指每个格点的 q 个 PDF 在内存中是连续排布的，而后者是指不同格点同一方向的 PDF 在内存中是连续排布的。如果用 C 语言的数组记法表示，对于前者分布函数声明为`f[NZ][NY][NX][Q]`, 而对于后者则为`f[Q][NZ][NY][NX]`。在 GPU 实现时，为了保证相邻线程在同一时刻访问全局内存的地址是连续的，只能采用后一种形式，即 SoA。

(2) 减少全局内存访问

LBM 计算在逻辑上可以分为三个步骤即碰撞、迁移、更新宏观量，如果分开执行这些操作，则会多次从全局内存加载和存储数据。事实上利用 LBM 是一种显式方法这个特点，这三个逻辑步骤可以合并在一个 Kernel 中执行。通常在全局内存上开辟有两套 PDF 数组，如`F0` 和`F1`，在一个时间步，每个格点从`F0` 中加载自己的

PDF，随后计算宏观量，然后执行碰撞，并将新的 PDF 写入到 `F1` 中相邻格点对应的位置，在下一时间步执行相同操作，不过读取和写入位置置换，即从`F1` 中读取，碰撞后写入`F0`。注意在上述方法中迁移步已经掩藏在向周围点写入新 PDF 的过程中了。

(3) 利用共享内存辅助迁移过程

在迁移过程中每个格点不可避免地要从临近格点写入 PDF，如果临近格点的 X 坐标（这里假定按`f[Q][NZ][NY][NX]` 形式开辟数组）与自己不同，则写入过程肯定有一个`sizeof(type)` 的偏移，因而不能不满足内存对齐要求，实际显存带宽会大为下降。目前常见的做法是用共享内存辅助迁移。即格点首先将自己的 PDF 读到自己的共享内存中，完成碰撞操作后从共享内存将数据写回全局内存，穿出 Block 边界的 PDF 正好可以临时存在 Block 另一侧空出的位置，图4.1所示为向右迁移的情况。写会过程中从共享内存读的是相邻格点的数据，而写入全局内存时的是写到自己的位置。总体上看，读写全局内存时都满足内存地址对齐要求，理论存储效率可以达到 100%。

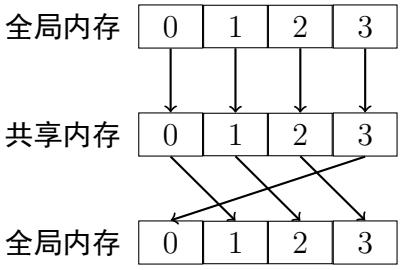


图 4.1: 利用共享内存辅助迁移

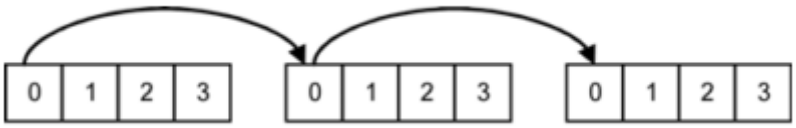


图 4.2: **LBExchange** 执行过程

由于在迁移过程中穿越 Block 边界的 PDF 没有被存到显存中正确的位置，所以在执行完碰撞步的 Kernel 后还要紧跟着调用另外一个 Kernel 来完成迁移步，这里我们按原文献命名为 **LBExchange**。它的作用就是交换这些没有被迁移到正确位置的 PDF。在三维情况，由于存储空间限制，不可能所有方向网格数都很大，如不会都超过 128 或 256，这时正好可以让一个相应大小的一维 Block 处理网格数最小的

方向上的“一条”格点，这时可以不用执行 **LBExchange** Kernel，而只需做一般的边界处理。（3）节中的方截面直管道流动模拟就利用了这个特点。

2. 验证算例及性能测试

这一节简要列出利用上节所述方法实现的不同算例 GPU 程序的程序性能，并将计算结果与相关文献或解析解做了对比以验证程序的正确性。

（1） 计算环境及程序性能测量方法

本工作使用的计算环境一台服务器搭载 Tesla C1060 型 GPU 的服务器，GPU 标配显存 4GB，CPU 为 Intel Xeon X5550 8 核处理器，操作系统为 CentOS 5.9 64bit 版，CUDA 工具箱版本为 3.2。本文中所有 CPU 程序均为串行程序，使用 icc V10.1 编译器编译，编译选项加上了`-fast`。

衡量 LBM 程序速度的标准单位是 MLUPS（Million Lattice Updates Per-second），意为每秒百万节点更新数。按照这个单位，程序运行速度其定义为

$$\text{Speed} = \frac{N_{\text{total}} \times T_{\text{LB}}}{10^6 \times t_{\text{comput}}} (\text{MLUPS}) \tag{4.1}$$

其中 N_{total} 表示流场中格点总数，T_{LB} 表示总的 LB 演化步数，t_{comput} 表示计算 T_{LB} 步所消耗的时间。考虑多孔介质流动情况时，还有一个只考虑流体格点的衡量单位 —— MFLUPS，其定义与 MLUPS 相比只是分子的 N_{total} 换为流体格点总数。因为多孔介质中存在大量不参与计算的固体格点，所以这个单位才能真实反映计算速度。

这里我们指出在计算 GPU 程序相对于 CPU 程序加速比时，CPU 程序性能对加速比影响很大（作为分母），而实际 CPU 程序执行的速度在相当程度上取决于 CPU 性能和所使用的编译器及编译选项，所以笼统的指出 GPU 程序的加速比并能不准确放映 GPU 程序的加速效果。在后文的性能测试中我们主要以 MLUPS 和 MFLUPS 为单位衡量 GPU 程序的绝对速度。

（2） 二维方腔流

二维方腔流边界条件简单，但是随雷诺数增大，却可以产生复杂的涡结构，因此被广泛地用作验证程序正确性的标准算例 [37]。这里我们采用的方腔流几何结构如图4.3(a)所示，方腔四边长都为 1.0m，空腔上边界以恒定速度 $U = 0.1\text{m/s}$ 水平向右移动。我们对比了雷诺数为 400 时的 GPU 程序计算结果与 Ghia [37] 等人的用多重网格方法计算得出的结果，并将 GPU 程序和对应的 CPU 程序计算速度作了对比。

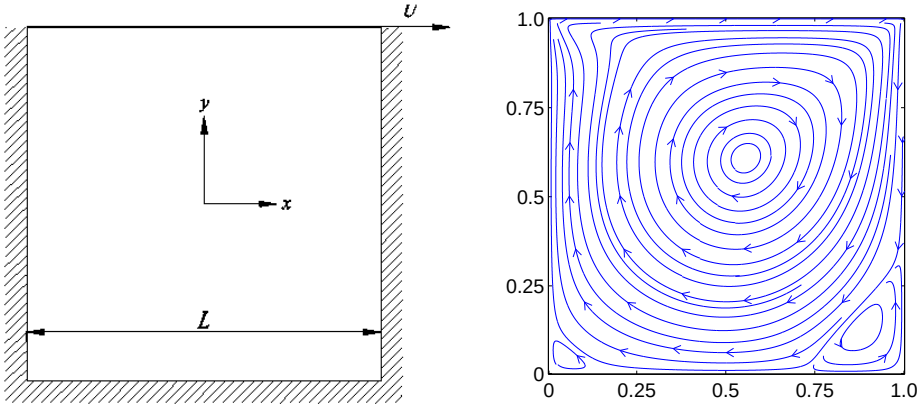
涡	一级涡		二级涡		三级涡	
位置	x_1	y_1	x_2	y_2	x_3	y_3
本文	0.5543	0.6070	0.8889	0.1235	0.0522	0.0475
Ghia	0.5547	0.6055	0.8906	0.1250	0.0508	0.0469

表 4.1: $Re = 400$ 时，各级涡中心的位置

我们采用的是标准 D2Q9 LBGK 模型，固壁采用半反弹格式，上边界采用非平衡外推格式，网格数为 256×256，在 GPU 上采用单精度计算，收敛标准为

$$\frac{\sqrt{\sum_{ij} |\boldsymbol{u}_{ij}^{n+1000} - \boldsymbol{u}_{ij}^{n}|^2}}{\sqrt{\sum_{ij} |\boldsymbol{u}_{ij}^{n}|^2}} \leq 1.0 \times 10^{-6} \tag{4.2}$$

表示相邻 1000 步直间速度场的相对误差小于 1.0×10^{-6}。图4.3(b)所示为流场稳定后的流线图，可以发现有三个涡结构分别出现在方腔中心和左、右下角。表 4.1列出了本文计算出的涡中心位置以及 Ghia [37] 的结果，可以看出本文结果和 Ghia 的结果相差很小。



(a) 二维方腔流示意图　　(b) $Re = 400$ 时稳定后流线图

图 4.3: 二维方腔流

我们发现 Block 的大小（本文采用的 Block 是一维的）和总体网格规模对计算速度都有影响，图4.4所示为不同 Block 大小和不同网格规模时的计算速度，可以

发现 GPU 最快速度快达到 924MLUPS，而与之相比，我们的 CPU 程序速度为 7.5MLUPS，可见 GPU 程序加速比达到了 123 倍。

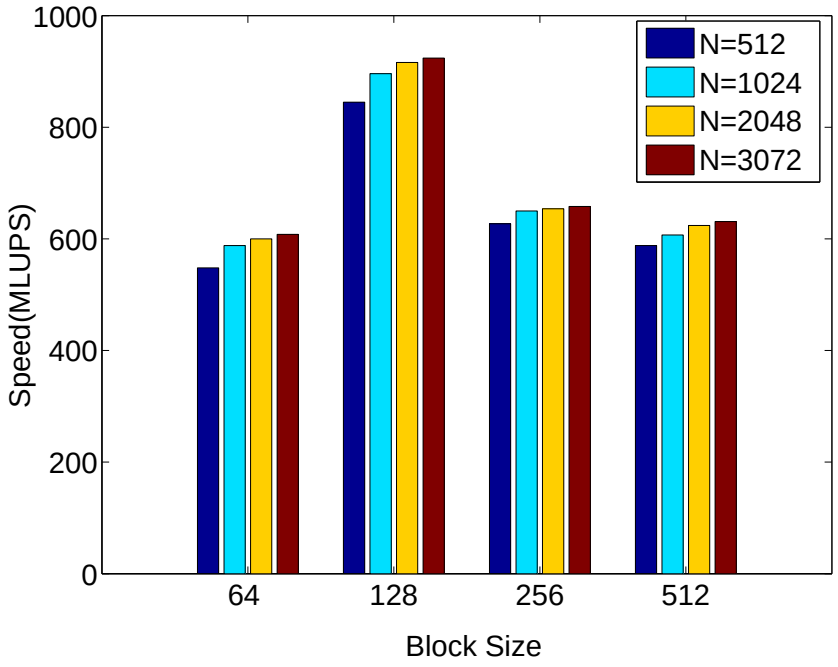


图 4.4: 网格规模和不同 Block 尺寸对计算速度的影响

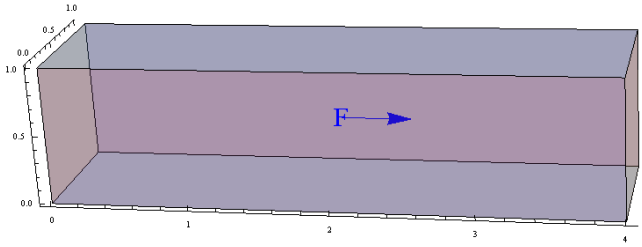
(3) 方形截面直管道流动

本小节将依然使用1.节中描述的方法模拟三维方形截面直管道流动，如图4.5(a)所示，这种流动类似于二维的泊肃叶流动，流体由沿管道方向的外力 $\boldsymbol{F}$ 驱动。由于流动沿管道方向具有周期性，所以管道长度不影响截面速度分布。截面流向速度分布只取决于外力大小 $\boldsymbol{F}$ 和管道宽度 a，其级数解析解为

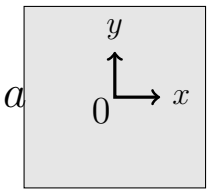
$$u(x,y) = U_0 \sum_{i=1}^{\infty} (-1)^i \left(1 - \frac{\cosh(\pi x i/a)}{\cosh(\pi i/2)}\right) \frac{\pi y i/a}{i^3} \tag{4.3}$$

其中 $U_0 = \frac{4aF}{\nu\pi^3}$。

数值模拟采用 D3Q15 标准 LBGK 模型，管壁边界采用非平衡态外推格式处理。参数 $U_0 = 0.001$, 收敛标准仍热使用式(4.2)。计算收敛后与解析解的相对误差为 0.14%。其中 $y = 0$ 处速度分布与解析解的对比见图4.6。GPU 程序中由于采用了 4.1 节中指出的方法，可以不用另外执行 **LBExchange** Kernel，因为流出 Block 的



(a) 几何结构



(b) 管道截面示意图

图 4.5: 外力驱动方截面直管道流

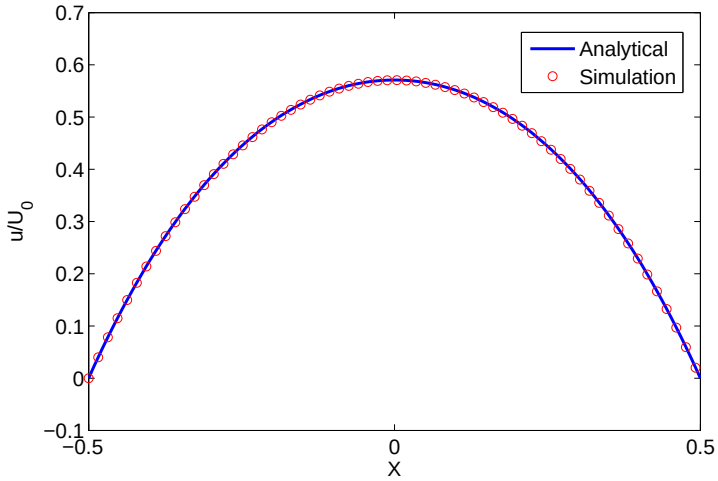


图 4.6: 截面上 $y = 0$ 处速度与解析解比较

PDF 又流回到了 Block 的另一侧，这正好符合周期性边界实现方式。固壁边界需要另外的 Kernel 来进行非平衡外推处理。我们对不同 Block 尺寸和不同网格规模下的计算速度做了测试，结果如图4.7所示。作为对比，CPU 程序在不同网格规模下的计算均为 4 ∼ 5MLUPS 左右。可见我们实现的 GPU 程序最快可以加速两个量级以上。

3. 小结

本章首先介绍了 LBM 在 GPU 上实现的基本方法及优化策略，然后利用按照这些方法实现的 GPU 程序模拟了二维方腔流和三维方形截面直管道流。模拟结果均与解析解或参考文献吻合良好，证明 GPU 程序正确的实现了 LBM 算法。之后又测试了 GPU 程序性能，并考察了影响计算速度的因素。两个算例的 GPU 程序的计算速度均为 CPU 版本的两个量级以上。下一章，我们将考虑流固边界较为复杂的渗

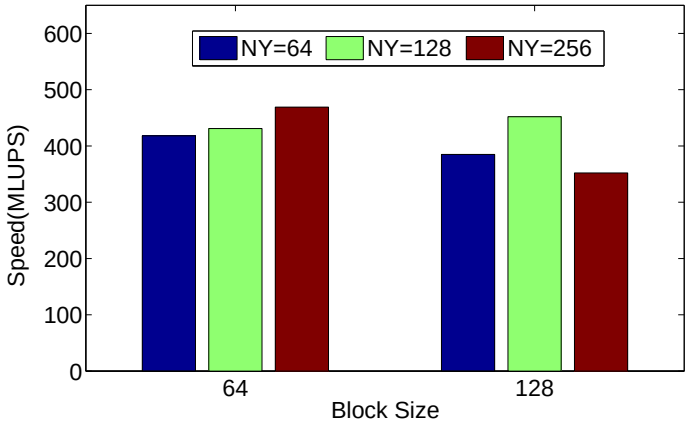


图 4.7: 不同 Block 尺寸和网格规模下的计算速度

流问题。

五 单相渗流 LB 模拟在 GPU 上的实现及优化

上一章中计算所使用的算例流场边界都比较规则、简单，流场内部格点都是流体格点，每个格点执行的操作都相同，线程中几乎没有逻辑判断分支语句。这在 GPU 实现时意味着同一个 Wrap 中的各个线程执行路径完全相同，SP 执行效率最高，因此用 LBM 模拟这种简单流场流动时能充分发挥 GPU 的并行效率。但在模拟实际问题时，流场结构通常并不规则，甚至非常复杂，如多孔介质。如何在 GPU 上高效地实现这中复杂边界 LBM 模拟依然是当前的一个研究热点。本章正是围绕这一问题探讨了从算法上考虑的各种优化途径，并以一个实际多空介质流动（渗流）算例验证了优化效果。

1. 对于多孔介质的一般处理方法

在多孔介质结构中，流体在相互连通的孔隙中流动，孔隙壁面是非规则的固体表面。将多孔介质在空间上离散后，计算区域被划分为流体格点和固体格点，如图5.1所示为二维多孔介质离散后的格点分布，图中黑色格点表示固体，白色格子表示孔隙（流体）。在程序实现时用一个整型数组`flag[NY][NX]` 来记录格点的类型，`flag[y][x] = 0` 表时格点`(y,x)` 为流体格点，等于 1 则为固体格点，演化时通过这个数组判断格点类型，若为固体格点，则不作任何处理。流固边界边界通常采用简单的标准反弹格式，处理时也需要判断周围格点类型。

上述处理方式虽然实现起来简单，但在 GPU 上实现时效率不会很高。原因有二，其一，处理每个格点时要周围格点类型，需要从全局内存加载周围格点 flag 数据，占用了显存带宽，并且访存时存在第四章中所述显存地址不对齐的情况；其二，确定周围格点类型时会引入判断分支语句，如果一个 wrap 中的线程对应的格点既有流体格点，又有固体格点，则会导致该 wrap 中的线程串行执行，执行效率将大为降低，在均匀的各向同性介质中，这种情况将尤为明显，因为几乎每个 wrap 中都产生了分支。上述方式除了在 GPU 上执行效率不高外，还存在另一个问题 —— 浪费存储空间，因为在开辟 PDF（速度分布函数）数组时，不参与计算的固体格点也占据了存储空间。并且多孔介质的孔隙率越低，无效格点所占比例越高，存储空间浪费越严重。

针对前一个问题，我们采用*位存储模式*结合*逻辑运算*方式解决，针对第二个问题，我们采*稀疏存储模式*解决。

另外，考虑到本文最终目的是要做多组分模拟，我们在这里并没有采用第四章中所描述的使用共享内存进行辅助迁移的方法，而是在碰撞之前从周围格点直接读

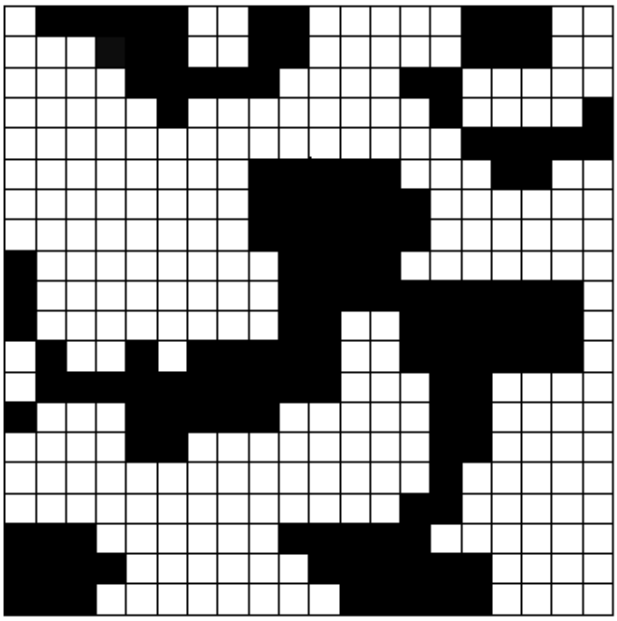


图 5.1: 二维多空介质示例

取 PDF。之所以这样做，是因为在多组分模拟时，每个格点需要迁移的 PDF 几乎翻倍，如果还是用共享内存辅助迁移，则 SM 的利用率会因共享内存数量限制而降低，进而影响程序性能。Obrecht 等人对这两种执行迁移步骤的方法做了详细对比，并指出第二种方法在模拟复杂问题如非等温情况时，会比第一种方法效率更高 [38]。

2. 优化技术一　位存储和逻辑运算

因为每个格点只能是流体格点或固体格点，所以理论上在 flag 中只用一个 bit 就可以反映格点类型。为了避免在演化过程中从加载周围格点的 flag 位，可以事先把这些信息存储在当前格点的 flag 中。按这种方法，对于 DnQq 模型，每个格点的 flag 要存储 $q$bit 的信息，其中 1bit 反映该格点本身类型，另外 $(q-1)$bit 反映周围格点类型。对于 D3Q19 模型，可以用一个 `int` 类型（32bit）来表示 flag，如图5.2所示，最低位表示当前格点本身类型，第 1 到 18 位分别表示第 1 到 18 个方向上相邻格点的类型（0 或 1）。利用这种位存储模式，解决了占用存储带宽的问题，接下来介绍利用逻辑运算来统一处理内部流体格点和固体边界处流体格点的方法。

通常在处理流场内部格点和边界流体格点，程序会进入不同的分支语句。这里以修正反弹格式为例，在这种反弹格式中，边界流体格点也需要执行碰撞，但它们与内部格点不同的是，在从周围格点读取 PDF 时，如果该在该方向上临近格点是

图 5.2: D3Q19 模型 falg 的位存储模式示意

固体点，则从其本身的反方向读取 PDF。这里笔者提出一种能同一处理内部格点与边界格点的方法，这种方法可以保证同一个 wrap 中各个线程执行的指令完全相同，指令吞吐量最高。代码2示例了如何使用逻辑运算实现这种方法。

如果周围格点时流体点则 `readShift=1`，表示读取位置有偏移，否则从自己读，`readShift=0`，没有偏移。如果读取位置有偏移，则从 `i` 方向的反方向（`i_reverse` 读（此时 `i_reverse != i`）。因为我们在定义离散速度方向时，保持了第 `i` 个与第 `i+1` 个方向正好相反（`i` 为偶数），所以可以通过 `i` 的奇偶性判断其反方向。

值得注意的是上述优话指令流的方法也可以运用于其他需要避免 `if` 语句的地方，具有一定的通用性。

3. 优化技术二　稀疏存储模式

稀疏存储模式是指只存储流体格点 PDF，目的是为了节省内存或显存（GPU 程序），通常 GPU 显存并不大（几 GByte），因此在 GPU 程序中，这一方法更为重要。稀疏存储模式中，首先遍历 `flag` 数组，统计出流体格点个数，然后动态分配一维线性内存/显存存储流体格点 PDF，流体格点的 PDF 按格点在遍历过程中出现的顺序插入该线性内存，如图5.3。由于流体格点分布不规则，所以流体格点之间的相对位置关系丢失了，在迁移时不能确定周围格点 PDF 的内存/显存地址。解决这个问题的办法是引入一个辅助数组来保存每个格点各个方向上相邻格点在的位置, 我们称这个数组为 `node_map`。在 GPU 上实现时，`node_map` 按 `node_map[Q-1][num_F]` 形式存储数据，即依然保证访问地址连续。稀疏模式还有一个附加优点，因为在 `node_map` 中直接存储了 *每个周围格点的 PDF* 在 PDF 数组中的索引，这样可以在构造 `node_map` 时将反弹格点与周期边界格点的迁移源地址或目的地址地直接存在里面，而在演化过程中不需要显示处理这些特殊格点，因而 Kernel 函数大为简化。

引入辅助数组之后，虽然每个格点要存储的信息增多了，但在孔隙率低于某个阈值 (当 PDF 采用双精度时，该阈值约为为 0.7) 时总的内存/显存用量仍低于全存

```
...
//由于程序中使用了动态内存存储PDF
//所以用GID宏来做数组索引转换
//(z, y, x)格点（当前点）在一维数组中的位置
k = GID(z,y,x);
flag_k = flag_h[k]; //当前点的flag
//当前点为流体格点时才进行下面处理
if(!(flag_k & 1))
{
  ...
  for(i=1;i<Q;i++)  //读取各个方向上前一格点的PDF
  {
    //是否从本身读取PDF
    readShift = (flag_k & (1<<i)) == 0;
    //读取PDF的方向
    i_reverse = i + (readShift^1)*((i&1)*2 - 1);
    xp = x - readShift*e[i_reverse][0];
    yp = y - readShift*e[i_reverse][1];
    zp = z - readShift*e[i_reverse][2];
    kp = GID(zp, yp, xp);
    //正式加载PDF
    F[i] = fIn[i_reverse*size + kp];
    ...
  }
}
```

代码 2: 优化技术一

储模式 [39]。

值得指出的是，稀疏存储模式在 CPU 上和 GPU 上运用都可以节省存储空间，但在 CPU 上相对于全存储模式并不会提高计算速度，而在 GPU 上却可以显著提高。这是 GPU 和 CPU 构架差异造成的，当在 GPU 上运用全存储模式时，不参与计算的无效格点占据了 SP 执行时间，而采用稀疏存储模式时，则不存在这一问题。本章后面分析 GPU 程序性能将会验证这个现象。

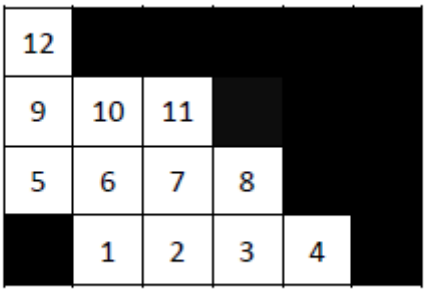


图 5.3: 稀疏存储模式

4. 算例一　BCC 结构渗透率的测定

（1） 问题描述

单相流体在外力或压差驱动下流过多孔介质时，其平均流速（达西速度）可由达西定律给出

$$u_d = -\frac{k}{\rho\nu}(\nabla p_z + \rho g) \tag{5.1}$$

其中 k 为多孔介质的渗透率，其具体值只有多孔介质结构本身确定，与其中所通过的流体性质无关。本算例中使用的多孔介质模型 BCC 结构的渗透率具有近似解析解，我们将其与数值模拟得出的渗透率对比来验证程序正确性。

BCC 结构是指体心立方结构（Body Centered Cubic），它是一种理想多空介质模型，其结构如图5.4所示，其中球体表示固体区域。因为球体排布具有空间周期性，所以计算区域只取其一个基本单元，如图5.5(a)所示，其二维投影如图5.5(b)。

BCC 结构参数为球体半径 r 和立方体单元边长 L。固体区域所占体积比例与孔隙率分别是

$$c = \frac{8\pi r^3}{L^3}, \quad \epsilon = 1 - c \tag{5.2}$$

其中 c 的最大值为 $c_{max} = \frac{\sqrt{3}\pi}{8}$，通常用一个取值范围为 0 到 1 的无量纲形状因子 $\chi = \left(\frac{c}{c_{max}}\right)^{1/3}$ 来表示球体填充的致密程度，$\chi = 1$ 时填充最为致密的情况，图5.5(a)就是这种情况。BCC 结构的绝对渗透率近似解析解形式为

$$k = \frac{2\epsilon r^2}{9d^*(1-\epsilon)} \tag{5.3}$$

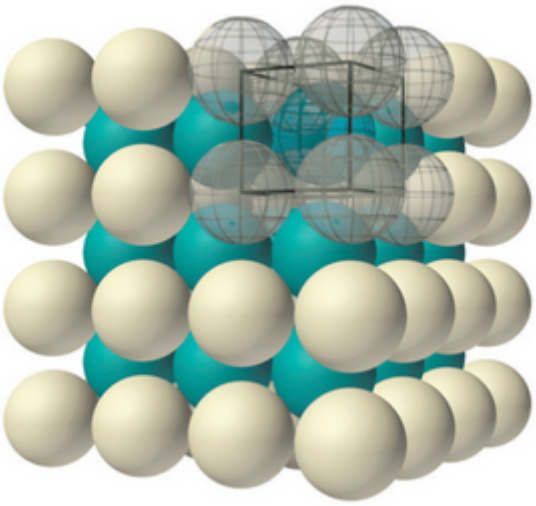
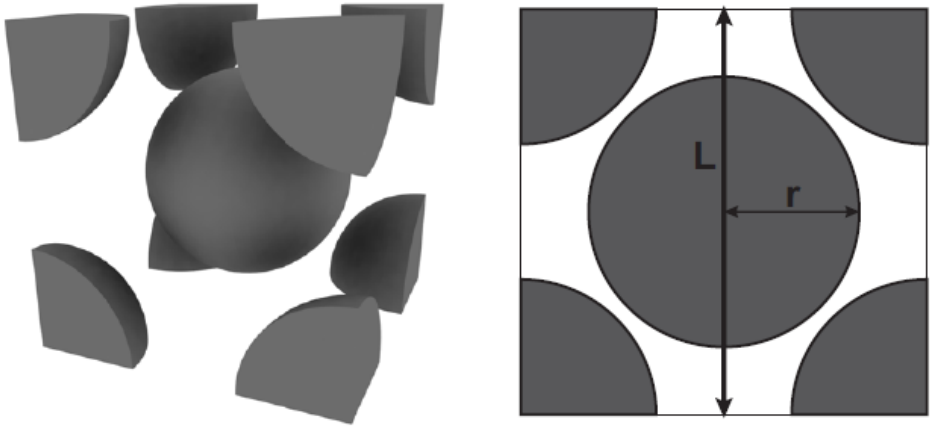


图 5.4: BCC 结构示意图



(a) BCC 结构基本单元　　(b) BCC 结构二维投影

图 5.5: BCC 结构及参数

其中 d^* 是单个球体所受流体拖拽力的无量纲阻力，在 $Re < 1$ 的情况下其级数近似解析解 [40] 为

$$d^* = \sum_{n=0}^{30} \alpha_n \chi^n \tag{5.4}$$

其中多项式系数 α_n 为常数，具体值见参考文献 [40]。

（2） 数值方法

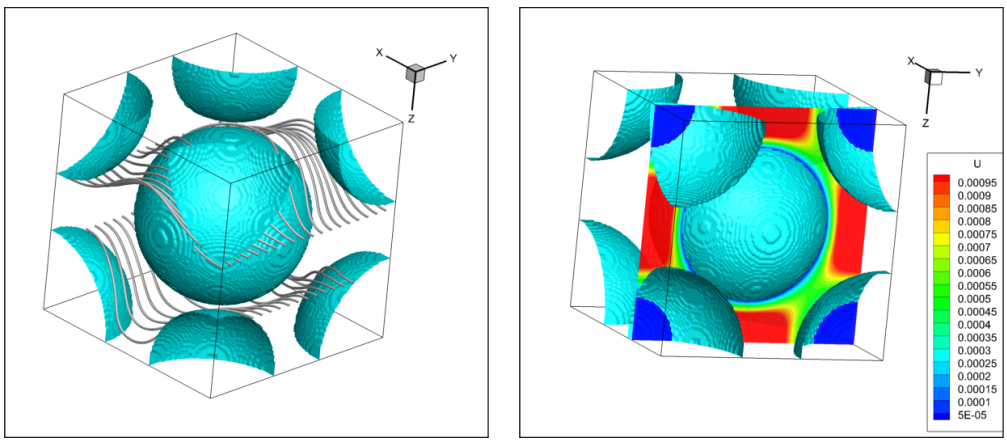
我们通过给定外力 g，在流场稳定后测出达西速度 u_d，然后根据式(5.1)得出渗透率数值解。流场的所有外边界均为周期性边界，流固边界采用修正反弹格式，外力离散格式采用 Luo 的二阶矩模型 [41]。模拟过程中保证 $Re < 1$，网格规模均为 128^3，立方体尺寸为 128m。计算收敛标准为计算所得渗透率相邻 1000 步的相对变化小于 10^{-5}。如无特别指明，所有工况在 GPU 上均用单精度计算。

Pan 等人指出用 LBGK 模型计算多空介质介质流动时，存在计算所得渗透率时与粘性相关的非物理现象，而 MRT 模型可以显著改善这一问题 [30]。这里我们也采用 D3Q19 LBGK 和 D3Q19 MRT 模型并验证了这一现象。在程序实现时我们分别采用了前两节所介绍的优化技术。

（3） 计算结果

MRT 与 LBGK

我们分别用 LBGK 和 MRT 模型计算了不同孔隙率和不同流体粘性对应的绝对渗透率。图5.6(a) 和图5.6(b)所示为 $\chi = 0.8, \tau = 0.8$ 时用 LBGK 计算得到的流线图和沿外力方向的速度分布图。



(a) 流线图　　(b) 沿外力方向的速度分量

图 5.6: LBGK 模型 $\chi = 0.8, \tau = 0.8$ 计算结果

图5.7和图5.8分别为不同孔隙率和不同流体粘性时计算得出的渗透率，可以看出使用 LBGK 模型计算出来的渗透率会随粘性不同而变化明显，尤其是在孔隙率较高时，而使用 MRT 模型计算出来的渗透率与粘性几乎无关。我们的计算得到的渗透

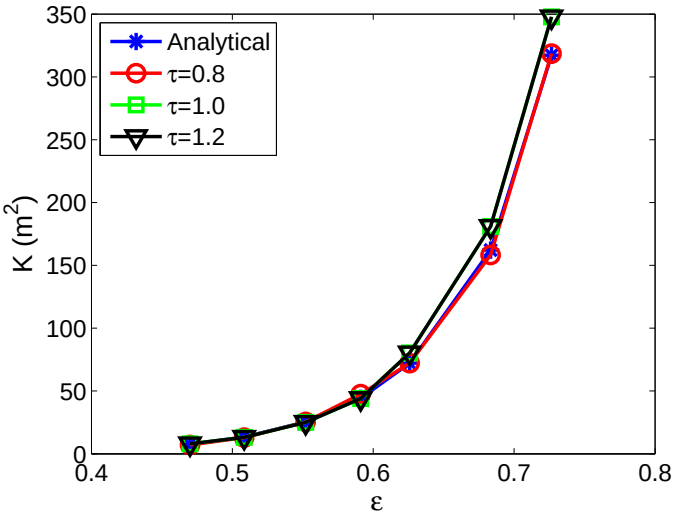


图 5.7: 使用 LBGK 模型时不同粘性和孔隙率下渗透率数值解

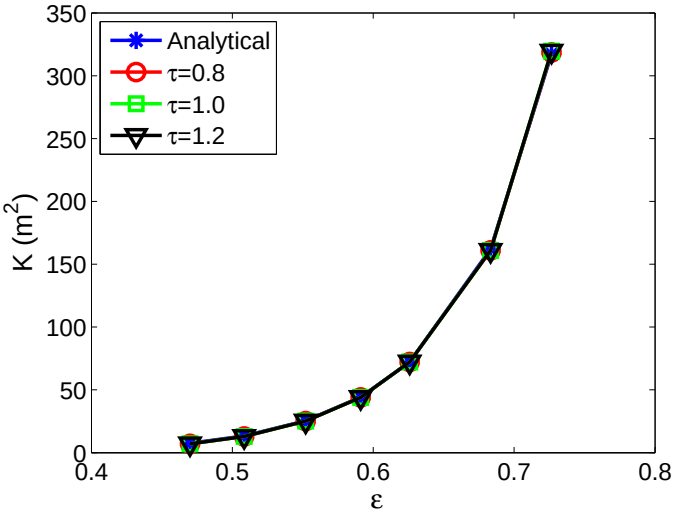


图 5.8: 使用 MRT 模型时不同粘性和孔隙率下渗透率数值解

率与解析解吻合良好，并成功验证了文献 [30] 的结论，这证明我们的程序的实现方法正确。

计算精度的影响

在目前的 GPU 构架上，通常在单精度计算要比双精度计算快得多（见 3.2 节），但单精度数只有 7 位有效数字，相比较而言双精度数有 16 位有效数字。对于大多数工程应用计算单精度已经足够，但对计算要求十分精确的场合必须使用双精度。本小节考察本节算例中单精度和双精度对计算结果的影响。

我们分别使用单精度和双精度计算了上小节中 $\chi = 0.6, \tau = 1.2$ 的工况，所用程序为 MRT 程序。我们比较了用两种精度计算时渗透率 k 的收敛过程，其结果如图5.9所示。可以明显发现二者有一定区别，单精度计算演化 39000 步之后收敛，收敛值为 315.45，而双精度计算演化 24000 步就已经收敛，收敛值为 319.74。单精度相对双精度相差 1.34%，这说明单精度计算只能要求不太高的工程问题计算。

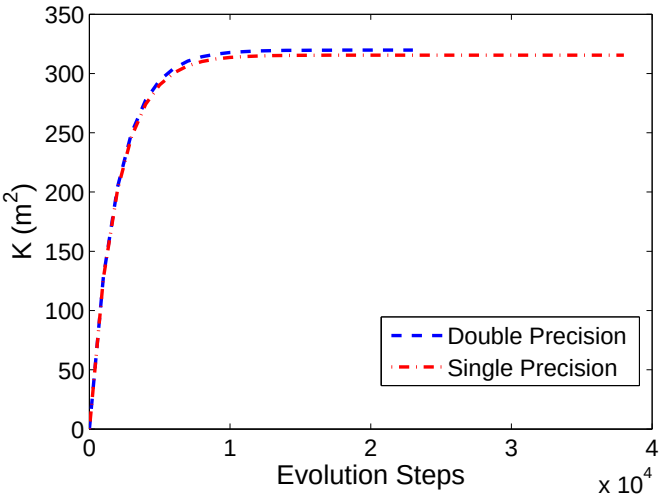


图 5.9: 使用单精度和双精度计算时渗透率 k 的收敛过程

(4) 计算速度分析

在这一小节中，我们分析各种影响计算速度的因素，包括所使用的优化算法（见 5.2、5.3 节）、模型（MRT 和 LBGK）、计算精度、Block 尺寸。在下面的比较中，如无特别指明，均采用单精度计算，Block 尺寸为 64。另外，我们在 4.2 节中指出衡量多孔介质模拟计算速度时有两种单位 —— MLUPS 和 MFLUPS，后者只统计流体格点更新，即考虑了孔隙率对计算量的影响，二者换算关系为

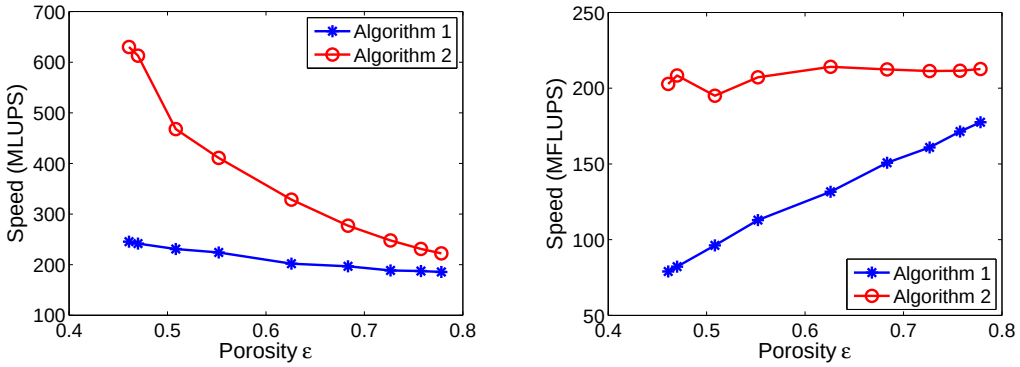
$$Speed(\text{MFLUPS}) = \epsilon \times Speed(\text{MLUPS}) \tag{5.5}$$

在本小节的图中，我们会明确标出所使用的衡量单位。为叙述方便我们在后文中称 5.2 节中中算法为算法一，而称采用 5.3 节中介绍的稀疏存储模式的算法为算法二。

为方便对比，我们还编写了同样计算这个问题的 CPU 版本程序，采用 intel icc 编译器编译，并加上了 `-fast` 编译选项（这是目前我们所能利用的最快的编译方法），在各种工况下观测到的最高流体格点更新速率为 3.61MFLUPS。

算法的影响

图5.10(a)所示为采用两种算法在不同孔隙率下的计算速度，单位为 MLUPS，图5.10(b)反映相同内容，不过单位为 MFLUPS。对比图5.10(a)中的红线和蓝线可以



(a) 衡量单位：MLUPS；　　(b) 衡量单位：MFLUPS

图 5.10: 算法对计算速度的影响

发现，随着孔隙率减小，采用算法二格点更新速度大幅提高，而采用算法一则速度只有很小的提高，并且在所考差的孔隙率范围内算法二格点计算速度均优于算法一。对比5.10(a)中的红线和蓝线可以发现，随着孔隙率减小，算法一的流体格点更新速度几乎线性减小而算法二则没有显著变化，我们分析这是因为采用算法一时，同一个 wrap 中的每个无效格点（固体）虽然不参与计算，但也占据了 SP 执行时间。

MRT 和 LBGK 模型的影响

MRT 模型相对于 BGK 模型只在计算量上有所增加，而访存模式即访存大小都没有变化。通常文献中反映其计算量约增加了 15% ∼ 20% [28]。这里我们分析了采用算法一时两者的计算速度的差别，其结果见图5.11，图中所示为在不同孔隙率下采用 MRT 模型计算速度相对 LBGK 模型的下降量，可以发现相对下降量都小于 10%。

单、双精度的影响

我们采用算法一结合 MRT 模型比较了单、双精度的计算速度，结果如图5.12所示，可以发现双精度计算速度为单精度的 34% 左右。在我们所使用的 Tesla C1060 GPU 上，双精度计算速度理论上是单精度的 1/12，而此处 34% 与之相差较大，我们分析这是因为 LB 的计算性能瓶颈在显存带宽而不在处理器执行速度，双精度计算相对单精度带宽需求只增大一倍，考虑到其他因素如访存效率影响，此处 43% 在

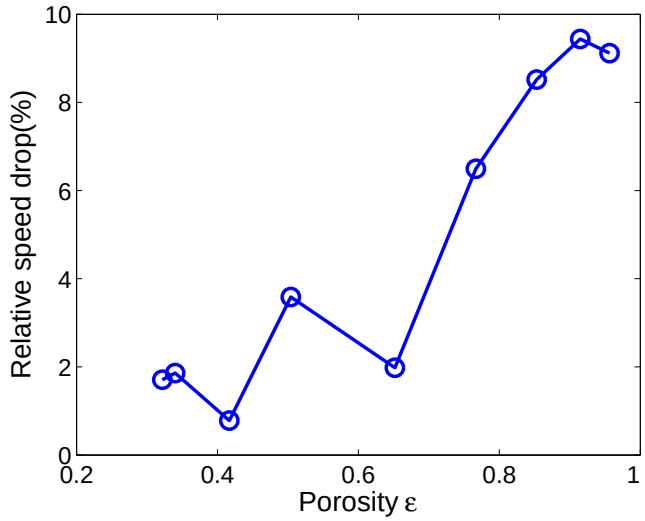


图 5.11: MRT 模型相对 LBGK 模型计算速度的相对下降量

预期范围内。

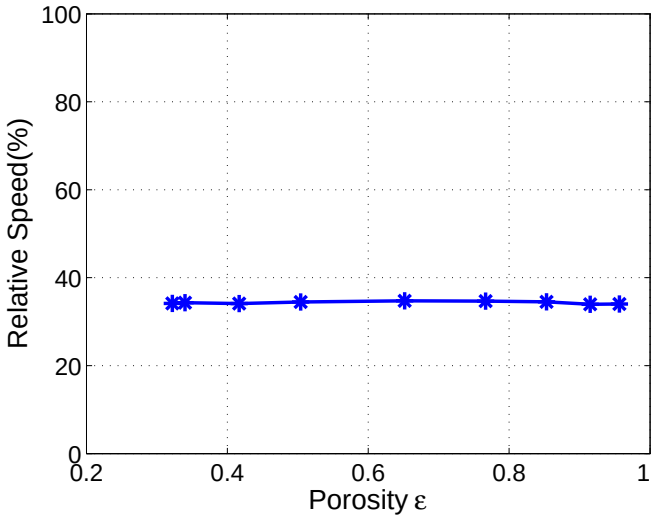


图 5.12: 双精度相对单精度的计算速度

Block 尺寸的影响

上面的速度测试 Block 大小都是 64，这里我们分别测试了 Block 大小为 96 和 128 时的计算速度。采用的的模型和算法分别是 MRT 模型与算法二。结果见图5.13，可以发现 Block 大小为 64 时速度最高，而为 96 和 128 时速度较小且差别不大，这是定型的计算效率受到寄存器或共享内存数量限制的现象。考虑到我们 Kernel 只使用了寄存器，而没有使用共享内存，所以此处 Block 尺寸增大时，性能下降的现象

应该是性能受到寄存器数量的限制引起的。

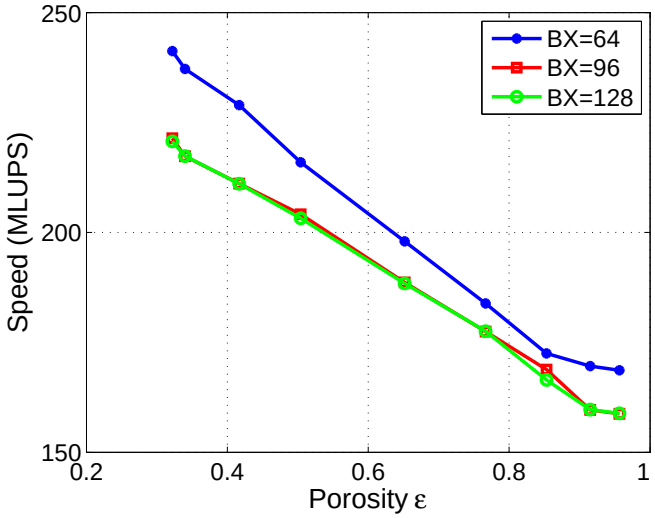


图 5.13: 不同 Block 大小时的计算速度

5. 小结

本章中我们首先介绍了 LBM 处理多孔介质的一般方法，并指出了两种针对在 GPU 上实现多孔介质流动模拟的优化技术 ——利用位操作结合逻辑运算减小访存量和优化指令流以及利用稀疏存储模式减小显存用量和提高程序计算速度。随后我们分别运用这两个优化技术编制了 LBGK 模型和 MRT 模型的 GPU 程序计算了 BCC 多孔介质模型的渗透率，与解析解对比发现吻合良好，证明了我们 GPU 程序实现正确。最后我们详细分析了上述 GPU 程序的计算速度，详细考察了各种影响计算速度的因素，在最优情况，我们的 GPU 程序计算速度可达 212MFLUPS 和 630MLUPS。超过 CPU 串行程序 50 倍。在下一章中我们将本章介绍的优化算法运用于更复杂的多孔介质内的多相流 LBM 模拟。

六 多相渗流模拟在 GPU 上的实现

多孔介质内的多相流动是一类常见的复杂流动现象，在化石能源开采，化工过程，新能源等应用广泛。目前 LBM 是模拟这类流动现象流行的数值方法，这得益于 LBM 在处理复杂流固边界和刻画多相多组分之间相互作用的独特优势（见第2章）。但与上一章介绍的多孔介质内单相流动一样，在孔隙尺度下直接运用 LBM 模拟多相渗流计算量通常相当大，一般需要利用并行计算加速。利用 GPU 加速 LBM 孔隙尺度多相流计算目前在文献中报道较少，这也是本文工作的一个重点。本章讨论基于 2.4 节中多组分/多相模型的 GPU 实现。我们用两个算例验证了多相流 GPU 程序实现的正确性，随后用其该模拟了二维和三维多孔介质中的两相流动，最后测试了其计算速度。

1. 两组分 LBM 的 GPU 实现

两组分 LB 模时，每个格点同时有两套 PDF 在演化，并且要考虑两组分间的相互作用对 PDF 变化的贡献。就数据访问模式而言，每个格点除访问本身和周围格点的 PDF 外，还要访问其宏观量。在单组份情况各个格点计算平衡态时只需要知道其本身的宏观量，即具有最理想的空间局部性，所以其计算过程可以融合在计算迁移的 Kernel 中，也不需要为宏观量开辟存储空间，但在两组分情况时，每个格点要访问周围格点的宏观量，若在碰撞迁移的 Kernel 中计算相邻格点的宏观量显然进行大量的重复计算，因此我们采取的策略是为每个组分的宏观量 ρ_k、$\boldsymbol{u}_k$ 开辟全局内存，并另外增加一个 Kernel——`LBUpdateMacros`，它在碰撞迁移的 Kernel 执行完后统计每个组分的宏观量。

2. 程序验证

（1） 静止气泡测试

静止气泡测试可验证 Laplace 定律，是两相流中标准的模型验证算例之一。Laplace 定律给出了两相界面间压差 Δp 和界面表面张力 γ 之间的关系

$$\Delta p = \sigma \frac{\gamma}{r} \tag{6.1}$$

对于二维情况 $\sigma = 1$，三维情况 $\sigma = 2$。对于球形界面 r 为球体半径，对于圆柱形界面 r 为圆柱半径。

在进行静止气泡测试时，在计算区域中心球形区域内放置一种组分（即所谓的“气泡”），球形区域以外放置另一种组分，待流场稳定后，压力场和气泡半径不再变化，如图6.1 所示。初始化流场时放置不同半径的气泡，稳定后得到不同的气泡半径 r_i 和界面压差 Δp_i。按照 Laplace 定律，Δp_i 和 $1/r_i$ 应该成正比。

这里指出，虽然我们编制的是 D3Q19 的 GPU 程序，但是为减少计算量，在实际计算时，X 方向只有一层格点，并且是周期边界，相当于计算的二维情况，后面的几个算例也是这种情况。

初始化时一个圆形状的组分被放在在 YZ 平面中心，平面四边均为周期边界，如图6.1所示。

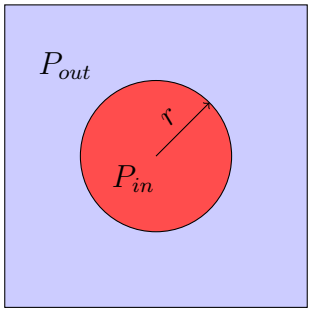


图 6.1: 气泡示意图

测试时两种流体粘性都为 1/6，密度都为 1，XY 平面格点数为 128×128，组分间相互作用系数 $g_{k\bar{k}} = 0.2$。测试结果如图6.2，可以看出计算结果与 Laplace 定律吻合良好。

（2） 层状两相流

层状两相流如图6.3所示，在两块水平放置的平板间，两相流体在沿 X 方向的外力驱动下一起向前运动。其中非润湿相处于中间层，厚度为 $2a$，而润湿相则贴近板壁，两板相距 $2L$。两相流体受到的外力可以不同，分别为 F_w 和 F_n，下标 n 和 w 表示润湿相（wetting phase）和非润湿相（no wetting phase）。两相流体 X 方向速度在沿 Y 方向分布的解析解为 [34]

$$u(y) = \frac{F_n}{2\nu_n\rho_n}(a^2 - y^2) + \frac{F_w}{2\nu_w\rho_w}(L^2 - a^2), \quad 0 \leqslant |y| \leqslant a \tag{6.2a}$$

$$u(y) = \frac{F_w}{2\nu_w\rho_w}(L^2 - y^2), \quad a \leqslant |y| \leqslant L \tag{6.2b}$$

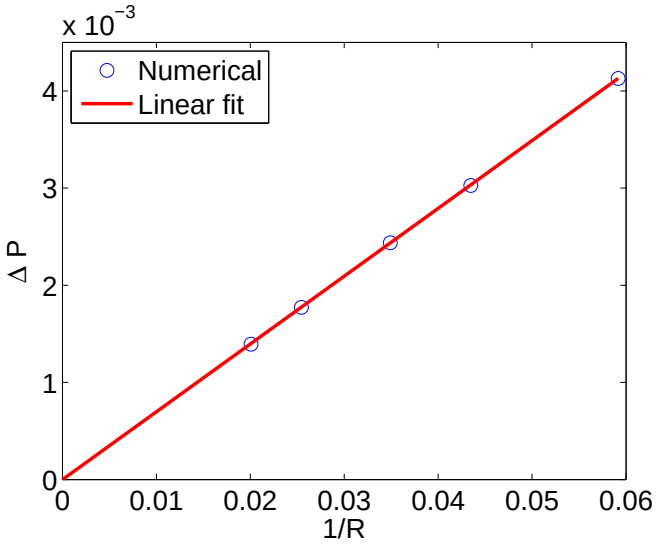


图 6.2: 气泡测试

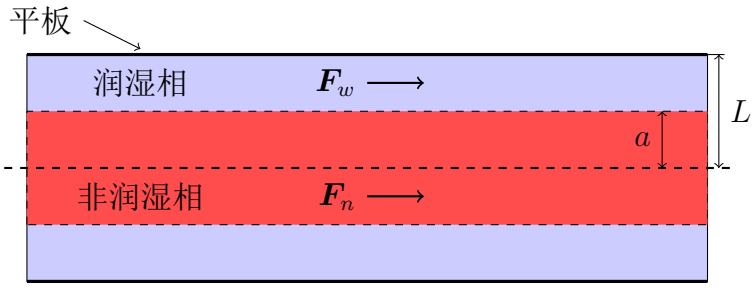


图 6.3: 层状两相流示意图

算例中设置的计算参数为 $\rho_n = \rho_w = 1.0$，$\nu_w = 10/6, \nu_n = 1/6$，$F_n = F_w = 1.0^{-6}$，组分间作用系数 $g_{k\bar{k}} = 0.27$，Y 方向网格个数为 256。计算结果与解析解的对比如图6.4所示，可以发现二者吻合良好。

（3） 液滴测试

为验证润湿性边界条件处理的正确性，我们进行了液滴测试，并获得了组分固体表面间相互作用强度与接触角的关系。组分与固体表面的接触角反应了两种组分对固体表面的相对润湿性。在 Shan-Chen 模型（或我们所使用的改进模型）中，接触角大小主要取决于两种流体组分（组分 g 与 f）与固体表面（s）间相互作用强度系数 G_{gs}、G_{gs} 以及组分间相互作用强度系数 G_{fg}。另外，G_{fg} 大小影响相界面厚度和数值稳定性，我们发现 $G_{fg} = 0.2$ 时比较理想，所以接下来的测试中取 $G_{fg} = 0.2$。

计算开始时，在一个长 800 高 200 的二维长管道的下底板放置一个半径为 50 半圆形组分 f，其他区域分布着另一组分 g，管道左右边界是周期边界，上下边界是固

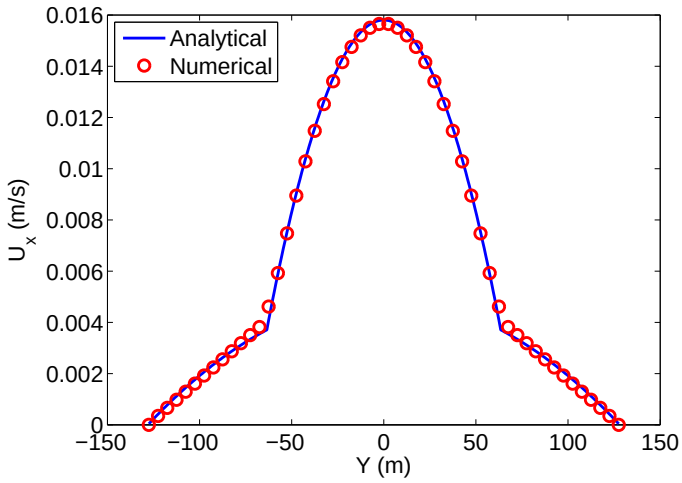


图 6.4: 沿 Y 方向的速度分布

壁，示意图见图6.5。

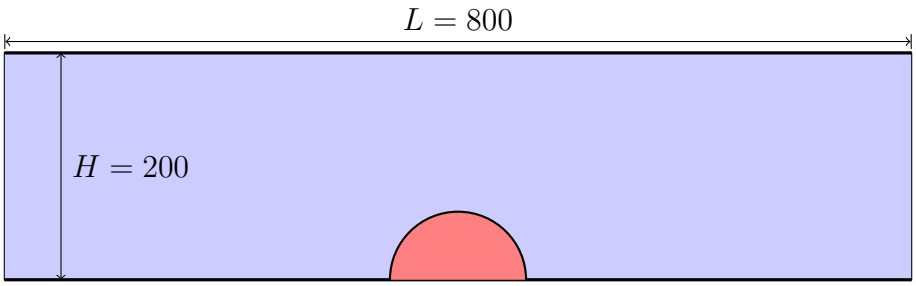


图 6.5: 液滴示意图

通过调整 G_{fs}、G_{gs} 的取值，液滴会呈现不同接触角，计算结果见图 6.6(a)至图6.6(f)。每幅图只画出了液滴附近的区域。

3. 多相渗流模拟

在上一节中的验证算例中，流固边界相对简单，我们在这一节中计算了实际多孔介质中的二维和三维两相渗流。

(1) 二维多相渗流模拟

计算过程使用的二维多孔介质如图6.7所示，图中黑色区域为固体，白色部分为孔隙，前处理时将其离散为 802×478 的个格子大小。初始化时，在孔隙中随机布满两种纯组分，即每个孔隙格点以相同的概率被设为组分 f 或 g。两种组分粘性为 $\nu_f = \nu_g = 1/6$，密度为 $\rho_f = \rho_g = 1$。流体在向右的大小为 $F = 10^{-4}$ 的外力驱动

下运动。组分间相互作用强度为 $G_{gf} = 0.15$, 组分与固壁间的相互作用强度分别为 $G_{fs} = 0.03, G_{gs} = -0.03$，根据上一节液滴测试的结果，在这组参数下，组分 f 形成的相是非润湿相，而组分 g 形成的相是润湿相，并且非润湿相接触角接近 180^o。图6.8(a)至6.8(e)所示为不同演化步时两相的分布，其中红色部分为润湿相（组分 g），蓝色部分为非润湿相（组分 f）。可以发现在流动过程中润湿相包围着固壁，而非润湿相则被润湿相包围，这是多相渗流基本的现象。

（2） 三维维多相渗流模拟

这一节中我们模拟了一种人工生成的多孔介质内的两相流，该多孔介质由大小和位置随机分布的球体堆积而成，其孔隙结构如图6.9(a)所示，其孔隙率大小为 0.7。各个面均为周期性边界条件，初始化时孔隙内各格点按相等概率随机给定组分 f 或组分 g。流体受沿 X 方向的外力驱动。演化 100000 步时，流场内部两相分布如图6.9(b)所示，其中蓝色部分为固体区域，红色部分为非润湿相，绿色部分为润湿相。

4. 性能测试

（1） 性能测试

性能预测

我们根据上一章中单组分流动稀疏存储算法 GPU 程序性能来预测本章多组分模拟 GPU 程序性能，二者在计算上的区别有如下几点:

- 两组分程序中每个格点存储两套 PDF，并且碰撞前需要访问相邻格点宏观量，访存量增加超过 1 倍；
- 两组分程序碰撞前需要计算组分间相互作用，计算量增大；
- 两组分程序中多了一个 `LBUpdateMacros` Kernel 函数；
- 两组分程序需特殊处理流固边界（润湿性边界），引入线程判断和分支；

考虑到 LB 的 GPU 计算性能瓶颈在访存带宽，所以上述几个因素中第一个因素对性能影响最大，即两组分相对单组份性能至少下降一半。另外我们运用 CUDA 工具箱中的 Visual Profiler 分析发现 `LBUpdateMacros` Kernel 函数占整个演化过程计算时间的 22% 左右。综合上述因素我们预测两组分相对单组份计算速度下降 70% 左右。上一章中采用稀疏存储算法的单组分计算 GPU 程序的流体格点更新速率为

210MFLUPS 左右（单精度，见图5.10(b)），因此我们预测本章的两组分计算程序速度大概为 60MLUPS。

网格数的影响

我们分别测试了二维和三维流场在不同网格数下的计算速度，流场中不含固体格点（即 Kernel 中没有判断分支），二维和三维计算采用的是同一个程序，二维情况只算一层流体格点（见（1）小节解释）。采用单精度计算时，速度测试结果如图6.10(a)和6.10(b)所示。可以发现二维和三维情况计算速度均为 70MLUPS 左右，三维情况最高速度可达 80MLUPS。

流固边界的影响

因为 Kernel 函数中处理流固边界润湿性时引入了分支判断语句，对 GPU 程序执行效率会有影响。我们测试在不同复杂程度的流场时的计算速度，所用三种流场如图6.11(a)至6.11(c)所示。图6.11(a)为最简单的情形，没有固体格点；图6.11(b)所示为上一节中使用的随机球体多孔介质模型，其流场较为复杂；图6.11(c)为随机生成的方块填充多孔介质结构，其流场结构最为复杂。

三种情况网格数均为 $64 \times 64 \times 128$，速度测试结果如图6.12所示。可以发现流场结构越复杂，计算速度越慢，最复杂的情况相对最简单的情况计算速度下降 33%。

5. 小结

本章介绍了两相渗流 LB 模拟在 GPU 上的实现。针对多组分 LB 模型在 GPU 上实现时的难点我们重新设计了演化的 Kernel 函数，并增加了一个 Kernel 单独计算宏观量。我们首先通过气泡测试、层状两相流、接触角测试验证了我们程序实现的正确性，随后分别模拟了二维和三维的两相渗流，计算结果显示的流动现象与实际情况相符。最后进行了程序性能测试，并分析了影响计算速度的因素，发现我们的两组分 GPU 程序的最高计算速度可达 80MFLUPS，而我们计算相同问题的 CPU 版本程序速度最快为 1.96MFLUPS，因此 GPU 加速比超过 40。

(a) $G_{fs} = -0.005, G_{gs} = 0.005$

(b) $G_{fs} = 0.0, G_{gs} = 0.0$

(c) $G_{fs} = -0.01, G_{gs} = 0.01$

(d) $G_{fs} = 0.01, G_{gs} = -0.01$

(e) $G_{fs} = -0.02, G_{gs} = 0.02$

(f) $G_{fs} = 0.02, G_{gs} = -0.02$

(g) $G_{fs} = -0.03, G_{gs} = 0.03$

(h) $G_{fs} = 0.03, G_{gs} = -0.03$

图 6.6: 接触角测试结果

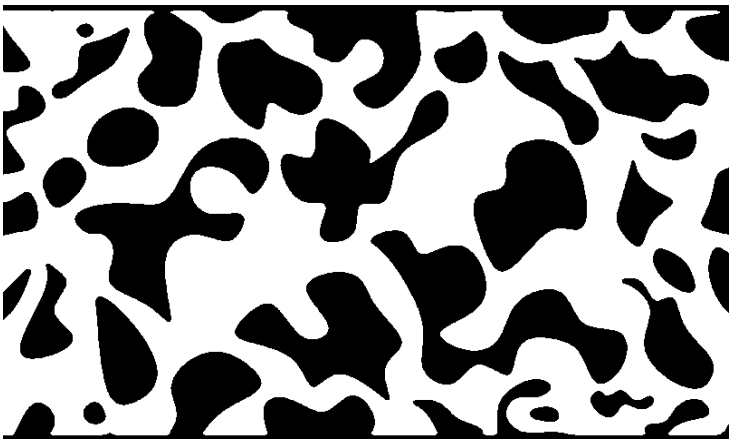
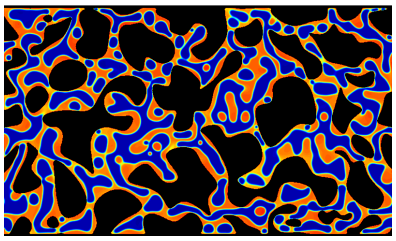
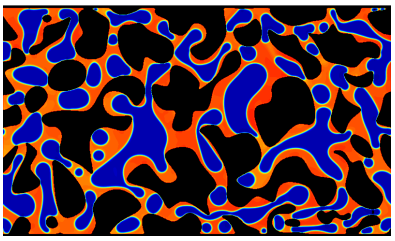


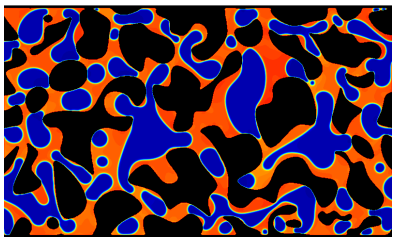
图 6.7: 二维多孔介质结构



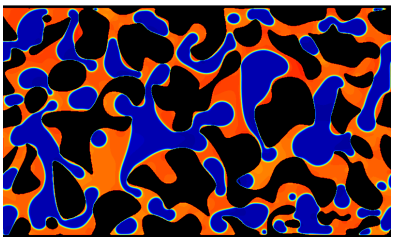
(a) 第 1000 步



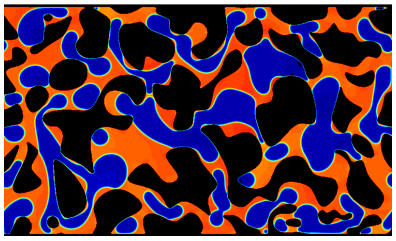
(b) 第 5000 步



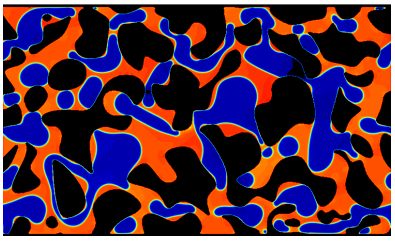
(c) 第 10000 步



(d) 第 20000 步

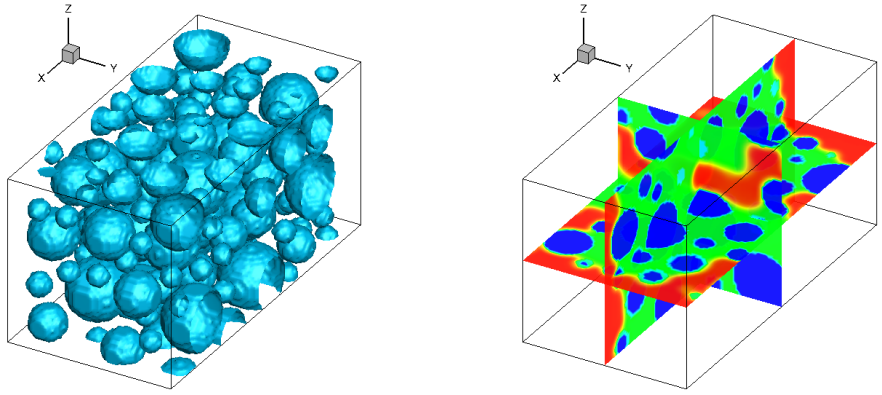


(e) 第 30000 步



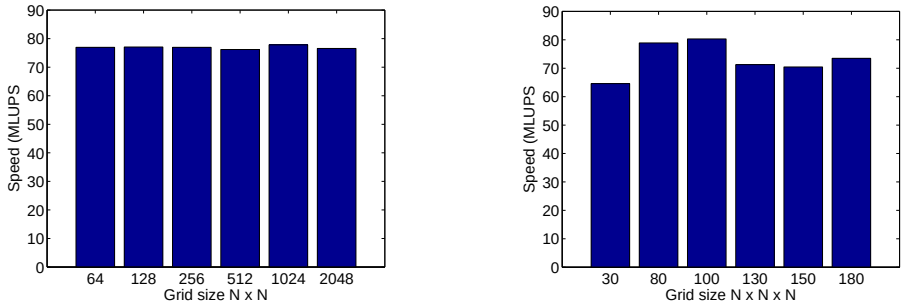
(f) 第 40000 步

图 6.8: 二维多相渗流模拟



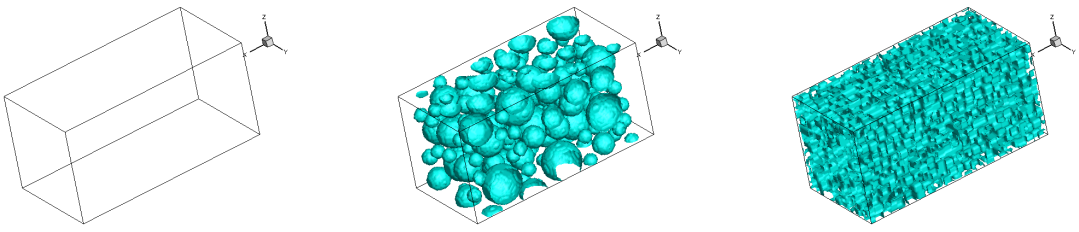
(a) 随机球体堆积多孔介质模型　　(b) 演化 100000 步时两相分布

图 6.9: 三维多孔介质两相流



(a) 二维　　(b) 三维

图 6.10: 网格数对计算速度的影响



(a) 没有固体格点　　(b) 随机球体　　(c) 随机方块

图 6.11: 不同复杂程度的流场结构

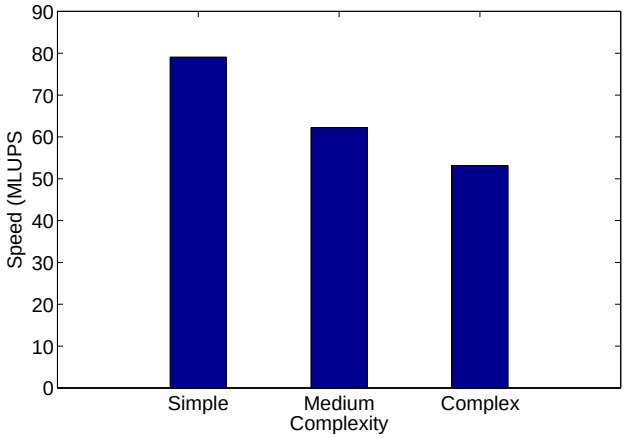


图 6.12: 流场复杂程度对计算速度的影响

七 全文总结

本文主要研究了复杂边界流动的格子 Boltzmann 模拟在 GPU 上的高性能并行实现。

我们首先按文献中常见的运用共享内存进行辅助迁移的方法实现了针对简单流动的 LBM 算法 GPU 程序，并用其模拟了二维顶盖驱动方腔流和三维外力驱动方截面直管道流。在验证了程序正确性后我们测试了其计算速度。我们发现在简单流场情况下，利用该方法可以达到相对于 CPU 程序两个量级的加速比。

考虑到 GPU 搭载的共享内存十分有限，在实现多组分 LBM 模型的时候共享内存大小容易成为性能瓶颈，我们在接下来实现的程序中并没有采用前面利用共享内存辅助迁移的方案，而是探究了其他优化方法，其一为利用位操作结合逻辑运算减少访存量并优化指令流，其二为利用稀疏存储模式大幅降低模拟多孔介质流动时的计算量。我们分别基于这两种优化方法编制了相应的 GPU 程序计算一种理想多孔介质模型（BCC 结构）的绝对渗透率，计算结果与解析解吻合良好，随后测试了两种优化方法的效果，发现第二种算法具有较大优势，能大幅减少显存耗用和计算量，尤其是在多孔介质孔隙率较低的情况下。

本文另外一个工作是针对我们所使用的多组分 LBM 模型计算特点，提出了该模型在 GPU 上的高效实现方法，并基于这个方法开发了具有一定通用性的 GPU 并行计算程序。在利用该程序其模拟了基本的多相问题，验证了其正确性之后，我们用它模拟了真实多孔介质中的多相渗流。最后，我们还对其进行了性能测试和分析。

本研究工作还有一些问题值得进一步研究，如目前的计算程序只使用了单 GPU 计算，实际进行大规模三维并行 LBM 模拟时，单个 GPU 提供的显存空间有限，必须结合使用多线程技术或 OPENMP/MPI 并行利用多个 GPU 并行计算才能满足存储空间要求。另外我们实现的 GPU 程序的通用性有待进一步提高，如实现不同如入口边界条件的设置等等。笔者将在今后的工作中着手解决这些问题。

致　谢

本文的研究工作是在导师煤燃烧国家重点实验室郭照立教授的悉心指导下完成的。郭老师学识渊博，是 LB 领域的知名学者，他为人爽朗，治学严谨，对学生和工作极为认真负责，让本人受益匪浅。每周的例会上，郭老师的点拨总能让我找到方向，消除困惑。除了每周例会，在毕业设计过程中遇到困难时，郭老师总是及时跟我交流，给我指导，让我的毕业设计工作得以顺利完成。郭老师的很多话我都记忆犹新，在以后的研究中，他的严谨作风仍会继续带给我积极的影响。在此，谨向郭老师致以最衷心的感谢。同时还要感谢煤燃烧实验室 606 郭老师课题组的各位师兄师姐们，是他们营造了一个团结、积极、活跃的学术氛围。在本论文的完成过程中，笔者曾无数次请教了课题组的娄钦和刘高洁两位博士师姐、王亮和杨康两位博士师兄另外还有数学院的黄昌盛博士，在这里也对他们表示衷心感谢。

最后感谢我的家人对我一如既往的关心和付出。

参考文献

[1] Xiaoyi He and Gary D Doolen. Thermodynamic foundations of kinetic theory and lattice boltzmann models for multiphase flows. *Journal of Statistical Physics*, 107(1-2):309–328, 2002.

[2] Thomas Pohl, Frank Deserno, Nils Thurey, Ulrich Rude, Peter Lammers, Gerhard Wellein, and Thomas Zeiser. Performance evaluation of parallel large-scale lattice boltzmann applications on three supercomputing architectures. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 21. IEEE Computer Society, 2004.

[3] Sebastian Geller, Manfred Krafczyk, Jonas Tölke, Stefan Turek, and Jaroslav Hron. Benchmark computations based on lattice-boltzmann, finite element and finite volume methods for laminar flows. *Computers & Fluids*, 35(8):888–897, 2006.

[4] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*, 2010.

[5] Jonas Tölke and Manfred Krafczyk. Teraflop computing on a desktop pc with gpus for 3d cfd. *International Journal of Computational Fluid Dynamics*, 22(7):443–456, 2008.

[6] NVIDIA. Cuda. `http://cudazone.nvidia.cn/cuda-action-research-apps/`.

[7] Youquan Liu, Xuehui Liu, and Enhua Wu. Real-time 3d fluid simulation on gpu with complex obstacles. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 247–256. IEEE, 2004.

[8] Nelson S-H Chu and Chiew-Lan Tai. Moxi: real-time ink dispersion in absorbent paper. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 504–511. ACM, 2005.

[9] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. Gpu cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 47. IEEE Computer Society, 2004.

[10] Hongbin Zhu, Xuehui Liu, Youquan Liu, and Enhua Wu. Simulation of miscible binary mixtures based on lattice boltzmann method. *Computer Animation and Virtual Worlds*, 17(3-4):403–410, 2006.

[11] Xiaoming Wei, Ye Zhao, Zhe Fan, Wei Li, Feng Qiu, Suzanne Yoakum-Stover, and Arie E Kaufman. Lattice-based flow field modeling. *Visualization and Computer Graphics, IEEE Transactions on*, 10(6):719–729, 2004.

[12] Jonas Tölke. Implementation of a lattice boltzmann kernel using the compute unified device architecture developed by nvidia. *Computing and Visualization in Science*, 13(1):29–39, 2010.

[13] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. Multi-gpu implementation of the lattice boltzmann method. *Computers & Mathematics with Applications*, 2011.

[14] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. The thelma project: Multi-gpu implementation of the lattice boltzmann method. *International Journal of High Performance Computing Applications*, 25(3):295–303, 2011.

[15] QinGang Xiong, Bo Li, Ji Xu, XiaoJian Fang, XiaoWei Wang, LiMin Wang, XianFeng He, and Wei Ge. Efficient parallel implementation of the lattice boltzmann method on large clusters of graphic processing units. *Chinese Science Bulletin*, 57(7):707–715, 2012.

[16] Qingang Xiong, Bo Li, Guofeng Zhou, Xiaojian Fang, Ji Xu, Junwu Wang, Xianfeng He, Xiaowei Wang, Limin Wang, Wei Ge, et al. Large-scale dns of gas–solid flows on mole-8.5. *Chemical Engineering Science*, 71:422–430, 2012.

[17] Johannes Habich. Performance evaluation of numeric compute kernels on nvidia gpus. *Master's Thesis, University of Erlangen-Nurnberg*, 2008.

[18] 黄昌盛，张文欢，侯志敏，陈俊辉，李明晶， 何南忠，施保昌. 基于 cuda 的格子 boltzmann 方法: 算法设计与程序优化. *科学通报*, 56(28-29):2434, 2011.

[19] Xiaoguang Ren, Yuhua Tang, Guibin Wang, Tao Tang, and Xudong Fang. Optimization and implementation of lbm benchmark on multithreaded gpu. In *Data Storage and Data Engineering (DSDE), 2010 International Conference on*, pages 116–122. IEEE, 2010.

[20] J Myre, SDC Walsh, D Lilja, and MO Saar. Performance analysis of single-phase, multiphase, and multicomponent lattice-boltzmann fluid flow simulations on gpu clusters. *Concurrency and Computation: Practice and Experience*, 23(4):332–350, 2011.

[21] Eirik O AKSNES and Anne C ELSTER. Porous rock simulations and lattice boltzmann on gpus.

[22] Jonas Tölke, Chuck Baldwin, Yaoming Mu, Naum Derzhi, Qian Fang, Avrami Grader, and Jack Dvorkin. Computer simulations of fluid flow in sediment: From images to permeability. *The leading edge*, 29(1):68–74, 2010.

[23] Carlos Rosales. Multiphase lbm distributed over multiple gpus. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 1–7. IEEE, 2011.

[24] Prasanna R Redapangu, SP Vanka, and Kirti Chandra Sahu. Multiphase lattice boltzmann simulations of buoyancy-induced flow of two immiscible fluids with different viscosities. *European Journal of Mechanics-B/Fluids*, 34:105–114, 2012.

[25] J. Toelke, G. De Prisco, and Y. Mu. Lattice Boltzmann multi-phase simulations in porous media using Multiple GPUs. *AGU Fall Meeting Abstracts*, page D8, December 2011.

[26] S. D. Walsh, J. Myre, M. O. Saar, and D. Lilja. Multi-GPU, Multi-core, Multi-phase Lattice-Boltzmann Simulations of Fluid Flow for the Geosciences. *AGU Fall Meeting Abstracts*, page D983, December 2009.

[27] Andrew K Gunstensen, Daniel H Rothman, Stéphane Zaleski, and Gianluigi Zanetti. Lattice boltzmann model of immiscible fluids. *Physical Review A*, 43(8):4320, 1991.

[28] 郭照立，郑楚光. *格子 Boltzmann 方法的原理及应用*. 科学出版社, 武汉, 2009.

[29] YH Qian, D d'Humieres, and P Lallemand. Lattice bgk models for navier-stokes equation. *EPL (Europhysics Letters)*, 17(6):479, 1992.

[30] Chongxun Pan, Li-Shi Luo, and Cass T Miller. An evaluation of lattice boltzmann schemes for porous medium flow simulation. *Computers & fluids*, 35(8):898–909, 2006.

[31] Dominique d'Humières. Multiple–relaxation–time lattice boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 360(1792):437–451, 2002.

[32] Zhaoli Guo, Chuguang Zheng, and Baochang Shi. An extrapolation method for boundary conditions in lattice boltzmann method. *Physics of Fluids*, 14:2007, 2002.

[33] Xiaowen Shan and Hudong Chen. Lattice boltzmann model for simulating flows with multiple phases and components. *Phys. Rev. E*, 47:1815–1819, 1993.

[34] Mark L Porter, ET Coon, Q Kang, JD Moulton, and JW Carey. Multicomponent interparticle-potential lattice boltzmann model for fluids with large viscosity ratios. *Physical Review E*, 86(3):036701, 2012.

[35] C Pan, M Hilpert, and CT Miller. Lattice-boltzmann simulation of two-phase flow in porous media. *Water Resources Research*, 40(1), 2004.

[36] Gerhard Wellein, Thomas Zeiser, Georg Hager, and Stefan Donath. On the single processor performance of simple lattice boltzmann kernels. *Computers & Fluids*, 35(8):910–919, 2006.

[37] U Ghia, KN Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[38] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. A new approach to the lattice boltzmann method for graphics processing units. *Computers & Mathematics with Applications*, 61(12):3628–3638, 2011.

[39] Chongxun Pan, Jan F Prins, and Cass T Miller. A high-performance lattice boltzmann implementation to model flow in porous media. *Computer Physics Communications*, 158(2):89–105, 2004.

[40] AS Sangani and A Acrivos. Slow flow through a periodic array of spheres. *International Journal of Multiphase Flow*, 8(4):343–360, 1982.

[41] Li-Shi Luo. Unified theory of lattice boltzmann models for nonideal gases. *Physical review letters*, 81(8):1618–1621, 1998.