Modeling and Simulation 建模与仿真, 2026, 15(4), 32-44
Published Online April 2026 in Hans. https://www.hanspub.org/journal/mos
https://doi.org/10.12677/mos.2026.154051

Hans汉斯

基于CUDA的IB-LBM并行算法设计与性能分析

李　赢[1]，方雨欣[1]，黄昌盛[2*]

[1]中国地质大学(武汉)数学与物理学院，湖北　武汉
[2]华中科技大学数学与统计学院，湖北　武汉

收稿日期：2026年3月8日；录用日期：2026年4月1日；发布日期：2026年4月8日

摘　要

浸入边界－格子Boltzmann方法(IB-LBM)已成为模拟可变形颗粒－流体相互作用的重要数值工具。然而，在大规模复杂流固耦合问题的数值模拟中，普遍存在内存访问效率低与计算冗余严重等挑战，限制了数值仿真的计算性能与网格规模。为此，本文基于CUDA并行计算平台，对IB-LBM算法进行了并行算法优化设计，其中提出了双指针稀疏矩阵存储策略以解决全矩阵存储模式下无效节点更新导致的计算冗余问题。在此基础上，以多孔介质流与分叉管道流为例开展测试，并对比分析不同孔隙率下两种存储模式的计算效率与显存占用情况。结果表明，在孔隙率小于0.85时，所提出的优化策略可显著减少无效内存访问，平均计算性能提升20%以上，同时有效降低显存消耗。本文研究结果为大规模复杂流固耦合系统的高效数值模拟提供了可行的并行优化方案。

关键词

浸入边界－格子Boltzmann方法，CUDA并行计算，双指针稀疏存储，可变形颗粒，流固耦合

Design and Performance Analysis of CUDA-Based Parallel Algorithm for IB-LBM

Ying Li[1], Yuxin Fang[1], Changsheng Huang[2*]

[1]School of Mathematics and Physics, China University of Geosciences (Wuhan), Wuhan Hubei
[2]School of Mathematics and Statistics, Huazhong University of Science and Technology, Wuhan Hubei

Received: March 8, 2026; accepted: April 1, 2026; published: April 8, 2026

*通讯作者。

文章引用: 李赢, 方雨欣, 黄昌盛. 基于 CUDA 的 IB-LBM 并行算法设计与性能分析[J]. 建模与仿真, 2026, 15(4): 32-44. DOI: 10.12677/mos.2026.154051

Abstract

The Immersed Boundary-Lattice Boltzmann Method (IB-LBM) has become a powerful tool for simulating deformable particle-fluid interactions. However, its application to large-scale fluid-structure interaction problems is often bottlenecked by inefficient memory access and significant computational redundancy. To address these issues, this paper presents a CUDA-based parallel algorithm optimization of the IB-LBM, and proposes a double-pointer sparse matrix storage strategy to eliminate the redundancy caused by invalid node updates inherent in conventional full matrix storage schemes. The presented algorithm is evaluated using porous media and bifurcation flow benchmarks, comparing the computational efficiency and GPU memory footprint of the sparse and full storage modes. The numerical results show that when porosities below 0.85, the proposed strategy significantly reduces invalid memory accesses, yielding a performance improvement of over 20% while substantially lowering memory usage. This work provides an effective parallel optimization scheme for high-performance simulations of complex, large-scale fluid-structure interaction problem.

Keywords

Immersed Boundary-Lattice Boltzmann Method, CUDA Parallel Computing, Dual-Pointer Sparse Storage, Deformable Capsule, Fluid-Structure Interaction

1. 引言

可变形颗粒的输运现象普遍存在于自然界和工业领域，并在诸多物理过程和应用中发挥重要作用[1] [2]。对于此类问题，由于流场结构复杂且涉及可变形颗粒的力学响应，传统的理论分析和实验观测往往难以深入揭示其中的物理机制。随着计算机性能的不断提升，数值模拟逐渐成为研究此类复杂现象的重要手段[3] [4]。在众多数值方法中，结合了格子 Boltzmann 方法(LBM)与浸入边界方法(IBM)各自优点的浸入边界–格子 Boltzmann 方法(IB-LBM)，在模拟受限空间内的弹性胶囊、细胞或柔性颗粒的迁移与变形方面得到了广泛的应用[5]-[8]。

然而，随着模拟问题规模的扩大以及物理模型复杂度的增加，IB-LBM 的计算成本也迅速上升，对计算平台性能提出了更高要求。传统基于单核或多核 CPU 的实现方式在大规模流固耦合问题中逐渐受到内存带宽和并行效率的限制。相比之下，图形处理器(GPU)凭借其高度并行的计算架构和不断提升的浮点运算能力，逐渐成为高性能科学计算的重要平台[9]。然而，在复杂几何和工程尺度问题中，实现高效的 GPU 并行算法仍面临诸多挑战。例如，一些学者对直接寻址方案与间接寻址方案的效率进行了研究，但对大规模计算区域流场问题的探讨仍相对不足[10]。同时，一些旨在提高数值稳定性或计算精度的算法往往引入额外的数据依赖关系或迭代求解步骤，使并行粒度变粗，从而难以在 GPU 平台上实现高效映射[11]。此外，格子 Boltzmann 方法的数值模拟通常属于 I/O 密集型应用，其中分布函数的访问占据了内存请求的绝大部分。因此，在 GPU 实现中通常采用 SOA (Structure of Arrays)数据布局替代 AOS (Array of Structure)布局，使同一线程束内的线程访问连续的地址空间，从而减少内存事务并提高访存效率[12]。针对复杂几何边界条件，一些研究还引入了稀疏存储策略以减少无效节点计算，例如，稀疏矩阵的存储通常采用压

缩稀疏行(CSR)、坐标格式(COO)以及 ELLPACK 等经典格式[13]，其中部分格式主要针对 CPU 架构的访存模式进行优化。本文借鉴类似的稀疏化思想，通过压缩组织计算域中的有效节点，以减少无效节点带来的计算冗余并降低 GPU 显存占用，从而提高 GPU 平台上的计算效率，这也是本研究的主要出发点。

2. 数值模拟方法

2.1. 格子 Boltzmann 方法

介观格子 Boltzmann 方法具有物理概念清晰、易于处理复杂边界和天然并行性等优点，因而自上世纪八十年代提出以来，已逐渐发展成为流体力学与复杂流动模拟的重要工具[14] [15]。在 LBM 中，计算区域首先被划分为规则网格，以流体粒子分布函数为演化对象。在每一个时间步内，流体微团首先在格点处发生碰撞以调整其分布，随后沿给定的离散速度方向迁移至相邻格点，其演化方程如下：

$$f_i\left(\boldsymbol{x}+\boldsymbol{c}_i\Delta t,t+\Delta t\right)=f_i\left(\boldsymbol{x},t\right)-\frac{1}{\tau}\left[f_i\left(\boldsymbol{x},t\right)-f_i^{\mathrm{eq}}\left(\boldsymbol{x},t\right)\right], \tag{1}$$

其中，$\boldsymbol{x}$ 表示网格节点，$\boldsymbol{c}_i$ 为第 i 个离散速度方向，Δx 与 Δt 分别为空间和时间步长，$i=1,2,\cdots,q$ 表示速度方向集合。函数 $f_i\left(\boldsymbol{x},t\right)$ 为时刻 t 在节点 $\boldsymbol{x}$ 上沿方向 $\boldsymbol{c}_i$ 的粒子分布函数，τ 为无量纲松弛时间，$f^{\mathrm{eq}}\left(\boldsymbol{x},t\right)$ 为局部平衡分布函数，定义如下[16]：

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

其中，ω_i 为方向 i 的权系数，c_s 为格子声速，其具体取值依赖于所采用的离散速度模型[13]。

通过对分布函数的各阶矩求和，可以得到流体的宏观物理量，密度 ρ 与速度 $\boldsymbol{u}$ 的计算公式为：

$$\begin{aligned}\rho\left(\boldsymbol{x},t\right)&=\sum_i f_i\left(\boldsymbol{x},t\right),\\ \rho\boldsymbol{u}\left(\boldsymbol{x},t\right)&=\sum_i \boldsymbol{c}_i f_i\left(\boldsymbol{x},t\right).\end{aligned} \tag{3}$$

本文将采用广泛应用的二维九速 D2Q9 模型进行数值模拟。

2.2. 浸入边界法

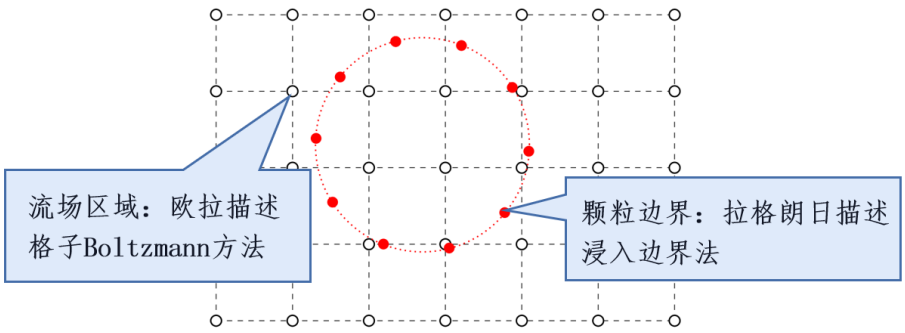

Figure 1. Schematic of the Eulerian and Lagrangian grids
图 1. 欧拉网格与拉格朗日网格

在利用 LBM 求解流体场的基础上，本文采用浸入边界法[17]处理流场与可变形颗粒膜之间的流固耦合。浸入边界法中使用两套不同的离散网格：一是固定的欧拉网格，用于描述流体区域；二是分布于颗粒膜表面的拉格朗日离散点，用于表征颗粒边界的形变与运动。不同于依赖贴体网格的传统方法，IBM

通过引用虚拟的体积力在固定的欧拉网格上实现流体与变形结构的耦合，能够在不重构网格的前提下有效地捕捉复杂边界的动力学行为，特别适用于处理复杂几何形状、多尺度界面及大变形移动边界问题。

如图 1 所示，可变形颗粒被建模为一个二维、零厚度的弹性膜结构，包裹着与外部流体性质相同的内部流体。膜通过拉格朗日标记点进行离散，这些标记点通过连接形成一个闭合且可变形的边界。标记点的位置会随时间演化，其运动受到周围流体流动和弹性膜力学特性的共同作用。当膜从其初始参考状态发生变形时，作用在膜上的力可表示为[18]：

$$\boldsymbol{F} = -\frac{\partial}{\partial s}\left(\tau \boldsymbol{t} + q\boldsymbol{n}\right), \tag{4}$$

其中，τ 表示膜面内的张力，q 为横向剪切张力，s 是弧长坐标，$\boldsymbol{t}$ 是沿 s 增长方向的位切向量，$\boldsymbol{n}$ 是指向外部流体的单位法向量。膜的面内张力 τ 遵循 neo-Hookean 型非线性本构关系：

$$\tau = k_s \frac{\varepsilon^3 - 1}{\varepsilon^{3/2}}, \tag{5}$$

其中 k_s 为膜的伸长系数，$\varepsilon = \left|\partial \boldsymbol{X}/\partial s_0\right|$ 为拉伸比，$\boldsymbol{X}$ 为膜上某标记点的位置向量，s_0 为该点的初始弧长。横向剪切张力 q 由下式给出：

$$q = \frac{\mathrm{d}m}{\mathrm{d}s}, \tag{6}$$

其中 m 为弯曲力矩，其表达式为：

$$m = k_b\left[\kappa(s) - \kappa_0(s)\right], \tag{7}$$

此处 k_b 为膜的弯曲模量，$\kappa(s)$ 为瞬时膜曲率，$\kappa_0(s)$ 为膜在静止初始状态下的初始曲率，此时弯曲力矩为零。方程中涉及的导数均通过有限差分方法进行计算。

为了在流体控制方程中体现流体-弹性结构的作用，在动量方程中引入外力项

$$\boldsymbol{F}_{ib}\left(\boldsymbol{x}_f, t\right) = \sum_b \boldsymbol{F}_{ib}\left(\boldsymbol{X}_b, t\right) D\left(\boldsymbol{x}_f - \boldsymbol{X}_b\right) \Delta s \Delta x. \tag{8}$$

该外力项通过将拉格朗日节点上的力 $\boldsymbol{F}_{ib}$ 分布到邻近的欧拉网格上得到，其中 Δs 表示相邻拉格朗日点之间的弧长。$D(\boldsymbol{x})$ 是离散 δ 函数，用于实现从拉格朗日点到欧拉网格的力分布：

$$D\left(\boldsymbol{x}_f - \boldsymbol{X}_b\right) = \frac{1}{\Delta x^2}\phi\left(\frac{x_f - X_b}{\Delta x}\right)\phi\left(\frac{y_f - Y_b}{\Delta x}\right), \tag{9}$$

其中

$$\phi(r) = \begin{cases} \dfrac{1}{4}\left(1 + \cos\dfrac{\pi r}{2}\right), & |r| < 2, \\ 0, & |r| \geq 2. \end{cases} \tag{10}$$

拉格朗日节点的速度通过邻近欧拉点的速度插值得到：

$$\boldsymbol{u}\left(\boldsymbol{X}_b, t\right) = \sum_f \boldsymbol{u}\left(\boldsymbol{x}_f, t\right) D\left(\boldsymbol{x}_f - \boldsymbol{X}_b\right) \Delta x^2. \tag{11}$$

通过上述方法，IBM 能够实现流体与可变形膜之间的双向耦合，保证了颗粒运动、形变以及流体动力学行为的高度一致性，使其在复杂边界条件和可变形颗粒模拟中具有较高的精度与稳定性。

3. 并行算法设计

LBM 与 GPU 并行计算技术具有良好的匹配性，本节将基于 CUDA 并行计算架构，系统地讨论 IB-LBM 的 GPU 并行算法设计与性能分析。

3.1. 数据存储设计

在 IB-LBM 程序中，每个时间步内对节点分布函数的访问通常发生两次：一次出现在碰撞步骤，另一次发生在流动步骤。为降低对全局内存的读写压力，我们将碰撞与流动两个步骤合并于同一核函数中执行。具体而言，每个线程负责一个节点，在执行完碰撞计算后立即进行流动处理，使得整个过程中对该节点的分布函数仅需一次读取和一次写入操作。需要注意的是，流动步骤中将分布函数传递至相邻节点可能导致原始数据被覆盖。为解决该问题，通常采用两套网格交替使用的策略。在本算法中，我们使用两个一维数组 f 和 F 存储所有格点的分布函数，称为分布函数数组。在偶数时间步，从数组 f 中读取分布函数，执行碰撞流动计算后将结果存入数组 F 。在奇数时间步则相反。

在数据存储与访问策略方面，传统的 LBM 算法实现通过使用完全矩阵存储模式。在此存储模式下，计算区域中的所有节点(包括流体点和固体点)均需分配完整的存储空间。假设计算网格的规模为 $NX \times NY$ ，则坐标为 (x,y) 的节点在速度方向 k 上的分布函数在数组中的索引可表示为

$$\mathrm{Index}(k,x,y) = k \times NX \times NY + y \times NX + x, \tag{12}$$

其中 k 为速度方向编号，范围为 $0 \le k < Q$ ，而 Q 为所采用格子模型的速度方向数。在该布局下，每个速度方向上的分布函数按照节点坐标 (x,y) 顺序存储，而不同方向的分布函数则以 k 为分块顺序依次排列。这种存储方式的主要优点在于其数据结构简单、索引规则统一，易于在 GPU 端并行实现。同时，线程在同一速度方向上的访问具有良好的连续性，有助于保证合并访问，从而提高显存带宽的利用效率。然而，该策略的不足之处也十分明显：首先，会造成较大的显存开销，即使在固体区域，所有节点依旧分配全部的分布函数存储单元；其次，固体点虽然不参与实际演化过程，但在内核执行中仍需进行访问和判断，导致分支开销增加；最后，在大规模模拟中，该模式的冗余存储和计算会使显存资源成为性能瓶颈。

为解决这一问题，本文提出了双指针稀疏矩阵存储模式。在该存储模式下，仅为计算区域中的流体节点分配存储空间，从而有效减少内存开销。由于仅存储流体节点，节点间的邻居关系在压缩映射过程中会丢失，因此需要额外的索引结构加以维护。与完全矩阵存储模式不同，此处不再使用 *flag* 数组标记节点类型，而是引入二维映射表 *toCompressIndex* ，其指向流体节点在实际分配的流体点数组中的实际存储位置。当 $toCompressIndex(i,j)$ 为−1 时，表示节点 (i,j) 为固体节点，不参与演化过程。

如图 2 所示，对于某一流体节点 p ，由于 *toCompressIndex* 数组保持了原始网格的拓扑结构，因此可以在压缩数组中为其增加一张反向索引表 *toBackIndex* ，用于恢复节点的物理坐标。具体而言，*toBackIndex* 定义为 4 字节无符号整型数，其中高 16 位用于存储 j 坐标，低 16 位用于存储 i 坐标。其编解码方式如下：

$$\begin{cases} i = toBackIndex[tid] \,\&\, 0\text{x}0000\text{FFFF}, \\ j = toBackIndex[tid] \gg 16. \end{cases} \tag{13}$$

其中“$\gg$”表示位运算中的右移操作符，“&”表示按位与操作符。通过该反向索引机制，可以由压缩数组下标 *tid* 唯一确定原拓扑结构中的物理坐标 (i,j) ，再结合 *toCompressIndex* 所保持的邻居关系，实现对流体节点邻域的快速定位与访问。

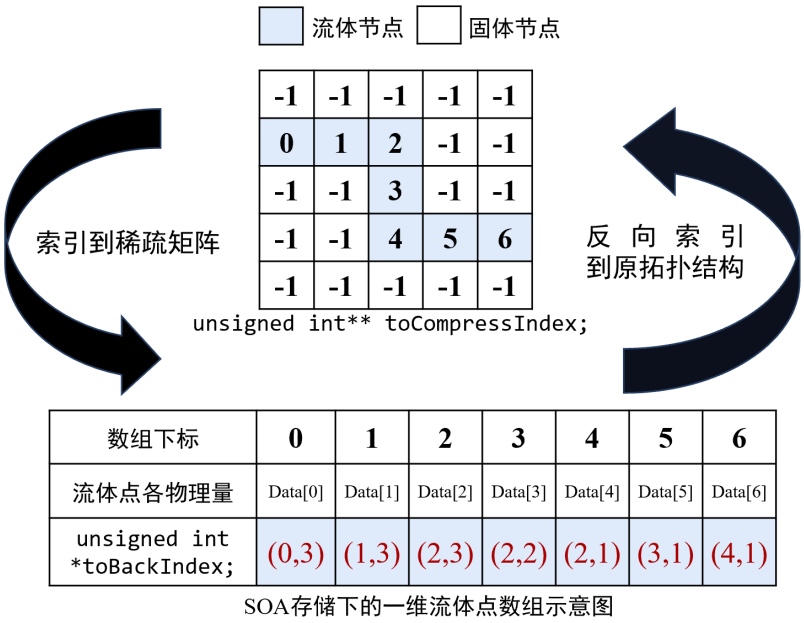

Figure 2. Schematic of the two-pointer sparse matrix access pattern
图 2. 双指针稀疏矩阵模式寻址示意图

3.2. GPU 并行算法设计

利用 CUDA 平台设计的算法通常遵循以下基本流程：(1) 在 CPU 与 GPU 上分别分配内存与显存空间，并在主机端完成数据初始化；(2) 将初始数据从主机内存传输至 GPU 内存(显存)；(3) 根据并行计算需求配置内核函数的 grid 与 block 维度，在 GPU 上启动核函数(kernel)执行并行计算；(4) 计算完成后，将结果从 GPU 内存拷贝回主机内存，并释放相关存储资源。

Algorithm 1: 使用 GPU 加速的 IBLBM 求解流程

输入: 初始流体场、可变形颗粒边界位置、最大时间步长 T_{max}
输出: 各时间步的流体分布函数与各宏观量数据

```
1 1: 初始化流体与边界数据，并将其拷贝至 GPU;
2 for t = 1 to T_max do
3     // 计算边界节点的弹性与约束力;
4     compute_particle_force<<<grid_ib, block_ib>>>(...);
5     // 将拉格朗日节点上的力分布至欧拉网格节点;
6     spread_force<<<grid_ib, block_ib>>>(...);
7     // 执行流体的碰撞与迁移步骤（LBM）;
8     collision_and_streaming<<<grid_lb, block_lb>>>(...);
9     // 计算欧拉网格节点的宏观物理量（密度与速度）;
10    macro_moment<<<grid_lb, block_lb>>>(...);
11    // 插值流体速度到拉格朗日节点并更新颗粒位置;
12    interpolation_velocity_and_moving<<<grid_ib, block_ib>>>(...);
13 end
14 将结果从 GPU 拷贝回 CPU 并输出。
```

Figure 3. Parallel IBLBM algorithm based on GPU acceleration
图 3. GPU 加速的 IBLBM 并行算法

图 3 和图 4 给出了本文基于 GPU 的 IB-LBM 并行算法的基础伪代码和算法流程图。在每一个时间步内，算法首先在拉格朗日边界点上通过求解结构弹性力得到边界受力信息；随后将该边界力传递至邻近欧拉流体格点，实现流固耦合。在完成虚拟体积力的传播后，在 GPU 上依次执行碰撞与迁移操作，更新各格点的分布函数，并计算密度、速度等宏观物理量。得到新时间步的流场速度后，再通过插值得到拉格朗日节点的速度，用于更新弹性颗粒边界的速度、位置，之后进入下一时间步的迭代。

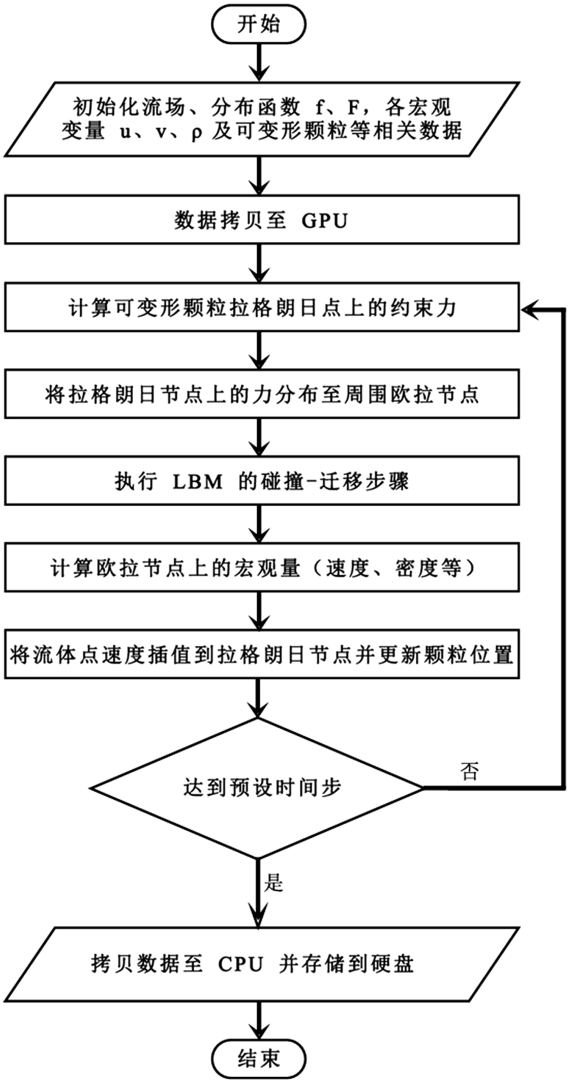

Figure 4. Flowchart of the GPU-accelerated IBLBM algorithm
图 4. GPU 加速 IBLBM 算法求解流程图

4. 数值模拟与性能分析

4.1. 算例设置

在本节中，我们以球体填充的多孔介质流动和可变形颗粒在分叉管道中的输运为算例，对程序的计算性能进行测试，并分析影响性能的主要因素。数值模拟的硬件平台配置为 AMD R9 7950X CPU 与 NVIDIA RTX 4090 GPU，编译环境采用 CUDA 12.4 版本。

为评估算法的性能，本文采用的每秒百万流体格点更新率(Million Fluid Lattice Updates Per Second, MFLUPS)作为指标。在多孔介质等低孔隙率场景中，固体节点数量较多且不参与流体演化，采用 MFLUPS 能更准确地反映算法的实际性能表现，具有更高的合理性。

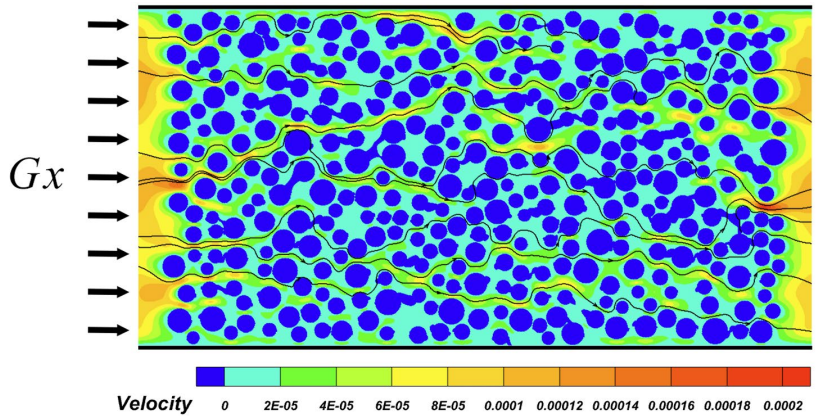

Figure 5. Schematic of the computational domain for flow in porous media
图 5. 多孔介质流计算区域示意图

多孔介质流动算例如图 5 所示，其中左右边界采用周期性边界条件，上下边界采用半反弹格式；同时在水平方向施加压力驱动(取 $Gx = 0.05$)。已有研究表明，当 *blockDim.y* 与 *blockDim.z* 取 1 时，更有利于实现合并访存，从而提升访存效率[19]。基于此，本文将 *block* 的设计维度设为 $(32,1,1)$。

可变形颗粒在分叉管道中的输运算例如图 6 所示，其中网格大小为 4096 × 2048，其他设置同上。

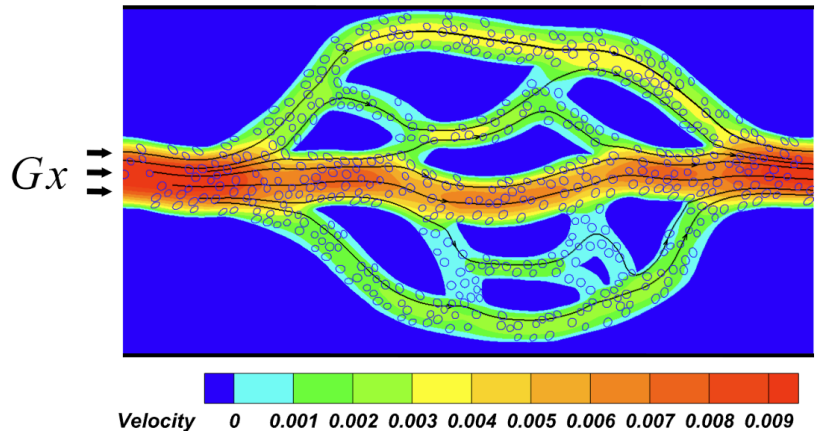

Figure 6. Computational domain for flow in a bifurcated channel
图 6. 分叉管道流计算区域示意图

4.2. 数值模拟结果

我们首先利用多孔介质流动为例讨论不同孔隙率下两种模式的优劣，网格大小为 2048 × 1024。不同孔隙率下的程序性能如表 1 所示。从表中可以看出，两种存储模式对于不同孔隙率的工况表现出明显差异。对于完全矩阵存储模式，当孔隙率为 1 时(不存在多孔介质)，其性能最佳，达到 1976 MFLUPS。然而随着孔隙率的降低，其性能逐渐下降，在孔隙率为 0.2 时仅为 891 MFLUPS。这表明，当固体节点数量增加时，大量线程需要处理无效计算，从而导致访存效率和并行效率下降，严重影响了程序的性能。

Table 1. Computational performance at different porosities (unit: MFLUPS)
表 1. 不同孔隙率下程序性能(单位：MFLUPS)

孔隙率	1.0	0.8	0.6	0.4	0.2
完全矩阵	1976	1750	1444	1127	891
稀疏矩阵	1820	1796	1752	1839	1844

图 7 展示了两种数据存储模式的性能对比。如图所示，双指针稀疏矩阵存储模式(图中算法 2)在不同孔隙率下的性能表现更加稳定。在孔隙率为 1 时，算法 2 的性能低于算法 1 (完全矩阵)，这是由于双指针稀疏矩阵存储需要维护额外的索引结构，并且引入了间接寻址开销，在流场中不存在多孔介质时这些开销无法被抵消，反而成为性能负担。但随着孔隙率逐渐减小，算法 2 的优势逐步显现。在孔隙率较低时，算法 2 可以有效跳过固体节点，从而避免了大量无效的计算与访存操作，使得性能几乎不随孔隙率的降低而衰减。此外，当孔隙率高于大约 0.85 时，算法 1 更具优势；当孔隙率低于该值时，算法 2 的稀疏存储策略能够发挥更大作用，性能明显超过算法 1。

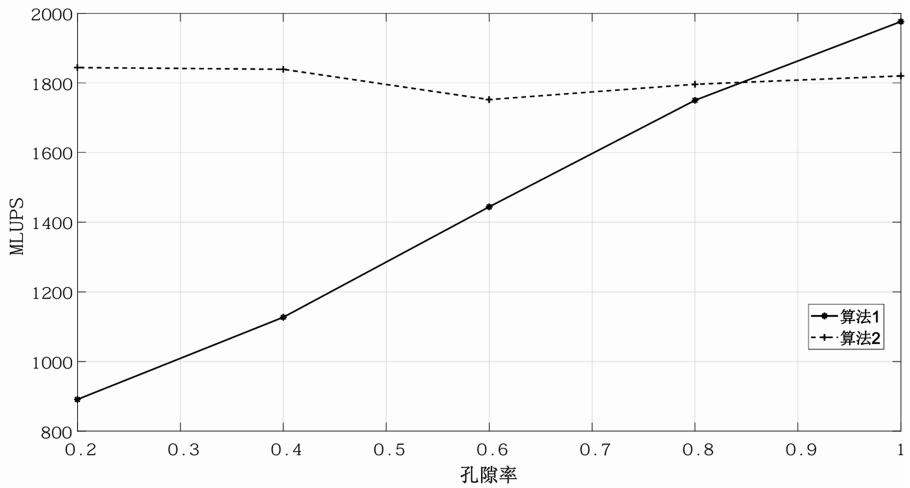

Figure 7. Computational performance (MFLUPS) of two memory layouts at different porosities
图 7. 不同孔隙率下两种存储模式的 MFLUPS

接着，我们详细对比了不同网格规模下本文所提出的算法分别在 CPU 与 GPU 平台上程序性能的差异，其中孔隙率设置为 0.6。测试结果汇总于表 2 中。

Table 2. Computational performance and speedup at different grid resolutions (unit: MFLUPS)
表 2. 不同网格下程序性能及加速比(单位：MFLUPS)

网格大小	512 × 256	1024 × 512	2048 × 1024	4096 × 2048	8192 × 4096
CPU	5.51	7.43	7.45	7.49	7.60
GPU	1142	1742	1744	1811	1925
加速比	207	234	234	241	253

从表中可以发现，当网格规模较大时 GPU 并行程序能取得更高的性能，MFLUPS 由 1142 快速增长到 1925，这说明更大规模的计算任务能够更充分地利用 GPU 的大规模线程并行架构，提高 SM 的占用率和访存合并效率，从而显著提升整体性能。加速比从 207 提升至 253，呈现出稳步上升的趋势，这表明

随着网格规模的增大，GPU 相对于 CPU 的优势愈发明显，并且在大规模问题中能够更高效地发挥其高带宽和高并行度的特性。

最后，本文评估了在分叉管道流场中，颗粒数量与半径对程序并行计算性能(MFLUPS)的影响。图 8 展示了固定颗粒半径时($R = 25\Delta x$)，不同颗粒数目及孔隙率下的性能变化。由图可见，当颗粒数目小于 64 时，MFLUPS 保持相对平稳，原因在于此阶段 GPU 程序的性能主要受到内核启动开销和函数调用等固定开销的影响。随着颗粒数目的增加，计算耗时占比逐渐上升，导致程序的 MFLUPS 出现近似线性下降趋势。此外，在固定颗粒数目条件下，不同孔隙率下的 MFLUPS 差异较小，表明孔隙率变化对计算效率无显著影响。

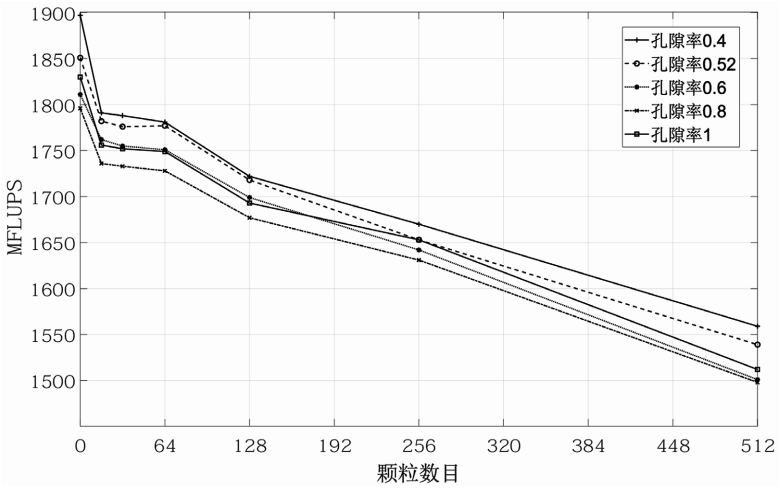

Figure 8. Performance of the sparse matrix mode with varying porosity and capsule number
图 8. 稀疏矩阵模式程序性能随孔隙率及颗粒数目变化

图 9 展示了固定颗粒数目($N = 256$)时，不同颗粒半径及孔隙率对性能的影响。结果显示，随着颗粒半径增大，MFLUPS 呈逐渐下降趋势。其原因在于颗粒半径的增加会导致单个颗粒所需的拉格朗日点数及相关联的流体节点数显著增多，从而引入额外的访存与计算开销。对比结果再次证实，孔隙率对整体 GPU 程序计算效率无显著影响。

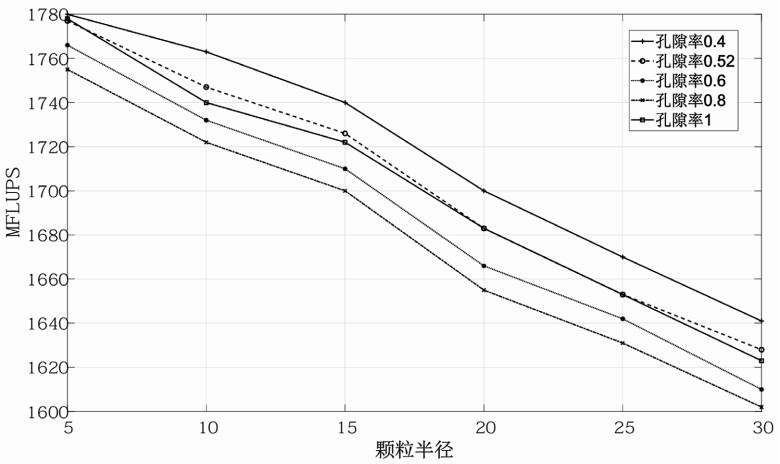

Figure 9. Performance of the sparse matrix mode with varying porosity and capsule radius
图 9. 稀疏矩阵模式程序性能随孔隙率及颗粒半径变化

4.3. 显存占用分析

本节将对比两种算法在内存(显存)消耗方面的差异。由于 IB-LBM 在数值实现中需要存储大量分布函数及宏观量，因此不同存储策略对内存需求的差异十分显著。假设 IB-LBM 模型的速度方向数为 Q，流场维度为 $NX \times NY$，孔隙率为 r，流体节点总数为 $N = NX \times NY$。

Table 3. GPU memory requirement for full matrix storage mode
表 3. 完全矩阵存储模式的显存需求

数组类型	单节点内存开销	总内存开销
分布函数 f, F	Q 个 double	$16NQ$
全局力场 f_x, f_y	1 个 double	$16N$
宏观量 u, v, ρ	1 个 double	$24N$
标识符 *flag*	1 个 int	$4N$
总计	/	$M_1 = N(16Q + 44)$

Table 4. GPU memory requirement for sparse matrix storage mode
表 4. 稀疏矩阵存储模式的显存需求

数组类型	单节点内存开销	总内存开销
分布函数 f, F	Q 个 double	$16rNQ$
全局力场 f_x, f_y	1 个 double	$16rN$
宏观量 u, v, ρ	1 个 double	$24rN$
toCompressIndex	1 个 int	$4N$
toBackIndex	1 个 unsigned int	$4rN$
总计	/	$M_2 = N(16rQ + 44r + 4)$

在完全矩阵存储模式中，需要存储整个计算域的所有节点的数据。相比之下，双指针稀疏矩阵存储模式仅针对流体节点进行存储，能够显著降低内存使用量，尤其是在孔隙率较低的情况下优势更加明显。

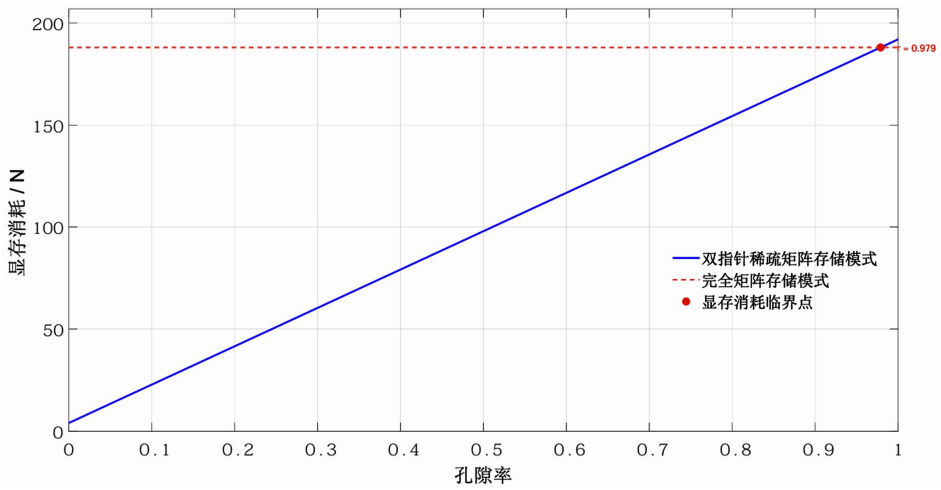

Figure 10. GPU memory requirement versus porosity for two storage modes
图 10. 不同存储模式下显存需求随孔隙率变化

为了直观展示这两种模式在存储开销上的差异，表 3 和表 4 分别列出了两种模式下需要存储的数组及其对应的内存消耗，并在图 10 中展示了两种存储模式下的显存需求。完全矩阵存储模式的显存开销与孔隙率无关，为一条水平线；而双指针稀疏矩阵存储模式的显存需求则随孔隙率的减小近似线性下降。两条曲线的交点表示两种模式显存开销相等的临界孔隙率。

由 $M_1 = N(16Q + 44)$ 与 $M_2 = N(16rQ + 44r + 4)$ 可知，当采用 D2Q9 模型时，临界孔隙率约为 $r = 0.979$。这意味着当流场孔隙率低于 0.978 时，双指针稀疏矩阵存储模式在显存使用上具有明显优势；随着孔隙率的降低，双指针稀疏矩阵存储模式的显存占用将以线性比例进一步下降，从而在复杂几何或多孔介质等低孔隙率场景中表现出更高的存储效率。

5. 结论

本文聚焦于浸入边界‐格子 Boltzmann 方法(IB-LBM)的 GPU 并行算法设计，重点对比分析了完全矩阵存储与双指针稀疏矩阵存储两种策略的差异及性能表现。借助二维多孔介质流与分叉管道流算例，系统评估了网格规模、孔隙率及颗粒几何参数(数量、半径)对并行算法性能的影响。结果表明，在孔隙率 $r < 0.85$ 时，本文所提出的优化策略能够有效减少无效内存访问，平均计算性能(以 MFLUPS 计)提升超过 20%。在含可变形颗粒算例中，数值模拟结果与理论分析高度吻合，验证了并行程序的正确性与数值稳定性。

进一步地，通过对两种存储模式的显存需求定量分析可以发现，当孔隙率 $r < 0.979$ 时，双指针稀疏矩阵存储模式在显存消耗方面优于完全矩阵模式。这表明在低孔隙率场景中(如多孔介质流、分叉管道流等)，稀疏存储模式能够显著提升计算资源的利用效率。

综上，本文完成了 IB-LBM 并行算法在 CUDA 平台上的 GPU 并行算法设计与性能分析，系统验证了稀疏矩阵存储策略在显存消耗与计算效率方面的优势，为复杂流场中可变形颗粒输运问题的数值模拟提供了有效的并行计算方案。

基金项目

本研究由国家自然科学基金(11702259)及湖北省教育厅科学技术研究计划指导性项目(B2023249)资助。

参考文献

[1] Cheng, X., Caruso, C., Lam, W.A. and Graham, M.D. (2025) Red Blood Cell Partitioning and Segregation through Vascular Bifurcations in a Model of Sickle Cell Disease. *Soft Matter*, **21**, 5793-5803. https://doi.org/10.1039/d4sm01519c

[2] Wu, N., Grieve, S.W.D., Manning, A.J. and Spencer, K.L. (2024) Flocs as Vectors for Microplastics in the Aquatic Environment. *Nature Water*, **2**, 1082-1090. https://doi.org/10.1038/s44221-024-00332-4

[3] Wu, T., Khani, M., Sawalha, L., Springstead, J., Null, J.K. and Qi, D. (2020) A Cuda-Based Implementation of a Fluid-Solid Interaction Solver: The Immersed Boundary Lattice-Boltzmann Lattice-Spring Method. *Communications in Computational Physics*, **23**, 980-1011. https://doi.org/10.4208/cicp.oa-2016-0251

[4] Xiang, X., Su, W., Hu, T. and Wang, L. (2023) Multi-GPU Lattice Boltzmann Simulations of Turbulent Square Duct Flow at High Reynolds Numbers. *Computers & Fluids*, **266**, Article ID: 106061. https://doi.org/10.1016/j.compfluid.2023.106061

[5] Zhang, J., Johnson, P.C. and Popel, A.S. (2007) An Immersed Boundary Lattice Boltzmann Approach to Simulate Deformable Liquid Capsules and Its Application to Microscopic Blood Flows. *Physical Biology*, **4**, 285-295. https://doi.org/10.1088/1478-3975/4/4/005

[6] Krüger, T., Varnik, F. and Raabe, D. (2011) Efficient and Accurate Simulations of Deformable Particle Suspensions with a Lattice-Boltzmann and Finite Element Method. *Journal of Fluid Mechanics*, **676**, 465-475.

[7] Krüger, T. (2016) Effect of Tube Diameter and Capillary Number on Platelet Margination and Near-Wall Dynamics.

Rheologica Acta, **55**, 511-526. https://doi.org/10.1007/s00397-015-0891-6

[8] Wang, Z., Sui, Y., Salsac, A., Barthès-Biesel, D. and Wang, W. (2016) Motion of a Spherical Capsule in Branched Tube Flow with Finite Inertia. *Journal of Fluid Mechanics*, **806**, 603-626. https://doi.org/10.1017/jfm.2016.603

[9] NVIDIA Corporation (2023) CUDA C Programming Guide. https://docs.nvidia.com/

[10] 朱红银. 基于 GPU 的晶格 Boltzmann 方法并行算法研究[D]: [硕士学位论文]. 桂林: 广西师范大学, 2021.

[11] Ames, J., Puleri, D.F., Balogh, P., Gounley, J., Draeger, E.W. and Randles, A. (2020) Multi-GPU Immersed Boundary Method Hemodynamics Simulations. *Journal of Computational Science*, **44**, Article ID: 101153. https://doi.org/10.1016/j.jocs.2020.101153

[12] 黄昌盛. LBM 的 GPU 算法及其在颅内动脉瘤血流动力学中的应用[D]: [博士学位论文]. 武汉: 华中科技大学, 2014.

[13] AlAhmadi, S., Mohammed, T., Albeshri, A., Katib, I. and Mehmood, R. (2020) Performance Analysis of Sparse Matrix-Vector Multiplication (SpMV) on Graphics Processing Units (GPUs). *Electronics*, **9**, Article 1675. https://doi.org/10.3390/electronics9101675

[14] 郭照立, 郑楚光. 格子 Boltzmann 方法的原理及应用[M]. 北京: 科学出版社, 2009.

[15] 何雅玲, 王勇, 李庆. 格子 Boltzmann 方法的理论及应用[M]. 北京: 科学出版社, 2009.

[16] Qian, Y.H., D'Humières, D. and Lallemand, P. (1992) Lattice BGK Models for Navier-Stokes Equation. *Europhysics Letters* (*EPL*), **17**, 479-484. https://doi.org/10.1209/0295-5075/17/6/001

[17] Peskin, C.S. (2002) The Immersed Boundary Method. *Acta Numerica*, **11**, 479-517. https://doi.org/10.1017/s0962492902000077

[18] Ma, J., Wang, Z., Young, J., Lai, J.C.S., Sui, Y. and Tian, F. (2020) An Immersed Boundary-Lattice Boltzmann Method for Fluid-Structure Interaction Problems Involving Viscoelastic Fluids and Complex Geometries. *Journal of Computational Physics*, **415**, Article ID: 109487. https://doi.org/10.1016/j.jcp.2020.109487

[19] 黄昌盛, 张文欢, 侯志敏, 等. 基于 CUDA 的格子 Boltzmann 方法: 算法设计与程序优化[J]. 科学通报, 2011, 56(28): 2434-2444.