# 格子玻尔兹曼机的并行实现

杨昊光　　2014011619

June 29, 2017

## 1. 引言

在计算流体力学（CFD）中，有两大类求解方法：基于 N-S 方程的有限差分、有限体积法，以及基于分子动力学的方法。格子玻尔兹曼机就是后者的一种，通过模拟流体分子或微团之间的相互作用，来描述流体的运动特点。格子玻尔兹曼机（Lattice Boltzmann Machine, LBM）基于格子气模型，后者将空间分为许多细小的网格，每个格点被分子占据或空置，某个格点下一时刻的状态（有无分子，分子的速度、密度）根据上一时刻该格点周围的分子占据情况计算得出，常见的计算模板包括平移和碰撞等。常见的平面格点形式有四方格点和六方格点，后者因为能模拟分子的随机性而被较为广泛的使用。在三维情况中，为了方便表示，常见的格点是立方格点。格子气模型因为直接从分子级尺度考虑问题，能够模拟出湍流等分级耗散机制以及复杂几何边界条件的更加细微的模型，但同时缺点也非常明显：为了模拟得足够精细，格点的密度和参与运算的分子数很大，这在过去计算机能力并不发达的时候是难以实现的。然而由于其运算是模板化的，在今天并行机普及的背景下，可以通过对其算法进行高效的并行，实现很好的模拟效果。本次大作业，笔者将分析格子玻尔兹曼机的算法并使用混合并行的方式对其进行加速。

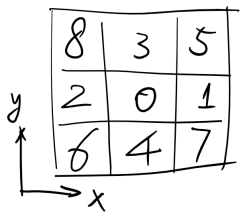
## 2. 算法简述

考虑一个网格，每个格点上有一个分子，可能往相邻的格点进行迁移，其格子玻尔兹曼方程可以写为：

$$f_i(\boldsymbol{x}+\boldsymbol{c_i}\Delta t, t+\Delta t) = f_i(\boldsymbol{x},t) + \frac{\Delta t(f_i(\boldsymbol{x},t) - f_i^{eq}(\boldsymbol{x},t))}{\tau}$$

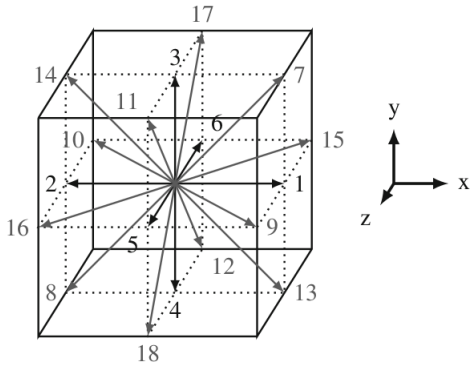
其中平衡态由 $f_i^{eq}(\boldsymbol{x},t) = w_i\rho\left(1 + \frac{\boldsymbol{u}\cdot\boldsymbol{c_i}}{c_s^2} + \frac{(\boldsymbol{u}\cdot\boldsymbol{c_i})^2}{2c_s^4} - \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2c_s^2}\right)$ 给出，其中的权重 $w_i$ 由所选组合决定。

对于二维的 D2Q9 组合，有：




| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $w_i$ | $\frac{4}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{9}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ |
| $c_{ix}$ | 0 | +1 | -1 | 0 | 0 | +1 | -1 | +1 | -1 |
| $c_{iy}$ | 0 | 0 | 0 | +1 | -1 | +1 | -1 | -1 | +1 |

对于三维 D3Q19 组合，有：




D3Q19

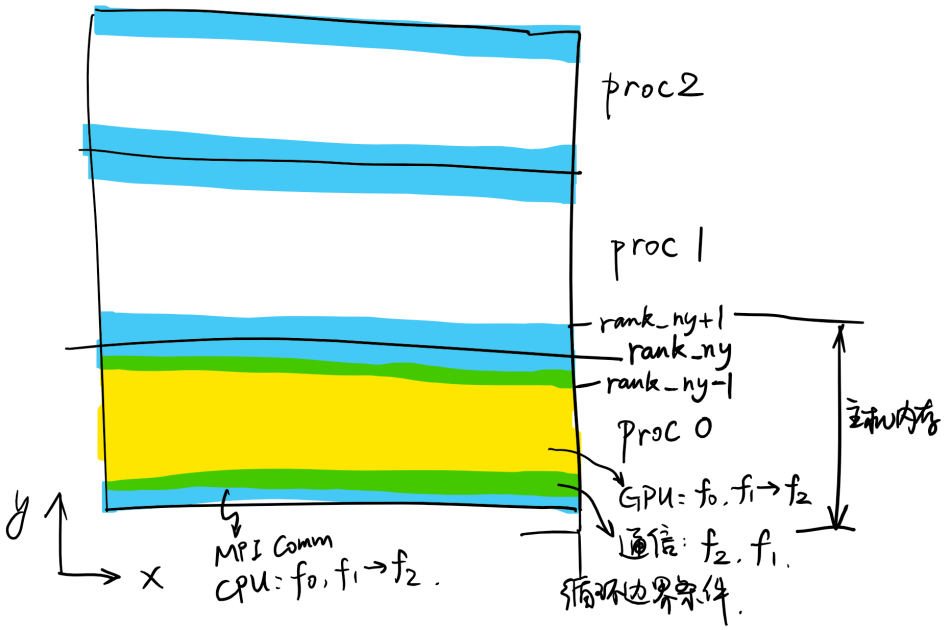
| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $w_i$ | $\frac{1}{3}$ | $\frac{1}{18}$ | $\frac{1}{18}$ | $\frac{1}{18}$ | $\frac{1}{18}$ | $\frac{1}{18}$ | $\frac{1}{18}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ | $\frac{1}{36}$ |
| $c_{ix}$ | 0 | +1 | -1 | 0 | 0 | 0 | 0 | +1 | -1 | +1 | -1 | 0 | 0 | +1 | -1 | +1 | -1 | 0 | 0 |
| $c_{iy}$ | 0 | 0 | 0 | +1 | -1 | 0 | 0 | +1 | -1 | 0 | 0 | +1 | -1 | -1 | +1 | 0 | 0 | +1 | -1 |
| $c_{iz}$ | 0 | 0 | 0 | 0 | 0 | +1 | -1 | 0 | 0 | +1 | -1 | +1 | -1 | 0 | 0 | -1 | +1 | -1 | +1 |

3. 并行化

本程序演示的三维格子玻尔兹曼机根据 Timm Kruger 的 *The Lattice Boltzmann Method: Principles and Practice* 第 13 章的二维示例程序改编而成，综合采用了 GPU、OpenMP 和 MPI 三种并行方法，以期获得更大的并行度。程序包括初始化、核心计算和导出量计算三部分，其中核心计算在 GPU 中完成，MPI 相关的通信边缘量和校核所需的导出量计算和集群通信在 CPU 中完成。由于校核所用的 Taylor-Green 流在二维情况下才有解析解，因此本程序主要关注二维格点 D2Q9 的情况。三维格点 D3Q19 的纯 CUDA 计算程序另附。

(a) 混合并行：分工与数据传输

由 LBM 的原理，每个格点具有 9 个场量，其中 $f_0$ 为非迁移量，仅由上一时刻的 $f_0$ 决定；$f_1$ 到 $f_8$ 均为迁移量，需要在迭代中联系周围格点的相关量进行计算。为此，我们独立拆分 $f_0$ 和剩下的 8 个场量，对 $f_0$ 进行原位迭代，除了储存和校核外不进行数据传输；对其余 8 个场量合并为一个数组进行模板计算，对边缘值进行数据传输。考虑到 CUDA 并没有直接的 MPI 接口，我们使用 CUDA 对内部点进行计算，使用 OpenMP 并行化对边缘点进行计算并使用 MPI 标准的接口更新数据，如下图所示。为方便地址的表示，本程序仅采用了简单的一维子网格划分。



(b) 基于 CUDA 的并行化

对 CUDA 部分，并行化主要是对模板运算应用于各个节点的并行，实现比较简单。提升性能的关键部分在于数据传输部分。笔者一开始测试了每次完全拷贝整个场量数组到内存，再由内存完成指针交换，在拷贝到显存的方法，效率极其低下，只有最终性能的约 10%。

通过分析流程我们不难发现，对于场量内部的数据除了储存和校核之外，并不需要每次都进行拷贝，于是我们只对边缘值进行复制，并在显存内也进行指针交换。经过这个调整，程序性能得到大幅提升。

经过 Profiling 发现，CPU 的运算能力较弱，对于比较强大的计算卡，如本次测试的 Tesla K40M ，容易出现显卡和 CPU 交错工作的情况。经分析认为显存和内存之间的复制如果改成非同步的方式可以改善该问题。经过调整程序执行过程，我们将部分这样的复制改为非同步 (cudaMemcpyAsync，程序处理后不等待结束就返回，需要用 cudaDeviceSynchronize() 进行同步)，测试过程中 CPU 和 GPU 均维持在接近满载的情况，仅在进行磁盘数据存储的时候出现短暂的等待过程，性能较优化前提升 12% 左右。

(c) 使用 OpenMP 并行

对边缘值的计算采用 OpenMP 并行，使用 schedule(guided) 和指定的 shared 变量能够处理大部分的并行化问题。对于导出值中 L2 误差的计算，平方求和的过程使用 reduction(+) 的方式进行并行。

(d) 使用 MPI 并行

为使通信和计算并行化，采用 non-blocking 的方式，传输子网格的边缘值。更新传输的位置

经分析置于 GPU 计算之前，以充分分散各个数据传输步骤于计算步骤之间来得到更好的异步执行并行特性。

4. 算例测试：Taylor-Green 流的 LBM 数值解与解析解比较

我们给定一个 8192×8192 的网格，模拟 4 个对旋的泰勒涡单元。解析解的速度场给出如下：

$$u_x = -u_0 \cos\left(\frac{2\pi x}{L_x}\right) \sin\left(\frac{2\pi y}{L_y}\right) \exp\left(-t\nu\left(\left(\frac{2\pi}{L_x}\right)^2 + \left(\frac{2\pi}{L_y}\right)^2\right)\right)$$

$$u_y = u_0 \sin\left(\frac{2\pi x}{L_x}\right) \cos\left(\frac{2\pi y}{L_y}\right) \exp\left(-t\nu\left(\left(\frac{2\pi}{L_x}\right)^2 + \left(\frac{2\pi}{L_y}\right)^2\right)\right)$$

使用 LBM 进行数值求解，每隔 2000 步输出系统的总能量 $E$ 和密度、两个方向上速度的 L2 误差，并对速度场和密度场进行磁盘存储，求解 8000 步并计算程序运行速度。

一个典型的测试结果如下所示（单机，24 CPU 线程，2 E5-2680 v3，1 Tesla K40M）：

```
=================CUDA information=================
device: 0
name: Tesla K40m
multiprocessors: 15
compute capability: 3.5
global memory: 11439.9 MiB
free memory: 11293.6 MiB
Maximum number of threads per multiprocessor: 2048
Maximum number of threads per block: 1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
**************************************************
Simulating Taylor-Green vortex decay
domain size: 8192x8192
nu: 0.166667
tau: 1
u_max: 0.02
```

```
rho0: 1
timesteps: 8000
save every: 2000
message every: 2000
MPI information processes: 1
Rank 0: 8192 nodes from y = 0 to y = 8191
Steps, Energy, Density L2 Norm, Ux L2 Norm, Uy L2 Norm
0,    13421.8,   6.02372e-15, 1.56868e-16,  1.50865e-16
2000, 13411.3,   0.00263866,  0.000184236,  0.000184236
4000, 13400.7,   0.00492103,  2.6618e-05,   2.66182e-05
6000, 13390.2,   0.00658027,  0.00014325,   0.000143251
8000, 13379.7,   0.0075498,   7.74777e-05,  7.7478e-05
----- performance information -----
memory allocated: 10242.0 (MiB)
timesteps: 8000
runtime: 1038.360 (s)
speed: 517.04 (Mlups)
bandwidth: 69.3 (GiB/s)
```

可见计算结果是正确的。

作为比较，笔者在相同机器上运行了对等的纯 OpenMP/MPI 代码，使用 24 线程的运算速度大约为 250 Mlups，对比下 GPU 将这个过程加速了整整一倍。

5. 结论

通过 GPU、CPU 和集群机之间的混合并行，可以充分感受到不同粒度并行方式的不同。充分利用不同并行级别的特点可以有效帮助我们解决问题，使高性能计算的特点得到充分的体现。本次作业以并行计算在 CFD 领域的一个应用示例，展示了如何利用 GPU 集群解决大规模科学计算的问题，而且取得了比较好的效果。