计算机科学
COMPUTER SCIENCE

基于MPI+CUDA的DSMC/PIC耦合模拟异构并行及性能优化研究

林拥真, 徐传福, 邱昊中, 汪青松, 王正华, 杨富翔, 李洁

引用本文

林拥真, 徐传福, 邱昊中, 汪青松, 王正华, 杨富翔, 李洁. 基于MPI+CUDA的DSMC/PIC耦合模拟异构并行及性能优化研究[J]. 计算机科学, 2024, 51(9): 31-39.

LIN Yongzhen, XU Chuanfu, QIU Haozhong, WANG Qingsong, WANG Zhenghua, YANG Fuxiang, LI Jie. Heterogeneous Parallel Computing and Performance Optimization for DSMC/PIC Coupled Simulation Based on MPI+CUDA [J]. Computer Science, 2024, 51(9): 31-39.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230300188

基于 MPI+CUDA 的 DSMC/PIC 耦合模拟异构并行及性能优化研究

林拥真[1,2]　徐传福[1,2]　邱昊中[1,2]　汪青松[1,2]　王正华[2]　杨富翔[3,4]　李　洁[3]
1 国防科技大学计算机学院量子信息研究所兼高性能计算国家重点实验室　长沙 410000
2 国防科技大学计算机学院　长沙 410000
3 国防科技大学空天科学学院　长沙 410000
4 军事交通学院　安徽 蚌埠 233000
(lyznudt@nudt.edu.cn)

摘　要　DSMC/PIC 耦合模拟是一类重要的高性能计算应用，大规模 DSMC/PIC 耦合模拟计算量巨大，需要实现高效并行计算。由于粒子动态注入、迁移等操作，基于 MPI 并行的 DSMC/PIC 耦合模拟往往通信开销较大且难以实现负载均衡。针对自主研发的 DSMC/PIC 耦合模拟软件，在原有 MPI 并行优化版本上设计实现了高效的 MPI+CUDA 异构并行算法，结合 GPU 体系结构和 DSMC/PIC 计算特点，开展了 GPU 访存优化、GPU 线程工作负载优化、CPU-GPU 数据传输优化及 DSMC/PIC 数据冲突优化等一系列性能优化。在北京北龙超级云 HPC 系统的 NVIDIA V100 和 A100 GPU 上，针对数亿粒子规模的脉冲真空弧等离子体羽流应用，开展了大规模 DSMC/PIC 耦合异构并行模拟，相比原有纯 MPI 并行，GPU 异构并行大幅缩短了模拟时间，两块 GPU 卡较 192 核的 CPU 加速比达到 550%，同时具有更好的强可扩展性。

关键词：DSMC/PIC 耦合；粒子模拟；异构并行；MPI+CUDA
中图分类号　TP391

Heterogeneous Parallel Computing and Performance Optimization for DSMC/PIC Coupled Simulation Based on MPI+CUDA

LIN Yongzhen[1,2], XU Chuanfu[1,2], QIU Haozhong[1,2], WANG Qingsong[1,2], WANG Zhenghua[2], YANG Fuxiang[3,4] and LI Jie[3]
1 Institute for Quantum Information & State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410000, China
2 College of Computer, National University of Defense Technology, Changsha 410000, China
3 College of Aerospace Science and Engineering, National University of Defense Technology, Changsha 410000, China
4 Army Military Transportation University, Bengbu, Anhui 233000, China

Abstract　DSMC/PIC coupled simulation is an important high-performance computing application that demands efficient parallel computing for large-scale simulations. Due to the dynamic injection and migration of particles, DSMC/PIC coupled simulations based on MPI parallelism often suffer from large communication overheads and are difficult to achieve load balancing. To address these issues, we design and implement efficient MPI+CUDA heterogeneous parallel algorithm based on the self-developed DSMC/PIC simulation software. Combining the characteristics of the GPU architecture and the DSMC/PIC computation, we conduct a series of performance optimizations, including GPU memory access optimization, GPU thread workload optimization, CPU-GPU data transmission optimization, and DSMC/PIC data conflict optimization. We perform large-scale DSMC/PIC coupled heterogeneous parallel simulations on NVIDIA V100 and A100 GPUs in the Beijing Beilong Super Cloud HPC system for the pulsed vacuum arc plasma jet application with billions of particles. Compared to the original pure MPI parallelism, the GPU heterogeneous parallelism significantly reduce simulation time, with a speedup of 550% on two GPU cards compared to 192 cores of the CPU, while maintaining better strong scalability.

Keywords　Coupled DSMC/PIC, Particle simulation, Heterogeneous parallel, MPI+CUDA

1　引言

粒子模拟是一类关键的高性能科学和工程计算应用，在很多领域应用前景广阔。当前主流粒子模拟方法通常耦合了直接蒙特卡罗模拟(Direct Simulation Monte Carlo，DSMC)方法[1]和粒子网格(Particle In Cell，PIC)方法[2-3]。DSMC 可以对稀薄气体流动微观层面的问题进行建模，模拟粒子之间的运动和碰撞等。利用经典的 Bird 算法[4]，DSMC 可以得到与

到稿日期：2023-03-23　返修日期：2023-06-29
通信作者：徐传福(xuchuanfu@nudt.edu.cn)

玻尔兹曼公式一致的结果；PIC 可以跟踪大量粒子的相互作用。耦合 DSMC 和 PIC 方法可以在微观层面模拟大量的粒子及其行为。

等离子体羽流[5-6]模拟是 DSMC/PIC 的重要应用领域之一。等离子体羽流通常包括蒸汽、等离子体、分子簇和表面碎片。蒸汽和等离子体以每秒几十公里的超高速度反喷。脉冲真空电弧产生的等离子体羽流在高稳定性等离子体器件中有重要的应用价值，是航空航天推进[7-9]、核聚变反应堆[10-11]等领域研究的关键。脉冲真空弧等离子体羽流通常在毫米范围内流动，在微秒内引起一系列的复杂热化学非平衡反应和壁相互作用。由于蒸发的粒子数量多、粒子密度大($10^{18}/m^3 \sim 10^{22}/m^3$)，相应的等离子体羽流非常稀薄，克努森数大于0.1，因此无法采用基于纳维-斯托克斯方程的连续介质计算流体动力学(Computational Fluid Dynamics，CFD)方法进行模拟。

虽然 DSMC/PIC 耦合方法可以实现精确的粒子模拟，但相对于传统的 CFD 模拟，大规模 DSMC/PIC 模拟通常需要更多的内存和计算资源[12-13]，因此针对其设计实现高效的并行计算至关重要。传统 MPI 并行 DSMC/PIC 模拟中，由于粒子动态注入、迁移等操作，随着并行规模增大，并行通信开销较大，且随着模拟推进容易出现严重负载不均衡，难以适应数亿量级粒子规模模拟需求。GPU 作为主流的异构并行架构，近年来已成为构建高性能计算系统的重要技术途径，很多科学计算应用在 GPU 上获得了优异性能，结合 GPU 架构和 DSMC/PIC 耦合模拟特点，实现高效 DSMC/PIC 耦合异构并行模拟具有重要意义。

针对自主研发的基于嵌套双重非结构网格 DSMC/PIC 耦合模拟软件，在原有 MPI 并行优化版本上设计实现了大规模高效 MPI+CUDA 异构并行算法，结合 GPU 并行体系结构和 DSMC/PIC 耦合计算特点，开展了 GPU 访存优化、GPU 线程工作负载优化、CPU-GPU 数据传输优化及 DSMC/PIC 数据冲突优化等一系列性能优化。在北京北龙超级云 HPC 系统的 NVIDIA V100 和 A100 GPU 上，成功实现了数亿粒子规模的脉冲真空弧等离子体羽流大规模 DSMC/PIC 耦合异构并行模拟，相比原有纯 MPI 并行，GPU 异构并行大幅缩短了模拟时间，两块 GPU 卡较 192 核的 CPU 加速比达到 550%，同时具有更好的强可扩展性。

2 DSMC/PIC 耦合求解器

本文采用的 DSMC/PIC 耦合求解器包括 DSMC 求解器和 PIC 求解器两个相对独立的部分。DSMC 求解器主要由国防科技大学开发，该求解器除了实现文献[4]中的经典算法模型，同时集成了一些自主研发的模型，模拟流场中粒子的碰撞、壁面相互作用和化学反应等过程[14-15]。PIC 求解器主要由中国工程物理研究院开发，用于模拟电场作用下带电粒子的运动。由于 DSMC，PIC 求解的网格单元大小分别受不同类型粒子限制，为了实现 DSMC 与 PIC 的耦合求解，设计了基于嵌套双重非结构网格的耦合方法。如图 1 所示，DSMC 模拟采用粗网格，网格单元尺度满足分子平均自由程约束[1)]；PIC 模拟采用细网格，网格单元尺度符合德拜长度[2)]要求。具体而言，在网格生成时首先生成 DSMC 粗网格，再将每个 DSMC 粗网格单元划分为 8 个 PIC 细网格单元，所有的 PIC 细网格单元完全嵌入一个 DSMC 粗网格单元中。

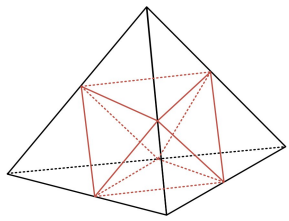
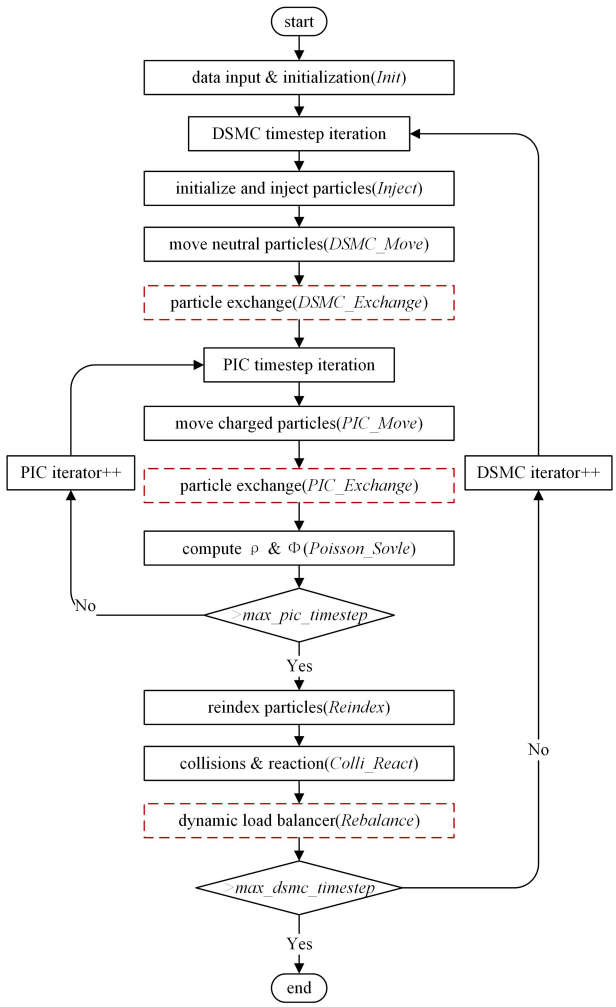


图 1 一个粗四面体 DSMC 网格单元有 8 个 PIC 细网格单元

Fig. 1 A coarse tetrahedral DSMC grid cell with 8 fine-grained tetrahedralgrid cells for PIC



注：虚线矩形表示 MPI 优化版本实现的并行通信和负载均衡。

图 2 DSMC/PIC 耦合求解器的整体流程

Fig. 2 Overall workflow of coupled DSMC/PIC solver

基于嵌套双重非结构，以 DSMC 求解流程为主，集成 PIC 求解流程，实现了 DSMC/PIC 耦合求解。图 2 给出了耦合求

1) 平均自由程是粒子(例如原子、分子或光子)运动时与其他粒子的一次或多次连续碰撞而显著改变其方向或能量的平均距离。

2) 德拜长度也叫德拜半径，一般指宏观电场有意义的最小长度。

解的整体工作流程。程序读入网格、模拟参数并初始化相关数据结构后进入时间步迭代。

DSMC 时间步主要包括：

1）粒子注入（Inject）：根据麦克斯韦分布和入口位置等条件注入并初始化模拟粒子。

2）模拟中性粒子运动（DSMC_Move）：每个 DSMC 时间步中性粒子会以恒定的速度直线移动，可以跨越不同的网格单元，甚至移动出计算域。

3）PIC 时间步迭代：一个 DSMC 时间步通常包含多个 PIC 时间步，DSMC 或 PIC 时间步的大小取决于粒子的平均自由程[16]，PIC 时间步迭代过程参考后文的介绍。

4）粒子重编号（Reindex）：粒子位置很可能在迭代过程中发生变化，求解器以统一的方式对所有粒子重编号，同时删除移出计算域的粒子，以确保每个粒子都有唯一的全局索引。

5）碰撞和反应（Colli_React）：采用 NTC 方法[17]选择碰撞对，根据粒子类型和碰撞能量来判断两个粒子是否会发生化学反应，实现了各种碰撞和化学反应模型[15]。

PIC 时间步包括两个主要步骤：

1）模拟带电粒子运动（PIC_Move）：每个 PIC 时间步中，带电粒子由前一个时间步的电场驱动。

2）求解泊松方程（Poisson_Solve）：带电粒子的速度 v 和位置 r 可以通过求解等离子体物理动力学方程[15]得到：

$$\begin{cases}\dfrac{\mathrm{d}r}{\mathrm{d}t}=V\\ m\dfrac{\mathrm{d}v}{\mathrm{d}t}=q(E+V\times B)\end{cases}\tag{1}$$

其中，m 为粒子的质量（由用户给出），q 为粒子的电荷，B 为磁场密度，E 为电场强度。本文只考虑静电场，假设不存在磁场（$B=0$）或恒定磁场（即 B 为用户给出的常数）。本文采用 *Boris* 方法[18]计算速度 v 的数值，然后通过向网格节点插值粒子电荷来计算电荷密度 q，跟踪带电粒子的运动。电场强度 E 通过求解静电的泊松方程来计算。

$$\nabla\cdot E=\frac{\rho}{\varepsilon_0}\tag{2}$$

其中，ρ 是一个总体积电荷密度，ε_0 是介质的介电常数，∇ 是拉普拉斯算子。电场强度 E 可以表示为电势的负梯度 $\varnothing$：

$$E=-\nabla\varnothing\tag{3}$$

通过用 $-\nabla\varnothing$ 替换 E，泊松方程转换为：

$$\nabla\cdot E=\nabla\cdot(-\nabla\varnothing)=-\nabla^2\varnothing=\frac{\rho}{\varepsilon_0}\tag{4}$$

在非结构网格上使用有限体积法[19]对式（4）进行离散求解需要构建线性系统：

$$K\varnothing=b\tag{5}$$

其中，K 为根据网格拓扑构造的对角占优刚度矩阵，b 由边界条件和电势推导而来。通过线性方程组求解获得 $\varnothing$ 后，根据式（3）计算 E。

前期工作中[20-21]，课题组针对 DSMC/PIC 耦合求解器设计实现了基于集中通信策略和基于分布式通信策略的 MPI 并行，实现了仿真粒子在任意 MPI 进程之间的迁移。此外，针对模拟过程中粒子动态注入、迁移等导致的负载不均衡，设计实现了动态复杂均衡算法，大幅提升了大规模 MPI 并行效率，形成了 MPI 优化版本的 DSMC/PIC 耦合计算软件。图 2 中的虚线框给出了 MPI 并行通信（PIC_Exchange 和 DSMC_Exchange）以及动态负载均衡（Rebalance）部分。本文在此基础上设计实现了 MPI+CUDA 版本的 DSMC/PIC 耦合程序，该程序除了输入、输出、粒子通信以及负载均衡模块，图 1 中的 Inject，DSMC_Move，PIC_Move，Poisson_Solve，Reind-ex 和 Colli_React 模块均在 GPU 端实现。

3 MPI+CUDA 异构并行

尽管 MPI 并行 DSMC/PIC 耦合模拟软件实现了动态负载均衡，一定程度提升了 MPI 并行效率，但随着粒子模拟规模的增加，大规模 MPI 并行可扩展性下降，无法满足亿级粒子以上复杂应用的模拟需求。在 MPI 并行基础上，面向 GPU 异构高性能计算机，实现 MPI+CUDA 异构并行，能够发挥 GPU 计算能力强的优势，大幅减少模拟时间。此外，在模拟同等问题规模时，MPI+CUDA 异构并行所需的 MPI 并行进程更少，对于超大规模 DSMC/PIC 耦合模拟具有重要的应用价值。下面将结合自主 DSMC/PIC 耦合软件计算流程、数据结构和 GPU 并行体系结构的特点，介绍 GPU 并行算法设计及其性能优化。

3.1 线程工作负载优化

通常一个 GPU 线程可模拟一个仿真粒子，但模拟中性粒子和带电粒子运动时需要使用随机数，粒子规模增加时（例如本文算例包含数亿粒子），每个 GPU 线程运行 *curand_init*() 函数初始化随机数产生器开销巨大（对于本文算例耗时十几分钟），严重影响了整体模拟效率。针对粒子运动（DSMC_move 和 PIC_move），优化设计了线程工作负载方式，减少了线程数量和随机数生成器初始化开销。如图 3 所示，圆圈表示粒子，矩阵表示线程。本文将 GPU 线程与粒子的关系改为一对多的映射关系，由一个 GPU 线程处理多个粒子，这样既能缩短随机数的初始化时间，也能提高内核对数据的重用。为了提高 GPU 全局内存访问效率，一个线程束中的 32 个线程分别模拟连续的 32 个粒子，以实现合并内存访问。具体随机数初始化算法和线程工作负载优化方法如算法 1 和算法 2 所示。

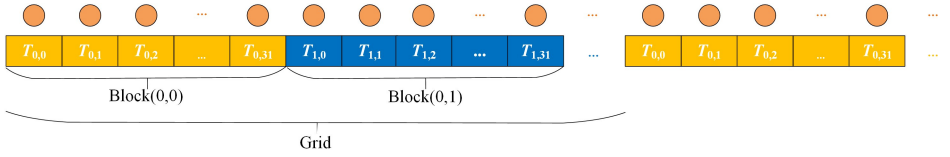


图 3 线程与粒子映射关系图

Fig. 3 Thread and particle mapping diagram

算法 1 随机数初始化算法

```
Input:seed
Output:curandStates[]
1. function RNG(seed,curandStates[])
2.    RNG_num←2E20
3.    block_x ←32
4. grid_x←RNG_num / block_x
5. init<<<grid_x,block_x>>>(seed,curandStates[])
6. end function
7. function init(seed,curandStates[])
8.    index←threadIdx.x + blockDim.x * blockIdx.x
9.    curand_init(seed,index,0,curandStates[index])
10. end function
```

算法 2 线程工作负载优化算法

```
Input:curandStates[];particle_num
Output:randnum;particle_move
1. function move(curandStates[])
2.    block_x ←32
3.    if particle_num <RNG_num then
4.       grid_x←particle_num / block_x
5.    else
6.       grid_x←RNG_num / block_x
7.    end if
8.    process<<<grid_x,block_x>>>(curandStates[])
9. end function
10. function process(curandStates[])
11.    index←threadIdx.x + blockDim.x * blockIdx.x
12.    for i=threadIdx.x→ particle_num do
13.          使用 curand_uniform_double(curandStates[index])生成随
             机数
14.          模拟粒子 i 的运动
15.          index←index + gridDim.x * blockDim.x
16. end for
17. end function
```

算法 1 根据模拟的粒子规模，初始化固定数量的随机数生成器，意味着在模拟中性粒子和带电粒子运动时核函数能启动的最大线程数应小于等于随机数生成器的数量。算法 2 中每个线程负责模拟多个粒子的运动，且固定使用同一个随机数生成器。在线程处理粒子的循环中，线程首先处理编号与其在线程网格中的全局编号一致的粒子，一次循环的跨度是一整个线程网格规模，这样同一个线程束处理的一定是相邻的粒子，提高了内存带宽的利用率。

3.2 数据冲突处理

粒子重编号模块(Reindex)主要是将同网格中的各类型粒子进行统一编号。它的实现包括 4 个紧密耦合的步骤：第一步统计各类型粒子的数量，同时分别统计每个 DSMC 网格中各种粒子的数量；第二步统计各类型粒子在全局编号中的起始位置，这里需要用到第一步中统计的各类型粒子的数量；第三步需要用到第二步中得到的各类型粒子在全局编号中的起始位置，继而求出每个 DSMC 网格中各类型粒子在全局编号中的起始位置；第四步需要用到求得的各 DSMC 网格中各类型粒子在全局编号中的起始位置以及在第一步中获得的各 DSMC 网格中各种粒子的数量，继而给每个粒子一个唯一的全局索引。为解决上述各步骤中的数据依赖问题，本文采用 kernel 分割以实现全局同步。kernel 分割方法具体如算法 3 所示。

算法 3 kernel 分割算法

```
Input:particles[]
Output:index[]
1. processStageOne<<<…,…>>>(…)
2. cudaDeviceSynchronize()
3. processStageTwo<<<…,…>>>(…)
4. cudaDeviceSynchronize()
5. processStageThree<<<…,…>>>(…)
6. cudaDeviceSynchronize()
7. processStageFour<<<…,…>>>(…)
8. cudaDeviceSynchronize()
```

此外，在模拟带电粒子运动的模块中，带电粒子的运动会导致电荷密度的更新。由于电荷密度是存储在非结构网格节点上的，在更新相邻的两个网格单元时可能会引起数据冲突。为了解决该潜在的数据冲突，本文利用原子函数对电荷密度数组的更新进行加锁，防止 GPU 线程之间互相干扰。

3.3 访存优化

粒子注入模块(Inject)是 DSMC/PIC 耦合模拟中耗时最长的部分，本文在将该模块移植到 GPU 时，发现其耗时仍然很长，加速效果不理想。其原因是粒子注入算法需要搜索注入的带电粒子的 DSMC 网格编号，由于大规模模拟时 DSMC 网格单元数较大，因此 GPU 全局内存访问延时高。基于以上分析，本文使用共享内存显式管理缓存，降低访存延时。访存优化的具体方法如算法 4 所示。

算法 4 访存优化算法

```
Input:inlet_cell[]
Output:particle_cell
1. is_find_cell←false
2. if threadIdx ==0 then
3.    unfind_cell_threadnums(shared)←blockDim
4. end if
5. __syncthreads()
6. while unfind_cell_threadnums ! =0 do
7.    if is_find_cell ==false then
8.       计算粒子位置、速度等参数
9.    end if
10.   iteration_num←inlet_cell_num / shared_array_size
11.   for i=0 →iteration_num do
12.      for myduty=threadIdx →shared_array_size do
13.         offset ←i * shared_array + myduty
14.            shared_array[myduty]=inlet_cell[offset]
15.         myduty←myduty + blockDim
16.   end for
17.      __syncthreads()
18.      for j=0 →shared_array_sizedo
19.         if is_find_cell ==true then
20.            return
21.         end if
22.         判断 shared_array[j] 是否为粒子网格
23.         if is_find_cell ==true then
24.            atomicSub(&unfind_cell_threadnums,1)
25.               break
26.            end if
27.      end for
```

```
28.     _syncthreads()
29.       if unfind_cell_threadnums ==0 then
30.     break
31.   end if
32. end for
33. end while
```

在访存优化算法中，本文利用共享内存来减少线程对全局内存的访问次数，充分利用共享内存的低延迟。该算法中，一个线程负责一个粒子的注入。每个线程块内的各个线程首先需要并行地将全局内存中的网格数据存入共享内存中，然后各线程遍历共享内存中的网格数据，减少访问全局内存的次数。由于线程块内的线程找到 DSMC 网格的进度不一致且循环内有同步，因此不能简单地让某个线程跳出循环，否则会造成块中线程无期限地等待。线程块内各线程需等待块中所有线程都执行完成，才能跳出寻找 DSMC 网格的循环。本算法使用变量 *unfind_cell_threadnums* 来控制循环的结束，使用变量 *is_find_cell* 来控制线程的执行路径，以使各线程能按不同执行路径到达同步点。

假设 DSMC 网格数为 N 个，线程块大小为 M。那么如果不使用共享内存，考虑最差的情况，一个线程块需要访问全局内存 NM 次。引入共享内存后，虽然增加了同步的开销，但一个线程块只需要访问全局内存 N 次，全局内存访问次数减少为原来的 $1/M$。在实际测试中，我们也验证了使用共享内存能带来更好的性能。

3.4 CPU 与 GPU 数据交互并行实现

在基于 MPI+CUDA 的 DSMC/PIC 耦合模拟中，粒子可能在不同 MPI 进程间不断迁移。由于粒子的注入和运动都是在 GPU 上实现的，MPI 并行运行在 CPU 上，因此需要在 CPU 与 GPU 之间传输粒子。在 MPI+CUDA 异构并行程序各进程数据交互的过程中，每个进程首先将迁移的粒子从 GPU 传输到 CPU，再通过 MPI 将粒子传输到目的进程，最后目的进程把接收到的粒子从 CPU 传输到 GPU。

在 CPU 与 GPU 数据交互中，统计迁移的粒子数量和打包迁移粒子非常耗时，占据了整个数据交互过程的大部分时间，因此有必要设计一种高效的 GPU 并行策略来统计迁移粒子的数量以及打包粒子。算法 5 是本文提出的 CPU 与 GPU 数据交互的并行算法和数据传输优化方法。在该算法中，首先调用 *countParticlesNum* 函数分别统计迁移到各个进程的粒子数，这里使用原子锁实现各线程互斥写入。为了充分利用内存带宽，本文将需要从 GPU 上传输下来的粒子进行打包，包括对粒子各参数进行打包和对各进程迁移的粒子进行打包。由于粒子的参数有 double 和 int 两种类型，因此采用两个数组进行打包。打包数组存储迁移到各进程的所有粒子，这里利用 *pos* 数组来记录迁移到各进程的粒子在打包数组中的起始位置。在 *package* 函数(将需要传输的数据进行打包)中，首先将 *counter* 数组各元素初始化为 0，然后利用原子锁对 *counter* 数组上的各元素进行原子自增，原子函数返回自增前的值，作为粒子唯一的编号 idx。迁移的粒子存入打包数组的位置是唯一的，由其迁移的目的进程号和原子锁返回的编号决定，因此可将数据并行打包。需要注意的是，迁出的粒子不做实际删除，仅仅只是作标记，在后续模拟中，跳过对已标记粒子的处理，这样虽然会浪费一定存储空间，但是可以节省顺序存储数组时删除操作的开销。

算法 5 CPU 与 GPU 数据交互并行算法

```
Input:particles[]
Output:new_particles[]
1. function countParticlesNum(particles[],count[])
2.     index←threadIdx.x + blockDim.x * blockIdx.x
3.     if partciles[index].rank ! =myRank then
4.       atomicAdd(&count[partciles[index].rank],1)
5.   end if
6. end function
7. funtion package(particles[])
8.   index←threadIdx.x + blockDim.x * blockIdx.x
9.   if particles[index].rank ==myRank then
10.    return
11.  end if
12.  offset_int←pos[particles[index].rank] * 5
13.  offset_double←pos[particles[index].rank] * 6
14.  idx←atomicAdd(&counter[particles[index].rank],1)
15.  result_int[offset_int + idx * 5 + 0]=particles[index].arg0
     …
16.    result_int[offset_int + idx * 5 + 4]=particles[index].
arg4
17.  result_double[offset_double+idx*6+0]=particles[index].
     arg5…
18.  result_double[offset_double+idx*6+5]=particles[index].arg9
19.  对迁出粒子作移除标记
20. end function
21. function communication(particles[])
22.  countParticlesNum<<<…,…>>>(particles[],count[])
23.  pos[0] ←0
24.  for i=1 →rank_num then
25.    pos[i]←count[i−1] + pos[i−1]
26.  end for
27.  package<<<…,…>>>(…)
28.  使用分布式通信将迁出的粒子传输给各进程
29.  将接收到的粒子传输到设备端
30. end function
```

4 实验结果与分析

4.1 实验设置

本文实验是在北京北龙超级云计算平台的 N26 分区和 N32-E 分区上进行的。N26 分区每个计算节点配置 8 块 NVIDIA Tesla V100-SXM2 32GB 显存的 GPU，以及一块 Intel(R) Xeon(R) Platinum 8255C CPU@2.50 GHz，计算节点间通过 25 Gbps 的 TCP 网络互联，不支持 RDMA 协议。N32-E 分区每个计算节点配置 8 块 NVIDIA A100 PCIe 40GB 显存的 GPU，以及两块 Intel 6248R 48C CPU@3.0 GHz。计算节点间通过 100 Gbps RoCE 网络进行高速互联，支持 RDMA 协议。本文采用 C++语言实现 DSMC/PIC 耦合模拟，编译器为 GCC v4.8.5 和 CUDA v11.5.0。MPI 版本是 OPENMPI v4.1.2，其他使用的软件或库包括 METIS v5.1.0，LAPACK/3.5.0 和 BLAS/3.5.0。

本实验的基准程序采用的是由课题组前期迭代开发的最优版本的 MPI 程序[20-21]，该程序采用分布式 MPI 并行通信策略以及动态负载均衡算法。测试平台为北京北龙超级云计算平台的 T6 分区，其每个计算节点的配置是：Intel(R) Xeon(R) Platinum 9242 CPU @ 2.30GHz，96 核，内存 384GB，节点间采用 Infiniband 互连。使用的编译器为 GCC v4.8.5，MPI 版本是 OPENMPI v4.1.2。其他使用的软件或库包括 METIS v5.1.0，LAPACK/3.5.0 和 BLAS/3.5.0。

测试算例模拟了脉冲真空电弧在三维圆柱喷管内等离子体羽流的扩散、碰撞和反应过程，网格包含 2 242 948 个用于 PIC 模拟的细四面体网格单元以及 268 479 个用于 DSMC 模拟的粗四面体网格单元。中性粒子密度设置为 1.4×10^{20}，模拟的中性粒子规模达到 3.2×10^{9} 个。带电粒子的密度设置为 1.25×10^{12}，需要处理的带电粒子数为 2×10^{9}。DSMC/PIC 耦合模拟运行 100 个 DSMC 时间步，每个 DSMC 时间步包含两个 PIC 时间步，取多次模拟运行时间的平均值。

4.2 实验结果分析

图 4 和表 1 分别给出了采用纯 MPI 并行和 MPI+CUDA 异构并行实现 DSMC/PIC 耦合模拟的整体性能，MPI+CUDA 异构并行分别运行在 V100 和 A100 两种 GPU 上。由于本文的算例配置计算开销较大，无法串行运行，因此将两个 CPU 计算节点(192 个处理器核)的 MPI 并行程序作为性能基准，扩展到 32 个计算节点(3072 个处理器核)。作为对比，GPU 卡从 2 块扩展到 32 块，MPI+CUDA 异构并行程序采用一个 MPI 进程控制 1 块 GPU 卡的方式运行。从图 4 可以看出，异构并行性能优于纯 MPI 并行：V100 上异构并行与纯 MPI 并行相比，加速比最多达 443%，最小 367%；A100 上异构并行与纯 MPI 并行相比，加速比最多达 550%，最小能达到 466%。A100 相比 V100 平均性能提升达 23%。同时也可以观察到，异构并行在 2 个 A100 卡时加速比最大，扩展到 32 块 A100 卡时逐渐下降；在 V100 上加速比整体却是逐渐上升的。图 5 给出了异构并行和纯 MPI 并行的强可扩展性。可以看出，异构并行在 V100 上的强可扩展性也优于纯 MPI 并行。但在 A100 上的强可扩展性弱于纯 MPI 并行，尤其是 GPU 卡较多时。导致 A100 加速比和强可扩展性弱于 V100 的原因可能是算例问题规模不够大。

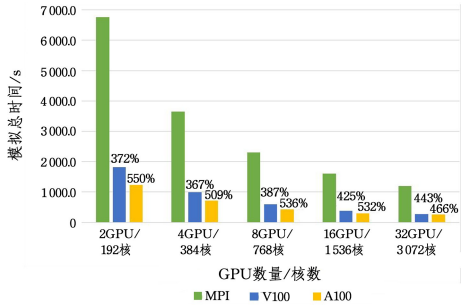


图 4 异构并行程序和 MPI 并行程序执行时间对比

Fig. 4 Comparison of execution times of heterogeneous parallel program and MPI parallel program

表 1 各版本程序模拟总时间

Table 1 Total simulation time for each version

(s)

	2GPU/192 核	4GPU/384 核	8GPU/768 核	16GPU/1536 核	32GPU/3072 核
MPI	6762.0	3643.1	2297.8	1601.1	1193.6
V100	1818.5	991.5	593.8	376.9	269.5
A100	1228.6	716.1	428.8	300.7	255.9

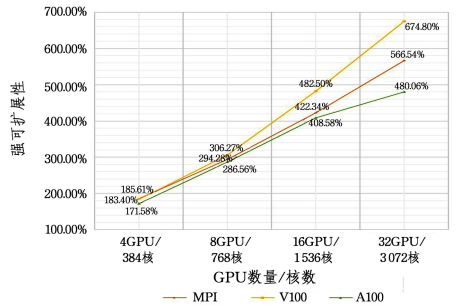


图 5 异构并行程序和 MPI 并行程序的强可扩展性

Fig. 5 Strong scalability of heterogeneous parallel program and MPI parallel program

表 2 列出了 MPI 并行和 A100 上异构并行各模块的加速比。可以看到，除了 Reindex，Colli_React 和 DSMC_Exchange，其他模块的加速比都很理想。类似地，表 3 列出了 MPI 并行和 V100 上异构并行各模块的加速比，整体而言，V100 异构并行加速比差于 A100。

表 2 MPI 并行程序和在 A100 计算节点上运行的异构并行程序主要模块执行时间

Table 2 Main module execution times of MPI parallel program and heterogeneous parallel program running on A100 compute node

(s)

模块		2GPU/192 核	4GPU/384 核	8GPU/768 核	16GPU/1536 核	32GPU/3072 核
DSMC_Move	MPI	37.68	19.02	9.88	5.49	2.96
	A100	5.20	3.03	2.16	1.17	0.65
	加速比/%	724.59	627.17	457.70	469.03	454.32
DSMC_Exchange	MPI	14.44	9.54	6.50	5.84	8.58
	A100	11.81	12.64	13.60	10.35	11.21
	加速比/%	122.26	75.52	47.77	56.42	76.48
PIC_Inject	MPI	6111.87	3030.43	1513.05	767.86	379.98
	A100	1073.68	566.17	294.93	173.18	116.86
	加速比/%	569.25	535.25	513.01	443.40	325.15
PIC_Move	MPI	53.97	40.14	35.69	22.40	12.84
	A100	7.70	5.17	4.13	2.53	1.67
	加速比/%	701.24	776.30	864.12	884.75	767.08

（续表）

模块		2GPU/192 核	4GPU/384 核	8GPU/768 核	16GPU/1536 核	32GPU/3072 核
PIC_Exchange	MPI	108.43	100.10	159.38	222.62	236.19
	A100	24.25	25.15	21.88	18.57	26.57
	加速比/%	447.20	397.96	728.56	1198.84	888.99
Poisson_Sovle	MPI	339.51	348.63	371.52	388.59	405.48
	A100	55.61	59.68	57.97	52.76	52.02
	加速比/%	610.57	584.17	640.85	736.48	779.45
Reindex	MPI	1.46	0.87	0.67	0.34	0.19
	A100	17.54	11.95	9.84	4.49	0.75
	加速比/%	8.31	7.30	6.79	7.55	25.61
Colli_React	MPI	0.14	0.15	0.12	0.09	0.08
	A100	6.96	6.34	5.81	1.36	0.25
	加速比/%	2.08	2.39	2.11	6.36	30.06

表 3　MPI 并行程序和在 V100 计算结点上运行的异构并行程序主要模块执行时间

Table 3　Main module execution times of MPI parallel program and heterogeneous parallel program running on V100 compute node

（s）

模块		2GPU/192 核	4GPU/384 核	8GPU/768 核	16GPU/1536 核	32GPU/3072 核
DSMC_Move	MPI	37.68	19.02	9.88	5.49	2.96
	A100	8.79	4.98	3.56	1.83	1.05
	加速比/%	428.62	381.83	277.30	300.42	282.88
DSMC_Exchange	MPI	14.44	9.54	6.50	5.84	8.58
	A100	9.80	10.46	12.22	15.31	11.17
	加速比/%	147.36	91.25	53.16	38.15	76.80
PIC_Inject	MPI	6111.87	3030.43	1513.05	767.86	379.98
	A100	1618.37	816.08	420.41	220.32	116.82
	加速比/%	377.66	371.34	359.90	348.52	325.28
PIC_Move	MPI	53.97	40.14	35.69	22.40	12.84
	A100	24.41	13.20	9.17	4.19	1.58
	加速比/%	221.13	304.08	389.37	534.08	815.36
PIC_Exchange	MPI	108.43	100.10	159.38	222.62	236.19
	A100	24.08	21.62	21.78	24.75	24.45
	加速比/%	450.28	462.97	731.78	899.60	965.94
Poisson_Sovle	MPI	339.51	348.63	371.52	388.59	405.48
	A100	99.14	98.93	108.57	85.12	70.07
	加速比/%	342.47	352.42	342.19	456.54	578.69
Reindex	MPI	1.46	0.87	0.67	0.34	0.19
	A100	13.41	7.90	5.92	2.19	0.75
	加速比/%	10.86	11.04	11.28	15.53	25.42
Colli_React	MPI	0.14	0.15	0.12	0.09	0.08
	A100	7.83	5.93	5.37	1.04	0.22
	加速比/%	1.85	2.56	2.28	8.25	34.50

图 6 给出了访存优化采用不同大小共享内存对程序执行时间的影响。大小为 32 的共享内存大小指开辟能够存储 32 个 DSMC 网格单元信息的共享内存空间，以此类推。该结果是在 2 块 V100 GPU 上测试的，由于 V100 上每个流多处理器可使用的共享内存空间有限，仅测试到 480。可以看到，开辟大小为 128 的共享内存执行时间是最优的。

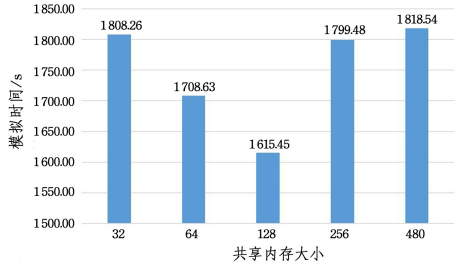


图 6　共享内存大小对 PIC_Inject 模块执行时间的影响

Fig. 6　Influence of shared memory size on PIC_Inject module execution time

图 7 给出了 V100 上访存优化对 PIC_Inject 模块的优化效果，该结果是在 2 块 V100 上测试的。可以看出，与没有优化的 baseline 版本相比，优化后的程序加速比最高能达到 243.77%，最小也能达到 140.70%。

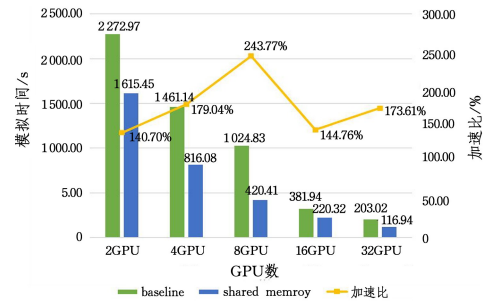


图 7　访存优化对 PIC_Inject 模块执行时间的影响

Fig. 7　Influence of access optimization on PIC_Inject module execution time

表 4 列出了在不同的线程工作负载方式下随机数初始化

的执行时间，该结果是在 2 块 V100 上测试的。表 4 中细粒度指一个线程在一个模拟时间步内仅处理一个中性粒子或带电粒子的运动，意味着需要给每个线程初始化一个随机数。粗粒度指一个线程在一个模拟时间步内处理多个粒子，核函数启动的线程数量是固定的，则初始化随机数的数量也是固定的。可以看出，粗粒度线程工作负载的随机数初始化时间远短于细粒度方式。

表 4 不同的线程工作负载方式下随机数初始化执行时间

Table 4 Random number initialization execution time under different thread workloads

线程工作负载方式	2GPU	4GPU	8GPU	16GPU	32GPU
细粒度	1072.84	741.41	174.21	89.34	41.32
粗粒度	1.49	1.68	2.15	2.15	2.16
加速比/%	72183	44094	8114	4146	1912

图 8 给出了不同线程工作负载方式下 PIC_Move 和 DSMC_Move 模块的执行时间，该结果是在 2 块 V100 上测试的。可以看出，大多数情况下粗粒度的线程工作负载方式是优于细粒度的线程工作负载方式的，这说明增加每个线程的工作量，某些情况下确实会提高内核对数据的重用，继而提升性能。

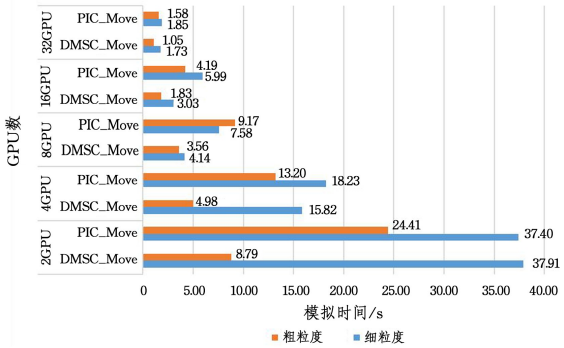


图 8 不同的线程工作负载方式下 PIC_Move 和 DSMC_Move 模块的执行时间

Fig. 8 Execution times of PIC_Move and DSMC_Move modules under different thread workloads

表 5 和表 6 分别列出了在 A100 和 V100 上各模块的核函数在使用不同线程块维度时的执行时间。这里仅列出维度为 32，64，128，256，512，1024 的测试结果。

表 5 A100 上各模块核函数使用不同块维度的执行时间

Table 5 Execution time of each module kernel function using different block dimensions on A100

(s)

BlockDim	32	64	128	256	512	1024
DSMC_Move	**4.29**	4.33	4.34	4.37	4.43	5.80
PIC_Inject	1112.61	1141.76	1139.65	1112.28	1083.67	**1070.07**
PIC_Move	**7.19**	7.22	7.24	7.24	7.34	8.07
Reindex_kernel1	11.14	**11.01**	11.28	11.07	11.06	12.71
Reindex_kernel2	3.64	3.30	3.30	3.30	**3.28**	4.93
DSMC_Exchange	3.62	3.55	**3.53**	3.67	3.66	3.76
PIC_Exchange	5.58	**5.02**	5.22	5.09	4.65	5.17
Colli_React	**6.84**	6.95	6.99	7.02	7.05	7.80

表 6 V100 上各模块核函数使用不同块维度的执行时间

Table 6 Execution time of each module kernel function using different block dimensions on V100

(s)

BlockDim	32	64	128	256	512	1024
DSMC_Move	**7.92**	7.94	7.94	7.96	7.94	30.00
PIC_Inject	1686.04	**1615.29**	1720.18	1814.65	3114.37	4144.45
PIC_Move	25.23	24.85	24.29	**23.97**	24.55	51.32
Reindex_kernel1	8.67	8.68	8.67	8.67	8.66	**8.65**
Reindex_kernel2	1.91	1.91	**1.90**	**1.90**	1.91	1.91
DSMC_Exchange	1.26	1.16	**1.15**	1.16	1.18	1.20
PIC_Exchange	**2.98**	3.22	3.22	3.17	3.07	**2.98**
Colli_React	**6.79**	6.97	7.03	7.84	9.18	9.24

可以看出，在 A100 上不同块维度下核函数的执行时间波动不大。但在 V100 上有部分核函数(DSMC_Move，PIC_Inject，PIC_Move)在块维度为 1024 时执行时间明显增加，这是因为每个线程块能使用的寄存器数量有限制，导致线程块中每个线程能用到的寄存器数变少，而这些模块中线程使用寄存器的数量较多，因此将块维度设置为 1024 时执行时间波动较大。

结束语 本文针对自主 DSMC/PIC 耦合模拟软件开展了基于 MPI+CUDA 的异构并行计算和性能优化研究。结合 GPU 体系结构、DSMC/PIC 耦合模拟特点，设计实现了大规模高效 MPI+CUDA 异构并行算法及一系列 GPU 性能优化方法，大幅提升了 DSMC/PIC 耦合模拟性能，对于支撑 DSMC/PIC 耦合模拟大规模工程应用具有重要价值。

参 考 文 献

[1] BIRD G A. Molecular gas dynamics [J]. NASA STI/Recon Technical Report A, 1976, 76: 40225.

[2] BIRDSALL C K, LANGDON A B. Plasma physics via computer simulation[M]. CRC Press, 2004.

[3] HOCKNEY R W, EASTWOOD J W. Computer simulation using particles[J]. SIAM Review, 1983, 3(25): 1025102.

[4] BIRD G A. Direct simulation and the Boltzmann equation [J]. The Physics of Fluids, 1970, 13(11): 2676-2681.

[5] COPPLESTONE S, ORTWEIN P, MUNZ C D, et al. Coupled PIC-DSMC simulations of a laser-driven plasma expansion[C]// High Performance Computing in Science and Engineering'15: Transactions of the High Performance Computing Center, Stuttgart(HLRS) 2015. Springer International Publishing, 2016: 689-701.

[6] COPPLESTONE S, MUNZ C D, PFEIFFER M. PIC-DSMC simulations of plasma plume expansions with ionization and recombination processes[C]// 2016 IEEE International Conference on Plasma Science(ICOPS). IEEE, 2016.

[7] SMITH B D, BOYD I D, KAMHAWI H, et al. Hybrid-PIC modeling of a high-voltage, high-specific-impulse hall thruster [C]// 49th AIAA/ASME/SAE/ASEE Joint Propulsion Conference. 2013: 3887.

[8] KORKUT B, LI Z, LEVIN D A. 3-D simulation of ion thruster plumes using octree adaptive mesh refinement[J]. IEEE Transactions on Plasma Science, 2015, 43(5): 1706-1721.

[9] BRIEDA L, ZHUANG T S, KEIDAR M. Near plume modeling

of a micro cathode arc thruster[C]// 49th AIAA/ASME/SAE/ASEE Joint Propulsion Conference. 2013:4120.

[10] TACCOGNA F, MINELLI P, BRUNO D, et al. Kinetic divertor modeling[J]. Chemical Physics, 2012, 398: 27-32.

[11] GLEASON-GONZÁLEZ C, VAROUTIS S, HAUER V, et al. Simulation of neutral gas flow in a tokamak divertor using the Direct Simulation Monte Carlo method[J]. Fusion Engineering and Design, 2014, 89(7/8): 1042-1047.

[12] XU C, ZHANG L, DENG X, et al. Balancing cpu-gpu collaborative high-order cfd simulations on the tianhe-1a supercomputer [C]// 2014 IEEE 28th International Parallel and Distributed-Processing Symposium. IEEE, 2014: 725-734.

[13] XU C, DENG X, ZHANG L, et al. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer[J]. Journal of Computational Physics, 2014, 278: 275-297.

[14] LI J, INGHAM D, MA L, et al. Numerical simulation of the chemical combination and dissociation reactions of neutral particles in a rarefied plasma arc jet[J]. IEEE Transactions on Plasma Science, 2017, 45(3): 461-471.

[15] SU Y, LI J, WANG H, et al. Numerical simulation of chemical reactions on rarefied plasma plume by DSMC method[J]. IEEE Transactions on Plasma Science, 2021, 49(3): 1214-1226.

[16] BIRD G A. Definition of mean free path for real gases[J]. The Physics of fluids, 1983, 26(11): 3222-3223.

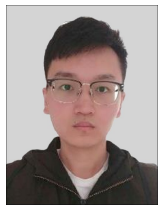
[17] BIRD G A. Molecular gas dynamics and the direct simulation of gas flows[M]. Oxford University Press, 1994.

[18] BORIS J P. Relativistic plasma simulation-optimization of a hybrid code[C]// International Conference on Numerical Simulation of Plasmas. 1970: 3-67.

[19] LIMA E R A, TAVARES F W, BISCAIA JR E C. Finite volume solution of the modified Poisson-Boltzmann equation for two colloidal particles[J]. Physical Chemistry Chemical Physics, 2007, 9(24): 3174-3180.

[20] QIU H Z. Parallel computing of DSMC/PIC Hybrid Numerical Simulation for Three-dimensional Unsteady pulsed Vacuum Arc Plasma Plumes [C]// 2021 HPC China. 2021: 483-491.

[21] QIU H, XU C, LI D, et al. Parallelizing and Balancing Coupled DSMC/PIC for Large-scale Particle Simulations [C] // 2022 IEEE International Parallel and Distributed Processing Symposium(IPDPS). IEEE, 2022: 390-401.



LIN Yongzhen, born in 1997, postgraduate. His main research interests include parallel computing and applications.

XU Chuanfu, born in 1980, Ph.D, associate researcher, master supervisor. His main research interests include parallel computing and large-scale science and engineering computing.

（责任编辑：喻藜）