基于 NVIDIA Kepler 的 PIC 方法并行*

文敏华[1+]，林新华[1]，Simon Chong Wee See[1,2]

[1](上海交通大学 高性能计算中心,上海 200240)

[2](NVIDIA Corporation)

A NVIDIA Kepler Based Acceleration of PIC Method

WEN Minhua[1+], James LIN[1], Simon Chong Wee SEE [1,2]

[1](Center for High Performance Computing, Shanghai Jiao Tong University, Shanghai 200240, China)

[2](NVIDIA Corporation)

+ Corresponding author: Phn: +86-13917675084, E-mail: wenminhua@sjtu.edu.cn

Abstract: The PIC (Particle-In-Cell) method has been widely used in computational plasma physics. However, a large number of computational particles need to be simulated in order to get high accuracy, which demands for great compute capacity. Therefor it becomes necessary to accelerate the PIC method to reduce the time cost. In this paper, we design a NVIDIA Kepler GPU based PIC algorithm and implement it using CUDA (Compute Unified Device Architecture). The most time consuming parts of PIC method, namely **collision** and **mover**, are ported to GPU platform. In our experiments, NVIDIA's newly released Kepler K20 is used to evaluate the performance and MAX 30x speedup is achieved compared with Intel Sandy Bridge E5-2650.

Key words: PIC method, CUDA, NVIDIA Kepler

摘要：PIC 方法是计算等离子体物理中广泛使用的一种计算方法。通常情况下需要使用大量的计算粒子以达到高的计算精度，这导致非常庞大的计算量。因而 PIC 方法的加速研究对于减少其时间成本非常有意义。在本文中，我们设计了一个基于 NVIDIA Kepler GPU 的 PIC 算法，并使用 CUDA 在 GPU 上将该算法实现。在 PIC 方法中最耗时间的两个函数 collision 和 mover 被移植到 GPU。在实验中使用了 NVIDIA 新发布的 Kepler K20 GPU 进行这两个函数的性能测试，相比于 Intel Sandy Bridge E5-2650，最高获得了 30 倍的加速。

关键词：PIC 方法，CUDA，NVIDIA Kepler

中图法分类号: O53 **文献标识码**: B

* Supported by SJTU CUDA Center Of Excellence

+**作者简介**: **文敏华**(1988－),男,江西会昌人,硕士,主要研究领域为高性能计算;**林新华**(1979－),男,浙江绍兴人，硕士,上海交大高性能计算中心副主任，主要研究领域为高性能计算; Simon Chong Wee See(1966－),男,新加坡人，博士，NVIDIA 公司亚太区首席技术总监，上海交大高性能计算中心科学计算总监，主要研究领域为高性能计算.

1 引言

PIC[1][2]方法在等离子体物理模拟中具有广泛应用。对于等离子体模拟，目前主要使用两种方法[3]：流体方法（fluid approach）和动力学方法（kinetic approach）。流体方法模拟每个网格的平均密度、速度和温度的变化，以获得整个系统的宏观物理量变化情况，但是这种方法精确度不高。动力学方法从第一性原理直接模拟等离子体粒子的运动，其精确度非常高，但是对计算能力的要求也非常高。PIC 方法基于动力学方法但是只处理包含计算粒子的相空间，这使得在保持高精度的情况下计算量极大下降。但是，PIC 方法仍然对计算能力有较大需求。

近年来 GPU 上数量众多的计算单元和通用可编程 GPU 计算技术的飞速发展为大规模 PIC 并行计算提供一种新的可能。GPU 的大规模并行运算潜力既能实现较强的运算能力又可以极大地降低运行成本。2010 年 NVIDIA 发布的基于 Fermi 架构的 Tesla C2050 GPU 单精度浮点峰值性能达到了 1TFLOPS 以上，是核心频率 3.0 GHz 四核 CPU 的 20 倍，计算能力几乎可以与机群相比，但是其成本和功耗却只有机群的 1/10 和 1/20。NVIDIA 最新发布的基于 Kepler 架构的 K20 GPU 性能相比 Fermi 架构则又有了质的飞跃，单精度计算能力达到了 3.54TFLOPS，但是其功耗却与前一代 GPU 几乎保持不变。与此同时，基于 GPU 的通用可编程技术的发展，如 CUDA 和 OpenCL（Open Computing Language）的发布，极大降低了基于 GPU 的编程难度，人们在不具备专业图形学知识的情况下即可利用 GPU 的强大计算能力。

目前已有一些针对PIC在GPU上的并行化研究。Decyk[4]等基于一个简单的2D PIC算法开发了一个参数化GPU程序，同时在数据结构上做了一些优化。Burau等[5]开发了PIConGPU，是首个在集群上的可扩展的基于GPU的并行程序。Stanchev等[6]开发的GPU程序则注重于粒子到网格差值的优化。然而以上GPU化的工作主要基于较为简单的物理模型，只将一些简单理论模型移植到GPU上。

X. F. Meng等[7]将最新版本的GTC（Gyrokinetic Toroidal Code）在天河1A集群上进行了移植，文中对真实的等离子体湍流进行了模拟，同时对GPU程序在集群上的可扩展性做了详细的讨论。

与前面的工作相比，本文主要有以下不同：1）移植的程序基于真实的物理实验，所使用的算例均经过实验验证，为了使数值计算更好吻合物理实验结果，程序中加入了更加复杂的模型，比如粒子的双体碰撞模拟和之字形算法统计电流，这也使得程序结构更加复杂；2）CPU程序中，电流统计和分配粒子到碰撞网格都是高度串行的算法，在GPU实现中，我们使用原子操作保证电流统计网格写入的互斥性，以确保结果正确，同时，我们针对GPU架构重新设计了碰撞网格的生成算法。

本文将程序中最耗时间的两个部分移植到了GPU上。结果显示GPU程序与CPU程序的计算结果十分吻合，移植的两个部分均获得了10倍以上的加速，最大加速30倍，而整体性能提升在4倍左右。

2 PIC 方法简介

2.1 PIC方法流程

在PIC方法中，大量的计算粒子用于模拟真实的等离子体粒子。每个粒子最初的位置和动量等信息由宏观的初始条件计算出，之后的计算包括粒子间碰撞、粒子移动、诊断、电场磁场计算等步骤，这些步骤循环迭代至终止条件，具体流程图如图1所示。

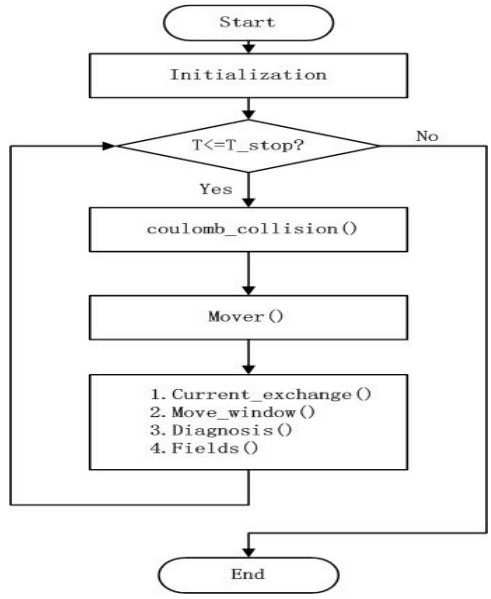

Fig. 1. PIC Flowchart

图 1 PIC 算法流程图

2.2 CPU程序剖析（Profiling）

将PIC程序移植到GPU之前，我们先使用Vampire对初始的CPU程序进行了剖析（Profiling），以查找程序的热点，一个时间步长内分析结果如图2和图3所示。初始的CPU程序已使用MPI（Message Passing Interface）并行，在剖析用的算例中我们使用4个MPI进程。从图3发现不同的进程花费时间相差很大，这主要是各个进程处理的计算区域不一样，模拟的粒子数量差异也很大，因而造成了负载不均衡，我们只要考虑花费时间更大的进程，即p0和p1。从图4可以看到，collision和mover函数花费的时间最多，约为94%，因而GPU化的工作将主要针对这两部分进行。

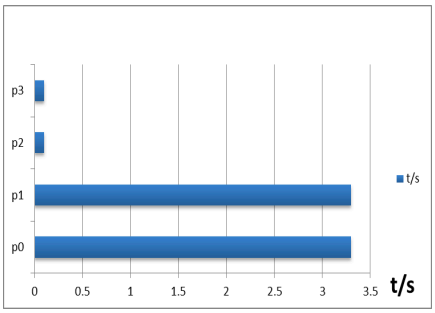

Fig. 2. Load imbalance of multi-process

图 2　多进程的负载不均衡

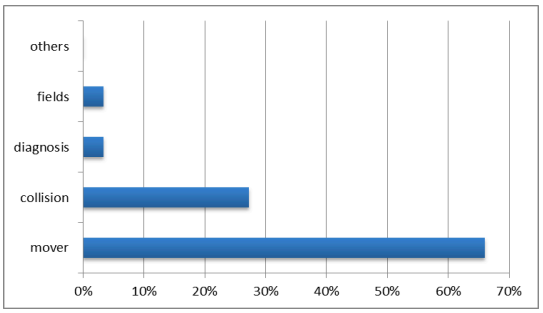

Fig. 3. Hotspots of PIC method: within one process

Fig. 3. 单进程内的 PIC 热点

3　PIC 方法在 GPU 上的并行

在并行实现中，我们将最耗时间的两个函数移植到GPU上。程序的初始化在CPU上实现，然后将粒子和电场磁场信息传输到GPU上，随后在GPU上进行碰撞和粒子移动的计算，再将数据返回CPU端进行其它运算，具体流程如图4所示。

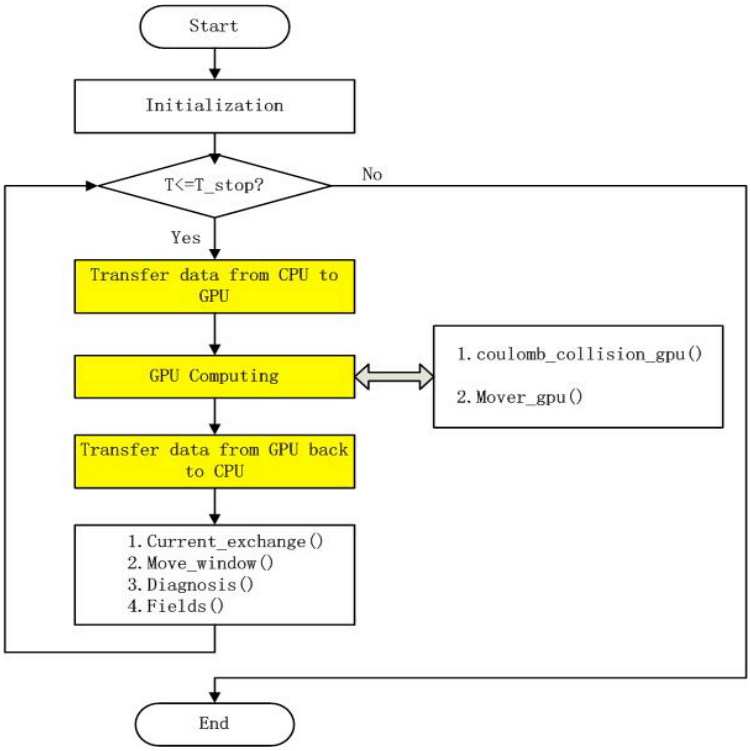

Fig. 4. PIC Flowchart on GPU

图 4 GPU 版本 PIC 算法流程图

3.1 collision

在collision函数中，所有的粒子按照其位置被分配到各个矩形网格，然后进行粒子间的碰撞计算，串行算法如图5所示。

```
p=head[isp]    // 粒子的信息存储在 p 为头指针的链表结构中
while(p!=NULL) //依次判断各个粒子所在的网格
{
    decide which cell particle p belongs to according to its position;
    add p to cell_index; //cell_index is organized by linked list
    p=p->next
}
for each cell   //计算每个网格粒子间的碰撞
{
    calculate actual collision time;
    choose collision pairs for calculating the collision;
    compute collision between the pairs;
}
```

Fig. 5. Serial algorithm of collision function

图 5 collision 函数串行算法

在串行实现中，粒子的存储使用了链表数据结构，这是一种高度串行的数据结构，粒子信息可能不会存储在连续的显存中，不同线程无法同时读取显存中的粒子信息，无法利用GPU高带宽的优点，因而不适合在GPU上实现。不仅如此，同一个网格里的粒子之间的索引也是使用链表来进行，链表的next指针指向同一个网格的下一个粒子，这样在计算粒子间碰撞的时候也无法高效读写粒子信息。为了将程序移植到GPU，我们在数据传输到GPU之前将数据转化为数组数据结构，这样做的好处是在GPU上更好利用了高显存带宽的优点，而坏处则是每次迭代需要进行数据结构的转换，链表到数组的赋值是一个串行的过程，效率非常低下，造成较为庞大的额外开销。

在算法上，判断每个粒子所属的网格以及计算各个网格内的粒子碰撞都是高度并行的过程。而将粒子按网格重新排列则是并行实现的难点。我们在实现过程中通过2个kernel和一个库函数来完成。

• cell_list_init。根据粒子的位置，判断所属网格，每个GPU线程处理一个粒子，每个粒子将所属的网格号写入cell_id数组。同时生成cell_list数组，每个线程将自己在kernel中的索引idx按顺序写入该数组。

• thrust::sort_by_key。这一步将所有的粒子按所属的网格排序，目前在GPU中已经有很多的高效并行排序算法，这里我们使用CUDA Thrust库里提供的排序函数sort_by_key，可以很方便的进行排序，极大节省编程成本，同时cell_list也根据cell_id进行重排，这样属于同一个网格的粒子序号在cell_list中连续存储，如图6。

• cell_idx_generator。根据排序后的cell_id数组，将每个网格第一个粒子在cell_list的索引计算出，存储在cell_idx数组，这样即可很方便索引到各个网格的粒子。以第n个网格为例，cell_list数组上第cell_idx[n]和cell_idx[n+1]之间的粒子即是网格n所有粒子的索引，而cell_idx[n+1]-cell_idx[n]则是该网格的粒子数量。

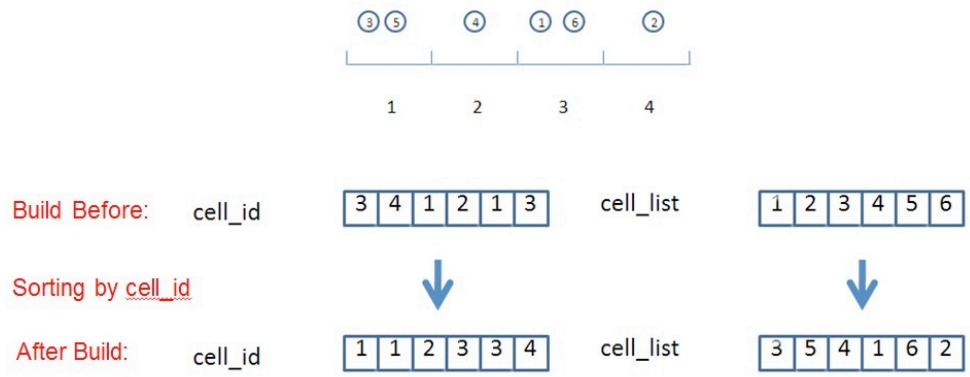

Fig. 6. Before build V.S After build

图 6 建立前 V.S 建立后

建立好cell_idx和cell_list数组后，便可进行碰撞计算。网格之间的碰撞相互独立，因而可以每线程处理一个网格的碰撞计算。在一个网格内，每个线程只需处理本网格的粒子，通过cell_idx和cell_list数组可快速索引到本网格粒子进行计算。

3.2 mover

mover函数计算粒子在电磁场作用下的运动以及运动产生的电流，这部分函数所花时间占程序总运行时间的60%以上，它的并行化对加速程序最有意义。每个粒子在电磁场作用下位置和动量发生变化，粒子之间互不影响，因而这部分的计算可以很轻易移植到GPU，每个线程处理一个粒子。

带电粒子在移动过程中产生电流，mover函数中还要计算每个带电粒子对电流的贡献。电流的计算是按网格来统计的，不同的粒子可能在同一个网格，因而统计粒子对网格的电流贡献时可能会有多个线程对同一个地址进行写操作，这种竞写会造成结果的不正确，因而需要每个线程对电流数组进行互斥操作，可以选择原子操作。尽管原子操作相对于一般的运算操作需要更多的时钟周期，而且不同线程对同一地址操作时甚至是

一个串行过程，但是原子操作的使用可以极大减小编程成本，使得电流计算算法比较容易实现，同时考虑到Kepler架构的原子操作性能相对于Fermi架构有了非常大的提升[8]。因而在我们的并行实现中，使用原子操作函数atomicAdd进行电流计算，在降低算法移植难度的同时保证了结果正确性。

4 实验结果及分析

4.1 正确性检查

实验所用平台如表 1 所示，所用 CPU 为 Intel Sandy Bridge 架构 E5-2650，搭配 Nvidia 最新发布的 Kepler K20 GPU。

Table 1 Experimental testbed

表 1 实验平台

CPU	Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz , 8 core
Memory	64GB DDR3 1600MHz
GPU	NVIDIA Kepler K20
OS	Red Hat Enterprise Linux Server 6.3
Compiler	MPICH2 1.2.1p1+GCC 4.4.6 + NVCC 5.0
Compiler option	-O2

为比较结果正确性，我们将 CPU 程序和 GPU 程序的结果输出并作比较。选用 PIC 方法里面比较重要的三个参量用于比较，分别是粒子信息、电场信息和电流分布。比较结果如表 2 所示，结果显示，误差在可接受范围内。

Table 2 Error analysis

表 2 误差分析

误差分析	绝对误差			相对误差		
	粒子信息	电场	电流	粒子信息	电场	电流
MAX	10^{-13}	10^{-10}	10^{-12}	10^{-13}%	10^{-12}%	10^{-10}%
MIN	0.0000	0.0000	0.0000	0.00%	0.00%	0.00%

4.2 加速比

为了测试 GPU 版本程序的性能提升效果，我们测试并比较了两个版本程序在迭代步数达到 400 步时一个时间步长所耗的时间(此时每时间步长各函数所花时间已趋于稳定)，结果如表 3 所示。collision 函数获得了 30 倍加速。而对于 mover 函数，我们获得了 10 倍左右的加速比。mover 函数的加速效果相对较低，我们认为主要是由于 kernel 内大量原子操作使用的原因。整个程序的加速在 4 倍左右，这主要是每次迭代链表和数组的数据结构转化和 CPU 与 GPU 之间的数据传输造成了额外的开销。

Table 3 Speedup

表 3 加速比

time / function	CPU 时间/ms	GPU 时间/ms	加速比
collision	900	30	30
mover	2200	212	10.3
total	3300	830	3.97

5 结论和下一步工作展望

本文设计了一个基于 GPU 的 PIC 算法，并使用 CUDA 将程序最耗时间的两个部分 mover 和 collision 移植到 NVIDIA Kepler K20。对于源程序中不适合在 GPU 上并行的链表数据结构，我们将其转化为数组然后使用到 GPU 计算中；在 collision 函数中，将粒子分配到各个网格是一个高度串行的算法，我们则针对 GPU 架构重新设计了算法，并实现了同样的功能。相对于 Intel Sandy Bridge E5-2650，移植的两个函数获得了最高为 30 倍的加速比。然而，由于 CPU 端和 GPU 端在每次计算迭代中都有数据传输，程序的整体性能提升比较有限。除此以外，链表和数组数据结构的转化也增加了程序的运行开销。

下一步，我们将重构 CPU 版本程序，将不适合并行的链表数据结构完全替换为数组，并将程序的其它部分也移植到 GPU 以提高整体性能。同时，我们还将在 GPU 集群上运行程序以获得更高的性能提升。

References:

[1] Birdsall C K and Landon A B. Plasma Physics via Computer Simulation. Longdon: Institute of Physics Publishing, 1991.

[2] Matsumoto H and Sato T. Computer Simulation of Space Plasmas. tyoko: Terra Scientific Publishing Company, 1985

[3] S Markidis, Lapenta. Multi-scale simulations of plasma with iPIC3D. Mathematics and Computers in Simulation, 2010

[4] V. K. Decyk and T. V. Singh, Adaptable particle-in-cell algorithms for graphical processing units. Computer Physics Communications, 182(3):641-648, 2011.

[5] H. Burau, R. Widera, W. Honig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T. E. Cowan, R. Sauerbrey, M. Bussmann, PIConGPU: A Fully Relativistic Particle-in-Cell Code for a GPU Cluster, IEEE Transaction on Plasma Science, 38 (10), 2831-2839, 2010

[6] G. Stantchev, W. Dorland, and N. Gumerov. Fast parallel particle-to- grid interpolation for plasma PIC simulations on the GPU. Journal of Parallel and Distributed Computing, 68(10):1339-1349, 2008.

[7] X. F. Meng, X. Q. Zhu, P. Wang, Y. Zhao, X. Liu, B. Zhang, Y. Xiao, W. L. Zhang, Z. H. Lin, Heterogeneous Programming and Optimization of Gyrokinetic Toroidal Code and Large-Scale Performance Test on TH-1A, 28th International Supercomputing Conference, ISC 2013, Leipzig, Germany, June 16-20, 2013. Proceedings.

[8] http://on-demand.gputechconf.com/gtc-express/2012/presentations/inside-tesla-kepler-k20-family.pdf