第47卷 第8期 计 算 机 工 程 2021年8月
Vol.47 No.8 Computer Engineering August 2021

·先进计算与数据处理· 文章编号：1000-3428(2021)08-0100-09 文献标志码：A 中图分类号：TP391

GPU在缪子快速模拟中的应用

易培淮[1,2]，李卫东[1,2]，林 韬[2]，邹佳恒[2]，邓子艳[2]，刘 言[1,2]

（1.中国科学院大学 核科学与技术学院，北京 100049；2.中国科学院高能物理研究所，北京 100049）

摘 要：江门中微子实验（JUNO）拥有当前世界上能量精度最高、规模最大的液体闪烁体探测器。缪子是JUNO的主要本底，每个缪子事例在大型探测器中产生百万量级的光子，但复杂的光子模拟计算量巨大，传统串行计算方式耗时较长。为此，提出一种基于GPU的分布式缪子快速模拟方法。利用多GPU卡并行加速闪烁光在液闪探测器中的传输过程，采用信息传递接口通信向多节点分发模拟任务和收集结果。测试结果表明，GPU方法具有良好的加速比，和CPU方法相比，加速比最高可达约250倍。

关键词：GPU并行计算；分布式通信；信息传递接口；缪子模拟；江门中微子实验

开放科学（资源服务）标志码（OSID）：

中文引用格式：易培淮，李卫东，林韬，等.GPU在缪子快速模拟中的应用[J].计算机工程，2021，47(8)：100-108.

英文引用格式：YI P H，LI W D，LIN T，et al.Application of GPU in fast Muon simulation[J].Computer Engineering，2021，47(8)：100-108.

Application of GPU in Fast Muon Simulation

YI Peihuai[1,2]，LI Weidong[1,2]，LIN Tao[2]，ZOU Jiaheng[2]，DENG Ziyan[2]，LIU Yan[1,2]

（1.School of Nuclear Science and Technology，University of Chinese Academy of Sciences，Beijing 100049，China；
2.Institute of High Energy Physics，Chinese Academy of Sciences，Beijing 100049，China）

【Abstract】For the Jiangmen Underground Neutrino Observatory（JUNO）experiment，a liquid scintillator detector with the largest size and highest energy accuracy in the world is used.Cosmic Muon is one of the major backgrounds of the experiment and each of them generates millions of photons in the large detector.However，the complicated simulation of photons involves massive calculation and the traditional sequential computing is time-consuming.Thus，a fast Muon simulation method implemented on GPU is presented.Multiple parallel GPU cards are used to accelerate the transmission of scintillation photons in the liquid scintillator detector.Also，the Message Passing Interface（MPI）is used to distribute simulation tasks to nodes and collect processing results.The test results show that the GPU-based method has an excellent speedup ratio，which is up to about 250 times higher than that of the CPU-based method.

【Key words】GPU parallel computing；distributed communication；Message Passing Interface（MPI）；Muon simulation；Jiangmen Underground Neutrino Observatory（JUNO）

DOI：10.19678/j.issn.1000-3428.0058153

0 概述

江门中微子实验（JUNO）[1-2]通过探测核电站反应堆产生的电子反中微子来测量中微子质量等级[2]和精确测量中微子振荡参数，具有丰富的物理目标。用于探测中微子的中心探测器是世界上能量精度最高、规模最大的液体闪烁体探测器[3]。宇宙线缪子作为JUNO实验重要的本底之一[4-5]，需要经过精细重建后才能够排除，而精细的重建算法需要大量的模拟事例作为输入[6-7]。但缪子穿过探测器时会产生百万量级的闪烁光和契仑科夫光，导致基于Geant4[8]的全模拟计算量巨大。为此，文献[9]提出一种快速模拟方法，首先用Geant4模拟预先产生PMT的击中光子数和击中时间的分布，然后在快速模拟时读入这些分布直方图并对其进行抽样，用抽样的方法代替耗时的光子传输模拟。尽管该方法可

基金项目：国家自然科学基金青年科学基金（11805223）；中国科学院战略性先导科技专项（A类）（XDA10010900）；中国科学院青年创新促进会项目（2017021）。

作者简介：易培淮（1996—），男，硕士研究生，主研方向为并行计算；李卫东，研究员、博士、博士生导师；林 韬，助理研究员、博士；邹佳恒，副研究员、博士；邓子艳，研究员、博士、博士生导师；刘 言，博士研究生。

收稿日期：2020-04-23 **修回日期：**2020-09-04 **E-mail：**yiph@ihep.ac.cn

使单个事例模拟时间从约1 h缩减至10 min左右，但传统的串行计算方式仍然需要消耗较长的时间。

基于异构的GPU应用[10-12]以及跨节点的信息传递接口（Message Passing Interface，MPI）[13]被广泛地应用于高性能计算，并获得数倍至数万倍的加速比[14-16]。在国际高能物理模拟领域，目前出现了利用多线程加速模拟计算，比较典型的是欧洲核子中心大型强子对撞机（LHC）上的ATLAS实验模拟软件[17]，以及利用多节点进行模拟的Alice实验模拟软件[18]。此外，各个高能物理实验开发了快速模拟软件[19-20]来提升模拟速度，这些快速模拟软件都只能在CPU上运行。如何运用GPU并行Geant4模拟[21]以及将快速模拟软件移植到GPU上运行仍处于初步研究阶段，到目前为止尚无发现成功应用案例。

缪子在探测器中产生的闪烁光具有各向同性的属性，并且模拟过程相互独立，适合在GPU上并行。本文提出一种基于GPU的缪子快速模拟方法，将闪烁光的产生和模拟置于GPU中进行并行计算，以提高运算速度。对于多节点GPU集群，该方法采用MPI分布式通信在GPU集群中实现光子模拟分布式计算。

1　缪子快速模拟算法

JUNO模拟软件由物理产生子、探测器模拟、电子学模拟3个部分构成，如图1所示。物理产生子用于产生原初粒子信息，之后再由探测器模拟对原初粒子进行运输和跟踪。模拟软件基于Geant4和SNiPER[22]开发，用于模拟粒子和探测器之间的相互作用，并记录探测器中的响应信息，最后由电子学模拟对击中信息数字化，生成与真实实验相同的波形数据。JUNO实验采用1 GHz的采样率采集波形，电子学模拟中按照1 ns间隔读出最终的波形数据。

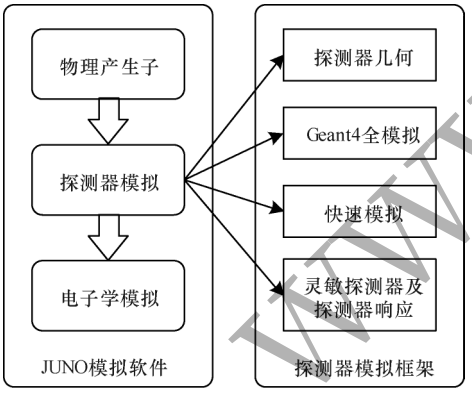

图1　JUNO模拟软件示意图

Fig.1　Schematic diagram of JUNO simulation software

快速模拟方法将缪子径迹分割成很多份短的步长（Step），如图2所示。对于每个Step，需要根据位置和沉积能量产生探测器的响应。Step所在位置可以确定径向位置R以及该位置与每个PMT之间的夹角θ，可以唯一确定需要加载PMT的光电子（nPE）直方图和击中时间（Hit time）直方图数据。快速模拟的第1步是根据每个Step的可见能量对nPE直方图进行抽样，产生该PMT上的nPE数目。第2步是为每个PE产生一个击中时间。Hit time直方图是使用初始时间为0的事例构建的，包括光子产生时间和光子飞行到达PMT的时间，最终的击中时间等于Step当前的时间加上抽样得出时间。

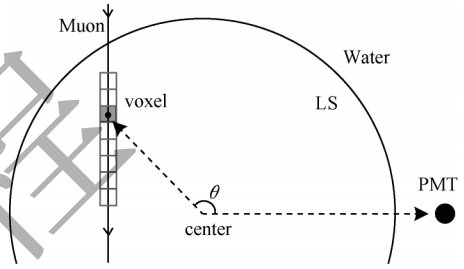

图2　缪子快速模拟几何关系示意图

Fig.2　Schematic diagram of Muon fast simulation geometrical relationship

快速模拟的抽样计算需要预先加载36 000份nPE直方图和Hit time直方图，才能够实现精确的快速模拟。快速模拟也需要加载约18 000支PMT的位置坐标，用于计算夹角θ。以上的直方图数据和PMT坐标数据都可以在模拟初始化阶段加载完成。

2　GPU缪子快速模拟

模拟缪子在穿过探测器时，Geant4会产生缪子的步长信息，快速模拟方法对每个步长进行计算得出每个PMT上的击中数目和时间。针对模拟过程中步长和PMT数目十分庞大的挑战，提出一种GPU缪子快速模拟方法，并完成了基于CUDA[23]的快速模拟并行软件。该方法的核心思想是：首先将缪子的步长集合拆分成子集，分配给不同的GPU卡处理，实现GPU卡间的并行；接着在每一个GPU卡上，把原来CPU串行遍历PMT改成按照GPU线程并行模拟PMT。GPU快速模拟方法通过上述2个层次的并行加快了运行速度。

GPU缪子快速模拟方法在初始化阶段将预生成的直方图数据经过序列化存储于GPU全局内存中，供后续内核算法访问。在事例循环阶段，该方法首先从Geant4收集步长信息，并进行实时序列化，用于后续传输至GPU内存。该方法使用多GPU及MPI进行并行模拟，对步长信息自动划分，并将划分后的数据传输至对应的GPU全局内存，解决了因单块GPU显存的限制无法处理所有步长信息的问题，从而实现大规模并行。该方法实现的CUDA Kernel函数将一个PMT的模拟映射到一个CUDA线程，通过启动大量的线程完成大规模的模拟计算，并且将中间结果存储于GPU全局内存。在每个事例终止时，

合并所有GPU的数据,并将结果传回CPU,从而完成一个事例的模拟。GPU缪子快速模拟的Kernel函数主要负责光子的模拟,用于替换串行的CPU算法。CPU算法中对于每个Step都需要遍历20 000个PMT的计算,GPU算法则将20 000个PMT的计算迁移至GPU并行,极大地缩短了运行时间。

GPU缪子快速模拟算法的具体实现包括1个运行于CPU的函数和2个运行于GPU的CUDA Kernel函数,如算法1所示。

算法1 GPU缪子快速模拟算法伪代码

```
1.procedure GPUMuonFastSimulation(e) //e为事例数据
2.hist ← LoadAllHistogram();//加载直方图
3.stepsInfo ← StepCollect(e);//收集Step信息
4.gpuStepsInfo ← cudaMemoryCopyHtoD(stepsInfo);
//拷贝Step信息到GPU
5.cudaKernel <<<block,thread >>>(gpuStepsInfo);
//启动GPU核函数计算
6. npeArrays, hitTimeArrays←cudaMemoryCopyDtoH();
//从GPU拷贝回CPU
7. return npeArray, hitTimeArray ← reduce(npeArrays,
hitTimeArrays);
8.end procedure
9.procedure cudaKernel(gpuStepInfo)
10.if threadid > pmtNum then //PMT数量作为并发线程数
11.return;
12.end if
13.pmtid ← getThreadId();
14.for i = 0 → gpuStepInfo.size()-1 do
15.npe,hitTimeArray ← SimulationHits(pmtid,gpuStep
Info[i]); //光子模拟
16.npeArray ← npeArray.add(npe);
17.end for
18.end procedure
19.procedure SimulationHits(pmtid,step) //step为步长数据
20.R,Theta ← CalculateRTheta(pmt[j],step);//计算参数
21.npeHist,hitTimeHist ← getHistgram(R,theta);
22.npe ← npeHist.getRandom();//直方图抽样产生最大
//击中光电子数
23.energyInt,energyFraction ← step.energy();
24.for i = 0 → energyInt.size()-1 do //确定性击中
25.for j = 0 → npe-1 do
26.hitTime ← hitTimeHist.getRandom();//直方图抽样
//产生击中时间
27.hitTimeArray ← hitTimeArray.merge(hitTime);
//合并同1ns内发生的击中
28.end for
29.end for
30.for j = 0 → npe-1 do //有概率击中
31.if random(0,1) < energyFraction then
32.hitTime ← hitTimeHist.get Random();
33.hitTimeArray ← hitT'imeArray.merge(hitTime);
34.end if
35.end for
36.return npe,hitTimeArray;
37.end procedure
```

在算法1中,第1行~第8行是CPU部分的代码实现,主要负责CPU与GPU之间的数据拷贝以及CUDA Kernel函数的调用。对于每个物理事例,收集Geant4产生的步长信息,然后拷贝至GPU,对应于第3行、第4行。随后,调用CUDA Kernel函数,通过block和thread 2个参数设置并发线程数,且总的并发数不少于PMT数目,对应于第5行。模拟完成后,再将结果合并拷贝回CPU,对应于第6行、第7行。

第9行~第18行是CUDA Kernel函数的实现,在GPU上并行执行,主要负责PMT的模拟。由于总的并发线程数将大于PMT数目,对于超过PMT数目的线程,设置不执行后续的计算,对应于第10行~第12行。通过CUDA提供的线程索引,计算出当前线程将要模拟的PMT ID,对应于第13行。第14行~第17行对传入的步长信息进行循环,并调用另一个Kernel函数SimulationHits进行光子的模拟。模拟产生的结果被存储于GPU全局内存,用于后续返回。

第19行~第37行对应于SimulationHits的实现,该函数的输入参数为PMT ID以及步长信息,返回的结果为当前步长生成的光子数目以及对应的击中时间。该函数从步长的位置计算得出径向距离R;再从PMT与当前步长的位置关系得出角度θ。基于距离R和角度θ 2个变量,从全局内存中分别定位到nPE和Hit Time 2套直方图数据,对应于第21行。第22行~第35行实现了从2套直方图分别抽样nPE和hit time。由于直方图是基于1MeV能量产生的,算法中将当前步长的能量分成整数energyInt和小数energyFraction 2个部分:对于整数部分,重复进行抽样产生hit time;对于小数部分,若产生的随机数小于小数部分,再产生hit time,对应于第31行~第34行。

对于伪代码中SimulationHits算法所描述的直方图抽样的过程,即代码第26行和第32行中的getRandom,实现过程如下:首先在初始化加载直方图数据时,将概率分布直方图转化为累计分布函数直方图,直方图在全局内存中以定长数组的形式保存。GPU线程通过Step信息中的R和θ找到对应的累积分布直方图后,调用CUDA随机数算法cuRAND生成在范围为0~1的概率值,遍历直方图数组中找到刚好不小于概率值的元素,元素的下标即为抽样的结果。

在并行模拟时,线程束(Warp)内的32个线程从GPU全局内存读取直方图数据用于抽样计算,如图3所示。直方图在内存的放置策略按照距离R和角度θ 2级索引的方式,保证Warp中的线程能够访问连续内存空间的直方图:第1级索引为距离R,长度为200;第2级索引为角度θ,长度为180。在图3中,32个线程访问的直方图具有相同的距离R和相

近的角度θ，能够合并访问GPU全局内存空间，从而提高数据访问效率。

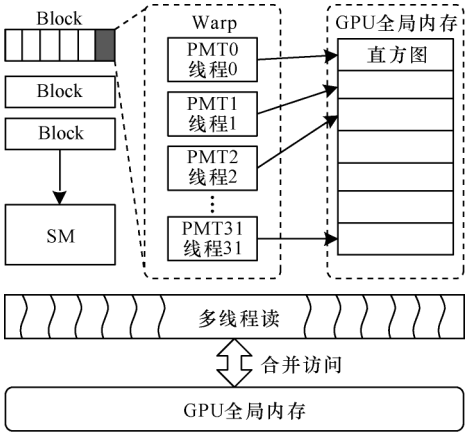

图3　GPU内存优化策略

Fig.3　GPU memory optimization strategy

除访问直方图数据外，GPU线程还要访问步长信息。预先产生的直方图数据数目固定，而步长的数目则与模拟事例的能量相关，能量越高产生的步长数目越多，所以对步长信息的实现与直方图数据不同。针对CUDA多次启动核函数开销高的问题，GPU算法并不对每个Step单独调用快速模拟的方法，而是创建步长信息的集合，将Step信息集中收集，待Step收集完成后，再一次性或分批调用并行化快速模拟算法，从而提高算法效率。通过引入步长集合的设计，将一个高能的缪子事例拆分成多批进行模拟，也可将多个低能的缪子事例合并成一批进行模拟。

缪子快速模拟并行软件以面向对象方法开发，并与JUNO离线软件系统整合，如图4所示。以SNiPER和Geant4作为底层软件，缪子快速模拟软件包括CPUFastSim、Parallel-FastSim和Helper 3个组件，分别实现CPU串行模拟、GPU并行模拟和辅助工具。Helper主要提供从文件加载直方图等通用功能。在Parallel-FastSim组件中，除了运行于GPU的Kernel函数和Buffer以CUDA实现，其他的类都派生自SNiPER框架，并以C++实现运行于CPU中。负责GPU快速模拟的GPUFastSimTool派生自ToolBase，与串行版本CPUFastSimTool具有相同的接口，能够在Geant4的EventAction与SteppingAction中被调用，软件能够支持串行和并行模式，并且复用FastSimHelper中提供的功能。负责MPI通信的MPISvc派生自SvcBase，能够在配置跨节点运行时按需被GPUFastSimTool调用。负责GPU Context的GPUSvc类派生自SvcBase，这样既可以被GPUFastSimTool调用，也可以被MPISvc调用。通过配置参数，软件在运行时将动态创建相应的对象，从而支持单GPU、多GPU、MPI-GPU3种加速模式。

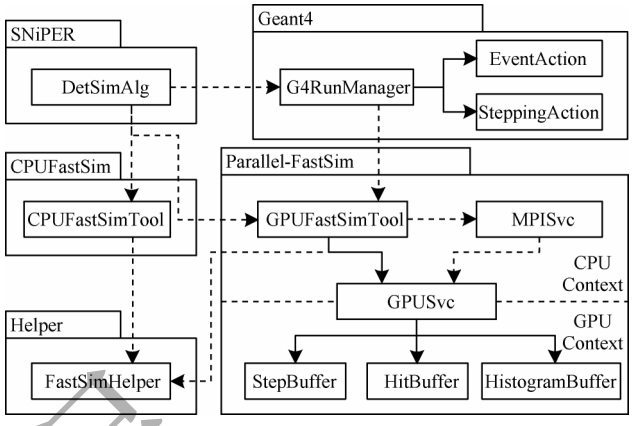

图4　GPU缪子快速模拟的UML类图

Fig.4　UML class diagram of GPU Muon fast simulation

多GPU模式的缪子快速模拟软件复用了和单GPU一致的流程，如图5所示。在初始化阶段，GPUSvc按照设定的GPU数量初始化GPU设备列表，并检测是否支持NVLink。该功能可在任意2块GPU之间互传数据，速度远高于PCIe总线传输。在多GPU模式下，初始化和结果合并阶段使用GPU间点对点传输的方式，以提高数据传输效率。CPU将直方图数据传递给编号为0的GPU，然后其他GPU通过GPUDirect拷贝编号为0的GPU中的数据。在事例循环阶段，采取CPU多线程来控制和调度多GPU：每个CPU线程控制各自的GPU设备，将Step信息均匀分配至GPU，并异步启动Kernel函数。CPU主线程负责等待全部控制线程执行结束，实现多GPU设备同步。GPU Kernel函数的结果由线程保存在各自开辟的定长数组中，以避免线程间资源竞争。对结果进行合并的方式为：单GPU模式的结果直接由GPU传回CPU；多GPU模式的结果先汇总至编号0的GPU内，再由编号0的GPU传输给CPU。GPU计算完成后全局内存不释放，仅将结果置零，直到所有事例计算完成才释放全局内存。

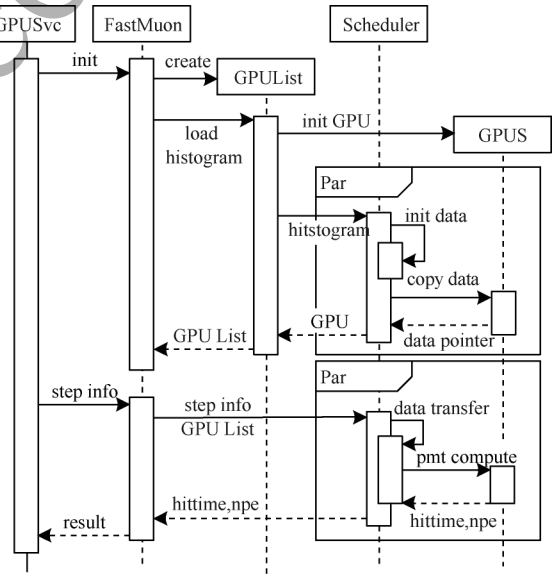

图5　GPU缪子快速模拟时序图

Fig.5　Sequence diagram of GPU Muon fast simulation

3 MPI-GPU并行计算

尽管单节点GPU并行计算可以加速光子模拟，但同一节点可搭载的GPU数量有限。计算集群GPU资源丰富，若聚合集群中多个节点的GPU算力，则光子模拟速度能进一步提升。将算法移植到集群上并行运行的关键是节点间信息的分发与收集，而MPI能提供跨节点高性能数据通信。MPI-GPU并行计算的核心思想是将步长集合进行切割，通过MPI消息将步长子集发送给GPU节点进行模拟计算，最后合并各个节点模拟的结果，从而完成跨节点并行计算。

如图6所示，MPI-GPU快速模拟包含主节点和多个从节点，主节点负责跨节点部分的流程控制、步长子集的分配以及模拟结果的合并，从节点完成相应步长子集的快速模拟。由于网络存在延迟，上述节点都以多线程的方式启动，并由专用线程负责节点之间的通信。在图6中，左右2个框图中的计算运行于主节点。主节点收集步长信息并暂存于数组中，收集完毕后将其拆分形成任务，并放入队列中。主节点中的专用线程会监听从节点的请求，并将任务分配给相应的从节点。在任务调度时，会出现拆分后的任务数目多于子节点数目或者子节点运行失败的情况，主节点会将未执行或执行失败的任务分发给空闲的从节点。完成相应任务后的从节点会发送消息通知主节点，由主节点对从节点的运行结果进行缓存，并在所有任务完成后进行合并。所有事例模拟完成后，主节点发送消息关闭从节点以及自身进程。图6中间框图为从节点，这些节点除了启动GPU并行计算线程，还会启动通信线程用于与主节点的通信。当从节点任务为空时，通信线程会与主节点通信，拉取模拟任务。获得步长子集后，从节点根据配置执行单GPU或多GPU的模拟，完成模拟后将模拟结果发送回主节点。接着从节点又进入空闲状态，等待主节点的任务分配。该流程不断循环，直至所有任务完成。

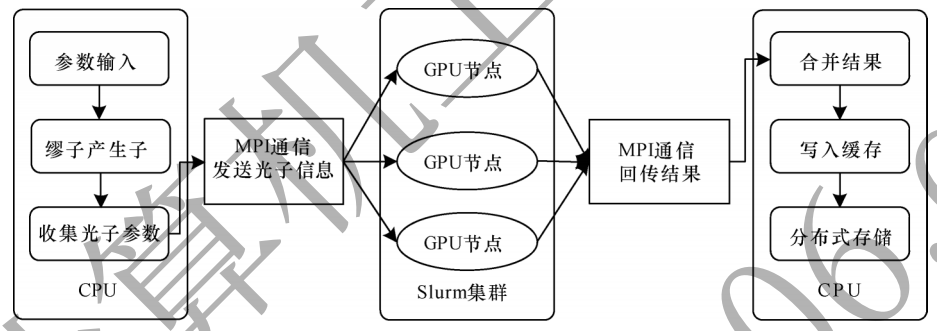

图6 MPI-GPU快速模拟示意图

Fig.6 Schematic diagram of MPI-GPU fast simulation

MPI-GPU缪子快速模拟的主从节点以独立的MPI程序启动，并且任务的分配与节点之间解耦合，从而实现通信独立性、系统容错性。实现的MPI通信服务在主从模式的基础上采用C/S模式维护主从之间的通信连接。主节点作为服务器，阻塞等待接收各个作为客户端的从节点的连接请求，任意客户端断开连接或者无响应都不会使服务器停止运行。主从模式与C/S模式结合无法保证数据一致性，但具备可用性和容错性，这与证明了数据一致性、可用性和分区容错性为不可能三角的CAP定理[24]相契合。通过哈希算法分配任务到节点时，需将任务与节点解耦，避免因从节点故障导致模拟失败。在实现时采用了带虚拟节点的一致性哈希算法[25]，在节点发送变化时能进行任务的自动分配：主节点在任务执行过程中，将所有未返回结果的任务暂时持久化；若在计算过程中突发从节点计算速度缓慢或者崩溃，该节点的任务将被重新分配到其他临近节点，从而实现轻量级的错误恢复与负载均衡。

MPI通信服务的设计如图7所示，主要由接口类和3个组件构成：对外接口MPISvc，管理MPI通信的Comm组件，负责节点内部事务的Context组件以及辅助工具。图7中GPUFastSimTool调用MPISvc，完成步长数据在不同节点之间的传输以及GPU快速模拟的调用。Comm组件底层的MPI通信由ClientSend、ServerSend和MsgRecv构成。ClientSend和ServerSend类为C/S模式的核心实现，负责主从节点MPI消息发送；MsgRecv用于接收处理MPI消息。Comm组件中CommManager类是控制节点消息分发的接口，主从节点分别派生实现相应的功能。Context组件以BaseContex为基类，以派生类MasterContext和SlaveContext类为核心，负责处理SNiPER框架异步通信事务与错误恢复功能。Util组件提供线程池（ThreadPool类）、进程内部的线程间通信（ThreadComm类）以及哈希值计算（MFSHash类）。MFSHash类对序列化后的步长数据计算哈希值从而得到目标GPU节点。

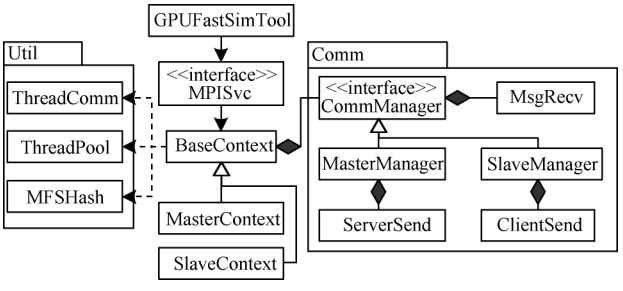

图7 MPI通信服务UML类图

Fig.7 UML class diagram of MPI communication service

4　性能测试与分析

针对本文完成的GPU缪子快速模拟进行如下的性能测试：首先对模拟的结果进行验证，确保方法的正确性；然后分别对单GPU、多GPU、MPI-GPU缪子快速模拟的3种模式进行性能测量。对于单GPU，测量了总光子数与加速比的关系，优化得出最佳的光子数模拟区间。对于多GPU，测量了GPU数目与加速比之间的关系，优化多GPU场景下程序的可扩展性和线性加速比。对于MPI-GPU，更多的开销将来自数据在网络中的传输，通过测量跨节点情况下GPU节点的利用率能够优化跨节点快速模拟的参数。这些参数的优化为大规模数据产生提供重要的参考依据。

缪子快速模拟测试是在中国科学院高能物理研究所计算中心的GPU计算集群上完成的。GPU计算集群中的每个节点都配备了2颗主频为2.30 GHz的Intel Xeon Gold 6140 CPU处理器，每个CPU处理器包含18个物理内核。每个节点还搭载了8块NVIDIA Volta构架 Tesla V100 SXM2 GPU计算卡，同一个节点上的GPU卡之间通过NVLink互联，单个GPU拥有32 GB内存。InfiniBand网络交换机Mellanox MCS7520/MSB7800实现集群计算节点间的互联，节点间的数据传输速率为100 Gb/s，延时小于1 μs。整个GPU集群由Slurm[26]进行管理和作业的调度，集群节点软硬件环境如表1所示。由于该集群被多个用户共享使用，在测试过程中申请的计算资源都以整个节点为单位，避免其他用户的计算作业对测试结果的干扰。通过在专用节点进行测试，可以实现测试环境和生产环境的隔离，避免对测试环境的干扰。

表1　测试节点软硬件环境

Table 1　Software and hardware environment of test node

类型	描述
CPU	2×Intel Xeon Gold 6140,2.30 GHz,18 cores per CPU
GPU	8×NVIDIA Tesla V100 SXM2 GPU 32 GB
OS	Scientific Linux release 7.5
Memory	12×32 GB
CUDA	10.1
MPICH	3.2.1

CPU缪子快速模拟和GPU缪子快速模拟使用了相同的事例进行测试，以确保运行时加载相同的Step信息，验证GPU缪子快速模拟算法结果的正确性。在对结果进行校验时，选择PMT击中时间以及每个PMT上nPE的分布进行比较。图8为GPU缪子快速模拟和CPU缪子快速模拟的模拟结果比较，纵轴使用对数坐标展示事例数。图8(a)为缪子事例中光子击中PMT的时间分布，直方图为GPU缪子快速模拟结果，圆点为CPU缪子快速模拟的结果，结果显示击中时间的分布一致。图8(b)为缪子事例中击中PMT的光子数量分布，同样地，直方图为GPU缪子快速模拟结果，圆点为CPU缪子快速模拟的结果。

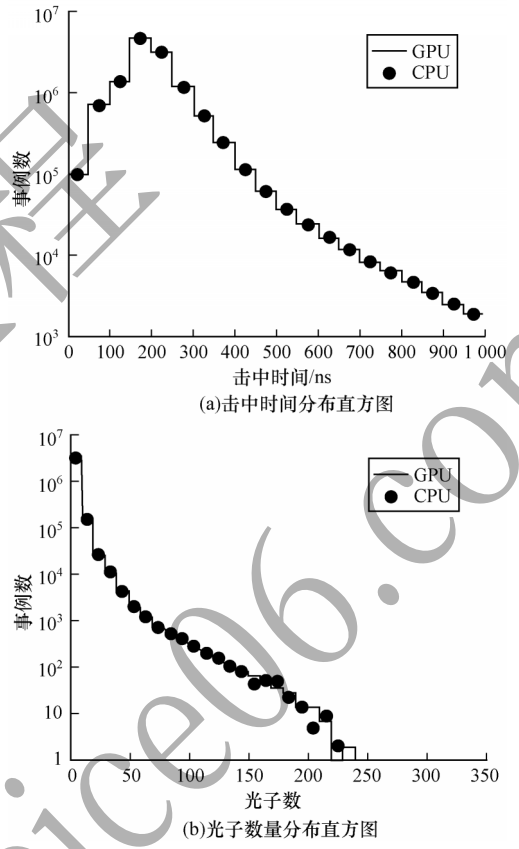

图8　GPU和CPU缪子快速模拟的结果验证

Fig.8　Result verification of GPU and CPU Muon fast simulation

从图8可以看出，在光子数量(nPE)小于150时分布一致，而nPE大于150时也在统计误差范围内。图8(a)和图8(b)中GPU缪子快速模拟和CPU缪子快速模拟的结果有细微的差别，这是由于两者在直方图抽样过程中使用了不同的随机数引擎造成的，CPU缪子快速模拟使用线性同余随机数(LCG)生成器，GPU缪子快速模拟使用线性反馈移位寄存器的变种XORWOW生成器。nPE>150时两者结果差距变大，是因为图8纵坐标取对数后，微小的差异在数据量较大时不明显，nPE>150时数据量小，抽样次数少，结果波动较大，差异更明显，但从整体上看，两者结果基本一致。

在测试单GPU加速模式中GPU算法对CPU算法的加速性能时，2个算法利用相同的参数生成了数量一致的模拟结果，CPU算法为单线程串行执行，

GPU算法为单GPU内部并行。测量过程中的运行时间仅包含快速模拟核心算法部分,以减少初始化加载参数直方图和数据传输等对测量结果的影响。每个测量点都重复100次后取平均值和标准差。在测试中需要改变产生的光子总数,测试过程中以215GeV缪子事例为基准,对Step进行采样和重复2种策略。例如,产生10%的总光子数时只需要按照10%的概率对每个Step进行筛选,产生2倍的总光子数时,则对相同的Step调用2次。图9给出单GPU加速模式下加速比与总光子数的关系。图中直方图为典型的215GeV缪子事例光子分布,加速比的定义为CPU快速模拟的运行时间除以GPU快速模拟的运行时间。

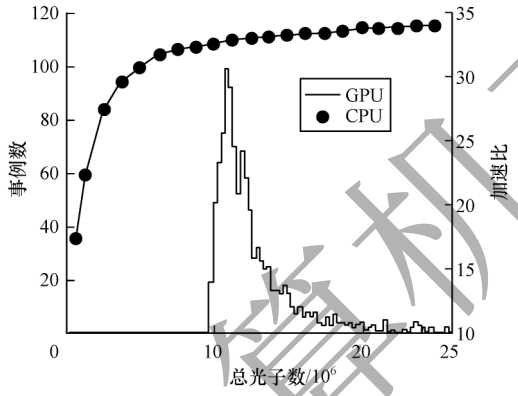

图9 单GPU加速模式加速比与光子数的关系

Fig.9 Relationship of single-GPU mode acceleration ratio and number of photons

从图9可知,在光子数较少的情况下,GPU加速效果不佳,未能充分利用GPU硬件资源;随着光子数增大,GPU处理的数据量增大,GPU算法加速比稳步提升,波动趋缓,最终稳定在33倍附近,达到单GPU性能极限。结合CPU缪子快速模拟对全模拟的加速约为6倍,单GPU缪子快速模拟对全模拟加速约为198倍。图中的直方图显示了能量为215GeV缪子事例的总光子数分布情况,可以看出大部分缪子事例产生的光子数处于GPU加速比稳定的区域。利用单GPU加速缪子快速模拟可取得良好效果,模拟生成1 000个事例数据从3天缩短至2 h,使短时间内生成大量数据成为可能。

在测量多GPU加速模式提升GPU算法的性能方面,本文比较在不同GPU数量的条件下GPU算法与CPU算法的时间,并以加速比的形式展现。在多GPU加速模式下,申请的配置为同一节点的单核CPU与多块GPU。图10显示了在不同GPU数量下,以同样的随机数种子生成相同的100个事例的GPU加速比。图中虚线由单GPU加速比的线形倍增得到,实线是多GPU加速比的测量值。从图10中可以看出,GPU数量越多,加速比越高,趋近于线形增长。当GPU数量大于3时,加速比增长率开始偏离理想情况,说明多GPU并行时系统运行效率下降,最大性能损耗为9.13%,仍保持较高效率。存在性能损耗的原因是:一方面,更多GPU导致数据传输时间有所增加;另一方面,GPU切换开销极大,结果合并时也将多次启动核函数计算。图中8块GPU并行实测加速比最高为253.7倍,是单卡性能的7.6倍,加速效果显著。

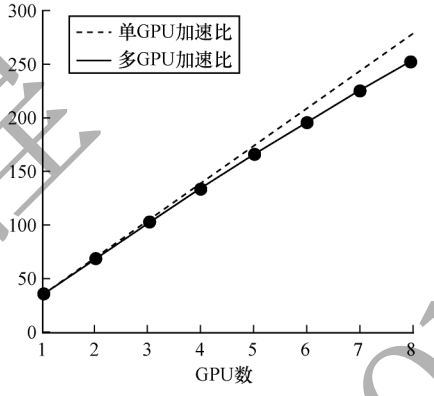

图10 多GPU模式加速比

Fig.10 Acceleration ratio of multi-GPU mode

GPU缪子快速模拟利用分布式计算可进一步扩大并行计算规模,但当需要并行计算的数据量不足时,分布式计算带来的速度提升不足以抵消分布式通信引入的额外开销。通过不断提高光子数并测量节点的GPU计算用时和系统总用时,来测试GPU缪子快速模拟的数据量能否使MPI分布式框架真实有效地利用GPU节点,两者的用时之比即为GPU节点的利用率。

MPI-GPU缪子快速模拟在集群中以集群作业的方式提交运行,在通常情况下,主节点不必等待所有从节点开始执行。在测试时主节点需等待从节点全部成功运行方可解除阻塞等待,作业排队的时间不包含在测试运行时间内,使结果更为准确。相比单节点的GPU并行计算,MPI-GPU缪子快速模拟在分布式通信时有额外的时间开销:MPI初始化的时间,计算时Step信息分发的时间,结果合并的时间。其中MPI初始化时间在集群作业无需排队的情况下可忽略不计。

在测试不同节点数量对MPI-GPU版本的影响时,使用2~4个节点进行测试,每个节点仅使用单核单GPU,其中包含1个主节点,主节点GPU参与计算,并多次测试发送总量相同的光子数。图11所示为MPI-GPU缪子快速模拟在跨节点分布式通信时,每次发送Step信息对应的光子数与运行时GPU节点利用率的关系。GPU节点利用率是GPU计算时间与从节点光子模拟总运行时间的比值。

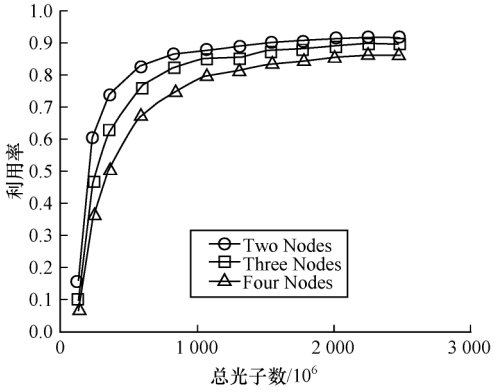

图11　分布式计算GPU节点利用率

Fig.11　GPU node utilization rate of distributed computing

从图11可以看出，在发送总光子数固定的前提下，单次发送光子数量较少时，GPU节点利用率低，因为单次发送光子数量越少，MPI通信次数越多，通信开销增长而计算开销不变。随着发送光子数增长至500×10^6后，GPU节点利用率持续增加直至平稳，GPU缪子快速模拟生成的光子数越多，分布式计算越能充分发挥多节点性能优势，且应尽可能减少通信次数，提高单次通信数据量，单次发送Step信息对应的光子数应保持在500×10^6，以提高节点利用率。随着节点数量增加，GPU利用率从90%下降至85%，因为节点数增多主节点通信开销增大，利用率稍有下降。当系统节点数量小于5个时，系统具有较好的横向扩展性，若配合纵向扩展，启用节点内部多GPU并行计算，可将加速比进一步倍增。

在国际上，尽管不同高能物理实验的探测器复杂度和物理过程不尽相同，但快速模拟方法相比于GEANT4全模拟的加速比一般在数十倍到数百倍之间。本文提出的GPU缪子快速模拟方法相比于GEANT4全模拟最终可以达到1 500倍，加速比相对于传统的快速模拟方法提高了一个量级。

5　结束语

缪子快速模拟对于中微子实验至关重要，但生成大量数据时存在性能瓶颈。为此，本文提出一种基于GPU的分布式缪子快速模拟方法。通过改进快速模拟算法，实现在GPU并行闪烁光的产生与模拟，减少CPU内存开销，并通过单GPU、多GPU、MPI-GPU 3种模式对快速模拟进行加速，利用GPU的高数据吞吐量，在单GPU加速模式中加速比高达33倍，多GPU加速模式最高可达约250倍；采用MPI分布式通信实现跨节点并行，4个节点以内GPU节点利用率高于85%，横向扩展性良好，此外MPI-GPU框架的任务分配、容错处理保证了快速模拟在集群中稳定运行。测试结果验证了GPU缪子快速模拟系统结果的正确性。但多GPU并行、MPI多节点并行依然存在一定损耗，GPU加速比还存在一定提升空间，下一步将提升MPI通信和多GPU并行效率，提供更通用的并行API接口。

参考文献

[1] DJURCIC Z, LI J Y, HU W, et al. JUNO conceptual design report[EB/OL]. [2020-03-10]. https://escholarship.org/uc/item/0jn34128.

[2] LI Y F, CAO J, WANG Y F, et al. Unambiguous determination of the neutrino mass hierarchy using reactor neutrinos[J]. Physical Review D, 2013, 88(1): 130-141.

[3] WANG Z M. JUNO central detector and PMT system[C]// Proceedings of the 38th IEEE International Conference on High Energy Physics. Washington D. C., USA: IEEE Press, 2017: 446-457.

[4] LU H Q. The JUNO veto detector system[C]//Proceedings of International Conference on Technology and Instrumentation in Particle Physics. Berlin, Germany: Springer, 2017: 197-201.

[5] ADAM T, BAUSSAN E, BORER K, et al. The OPERA experiment target tracker[J]. Nuclear Instruments and Methods in Physics Research, 2007, 577(3): 523-539.

[6] GENSTER C, SCHEVER M, LUDHOVA L, et al. Muon reconstruction with a geometrical model in JUNO[EB/OL]. [2020-03-10]. https://arxiv.org/abs/1906.01912.

[7] ZHANG K, HE M, LI W D, et al. Muon tracking with the fastest light in the JUNO central detector[J]. Radiation Detection Technology and Methods, 2018, 2(1): 1-6.

[8] AGOSTINELLI S, ALLISON J, AMAKO K, et al. GEANT4—a simulation toolkit[J]. Nuclear Instruments and Methods in Physics Research, 2003, 506(3): 250-303.

[9] LIN T, DENG Z Y, LI W D, et al. Fast Muon simulation in the JUNO central detector[J]. Chinese Physics C, 2016, 40(8): 86-91.

[10] 陈钢. 众核GPU体系结构相关技术研究[D]. 上海：复旦大学，2011.
CHEN G. Research on many-core GPU architectures[D]. Shanghai: Fudan University, 2011. (in Chinese)

[11] AYUBIAN S, ALAWNEH S, THIJSSEN J. GPU-based Monte-carlo simulation for a sea ice load application[C]// Proceedings of Summer Computer Simulation Conference. Berlin, Germany: Springer, 2016: 1-8.

[12] DEBNATH J K, GOLE A M, FUNG W K. Graphics-processing-unit-based acceleration of electromagnetic transients simulation[J]. IEEE Transactions on Power Delivery, 2015, 31(5): 2036-2044.

[13] GROPP W, Gropp W D, LUSK E, et al. Using MPI: portable parallel programming with the message-passing interface[M]. Cambridge, USA: MIT Press, 1999.

[14] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets[C]//Proceedings of HotCloud'10. [S. l.]: USENIX, 2010: 1-10.

[15] BINGMANN T, AXTMANN M, JÖBSTL E, et al. Thrill: high-performance algorithmic distributed batch data processing with C++[EB/OL]. [2020-03-10]. https://arxiv.org/abs/1608. 05634.

[16] MENSHOV I, PAVLUKHIN P. Highly scalable implementation of an implicit matrix-free solver for gas dynamics on GPU-accelerated clusters[J]. The Journal of Supercomputing, 2017, 73(2): 631-638.

[17] SIMONE A D, ACALU F. Modernizing the ATLAS simulation infrastructure[J]. Journal of Physics, 2017, 89(4): 420-431.

[18] WENZEL S. A scalable and asynchronous detector simulation system based on ALFA[J]. European Physical Journal, 2019, 214: 20-29.

[19] RAINE J. Fast simulation methods in ATLAS[C]// Proceedings of the 24th International Conference on Computing in High Energy and Nuclear Physics. Adelaide, Australia: [s. n.], 2019: 315-326.

[20] RATNIKOV F, ZAKHAROV E. Fast calorimeter simulation in LHCb[C]//Proceedings of the 39th IEEE International Conference on High Energy Physics. Washington D. C., USA: IEEE Press, 2019: 162.

[21] SEISKARI O, KOMMERI J, NIEMI T. GPU in physics computation: case Geant4 navigation[EB/OL]. [2020-03-10]. https://arxiv. org/abs/1209. 5235.

[22] ZOU J H, HUANG X T, LI W D, et al. SNiPER: an offline software framework for non-collider physics experiments[J]. Journal of Physics, 2015, 664(7): 720-753.

[23] GARLAND M. Parallel computing with CUDA[C]// Proceedings of IEEE International Symposium on Parallel & Distributed Processing. Washington D. C., USA: IEEE Press, 2010: 111-132.

[24] GILBERT S, LYNCH N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services[J]. ACM SIGACT News, 2002, 33(2): 51-59.

[25] KARGER D, LEHMAN E, LEIGHTON T, et al. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web[C]//Proceedings of the 29th Annual ACM Symposium on Theory of Computing. New York, USA: ACM Press, 1997: 654-663.

[26] YOO A B, JETTE M A, GRONDONA M. SLURM: simple Linux utility for resource management[C]//Proceedings of Workshop on Job Scheduling Strategies for Parallel Processing. Berlin, Germany: Springer, 2003: 44-60.

编辑 索书志