太湖之光上利用OpenACC移植和优化GTC-P

王一超[1] 林新华[1,2] 蔡林金[1] William Tang[3] Stephane Ethier[3] Bei Wang[3] Simon See[1,4] 松岗聪[2]
1（上海交通大学高性能计算中心 上海 200240）
2（东京工业大学 日本东京 1528550）
3（普林斯顿大学等离子体物理实验室 美国普林斯顿 08540）
4（英伟达公司 新加坡）
james@sjtu.edu.cn

摘要：
神威“太湖之光”是最新一期Top500榜单上排名第一的超级计算机，实测峰值性能约93PFLOPS。该系统提供了基于指导语句的并行编程工具OpenACC，兼容OpenACC 2.0编程标准，并添加了部分定制化功能。GTC-P是一个具有重要物理意义的科学应用，算法基于高性能计算领域中被广泛使用的particle-in-cell方法。我们利用神威OpenACC并行编程模型在“太湖之光”上成功移植了GTC-P应用。在移植过程中，鉴于OpenACC编译器尚无法解决的性能瓶颈，提出了3种基于中间代码二次开发的优化方法：1）消除原子操作；2）避免低效的全局访存操作；3）手动添加SIMD intrinsics指令．实验结果表明，在64个从核上相比1个主核，优化后的函数charge和push分别实现了1.6倍和8.6倍的加速比，同时GTC-P代码整体取得了2.5倍的加速比。优化结果证明了基于中间代码的手动优化对利用神威OpenACC移植的PIC算法在“太湖之光”上的性能提升非常重要。

关键词：太湖之光，GTC-P应用，PIC算法，神威，OpenACC语言

国家超级计算无锡中心的神威“太湖之光”超级计算机在2016年6月发布的世界Top500超算榜单中排名第一，其理论峰值性能达到125.4PFLOPS，Linpack实测效率74.15%，约为93PFLOPS[1]。值得一提的是，“太湖之光”搭载的处理器芯片是完全由中国自主研发的神威异构众核处理器SW26010。该芯片的架构面向高性能计算，通过统一指令系统、统一执行模型和支持一致性的主存共享，实现异构核心的深度融合[2]。

在软件层面，“太湖之光”的操作系统和编译环境等也全部由中国自主研发。该系统实现的并行编程模型神威OpenACC，兼容OpenACC 2.0标准，还添加了部分定制功能。基于指导语句的并行编程模型可以帮助用户在应用移植过程中，避免硬件架构不同带来的差异，提供高可移植性。本文将研究在神威众核处理器上利用OpenACC对于实际应用进行移植及优化。

由美国普林斯顿大学开发的高性能计算应用GTC-P（the Princeton gyrokinetic toroidal code）利用 PIC（particle-in-cell）算法求解Vlasov-Poisson方程，模拟粒子在托卡马克装置内的运动，具有重要的物理意义。又由于其极佳的可扩展性，GTC-P已在Top500排名前10的6台超级计算机上进行了性能测试[3]，并入选了美国NERSC国家超算中心的基准测试集[4]，是最具代表意义的PIC应用。

综上所述，本文使用OpenACC在“太湖之光”超级计算机上成功移植了GTC-P应用。针对神威OpenACC编译器尚无法解决的性能瓶颈，本文提出了3种基于中间代码二次开发的优化方法．实验证明优化方法有效，同时我们还对神威OpenACC结合

基金项目：国家重点研发计划（2016YFB0201400，2016YFB0201800）；美国自然科学基金跨学科合作项目(ACI-1440733)；NVIDIA GPU全球卓越中心；日本学术振兴会RONPAKU项目(113209)

MPI的多节点版本进行了强可扩展性的评估。

简而言之，本文有以下2点贡献。

1) 首次在“太湖之光”上利用OpenACC成功并行移植了基于PIC算法的实际应用GTC-P。

2)提出了3种基于中间代码的手动优化方法：1消除原子操作；2避免低效的全局访存操作；3手动添加SIMD intrinsics指令。实验证明优化方法对于PIC算法在“太湖之光”上的性能提升效果显著。

1. 相关工作

本文的相关工作主要有以下3类。

1)“太湖之光”上的应用移植：清华大学付昊桓等人[5]利用OpenACC在“太湖之光”上成功移植了大气模型CAM应用。实验结果显示，利用OpenACC移植后的应用，相比1个主核，在64个从核上整体实现了2倍的加速比。但该论文尚未提出在“太湖之光”上针对OpenACC移植后的性能优化方法。

2) GTC-P的移植与调优：普林斯顿大学Wang等人[6]对GTC-P应用在Mira, Sequoia, Hopper等超级计算机上的移植与调优做了介绍，并评估了可扩展性。在异构系统上的移植方面，美国劳伦斯伯克利国家实验室Ibrahim等人[7]介绍了利用CUDA在基于GPU的Titan超级计算机上移植GTC-P的方法。但这些工作都是在基于CPU, GPU, MIC加速卡构建的超算系统上完成的，与神威众核的架构存在差异。

3)利用OpenACC移植应用：Calore等人[8]和Hariri等人[9]分别利用OpenACC对基于Lattice Boltzmann和particle-in-cell算法的科学应用进行了并行移植，并介绍了常用的优化手段，移植平台为GPU和CPU。东京工业大学Hoshino等人[10]对于利用OpenACC在GPU上进行并行移植存在的性能瓶颈问题做了深入的研究，并指出缺乏对于GPU shared memory的控制是导致性能的原因。但本文中所提及的神威OpenACC实现上与OpenACC标准存在差异。

2. 背景介绍

本文的研究背景介绍将从神威众核处理器架构、神威OpenACC以及GTC-P应用展开。

2.1 神威众核处理器架构

神威众核处理器旨在用少量具备指令级并行能力的管理核心，集成众多面向计算开发的精简运算核心高效处理线程级并行，从而大幅提高芯片性能[2]。

“太湖之光”超级计算机上所使用的神威SW26010芯片架构如图1所示，1块芯片由4个核组组成，每个核组内64个从核按照8×8的mesh拓扑结构，由片上内部网络互联。

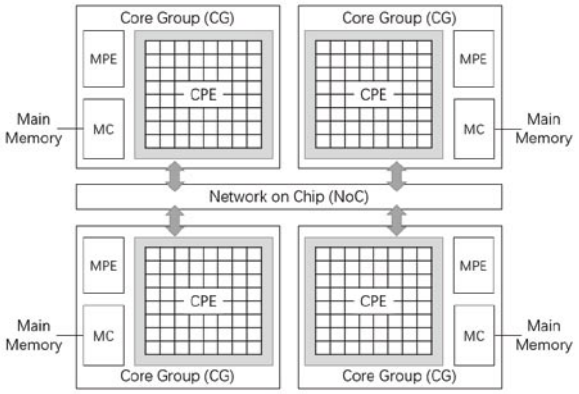

图1 神威众核处理器架构图

此外，主从核均支持256b向量化指令，以及乘加融合FMA（fused multiply add）指令。每个从核支持一条浮点数DP（double precision）流水线，主核支持双浮点数流水线。由此可得，主从核双精度理论峰值性能的计算公式如下：

$$1 \times 1.45 \times 8 \times 2 = 23.2\text{GFLOPS};$$
$$64 \times 1.45 \times 4 \times 2 = 742.4\text{GFLOPS}$$

不难发现，SW26010上逾98%的理论性能均源自于从核，所以应用必须在从核上得到并行化移植才能充分发挥“太湖之光”超级计算机的性能。

存储架构方面，主核拥有2级缓存：第1级缓存指令数据分离，即指令和数据各自拥有32KB缓存；第2级则为指令数据共享的256KB缓存。而每个从核拥有16KB的一级指令缓存。从核数据存储的设计是一种类CELL处理器的用户控制的草稿本方式存储，即SPM（scratch pad memory），每个从核的SPM容量为64KB。这种存储方式不仅省去了缓存实现上的控制开销，还避免了众多运算核心间一致性处理带来的设计复杂性和性能降级[2]。但也为应用开发中从核的数据管理带来了新挑战，即需要人为对于应用在SPM上的访存进行设计与规划。

SW26010片上集成了4个内存控制器，每个核组拥有8GB内存。但在具体实现上，内存管理器仍是基于DDR3的，芯片的实测访存带宽为136.51GB/s。

基于上述设计，相比NVIDIA GPU K20X或者Intel Xeon Phi 7110P在Stream (Triad)测试中实测180GB/s左右的访存带宽，SW26010的带宽差了近25%。由此，在SW26010上移植应用，性能更容易受限于访存带宽。充分利用SPM局存访问，缓解主存访问压力，就成了应用移植和优化的关键之一。

2.2 神威OpenACC

神威OpenACC是“太湖之光”上基于定制OpenACC 2.0标准实现的并行编程模型，面向从核的

并行化移植。其指导语句的设计与OpenACC 2.0标准存在一定区别，具体情况如表1所示：

表1 神威OpenACC与OpenACC 2.0标准的对比

Comparison	*Syntax*
Supported	*parallel, loop, copy/copyin/copyout, data, atomic, private*
Different	*async, tile, cache, gang, worker*
Not supported	*vector, independent, create*
Customized	*swap/swapin/swapout, local, pack/packin/packout, annotate*

这样的设计主要出于3方面的考虑：

1）由于神威架构与GPU和MIC等主流异构加速器存在区别，导致如tile，cache等语句在实现上与标准OpenACC的实现产生了差异。

2）基于神威众核的存储架构，添加如pack/packin/packout和swap/swapin/swapout等神威OpenACC定制的指导语句，为使用户更充分地利用较为有限的访存带宽。

3）自主研发的神威OpenACC编译器，与PGI，Cray等成熟的商业编译器相比，还存在不完善之处。尤其在数据管理方面，需要指导语句给予编译器更多的提示，这反映在了定制的annotate语句中。

此外，神威OpenACC编译器是基于ROSE开源框架开发的代码转译器，主要负责中间代码生成。其后端仍使用的是“太湖之光”的本地编译器。该编译器向用户开放中间代码，允许用户基于编译生成的中间代码进行二次开发。第4节的性能优化中，本文就会用到该功能。

2.3 GTC-P应用

GTC-P是由美国普林斯顿大学等离子体物理实验室开发的应用于磁约束聚变领域中全局性大规模并行模拟的高性能计算程序。该应用在近10年来，一直都是研究托卡马克装置内等离子体湍流运动等重要非线性物理问题的主要工具。

粒子在1束激光中的运动可以通过六维的Vlasov-Maxwell方程来模拟[11]。而对于托卡马克这种带有零阶稳态平衡导向场的等离子体系统，回旋动理学的简化降维又是非常有效的。在这种情况下，磁场中的带电粒子的回旋运动将近似于一个带电环。而PIC算法则是现如今最为普及的求解该系统方程的数值计算方法。

基于回旋动理学的PIC方法的每个时间步主要分为5个计算步骤，对应到代码中的主要函数。1）charge：使用4点陀螺平均法求解粒子到网格上的电荷沉积；2）poisson：在网格上求解回旋动理学的Poisson方程；3）field：计算网格上的电场；4）push：用矢量场和其他衍生物助推粒子；5）smooth：使电荷密度和潜在矢量平滑。其中，charge和push的计算强度均小于2。结合SW26010的峰值性能和访存带宽可知，计算强度低于20的应用在神威平台上的性能就会受限于访存性能了，因此这2个函数的访存实现将会成为移植过程中的主要难点。

3. 利用OpenACC移植GTC-P

在使用神威OpenACC进行从核并行化移植之前，本文首先将GTC-P在x86平台上实现的串行代码移植到SW26010的1个主核上。并记录了各部分函数的执行时间，分析了程序热点，结果如图2所示：

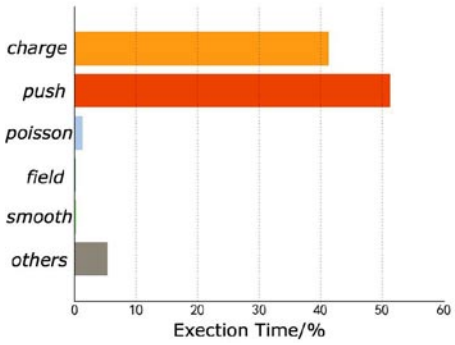

图2 GTC-P代码热点分析

可见，主核上的热点测试结果与上文的分析相印证，即访存密集型的函数charge和push是程序的主要热点。本文将对于GTC-P应用中的这2个函数进行从核并行化移植与优化，移植工作分为以下3个步骤：数据管理、循环并行化、原子操作。

3.1 数据管理

如2.1节中提到的，在神威众核架构中，存在着访存带宽性能有限的潜在瓶颈。所以从应用性能角度出发，将计算涉及的数据提前拷贝至访问延迟低的SPM局存中，是从核并行化移植的关键步骤。OpenACC2.0标准中的data指导语句下提供了如copy, present, cache等多种数据管理模式，在神威OpenACC中为提升DMA带宽添加了swap，pack等定制语句。本文采用了将所有在计算中涉及的数据优先传输至SPM的实现策略，具体实现方法如下：

1）按循环索引划分传输。若数组的索引变量与循环的索引变量紧耦合时，神威OpenACC编译器会自行将数组划分为64份，然后利用DMA的方式分别将64份数据传输至各从核的SPM中。具体使用如图3所示，其中 annotate(dimension(array(size))) 指导语句是为了指明数组长度，方便编译器划分数据。

2）完整传输。当访问的数组与循环索引无关联时，由于SW26010缺少共享的局部存储，编译器将无法判断如何划分数据，从而默认不会把该数组传输至SPM中。在此情况下，可以通过annotate(entire(array))指导语句提示编译器将数组的完整副本分别传输至64个从核的SPM上。

3）转置后传输。swap/swapin/swapout是神威OpenACC定制的指导语句。该语句旨在解决循环索引与数组索引不对应的问题。方法是在数据传输之前先在主核上对数组进行1次高效的转置操作，使数组的第1维索引与循环索引对应，而后再根据第1种方法传输数组。本文在charge中使用了该方法，若不使用，则编译器无法进行数据传输。

4）使用缓存。OpenACC 2.0中的cache指导语句在SW26010上实际对应的是SPM。又因为SPM没有实现硬件层的缓存管理机制，所以神威OpenACC的cache指导语句背后是一个软件实现缓存机制。

基于以上4种数据管理方法的函数charge实现如图3所示。其中，copyin语句中的z0，z1，z2等数组编译后会基于循环索引m均等划分，分别传输至64个从核的SPM中。而delt, mtheta, igrid, qtiny这4个数组则将被完整地传输至64个从核的SPM中，等同于在每个从核的SPM中保存了一份这4个数组的副本，方便从核随时进行访问。copyout指导语句中的数组传输方法与z0等相同，按照循环索引进行划分。值得一提的是cache指导语句中的pgyro和tgyro，这两个数组的访存是不规则的，无法根据循环索引进行均等划分；并且由于数组所占的空间大于500KB，远大于SPM的64KB空间，所以也无法完整传输。

```
1  #pragma acc parallel loop
2  copyin(z0,z1,z2,z4,z5) annotate(dimension(z0(mi),z1(mi),z2(mi),z4(mi),z5(mi)))
3  copyin(delt,mtheta,igrid,qtinv) annotate(dimension(delt(90+1),mtheta(90+1),igrid(90+1),qtinv(90+1)), entire(delt,mtheta,igrid,qtinv))
4  copyout(wpion,wtion0,wtion1,jtion0,jtion1) annotate(dimension(wpion(4*mi),wtion0(4*mi),wtion1(4*mi),jtion0(4*mi),jtion1(4*mi)))
5  cache(pgyro,tgyro)
6  for (m = 0; m < mi; m++) {
```

图3 tile基于4种数据管理策略的函数charge实现

3.2 循环并行化

在循环并行化方面，神威OpenACC与OpenACC 2.0标准保持一致，使用parallel和loop语句使代码在64个从核上并行执行。

需要注意的是，gang, worker, vector是OpenACC 2.0标准中的3层循环设计，由于神威众核架构在物理上并没有分层需求，所以神威OpenACC的实现是默认把gang设置为64，worker设为1，而vector也没有对SIMD功能做支持。

3.3 原子操作

为保证数据竞争情况下的结果正确性，神威OpenACC与OpenACC 2.0一样，都支持了原子操作。atomic指导语句可以保证数据在同一时间只会被某一个进程读写。在GTC-P的函数charge中，数组densityi_part由于不规则访问会产生数据竞争。为保证结果的正确性，需要添加atomic指导语句。

针对charge中的原子操作，本文利用“太湖之光”上提供的性能分析工具[12]测得执行一次原子操作的开销约为750个周期。同时，由于数据竞争问题的存在，导致原子操作的数组无法通过copy指导语句传输至64个从核的SPM中，使得计算部分必须进行主存访问，增加了额外的访存开销。

表2 从核并行化移植与主核执行效果对比

Function	*Execution Time on MPE*	*Execution Time on CPE*
charge	*4.68*	*2360.52*
push	*4.45*	*14.46*
poisson	*12.25*	*12.40*
field	*1.67*	*1.70*
smooth	*2.94*	*2.72*
Total	*26.34*	*2392.14*

表2为利用上述方法移植GTC-P后，主核与从核版本之间的执行时间对比，从核版本比主核版本慢了将近92倍．造成性能瓶颈的原因有以下2点：

1）神威众核处理器上的原子操作由锁机制实现，实际的运行开销大。

2）从核之间缺乏共享局部存储，为配合原子操作，无法将数据预存入各从核的SPM中。

以上原因与神威众核的硬件架构相关，目前仅依靠神威OpenACC指导语句是无法解决该性能瓶颈的。

4. 性能优化

针对上一段文末提出的性能瓶颈原因，本文提出不依赖于神威OpenACC编译器直接编译得到的可执行程序，而是基于中间代码的二次开发来优化性能。

4.1 消除原子操作

为了解决多从核之间的数据读写竞争问题，同时又避免原子操作，我们先对于函数charge中相关的代码段进行了分析，如图4所示。

这部分的数组操作为累加，在从核并行时并不涉及共享同步机制，因此本文采用的优化策略是在每个从核的SPM中保存一个数组副本，在代码段执行完之后对数组副本进行1次统一的归约求和计算。

```
1  #pragma acc atomic
2  densityi_part[ij1] += d1;
3  #pragma acc atomic
4  densityi_part[ij1+1] += d2;
5  #pragma acc atomic
6  densityi_part[ij1+mzeta+1] += d3;
```

图4 函数charge原子操作代码段

由于需要分别对64个从核SPM中的数组副本进行维护，这部分优化需要利用神威OpenACC编译器在中间代码中提供的acc_myid接口才能实现。通过用线程号维护densityi_part[acc_myid][]来实现数组副本的管理，代码转换如图5所示：

```
1  densityi_part[acc_myid][ij1] = d1;
2  densityi_part[acc_myid][ij1+1] = d2;
3  densityi_part[acc_myid][ij1+mzeta] = d3;
4  densityi_part[acc_myid][ij1+mzeta+1] = d4;
```

图5 数组根据从核进程号维护副本

因为数组densityi_part大于500KB无法完整存放到SPM中，为了在不规则访问的情况下尽可能避免主存访问，本文在该代码段前添加copyin语句，并结合变量索引进行DMA数据传输，即在每次对于数组副本densityi_part 进行累加前，先将涉及到的数据传输到SPM中。优化后的性能如表3所示：

表3 消除原子操作的优化效果

Execution Time on MPE	Execution Time on CPE w Atomic	Execution Time on CPE w/o Atomic
4.68	2360.52	12.58

在消除了原子操作和主存访问带来的额外开销之后，函数charge实现了约180倍的加速，性能接近于主核版本，但仍慢了将近1/3，需要进一步优化。

4.2 提升DMA传输带宽

如2.1节所述，在神威众核架构下，64个从核同时发起DMA访问时，访存带宽可以达到30GB/s。然而神威OpenACC中的copy/copyin/copyout指导语句默认却是按照单次循环涉及的数组元素进行传输的，若不进行优化，则无法提高DMA传输带宽。

指导语句tile是OpenACC02.0标准中的语句，在CPU和GPU上的实现是为了通过循环合并达到更好的数据局部性。神威OpenACC实现的tile指导语句则是通过设定循环合并的层数来控制每次DMA传输的数据块大小，从而达到提升传输带宽的目的。根据“ pragma acc data copy(z0) tile(250) ”的指导语句编译出的中间代码示例如图6所示。其中acc_sync_pe_m2l_nostride_copy为实际的DMA数据传输接口，而acc_cp_4_size则为传输数据的尺寸参数，则此次传输的数据块为250×8 B，即一次DMA传输约为2KB的数据。

```
1  for (acc_blockindex_m0_1 = acc_myid * 250; acc_blockindex_m0_1 <= _ldm_mi - 1; acc_blockindex_m0_1 += 250 * acc_corenum)
2      acc_loop_ub_tmp_var0 = _ldm_mi - 1 - acc_blockindex_m0_1;
3      acc_loop_ub_m = min(249, acc_loop_ub_tmp_var0);
4      acc_cp_4_size = (acc_loop_ub_m + 1) * 8;
5          _swacc_ret = acc_sync_pe_m2l_nostride_copy(_ldm_z0,&z0[acc_blockindex_m0_1],acc_cp_4_size);
```

图6 tile指导语句生成的中间代码

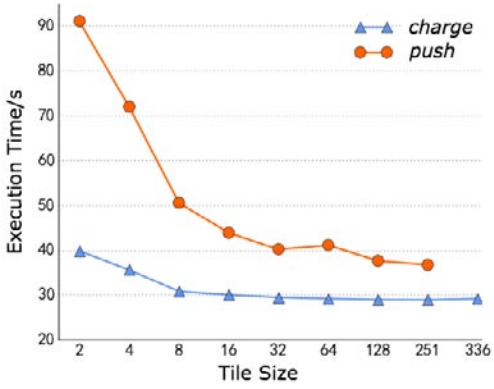

图7 神威OpenACC的tile尺寸测试

根据上述原理，本文针对charge和push中的tile尺寸设定做了测试实验，结果如图7所示。伴随tile尺寸增大，执行时间变小，性能得到优化，但受限于实际可用的SPM大约为58KB，在这2个函数中可用的tile尺寸最大分别为251和336。

与消除原子操作的版本相比，在提升DMA传输带宽后，函数charge和push分别实现了2.7及4倍的加速比。

4.3 避免低效的访存操作

从核访主存操作主要分为2类，通过全局离散访问主存gld/gst（global load/store）指令，或者通

过 DMA方式批量访问主存。神威OpenACC代码中的copy/copyin/copyout，swap/swapin/swapout指导语句都是利用DMA的批量访问实现的，而未指定的主存访问则默认采用全局离散访问方式。1条gld指令的延迟约为127个周期，是1次访SPM操作的30倍，属于低效的访存操作。此外，当64个从核同时发起gld/gst指令时，这些全局访存指令将排队依次执行，从而大大降低并行效率，造成更大的资源浪费。所以，冗余的gld/gst指令需要尽可能避免。

“太湖之光”上提供的性能分析工具中的penv2_slave_gld_count接口可以收集中间代码里的gld/gst指令数目，帮助用户发现冗余指令。该工具提供了一系列的函数调用接口，可以帮助用户收集硬件相关的统计数据，包括总指令数、load/store指令数、TLB miss数等[12]，使用代码如下所示：

```
//主核代码部分
penv2_slave_gld_init();
…
//从核函数调用
…
//从核代码部分
    penv2_slave_gld_count(&ic1);
    …
    penv2_slave_gld_count(&ic2);
```

在函数push中发现的冗余gld/gst指令，主要分以下2种情况：

1）直接访问保存在主存上的指针。对于这类情况的优化方法是，在SPM上重新声明一个局存指针，然后在代码段初始部分将指针地址赋值给局存指针，使其后的指针操作都是直接访SPM，而非通过gld指令的全局离散访主存。

2）“太湖之光”编译环境中默认链接的数学库需要进行离散访主存操作，而函数push 需要使用sin, cos, exp这3个数学库计算。本文调用了加载在SPM上的优化从核数学库[13]。该数学库完全在SPM上实现，所以不会产生gld/gst指令，但会占用更多的SPM空间，需要提前规划好SPM的使用。最新版本中双精度的函数cos和sin各需要4KB空间，而函数exp 占用9KB空间。

在对函数push进行消除低效的访存操作后，对比上一步的优化结果，进一步取得了6.7倍的加速比。

4.4 手动添加SIMD intrinsics

正如2.1节中的理论峰值性能公式所示，256b的SIMD向量化（相当于4个双精度数）操作对于性能起了重要作用。若所有计算都仅使用64b的双精度标量计算，实际上即是损失了75%的计算性能。神威OpenACC中尚未提供针对SIMD的支持，且自动SIMD向量化一直以来是编译器的实现难点。所以本文采用在中间代码中手动添加SIMD intrinsics指令的方法，不依赖编译器，手动向量化计算部分的代码。

本文针对函数charge 中的部分计算进行了向量化优化，实现示例如图8所示：

```
1  for (_ldm_i = 0; _ldm_i <= acc_loop_ub_i; _ldm_i += 1) {
2    vtmp = simd_set_doublev4(0,0,0,0);
3    for (_ldm_j = 0; _ldm_j <= 63; _ldm_j += 4) {
4    simd_load(vd,&ldm_swap_mydensityi_local0[_ldm_i][_ldm_j]);
5      vtmp = simd_vaddd(vd,vtmp);
6      _ldm_denloc_sum+=_ldm_swap_mydensityi_local0
                                [_ldm_i][_ldm_j];
7    }
8    simd_store(vtmp,&tmp[0]);
9    _ldm_densityi[_ldm_i] = tmp[0] + tmp[1] + tmp[2] + tmp[3];
10 }
```

图8 函数charge向量化优化代码

通过在中间代码中添加SIMD intrinsics指令，对比执行周期数，该部分代码实现了5.6倍的加速比。结果证明该优化方法效果明显．但由于该代码段在全局中所占的时间尚不足1%，所以对于整体的GTC-P性能并未产生影响。

5. 实验验证

本节在“太湖之光”上针对第4节提出的优化方法进行了实验验证。其中，手动添加SIMD intrinsics指令的方法对整体性能影响不大所以未加入本实验。

5.1 性能对比分析

本文将运行在1个主核上的串行版本GTC-P作为测试基准，分别与消除原子操作、提升DMA传输带宽、消除冗余gld/gst指令这3个优化方法在单核组的从核上进行了性能对比，结果如图9所示。该测试的问题规模为A，即回环中有3 235 896个粒子。

除了4.1节中提及的消除原子操作优化效果显著外，本节的实验结果还验证了以下2点：

1）通过tile指导语句优化DMA传输效率，对于神威众核上访存密集型的应用，是有效且必要的。该优化对比消除原子操作的版本，在函数charge和push上分别实现了2.7及4倍的加速比，并使得GTC-P在从核上的整体性能要高于主核上的。

2）去除中间代码中冗余gld/gst指令可以有效优化应用性能。优化后的函数push相比tile优化后的版本又实现了6.7倍的加速比。

在单核组实验中，函数charge和push在从核上的最终版本对比主核版本分别实现了约1.6倍和8.6倍的加速比，对于整个GTC-P应用而言，实现了2.5倍的加速比。

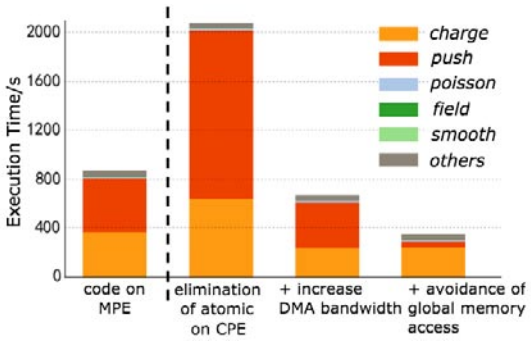

图9 GTC-P性能对比

5.2 强可扩展性分析

本文在利用OpenACC实现的单核组版本基础上，结合MPI[6]在“太湖之光”上测试了OpenACC+MPI版本，分析了强可扩展性。问题规模较之前的性能对比，使用了输入B（1 235 644 416个粒子），步长仍为100．其中，图10(a)为输入规模A的结果，图10(b)为规模B的结果。

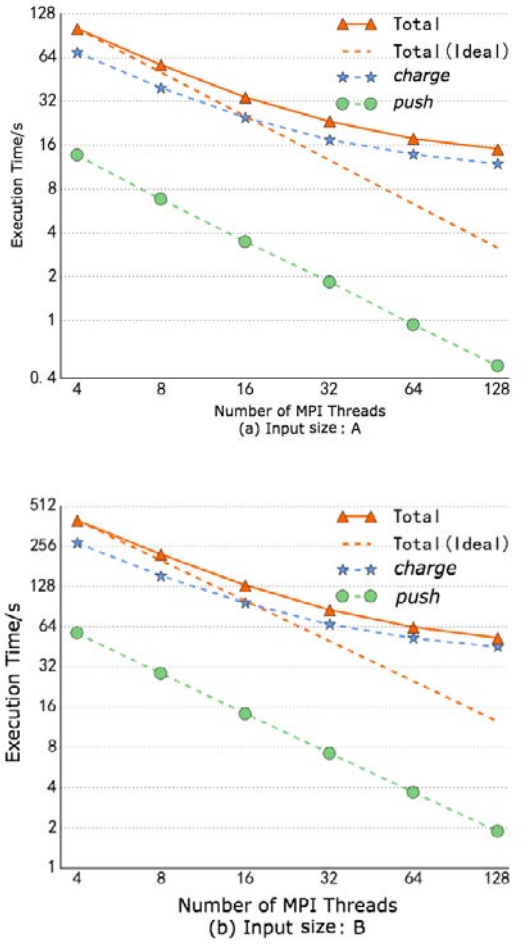

图10 针对不同问题规模的GTC-P在从核上的强可扩展性分析

图10中的横轴代表MPI的线程数，1个MPI线程控制1块SW26010芯片上的1个核组，即4个MPI线程控制整块芯片（即单节点）。实验的线程数从4一直扩展到了128，即从1个节点扩展到了32个节点。

伴随并行计算节点数的上升，函数push的强可扩展性表现要优于函数charge的，执行时间基本呈理想的线性递减趋势。而GTC-P应用的整体强可扩展性则未与理想的线性趋势相吻合。主要原因是由于GTC-P的整体执行时间由函数charge主导（占80%左右），而charge函数则受限于不规则访存导致的性能瓶颈，尚需进一步优化。

6. 总结

本文利用OpenACC在“太湖之光”上并行移植了GTC-P应用。在实现了完全基于指导语句的移植版本后，我们发现神威OpenACC编译器尚无法解决由于原子操作开销大以及离散访存操作代价高所导致的性能瓶颈。本文提出，不依赖于编译器，而是基于中间代码进行二次开发，通过消除原子操作和冗余gld/gst指令，在中间代码里手动添加SIMD intrinsics指令的优化方法，提升应用性能。实验结果证明所提出的优化方法有效。

此外，本文还通过与OpenACC2.0标准进行对比，提出了利用OpenACC移植应用到“太湖之光”上需要注意的地方，总结为以下3点：

1）在data指导语句使用过程中，需要注意数组索引与循环索引的关联性，尽可能使数组可以按索引划分并传输至从核的SPM中。若无法划分且所占空间小于64KB，则建议可利用annotate(entire(array))指导语句将数组完整传输至SPM。

2）swap/swapin/swapout是神威OpenACC中定制的指导语句，可以帮助由于索引问题无法被神威OpenACC编译器自动识别传输的数组，转置后实现正常传输，提升性能。

3）tile指导语句对于应用在“太湖之光”上的优化效果显著，GTC-P应用中部分函数通过该指导语句优化实现了3~4倍的加速比。

我们下一步将探索适应神威众核架构，尤其是针对64KB SPM局存设计的新算法，并利用底层编程方法对于GTC-P进行深度优化。

致谢

感谢国家并行计算机工程技术研究中心的刘鑫、尤洪涛、何香老师在本次研究中提供的大力支持。