DOI: 10.16452/j.cnki.sdkjzk.2021.04.009 文章编号:1672-3767(2021)04-0076-10

# 神威平台上 AceMesh 编程模型的构图优化

叶雨曦[1],傅 游[1],梁建国[1],孟现粉[2],刘 颖[3],花 嵘[1]
(1.山东科技大学 计算机科学与工程学院,山东 青岛 266590;
2.中科寒武纪科技股份有限公司, 北京 100191; 3. 中国科学院计算技术研究所,北京 100190)

**摘 要**:面向高性能计算领域的多核、众核处理器飞速发展,为了降低并行编程的难度,提高并行计算效率,数据驱动的并行编程模型成为高性能计算领域的研究热点。AceMesh 是数据流驱动的、支持多核和众核异构平台的任务并行编程模型,能自动发掘结构化网格应用中存在的数据驱动的任务图并行性。但如果任务粒度划分较细,其构图过程会造成很大开销。本研究结合"申威 26010"异构众核处理器的结构特点,从主、从核通信优化、内存池、无后继任务收集等方面对 AceMesh 构图过程进行优化,并采用航天飞行器应用中的 7 个热点子程序对优化效果进行测试。测试数据表明以上优化取得 5 倍的加速。为验证构图优化对 AceMesh 整体性能的提升,对航天飞行器应用分别在 Acemesh 和神威 OpenACC 的加速效果进行了测试,优化后的 AceMesh 加速效果约为神威 OpenACC 的 1.5 倍。

**关键词**:DAG 构图优化;任务并行编程模型;神威·太湖之光;申威处理器;性能
**中图分类号**:TP311.52 **文献标志码**:A

## Composition optimization method of AceMesh programming model on Sunway TaihuLight Platform

YE Yuxi[1], FU You[1], LIANG Jianguo[1], MENG Xianfen[2], LIU Ying[3], HUA Rong[1]
(1. College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, Shandong 266590, China; 2. Zhongke Cambrian Technology Co, Ltd, Beijing 100191;
3. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

**Abstract**: In recent years, the multi-core and many-core processors have developed rapidly. To reduce the difficulty of parallel programming and improve the efficiency of parallel computing, the data-driven task-parallel programming model has become a research hotspot in the field of high-performance computing. As a data-driven task-parallel programming model that supports multi-core and many-core heterogeneous platform, AceMesh can automatically discover the data-driven task graphs parallelism in structured grid applications. However, the composition process of AceMesh can be costly if the task is fine-grained. Based on the unique architecture of SW26010 processor, the composition process of AceMesh's task graphs was optimized by moving the communication variable to Local Data Memory (LDM), memory pool, and no follow-up task collection. The optimization effect was tested by using seven hot spot subroutines in an aerospace craft application. The test data shows that the optimization has brought about 5 times performance improvement for composition. To verify the improvement of AceMesh's overall performance by composition optimization, the acceleration effects of AceMesh and Sunway OpenACC on aerospace vehicle applications were compared and the results show that AceMesh has 1.5 times the speedup of Sunway OpenACC.

**收稿日期**:2020-03-03
**基金项目**:国家重点研发计划项目(2016YFB0200803);山东省重点研发计划项目(2019GGX101066)
**作者简介**:叶雨曦(1997—),男,山东德州人,硕士研究生,主要从事分布式处理与并行计算等方面研究.
花 嵘(1969—),男,江苏常州人,博士,副教授,主要从事高性能计算等方面研究,本文通信作者.
E-mail: huarong@sdust.edu.cn

**Key words**: DAG′s composition optimization; task-based parallel programming model; Sunway TaihuLight; SW26010 processor; performance

任务并行编程模型把任务作为并行的基本单位,为编程人员提供任务划分和任务同步接口。编程人员需要考虑如何对应用程序进行合理的任务划分,而任务具体的要在哪个物理核上执行以及如何实现任务之间的同步都由运行时系统完成。任务并行编程模型可看作底层体系结构和上层程序应用之间的桥梁,向上隐藏并行处理的细节,提高编程层次,简化并行编程;向下充分利用硬件资源,高效正确地执行并行程序[1]。支持任务并行调度的并行编程模型主要有 MIT 的 Cilk[2]/Cilk++[3]、Intel 的 Threading Building Blocks (TBB)[4]、BSC 的 OmpSs[5]、微软的 Task Parallel Library(TPL)[6]等。

中国科学院计算技术研究所并行编译组提出了一种面向网格应用,以数据为中心,应用于多核、众核平台上的任务并行编程模型 AceMesh[7-10]。AceMesh 主要包括编译器和任务调度系统。AceMesh 编译器作为一个源到源编译框架[5],根据平台特性将带指导语句的程序代码自动生成指定平台的并行化程序,编程人员只需要关注指导语言和应用本身即可。AceMesh 任务调度系统采用基于有向无环图[11](directed acyclic graph,DAG)的动态任务调度,能自动发掘结构化网格应用中存在的数据驱动的任务图并行性,其性能较其他的任务并行编程模型优越[12-13]。

AceMesh 作为数据流驱动的任务并行模型,细粒度的任务划分会使其构图开销显著增加,降低应用的并行可扩展性。王松等[14]提出了并发构图的方法,对 AceMesh 构图进行了优化,但该方法在访存、内存申请和任务收集方面存在一定的问题,导致 AceMesh 构图过程开销仍过大。为提高构图效率,本研究从减少通信开销、设计内存池和优化任务收集策略等角度对 AceMesh 构图过程进行优化,通过数据测试验证构图优化的可行性和有效性。

## 1 “神威·太湖之光”平台简介

“神威·太湖之光”[13-15]计算机系统采用“申威 26010”异构众核处理器,芯片工作频率 1.45 GHz,峰值运算速度 3.168 TFLOPS,采用了针对该处理器定制的 64 位申威指令系统,与 X86 指令系统不兼容。“申威 26010”异构众核处理器架构如图 1 所示。

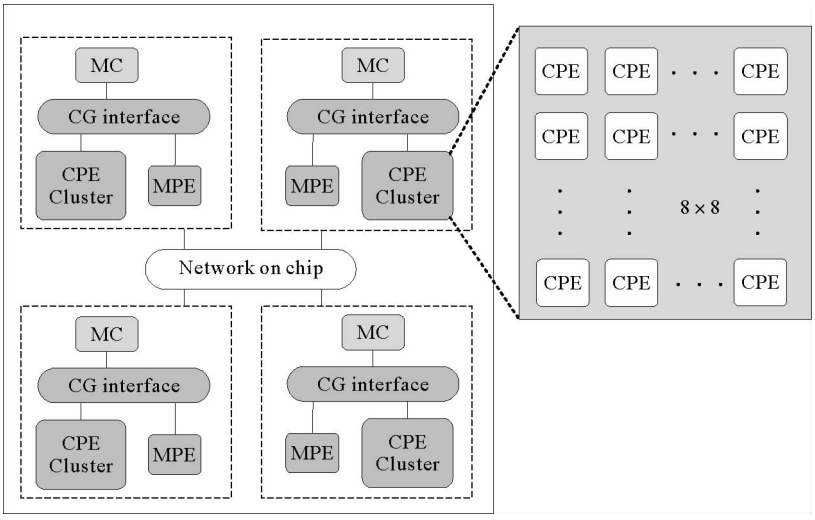


**图 1 “SW 26010”异构多核处理器体系结构**

Fig. 1 Architecture of “SW 26010” heterogeneous many-core processor

“申威 26010”异构众核处理器集成了 4 个运算核组,共 260 个运算核心,核组间支持 Cache 一致性,通

过高速片上网络相连。每个核组包含 1 个运算控制核心(management processing element, MPE)即主核、1 个运算核心(computing processing elements,CPE)即从核阵列和 1 个存储控制器(memory controller, MC),通过 iMC 与主存相连。众核处理器还集成系统接口总线用于连接标准 PCIe 接口,实现片间直连和互连,管理与维护接口负责系统管理、维护和测试。4 个核组和系统接口总线通过群间传输网络实现存储共享和通信。主核具有两级片上存储层次:L1 级数据和指令 Cache、共享的 L2 级 Cache。1 个计算核组中的 64 个从核共享 L2 级指令 Cache。每个从核具有私有 L1 级指令 Cache 和用于数据存储的 LDM 空间。应用程序由主核启动,借助高性能线程库 Athread 将计算任务加载到从核执行,双方通过同步接口协同。

## 2 AceMesh 任务调度系统的构图过程及其存在的问题

AceMesh 任务调度系统的工作机制如图 2 所示。用 DAG 将计算任务之间的依赖关系表达出来的过程称为构图,利用构建的 DAG 分析保持数据原有依赖关系的正确任务执行顺序,并对任务进行调度和计算的过程,称为图执行。DAG 的构建由主核独自完成,执行由主核和从核共同完成。

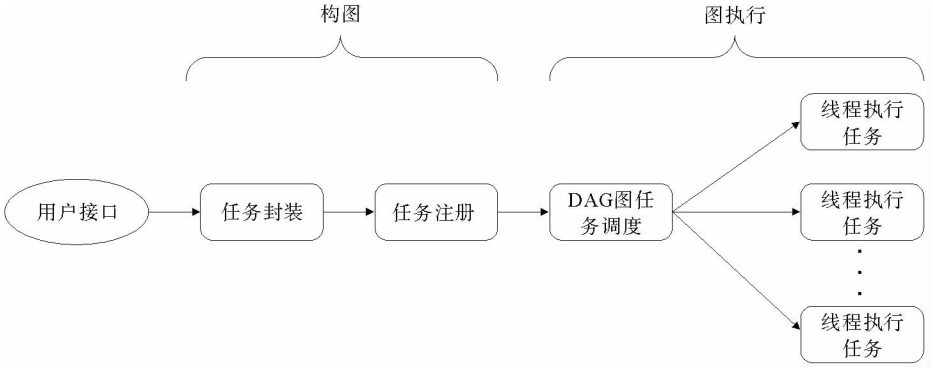


**图 2 AceMesh 任务调度系统工作机制**

Fig. 2 Working mechanism of AceMesh task scheduling system

构图过程包括两步:一是数据域的划分及计算任务封装。根据代码访问数据区域的不同,将整个数据域划分成若干个数据区域。为减少通信量,尽可能把具有依赖关系的数据区域分配到一个数据块中,并将数据块、对该数据块的操作和需要通信的邻居节点封装成一个计算任务;二是计算任务注册。为了更好地发掘缓存中的数据重用性,初始化计算任务的亲和性设置,根据最近、可达的数据访问信息如变量地址、邻居关系、读写信息等,构建正在注册的计算任务与前驱计算任务间的数据依赖边,并记录为后继任务;将没有前驱的任务存放在无前驱任务列表 need_spawn_tasks 中,作为最先调度的任务;查找无后继任务并将其存放在无后继任务列表 end_tasks 中,用于检查 DAG 执行结束状态。

在构建 DAG 的过程中存在下面 3 个问题:

1) DAG 构建由主核独立完成,为了在构建完成后唤醒从核,主核需要在主存上维护一个共享变量 is_run。主核构建 DAG 时将 is_run 赋值为 0,构建完成后将 is_run 赋值为 1;处于等待状态的从核会访问 is_run 的值,若值为 1 则从核进入工作状态,执行计算任务。因此主核构建 DAG 期间所有的从核都在不断地访问主存,造成了很大的访存开销。

2) 系统使用 new()、malloc()等申请分配内存,会调用操作系统的系统调用函数 mmap2(),时间开销大,应尽量减少系统调用的次数。AceMesh 任务调度系统中大量地使用动态数据结构,如任务的继承类、任务参数和任务后继列表等,当任务的总数很多时,这些动态数据结构的分配和释放造成大量的系统开销。

3) 无后继任务列表 end_tasks 是基于哈希表实现的,用于检查 DAG 执行结束状态。每个新任务注册后都需要对 end_tasks 列表执行添加和查找操作,每次更新后存在后继的任务需要从 end_tasks 列表中删

除。若无后继任务占比较低,随着注册任务数增加,end_tasks 列表的修改和查找开销会变得越来越大。

## 3 AceMesh 任务调度系统的构图优化

AceMesh 任务调度系统先构图后调度,合理高效的构图方法可以有助于减少线程的等待时间,提高程序的执行效率。本节针对第 2 节中 AceMesh 构图过程存在的 3 个问题进行构图优化。

### 3.1 主、从核通信优化

申威 26010 异构众核处理器单核组存储器访问方式如图 3 所示。每个核组的从核可以直接访问主存和局存,从核访问主存有 gld/gst 离散访存和 DMA 批量式访存两种方式。主核不能直接访问主存,但是可以通过 I/O 访问局存,借助宏 h2ldm(element,penum,cgnum)函数直接对从核 LDM 变量进行 I/O 存取操作,其中 element 是运算核心程序内的局存私有变量,penum 是核号,cgnum 是核组号。

AceMesh 任务调度系统中从核访问共享变量 is_run 的方式为 gld/gst 离散访存。在访存带宽方面,gld/gst 方式单个核组内的访存带宽为 1.5 GB/s,DMA 方式单个核组内的访存带宽为 26 GB/s;在延迟方面,gld/gst 方式延迟 278 cycle;DMA 方式延迟 29 cycle。可以看出,与 gld/gst 方式相比,DMA 方式带宽利用率高,延迟小。

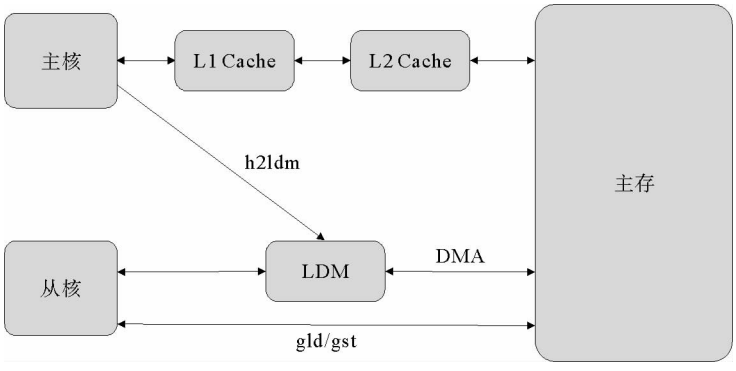


**图 3 单核组存储器访问图**

Fig. 3 Single-core group memory access cycle

但是 DMA 批量式访存也存在一定的局限性。图 4 所展示的单核组 DMA 带宽增长趋势表明,只有当单次拷贝的数据块大于 256 B 而且主存地址 256 B 对齐时,DMA 方式才能有效利用访存带宽。另一方面,从核在构图期间不断地进行变量的数据传输,会占用访存带宽。因此 DMA 方式不是有效的优化方式。

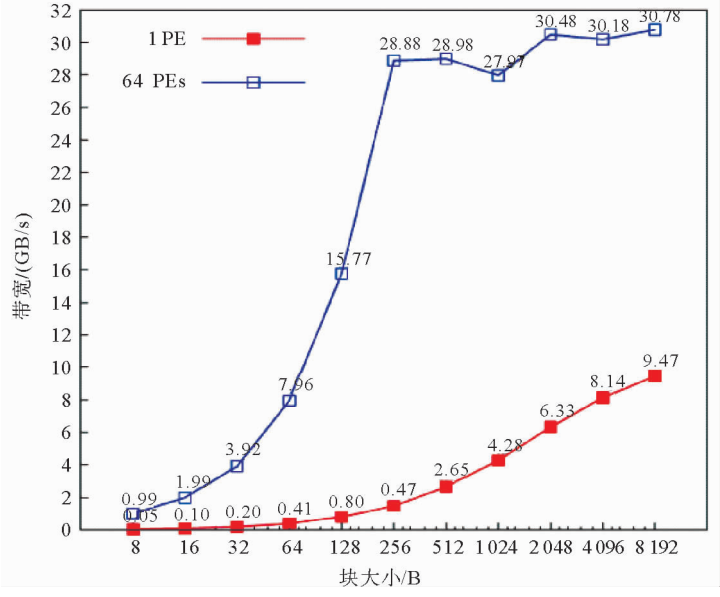


**图 4 单核组 DMA 访存带宽**

Fig. 4 Single-core group DMA access bandwidth

为了减少从核访问主存的带宽开销,将主存上的共享变量 is_run 改为从核 LDM 私有变量,每个从核维护一个自己的通信变量 is_run。如图 5 所示,主核构图完成后,通过 h2ldm(is_run,i,cgid)=1 访问每个从核的 LDM 空间并将私有变量 is_run 赋值为 1。从核判断自己维护的私有变量 is_run 的值为 1 后开始执行任务。当任务执行完成后,主核再通过 h2ldm(is_run,i,cgid)=0 通知从核停止任务执行,等待下一个任务图的执行。将 is_run 变量改为从核私有变量减少了从核访问主存的次数,有效降低了构图过程中的访存开销。

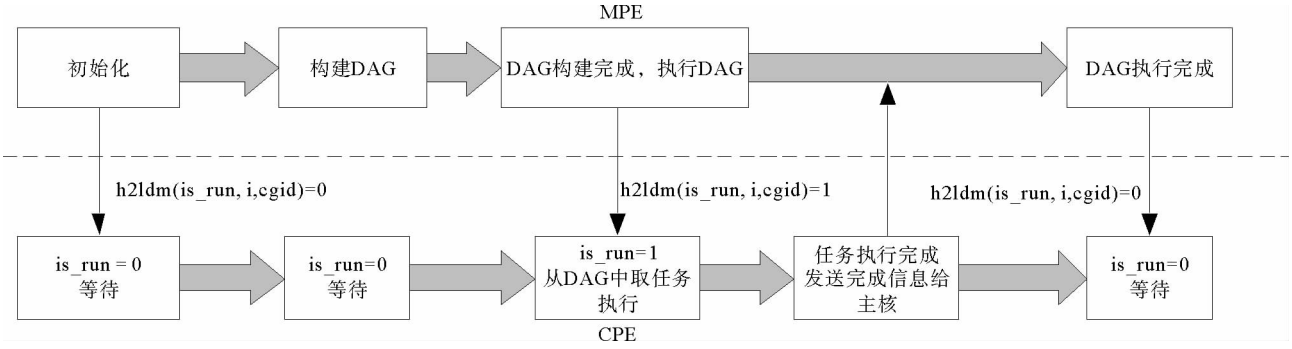


**图 5 优化后的主、从核通信流程**

Fig. 5 Process of the communication between CPE and MPE after optimization

### 3.2 内存池优化

AceMesh 允许用户为一个应用程序设置多个并行区，随着应用计算规模的增加，单个并行区内提交的任务数也不断增加，这些并行任务提交时申请的内存空间将在并行区结束后释放。频繁的内存申请和释放会带来严重的内存碎片问题，同时 malloc()和 free()函数的使用也会造成很大的时间开销。

为解决上述问题，引入内存池管理存储资源。内存池实质上是一种内存分配方式，在真正使用内存之前，通过调用系统内存分配函数预先申请适当大小的内存块作为备用，之后应用程序对内存的分配和释放则通过这个内存池来完成。当有新的内存需求时，就从内存池中分出一部分内存块，若内存块不够，需要动态扩展时，再继续申请新的内存块。

本研究提出一种采用内存池管理内存分配的方法，其内存池的申请和释放状态转换图如图 6 所示。内存池中可以申请多个内存块，申请的内存块中包含可分配的空闲节点，所有空闲块由内存池空闲列表管理，若空闲块被分配出去则将其从表中删除；若已分配的内存块被释放，则将其重新插入到空闲列表的头部。如果内存块中的空闲块不足，则申请新的内存块。内存池的分配和释放都是以固定大小的内存块为单位，不会产生内存碎片，因此可以有效避免内存泄露。

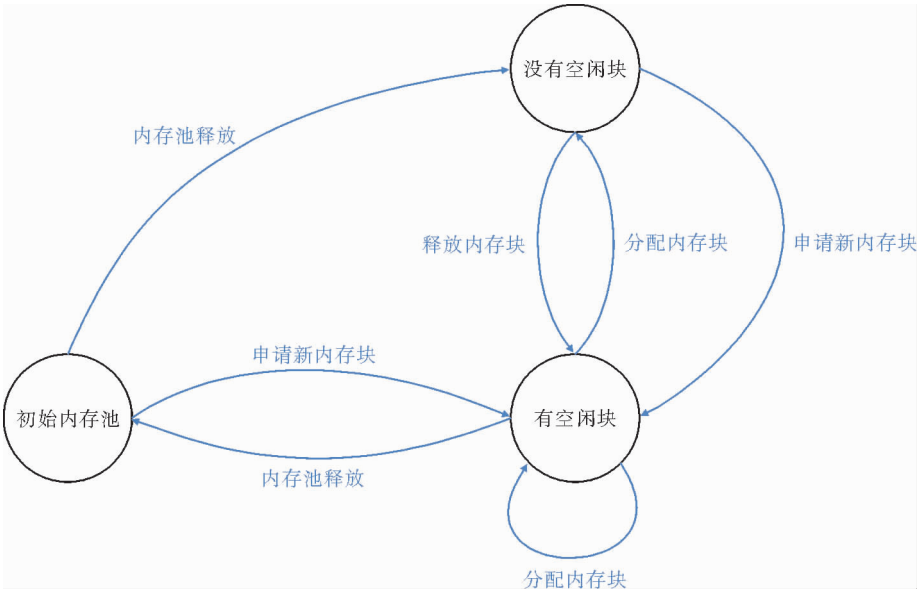


**图 6 内存池内存分配状态转换图**

Fig. 6 Status conversion diagram of memory pool allocation

内存池和内存块的数据结构为:

```
typedef struct node{
void *   data;
struct node *   next;
}node;//内存块
typedef struct MemPool{
void *   to;  //指向当前空闲块的尾部
void *   from;  //指向当前空闲节点的可用位置
struct node *   AllocatedMemNode;  //指向内存块的已分配列表
struct node *   FreeMemNode;  //指向内存块的空闲列表
}MemPool;//内存池
```

根据内存池的工作流程,内存池申请算法如下。

**Algorithm 1** 内存池申请算法

输入:需要使用的内存空间大小 data_size,一个内存块包含的内存节点数 num_node,每个内存节点的存储空间大小 node_size
输出:内存池变量 Mpool

```
1  //内存池初始化
2  struct MempoolMpool
3  Mpool.AllocatedMemNode = NULL
4  Mpool.FreeMemNode = NULL
5  //为内存池加入新的内存块
6   byte *new_memory_block = (byte *)malloc(node_size * sizeof(byte))
7  do i = 1, num_node
8          申请一个新的内存块 node[i]
9          node_i.data = new_memory_block + i * node_size
10          if (i == 1) then continue
11          else 上一个内存块 node[i-1].next 指向 node[i]
12          endif
13  end do
14  if (Mpool.FreeMemNode)  //空闲列表不为空
14  then 将新内存节点链表加入到 FreeMemNode 列表
15  else{
16          Mpool.FreeMemNode = node[1]
17          Mpool.from = node[1].data
18          Mpool.to = node[1].data + node_size}
19  end if
```

内存池分配算法如下。

**Algorithm 2** 内存池分配算法

输入:已分配好的内存池变量 Mpool,需要申请的内存大小 data_size,需要申请空间的指针变量 data_point
输出:分配好内存空间的指针 data_point

```
1  int node_n, node_size = Mpool.to - Mpool.from
2  根据 node_size 和 data_size 计算需要分配的节点数 node_n
3  void * point = Mpool.FreeMemNode->data
4  do i = 1, node_n
5          if (Mpool.FreeMemNode == NULL)//当前内存池内没有空闲节点
6          then 为内存池加入新的内存块
7          endif
8          将 Mpool.FreeMemNode 第一个节点取出
9          在 Mpool.AllocatedMemNode 加入新分配的节点
10  end do
11  data_point = point
```

从内存池的申请和分配算法中可以看出,只在新的内存节点申请时用到 malloc()函数,内存节点的分配则是通过传递指针和修改链表来完成的,减少了 malloc()和 free()函数的调用次数,从而降低内存申请的时间开销。

### 3.3 无后继任务收集优化

无后继任务列表 end_tasks 是为了检测 DAG 是否执行结束而引入的一个任务集,用来记录应用程序的所有无后继任务。AceMesh 任务调度系统中 end_tasks 列表的数据结构为 unordered_set,利用哈希表实现。哈希表是根据键值进行直接访问的数据结构,通过相应的哈希函数处理关键字得到相应的键值,键值对应着一个特定位置,用该位置来存取相应的信息,能以较快的速度获取关键字的信息,具有可快速查找、删除和添加的优点。在 AceMesh 构图机制中,end_tasks 列表的收集方法是在每个任务注册时将其添加到 end_tasks,若发现任务有后继,再从 end_tasks 中查找并删除。

图 7 是对航天飞行器应用中部分代码进行任务划分后得到的 DAG。该部分代码包含多个循环计算,通过对循环的划分来分配计算任务。每个圆圈代表一个计算任务,圆圈内第一个数字代表并行区内的循环的序号,第二个数字代表计算任务在循环内的序号,每个任务通过箭头指向其后继任务,相同颜色的计算任务在同一个计算线程上执行。这部分代码计算的数据访问很规则,所以计算任务之间的依赖关系比较简单,因为计算任务是按照循环的次序串行提交的,所以该 DAG 构建时,第一个循环内的任务(即第一行的任务)首先注册,并被加入到 end_tasks 列表中,第二行计算任务注册时,之前提交的计算任务都被检测到有后继任务,end_tasks 列表需要将第一个循环内的任务删除,第三行计算任务提交后同样需要删除第二行计算任务,这就造成了很多不必要的哈希表查找和修改操作。若无后继任务占比极低,会使得注册时收集到的任务绝大多数被删除,导致哈希表相应数据项的操作失去意义,造成不必要的开销。

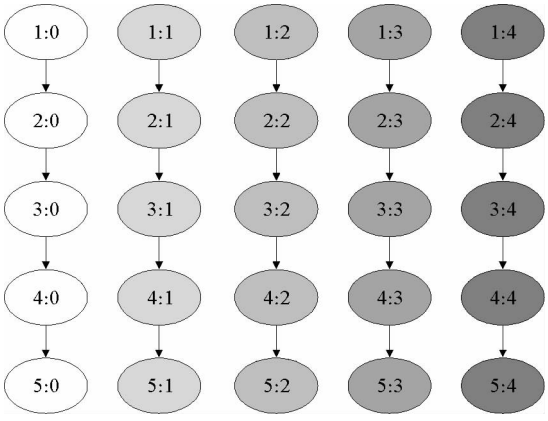


**图 7 航天飞行器应用规则部分 DAG**

Fig. 7 DAG of the aerospace craft application rules section

对于无后继任务占比较小的并行区,通过为用户提供指定无后继任务的开关,由用户直接判断无后继任务,从而省去构图过程中 end_tasks 列表频繁修改的开销。在实现上增加开关 SPECIFY_END_TASKS=true(false)和接口 acemesh_specify_end_tasks()。如果应用并行区内无后继任务占比较小,用户设置 SPECIFY_END_TASKS=true,AceMesh 构图时将不再进行无后继任务列表 end_tasks 的收集,由用户在提交任务时利用接口 acemesh_specify_end_tasks() 指定无后继任务并将其直接放入无后继任务列表 end_tasks 中,所有任务提交完成后无后继任务也收集完毕,避免对 end_tasks 列表频繁的插入和删除操作,提高构图效率。

## 4 构图优化测试

航天飞行器应用程序的循环有 6 层。外 3 层循环为速度空间,内 3 层循环为位置空间,外 3 层循环在进程级别对程序进行划分,本研究只对内 3 层循环即位置空间 $i,j,k$ 的 3 个维度进行划分。对航天飞行器应用程序进行分析得到具有代表性的 7 个计算循环,每个计算循环有不同的划分方向和分块尺寸,各循环的划分方向和分块尺寸如表 1 所示。各个循环在 3 个维度上的划分如表 1 所示。

将 7 个计算循环作为热点子程序进行性能测试,由于在“神威·太湖之光”平台上,AceMesh 任务调度系统的构图阶段是单线程执行的,故无论采用多少计算线程,构图时间均不变。对各热点子程序使用执行效率最高的线程数,在速度空间32×16×16、位置空间 100×19×31、进程网格 1×1×1 规模下,对 AceMesh 任务调度系统的构图阶段测试 5 次,取 5 次测试结果的平均值,对比构图优化前后的性能,说明构图优化的效果,如表 2 所示。

**表 1 计算循环分块表**

Tab. 1 Computational loop blocking table

| 计算循环 | $i$ 维 | $j$ 维 | $k$ 维 |
|---|---|---|---|
| Algorith1 | — | 10 | 1 |
| Algorith2 | — | 10 | 1 |
| Fcta1 | 25 | 1 | — |
| Fcta2 | 25 | 1 | — |
| Fita | 1 | — | 1 |
| Fksi | — | 1 | 1 |
| GaussLegendre | — | 1 | 1 |

表 2 中可见,对于不同的热点程序,构图优化对构图的性能提升程度不同,但加速比均在 4.7 以上,其中 GaussLegendre 的构图性能表现最好,加速比为 7.18。因为 AceMesh 任务调度系统分为构图和执行两个依次进行的阶段,所以计算线程数对构图性能无影响。

**表 2 AceMesh 任务调度系统的构图优化性能对比**

Tab. 2 Comparison of composition optimization performance for AceMesh task scheduling system

| 热点子程序 | 执行线程数 | 构图优化前/s | 构图优化后/s | 加速比 |
|---|---|---|---|---|
| Algorith1 | 62 | 2.19 | 0.35 | 5.19 |
| Algorith2 | 62 | 2.15 | 0.35 | 5.13 |
| Fcta1 | 38 | 2.84 | 0.50 | 4.70 |
| Fcta2 | 38 | 2.74 | 0.45 | 5.02 |
| Fita | 31 | 1.02 | 0.16 | 5.37 |
| Fksi | 31 | 1.01 | 0.16 | 5.28 |
| GaussLegendre | 31 | 1.37 | 0.17 | 7.18 |

任务调度系统各优化项构图优化贡献如图 8 所示,以优化前的构图作为基准,主、从核通信优化、内存池以及无后继任务收集等优化方法对构图有不同程度的加速贡献,其中无后继任务收集优化方法的加速贡献最大,加速比为 2.6 以上,所有优化方法总加速比最高达到 8.19。

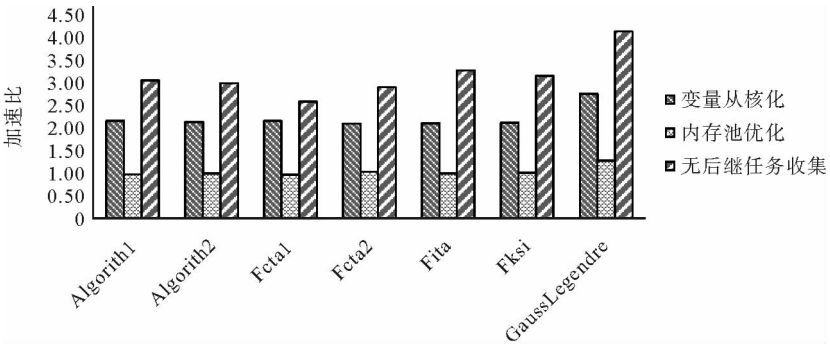


**图 8 AceMesh 任务调度系统各优化项构图优化加速比**

Fig. 8 Optimizing acceleration ratio of optimized composition items in AceMesh task scheduling system

AceMesh 任务调度系统构图优化的子程序总性能对比如图 9 所示,因为航天飞行器应用的 7 个子程序的任务数不同,构图优化对 7 个热点子程序的总体性能加速表现不同,但均在 1.3 倍以上,其中 GaussLeg-

endre 的优化加速最好,达到 3.13 倍;其次是 fcta1 和 fcta2,总性能加速比趋于相同,达到 2.6 倍以上。

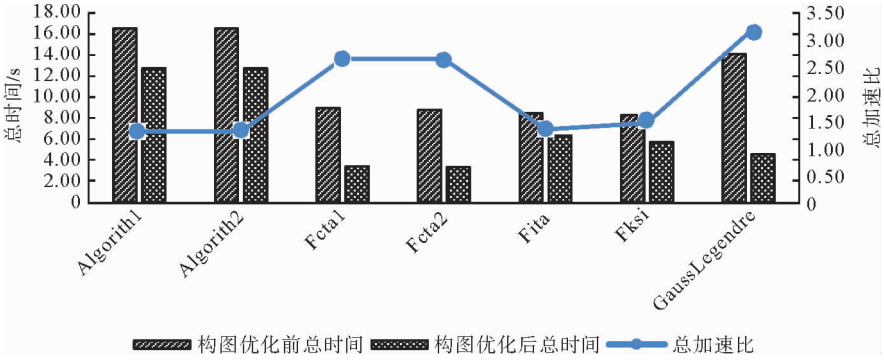


**图 9 AceMesh 任务调度系统构图优化子程序总性能对比**

Fig. 9 Total performance comparison of subprograms with composition optimization for AceMesh task scheduling system

测试采用"神威·太湖之光"平台的测试队列,由于使用权限所致,最大进程数只能为 64,运行时间最长为 60 min。在速度空间 64×64×64,位置空间 100×19×31 规模下,测试不同进程数下应用整体性能的提升。并对照神威 OpenACC 编程语言版本,说明 AceMesh 任务图并行化带来的性能提升,如表 3 所示。

**表 3 AceMesh 与神威 OpenACC 版本的航天飞行器应用性能对比**

Tab. 3 Performance comparison between AceMesh and SW-OpenACC

| 进程数 | AceMesh/s | OpenACC/s | 加速对比 |
|---|---|---|---|
| 24 | 175.90 | 264.48 | 1.504 |
| 32 | 127.38 | 193.37 | 1.518 |
| 36 | 120.35 | 180.41 | 1.500 |
| 48 | 87.82 | 133.29 | 1.518 |
| 64 | 63.99 | 97.41 | 1.522 |

表 3 数据表明,AceMesh 编程语言的 DAG 并行版本比神威 OpenACC 的运行效率要高,在不同的进程数下,加速效果都约为 OpenACC 的 1.5 倍。究其原因,一是因为乱序执行是 AceMesh 独有的特性,二是因为 AceMesh 通过对计算循环进行多维度的划分,在保证足够任务数的同时每个任务有充足的计算量,充分利用访存带宽,使得 DMA 传输的效率提高。随着进程数的增加,AceMesh 并行版本和神威 OpenACC 并行版本的执行时间均逐渐减小,在 64 线程时,运行速度最快,说明 AceMesh 和神威 OpenACC 任务并行编程模型都具有良好的扩展性。

## 5 结论

为提高 AceMesh 的构图性能,从主、从核通信优化、设计内存池、无后继任务收集等 3 个方面对 AceMesh 的构图进行优化,对航天飞行器应用的 7 个热点子程序在"神威·太湖之光"上进行了测试,结果证明优化为构图带来 5 倍的性能提升。后两个方面的优化具有普适性,可为所有超算平台上的优化提供参考。以上构图效率提高的主要原因来自无后继任务收集的优化,进一步可以考虑设计一种更高效的任务收集机制来减少哈希表的查找和修改开销。

**参考文献:**

[1]王蕾,崔慧敏,陈莉,等.任务并行编程模型研究与进展[J].软件学报,2013(1):77-90.
WANG Lei,CUI Huimin,CHEN Li,et al.Research on task parallel programming model[J].Journal of Software,2013(1):77-90.

[2]WENG T H,WANg T X,HSIEH M Y,et al.Parallel fast Fourier transform in SPMD style of Cilk[J].International Journal of Embedded Systems,2019,11(6):778.

[3]ASAI R,VLADIMIROV A.Intel Cilk Plus for complex parallel algorithms[J].Parallel Computing,2015,48,125-142.

[4]ALESSANDRINI V.Intel threading building blocks[J].Shared Memory Application Programming,2016:307-339.

[5]PLLANA S,XHAFA F.Programming multi-core and many-core computing systems[J].Hoboken:John Wiley & Sons,2017:99-120.

[6]MUHAMMED MARUF Z,ZENGIN A.Improved GUI testing using task parallel library[J].ACM SIGSOFT Software Engineering Notes,2016,41(2):1-9.

[7]CHEN L,TANG S,FU Y,et al.AceMesh:a structured data driven programming language for high performance computing[J].CCF Transactions on High Performance Computing,2020,2:309-322.

[8]高希然.“神威·太湖之光”上任务图并行调度优化研究[D].青岛:山东科技大学,2018:2-32.
GAO Xiran.Study of the parallel task graph scheduling optimaization on the Sunway TaihuLight[D].Qingdao:Shandong University of Science and Technology,2018:2-32.

[9]郭杰,高希然,陈莉,等.用数据驱动的编程模型并行多重网格应用[J].计算机科学,2020,47(8):32-40.
GUO Jie,GAO Xiran,CHEN Li,et al.Parallelizing multigrid application using data-driven programming model[J].Computer Science,2020,47(8):32-40.

[10]GÓMEZ-SOUSA H,ARENAZ M,RUBIÑOS-LÓPEZ Ó,et al.Novel source-to-source compiler approach for the automatic parallelization of codes based on the method of moments[C]//9th European Conference on Antennas and Propagation (EuCAP).IEEE,2015:1-6.

[11]LI J,FERRY D,GILL C,et al.Parallel real-time scheduling of DAGs[J].IEEE Transactions on Parallel & Distributed Systems,2014,25(12):3242-3252.

[12]孟现粉,花嵘,傅游.网格应用的 DAG 任务调度系统的并行构图方法[J].计算机工程与设计,2019,40(8):2174-2180.
MENG Xianfen,HUA Rong,FU You.Parallel composition method of DAG task scheduling system for grid application[J].Computer Engineering and Design,2019,40(8):2174-2180.

[13]傅游,王坦,郭强,等.“神威·太湖之光”上 Tend_lin 并行优化[J].山东科技大学学报(自然科学版),2019,38(2):90-99.
FU You,WANG Tan,GUO Qiang,et al.Parallelization and optimization of Tend_lin on Sunway TaihuLight system[J].Journal of Shandong University of Science and Technology (Natural Science),2019,38(2):90-99.

[14]王松,花嵘,孟现粉,等.数据驱动的任务图执行中并发构图方法[J].计算机工程与设计,2018,39(3):758-762.
WANG Song,HUA Rong,MENG Xianfen,et al.Concurrent composition method for data-driven task graph execution[J].Computer Engineering and Design,2018,39(3):758-762.

[15]王涛.“神威太湖之光”超级计算机[J].科学,2016(4):5.
WANG Tao.“Shenwei TaihuLight” Supercomputer[J].Science,2016(4):5.

(责任编辑:刘西奎)