一个面向任务图并行程序的错误检查工具

刘艳娜 陈 莉 唐生林
(中国科学院计算机研究所 北京 100190)

摘 要 AceMesh是一种基于数据流描述的任务并行编程语言,它允许程序员从串行程序出发,追加并行区域、并行循环的制导以及任务区的数据访问信息,AceMesh编译系统则自动把该程序转化为异步任务图并行的程序。分析了AceMesh程序改写中常见的并行化错误,介绍了其错误检查工具AceMeshCheck的结构,描述了访存轨迹的高效收集、存储方法以及逻辑形状推导的三维压缩算法。实验表明,AceMeshCheck不仅能分析出制导程序中的典型错误,而且开销较小。

关键词 任务并行语言,数据流信息,错误检查,动态分析,网格应用
中图法分类号 TP321 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.03.010

Error Checking Tool for DAG-based Task Parallel Programs

LIU Yan-na CHEN Li TANG Sheng-lin
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

Abstract AceMesh is a dataflow-driven' task parallel programming language, which allows programmers to parallelize traditional C/C++ program by using pragmas marking parallel regions, parallel loops and task regions with data inputs and outputs description. Then the program can be translated to a DAG-based task parallel program, being built dependency graphs at runtime and scheduled to multicore platforms efficiently. This paper analysed typical errors which may exhibit in parallelizing AceMesh programs, and introduced AceMeshCheck, a debug tool for them. The paper presented the implementation details of the tool, and discussed how it reduces the overhead of memory trace collection, and how to rebuild the three dimensional, rectangular access regions from linearized memory access sequences. Experimental results show that the tool can identify typical errors hidden in AceMesh programs with relatively low overhead.

Keywords Task parallel language, Dataflow, Error checking, Dynamic program analysis, Mesh application

1 引言

近年来,多核芯片的流行使得任务并行编程模型受到用户的欢迎。基于输入输出描述的任务并行语言[1-2],能支持有向无环任务图(Directed Acyclic Graph, DAG)的自动构建,成为了学术界的热点。

AceMesh是一种数据驱动的任务并行编程语言,其并行性抽象包括:DAG并行区制导;并行循环的for制导;Wait制导;任务并行块的task制导。针对结构网格应用中的分片存储和stencil计算的需求,它采用二元的数据访问描述,分别描述阴影区与中心区的访问模式。

图1的并行循环是对当前网格层的每个存储分片进行stencil计算(代码来自Chombo),图中展示了AceMesh的for制导和task制导。后文把有task制导的代码区域称为任务区代码,DAG并行区中的其他代码则称为构建区代码。图1中的for循环头和datInd的赋值语句都属于构建区代码。

```
#pragma acemesh for
for(dit.begin();dit.ok();++dit){
DataIndex datInd=dit();
#pragma acemesh task data(&a_phi[datInd]:〈RW,R〉,
&a_rhs[datInd]:〈R,N〉) firstprivate(datInd)
{ //task region
constBox* region=&dbl[datInd];m_bc(a_phi[datInd],region,m_
domain,m_dx,true);FORT_GSRBLAPLACIAN(CHF_FRA(a_phi
[datInd]),
CHF_CONST_FRA(a_rhs[datInd]),…);
} //end task region
}
```

图1 AceMesh制导示例

2 AceMech程序改写中的一些典型错误

AceMech程序在并行化过程中所引入的错误有的比较显然,比如用户提供的数组的注册区域与实际访问区域不相

到稿日期:2016-02-03 返修日期:2016-06-21 本文受国家自然科学基金(61432018),国家高技术研究发展计划(863)(2012AA010902)资助。
刘艳娜(1993-),女,硕士生,主要研究领域为并行程序设计语言和并行编译技术,E-mail:liuyanna@ict.ac.cn;陈 莉(1970-),女,博士,副研究员,主要研究领域为并行程序设计语言和并行编译技术,E-mail:lchen@ict.ac.cn;唐生林(1974-),男,硕士,工程师,主要研究领域为并行编译和并行编程环境,E-mail:tangshenglin@ict.ac.cn。

等，循环并行性的错误；而另外一些错误则比较复杂。

(1)任务区代码到构建区代码的数据依赖

在并行区域内，一些构建区代码可能带来从任务区到构建区的数据依赖。而一般情况下，构建区代码可能先于任务区代码的执行，于是会导致制导所描述的程序行为与原程序不一致。

(2)firstprivate 制导的缺失

如果在图 1 中漏掉 firstprivate 子句，则这个循环的并行化是不安全的。因为构建区对变量 *datInd* 进行了多次写操作，如果编译器在自动任务构造时没有采用值传参，则会带来并发执行中的错误。

(3)首地址的选择

AceMesh 用户倾向于用单个地址代表一个地址区域。如果来自不同并行循环的两个任务之间存在数组访问带来的数据依赖，则相关数组访问区域的注册地址必须相同，后文称为“区域注册的地址相同条件”。如果这个条件不满足，变换后的多线程程序很可能会出现数据竞争。

图 2 中的第 2，3 循环套在 ey 访问区域上有交集，但注册地址不一致，导致任务之间存在错误的依赖关系。

```
pragmaacemesh task(&ey[0][0]:⟨W,N⟩)
for (j=0;j<ny;j++)
    ey[0][j]=t;
#pragma acemeshfor
for (ii=1;ii<nx;ii+=tz){
  #pragma acemesh task(&ey[ii][0]:⟨RW,N⟩,\&hz[ii][0]:⟨R,
  R⟩)
  for (i=ii;i<min(ii+tz,nx);i++)
    for (j=0;j<ny;j++)
      ey[i][j]=ey[i][j]-0.5*(hz[i][j]-hz[i-1][j]);
}
…
#pragma acemeshfor
for(ii=0;ii<nx;ii+=tz) {
  pragmaacemesh task(&hz[ii][0]:⟨RW,N⟩,\&ex[ii][0]:⟨R,N⟩,
  &ey[ii][0]:⟨R,R⟩)
  for (i=ii;i<min(ii+tz),nx);i++)
    for (j=0;j<ny;j++)
hz[i][j]=hz[i][j]-0.7*(ex[i][j+1]-ex[i][j]+ey[i+1][j]-
ey[i][j]);
}
```

图 2　代表性地址不同的错误

(4)变量生存期问题

构建区往往会分配一些局部量，以供后继的任务区使用，如果这些变量的生存期小于并行区域，则会出现问题。因为构建区代码很可能在任务区代码执行之前结束，从而在任务区代码访问这些变量时出现野指针访问的问题。

下面是 Chombo 示例程序中一个并行区的片段，其中 timeInterp()函数调用的内部有并行循环和并发任务。这里局部量 phiCoarse 相对于当前并行区来说，生存期不足。

```
Void LevelWaveOperator::eval(…)
{
  if (m_hasCoarser) {
    …some checking
    LevelData⟨FArrayBox⟩ phiCoarse;
    phiCoarse.define(a_phiCoarseOld);
    //interpolate coarse data to the current time,
    //many parallel tasks
    timeInterp(phiCoarse,a_time,…);
  }
  …other codes in this parallel region
}
```

图 3　变量生存期错误

3　AceMeshCheck 工作原理

本工具有两个基本用法：1)用户只标记出并行区、任务区，并让工具检查是否存在从任务区到构建区的数据依赖、是否有生存期不足的变量；2)用户标记任务区的数据访问信息，让工具检查每个任务的注册信息是否与实际访问情形一致。

对应于上述两种用法，本工具在总体结构上分为在线检查和离线检查两部分。在线检查主要负责访存轨迹的收集以及前述 1)的检查，而离线检查则负责 2)的检查。

在内部机制的实现上，AceMeshCheck 使用 Valgrind 的客户端请求机制支持应用的自动插桩。工具向上层用户暴露了一些接口以传递必要的信息，比如并行区域/并行循环/并发任务的开始和结束、数组形状、逻辑访问区域、访问区域的最小内存地址等。

3.1　数据访问信息的收集优化

数据访问信息是错误检查的基础，在线分析的评测表明收集这些信息的开销较大，是优化的重点。

轨迹信息的组织与存储采用 Valgrind 提供的二叉平衡树和动态数组，而存储单位 *op* 可以是一条访存指令所对应的内存数据区域，也可以是地址连续的多个访存指令所对应的内存区域，或者是逻辑上具有特定的三维形状的内存区域。

收集到一个访存地址后，需进行以下两个操作：

(1)确定该地址所属的数据对象的存储类别。这里进行了两点优化：1)引入软件缓存来记录最近的变量信息，以减少对全局变量或局部变量的地址区间树等数据结构的查找；2)记录不同表(全局变量、局部变量、当前栈变量或堆变量)的访问热度以优化查找效率。

(2)将该地址添加到二叉树，存储方式上可考虑以下 3 种方案。

方案 1　使用二叉平衡树。每收集到数据对象的一个新的 *op*，就将其插入到对应的树上。因需插入的 *op* 过多，该方案的时间和空间开销都较大。

方案 2　先暂存到动态数组，当前任务处理完后再插入二叉树，但是要利用地址连续性进行 *op* 的合并。若数组中元素过多，合并动作的时间开销会较大，但合并动作会减少空间需求。

方案 3　与方案 2 的差别在于收集时不进行合并。这样虽然避免了合并的时间开销，但是空间开销过大。

由于上述 3 种方案都存在问题，根据局部性原理，对方案 2 进行优化，即搜索不再从数组起始位置开始，而是只遍历最后的 N 个元素(其中 N 是经验值)。

3.2 数组区域的逻辑形状推导

对于科学计算应用，其地址序列虽然物理上不连续，但是却常常具有规则的逻辑三维形状。识别出规则的三维形状，即可实现数据压缩。

针对 3.1 节获得的二叉平衡树，需要研究数据压缩的算法。压缩的规则是区域在两个维度上相同，且第三维邻接。难点在于为了实现有效的数据压缩，需要根据区域的重叠情况，进行大量的区域拆分，并考虑所有可能的邻接情况(可达 81 种之多)，导致实现过于复杂，不利于软件维护，于是引入逐层压缩的算法。

步骤 1 先进行一维压缩，即把地址连续的 *op* 进行合并，得到多个一维线性连续地址块，然后根据数组的逻辑形状将一维压缩的结果按行分解，得到行级连续的地址区域。

步骤 2 对前述的地址区域以行优先的顺序进行二维压缩，其具体原理与步骤 1 类似。合并阶段得到逻辑上连续的矩形面，其成为一个新的 *op*。由于这些 *op* 可能跨占不同的面(即 z 维)，需对它们按 z 维进行分解，得到 z 维厚度是 1 的二维矩形面。

步骤 3 对上述矩形面进行三维压缩，即可得到该数据访问区域的三维逻辑形状。

由于步骤 3 比较复杂，详细解释如下。每个矩形面由两个行坐标、两个列坐标共 4 条线段围成，对于两个矩形面，8 条线段总能围成 9 个封闭的区域(见图 4)，这里两个矩形面分属 Z 轴的相邻两层，上层记做 *opn*，即已经过压缩处理的部分，下层为 *op*，即当前需要压缩的部分。当存在某两条线重合时，这两条线之间的矩形区域的长度或者宽度为 0，这里不做特殊对待，统一处理。对这 9 个区域的操作即为对两个矩形区域的压缩处理。

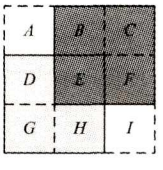

图 4 三维压缩图

具体方法如下：

对于 A 和 I，它们不属于两个矩形区域中的任何一个，标记为 0，以后不再处理；对于宽度或者长度为 0 的区域，该区域不存在，标记为 0，以后不再处理；对于 E，其为两个矩形区域的交集部分，标记为 2，该区域即为两个矩形面重叠时需要被压缩的部分；对于 *op* 的剩余区域，对其进行行连续优先压缩，并将压缩结果作为新的 *op* 递归进行压缩。对于 *opn* 的剩余区域，同样对其进行行连续优先压缩，并将压缩结果作为新的已压缩元素存储，以备后续压缩。

4 实验和结果分析

本节选取比较有代表性的 3 个测试用例进行测试，并以其检查结果来展示工具的正确性以及性能。1) *3d7p* 即 3 维 7 点的 stencil 计算；2) *fdtd* 案例，选自 pluto 测试集，它是个二维问题，包括 4 个循环套，对最外层循环进行并行划分；3)Chombo的 *relax*() 过程即 AMRPoisson 应用中的核心计算，它采用红黑高斯赛德尔松驰迭代法。

本文选择的测试平台是有 8 个 Intel(R) Xeon(R)CPU E7-8830 的服务器，每个 CPU 包含 8 个核心，主频2.129GHz。操作系统是Ubuntu12.04.2，内核版本 3.2.0-29-generic。编译器是 gcc-4.6/g++ 4.6，编译选项-O0-g。

4.1 检查功能的展示

对于不同的变量存储类别、不同的错误类型、不同维数的数据对象以及不同的注册类型，工具需要保证都能正确地完成错误检查功能并给出正确的错误提示信息。

下面以 *fdtd* 的一个变形为例，展示对工具的正确性测试结果。

需要指出的是，*foo*() 调用被用户作为构建区的代码，因此 *foo*() 先于第一个循环中的 *ey* 定值，从而导致从任务区到构建区的数据依赖。这个例子里没有变量生存期问题。部分注册信息如图 5 所示。

```
VALGRIND_ACEMESH__CHECK_LOOP_START;
VALGRIND_ACEMESH__CHECK_TASK_START;
VALGRIND_ACEMESH__CHECK_TASK_DATA_WRITE(&ey[ ][ ], ,          ,&p1,sizeof(double));
        for(j= ; j<ny; j++)
                ey[ ][j] = t;
VALGRIND_ACEMESH__CHECK_TASK_END;
VALGRIND_ACEMESH__CHECK_LOOP_END;

foo(ey);
```

图 5 注册信息图

实际检查结果如图 6 所示。

```
datachip accessed in task0 is wrote in other tasks: 1nop,id2,task0:11,memkey,addr:0x603d30
Pagerr: 2,LifetimeErr: 0
```

图 6 错误报告图

本工具定位出了由数组 ey 引入的、从任务区到构建区代码的数据依赖。错误报告中，task0 表示构建区代码，并报告“没有变量生存期错误(LifetimeErr)”。将来该工具会添加行号信息。

4.2 性能评估与分析

在线工具主要检查并行区域的合法性，轨迹收集是其主要开销。本节与 Memcheck 进行对比，评估了不同数据规模下所采取的优化措施的效果。

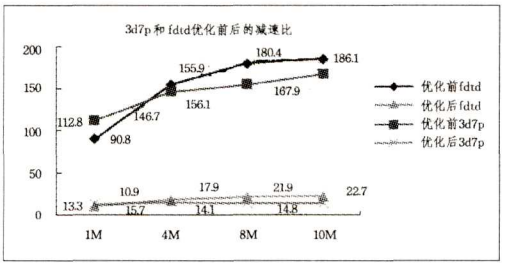

图 7 本工具相对于 Memcheck 的减速比(优化前和优化后)

图 7 中的数据来自 3d7p 和 fdtd，纵坐标是减速比(相对于 Memcheck 的运行时间)。

从这些数据的对比可以看出，AceMeshCheck 经过优化后的性能提升了 10 倍左右，这说明 3.1 节所述的优化策略的作用还是比较明显的。

离线工具主要检查每个任务的数据访问信息的注册是否恰当。其中文件读取和数据压缩(访问区域的三维重建)的耗时较多。以 Chombo 的 *relax*() 过程为例给出离线工具里各部分功能的时间分布图，如图 8 所示。

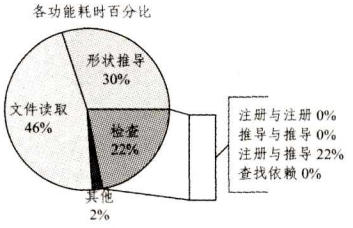

图 8　离线分析中各个分项的时间开销(Chombo 的 relax())

实验表明文件信息的读取开销巨大，为了优化这部分开销，将来需把离线分析改为在线分析。实验中访存区域的逻辑形状推导的开销也较大，这主要是因为轨迹信息的处理量较大。工具的分析检查则耗时较少，这得益于数据压缩的表示。

由图 8 可知，在所有的检查工作中，注册与推导的全检查是最耗时的部分，而其他由用户给出的特定 task 号的抽样性检查的耗时则几乎可以忽略。

结束语　本文介绍的 AceMeshCheck 能帮助用户定位 AceMesh 程序中并行相关的错误，并提供有用的反馈信息，从而降低 AceMesh 应用的并行化难度。相比 StarSscheck[3]，本工作的特色有：能检查一些全局性的错误；能解决网格应用中多维逻辑区域的形状识别问题。

未来会增加 AceMesh 编译器对 AceMeshCheck 的支持，自动把 AceMesh 制导转化为 AceMeshCheck 的宏功能接口。此外，也会进一步降低 AceMeshCheck 的时间开销，使其性能接近 Memcheck。

参 考 文 献

[1] ETSION Y, CABARCAS F, RICO A, et al. Task Superscalar: An Out-of-Order Task Pipeline[C]//Proceedings of the Annual International Symposium on Microarchitecture (MICRO-43). 2010: 89-100.

[2] GAUTIER T, LIMA J V F, MAILLARD N, et al. XKaapi. A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures[C]//27th IEEE International Parallel & Distributed Processing Symposium (IPDPS). 2013: 1299-1308

[3] CARPENTER P M, REMIREZ A, AYGUADE E. Starsscheck: A Tool to Find Errors in Task-Based Parallel Programs[M]// Euro-Par 2010-Parallel Processing. Springer Berlin Heidelberg, 2010: 2-13.

(上接第 19 页)

数据流和查询数据流大量涌入时的高吞吐量的移动对象 KNN 查询问题，提出了一种新的基于内存的移动对象 KNN 查询算法——DSRNKNN。算法采用了快照的处理方式，在每个快照内，基于移动对象的更新，重新构建索引，避免了复杂的索引维护操作，充分发挥了硬件的特性。对于查询，采用了批处理的方式。基于查询所在的边进行聚集，每次处理一条边上的查询。在执行过程中，首先计算边上两个端点的 KNN 结果，基于端点的 KNN 结果，能够快速地得到该边上所有查询的结果。同时，节点可能被多条边共有，只需要对节点计算一次，便可以为多边使用，减少了搜索范围，提高了查询效率。通过对 California 和 San Francisco 的实际路网的数据集进行实验，验证了算法的有效性。

参 考 文 献

[1] CHO H J, CHUNG C W. An efficient and scalable approach to CNN queries in a road network[C]//Proceedings of the 31st International Conference on Very Large Data Bases. VLDB Endowment, 2005: 865-876.

[2] MOURATIDIS K, YIU M L, PAPADIAS D, et al. Continuous nearest neighbor monitoring in road networks[C]//Proceedings of the 32nd International Conference on Very large Data Bases. VLDB Endowment, 2006: 43-54.

[3] LEE K C K, LEE W C, ZHENG B, et al. ROAD: a new spatial object search framework for road networks[J]. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(3): 547-560.

[4] CHEN Y J, CHUANG K T, CHEN M S. Coupling or decoupling for KNN search on road networks?: a hybrid framework on user query patterns[C]//Proceedings of the 20th ACM International Conference on Information and Knowledge Management. ACM, 2011: 795-804.

[5] CHEN Y J, CHUANG K T, CHEN M S. Spatial-temporal query homogeneity for KNN object search on road networks[C]// Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management. ACM, 2013: 1019-1028.

[6] BALKESEN C, TEUBNER J, ALONSO G, et al. Main-Memory Hash Joins on Modern Processor Architectures [J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(7): 1754-1766.

[7] TEUBNER J, ALONSO G, BALKESEN C, et al. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware[C]// 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE, 2013: 362-373.

[8] KIM C, KALDEWEY T, LEE V W, et al. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs[J]. Proceedings of the Vldb Endowment, 2009, 2(2): 1378-1389.

[9] SIDLAUSKAS D, SALTENIS S, JENSEN C S. Processing of extreme moving-object update and query workloads in main memory[J]. The Vldb Journal, 2014, 23(5): 817-841.

[10] LI F, CHENG D, HADJIELEFTHERIOU M, et al. On trip planning queries in spatial databases[M]// Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg, 2005: 273-290.

[11] LI F. Real Datasets for Spatial Databases: Road Networks and Points of Interest[OL]. http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm. 2004

[12] LEE K C K, LEE W C, ZHENG B. Fast object search on road networks[C]//Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 2009: 1018-1029.

[13] DITTRICH J, BLUNSCHI L, SALLES M A V. Indexing moving objects using short-lived throwaway indexes[M]// Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg, 2009: 189-207.

[14] BRINKHOFF T. A framework for generating network-based moving objects[J]. GeoInformatica, 2002, 6(2): 153-180