软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(12):2843−2858 [doi: 10.3724/SP.J.1001.2013.04405]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于数组生命期的数据分解算法*

丁 锐[1,2], 赵荣彩[1,2], 韩 林[1]

[1](解放军信息工程大学,河南 郑州 450002)

[2](数学工程与先进计算国家重点实验室,河南 郑州 450001)

通讯作者: 丁锐, E-mail: dr2012earth@gmail.com

摘 要: 划分是一种自动分配计算和数据到各个处理器的编译技术,是分布存储结构下并行编译的核心问题.以往的划分研究较少从生命期的角度考虑数据分解问题,分解在数组的不同生命期中不一致时会产生冗余通信.为解决上述问题,提出了一种数据分解算法,通过定义-引用图来表示数组的数据流信息,并使用分解映射表为数组不同的生命期建立各自的数据分解.对矩阵求逆等 9 个实际用例的实验结果表明,与以往不区分生命期的划分研究相比,使用所提算法能够在寻找数据分解时对并行收益做出更准确的评估,减少了通信冗余,从而提升了自动生成的并行代码的加速比.

关键词: 数据分解;数组生命期;自动并行化;分布存储

中图法分类号: TP314 **文献标识码**: A

中文引用格式: 丁锐,赵荣彩,韩林.一种基于数组生命期的数据分解算法.软件学报,2013,24(12):2843−2858. http://www.jos.org.cn/1000-9825/4405.htm

英文引用格式: Ding R, Zhao RC, Han L. Data decomposition algorithm based on array life cycle. Ruan Jian Xue Bao/Journal of Software, 2013,24(12):2843−2858 (in Chinese). http://www.jos.org.cn/1000-9825/4405.htm

Data Decomposition Algorithm Based on Array Life Cycle

DING Rui[1,2], ZHAO Rong-Cai[1,2], HAN Lin[1]

[1](PLA Information Engineering University, Zhengzhou 450002, China)

[2](State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Corresponding author: DING Rui, E-mail: dr2012earth@gmail.com

Abstract: Partition is a compiler technique that maps computation and data onto different processors, and is the key issue of automatic parallelization on distributed memory architecture. Array's life cycle has been less considered by previous researches on data decomposition, despite of the fact that the inconsistency of decomposition in different array life cycles often results in communication redundancy. This paper proposes a new data decomposition algorithm which represents data flow information of array by define-use graph, and creates own decomposition for each life cycle of array. The experimental results on Matrix-Inversion and other eight applications show that compared with automatic data decomposition methods that does not distinguish the life cycle of array, the proposed algorithm not only makes more accurate assessment of parallel benefits, but also reduce communication redundancy and rise up the speedup.

Key words: data decomposition; array life cycle; automatic parallelization; distributed memory

分布存储结构凭借其卓越的可扩展性,依然在现今的高性能计算机中非常流行,并越来越多地用于处理大规模科学计算问题.在分布存储的并行计算中,处理器对本地内存的访问速度远超于异地,所以,并行程序需要合适的划分计算和数据来增加数据引用的局部性和最小化通信.在划分中,计算到各个处理器的映射被称为计算分解.类似的,数据到各个处理器的映射被称为数据分解.通过并行编译器来选择良好的计算和数据分解,是

* 基金项目: 国家高技术研究发展计划(863)(2009AA01220); "核高基"重大专项(2009zx10036-001-001)
收稿时间: 2012-11-19; 定稿时间: 2013-04-02

自动并行化研究中的热点之一,以往的学者对此做了很多研究.

Anderson 和 Lam 基于贪婪算法和通信图提出了一种动态计算和数据分解算法,根据数组在循环嵌套之间的相关性分析来逐步合并划分的静态子集[1,2].Lim 和 Lam 提出一种最大化并行的仿射划分方法[3].Lee 和 Kedem 提出了一种提升数组等级的自动数据分解算法,在成分对准后优先分布片段内的主导数组,并通过流水并行对使用 ADI(alternating direction implicit)方法的程序取得了较好的加速比[4].文献[5,6]将通信图改为融合控制流的层次分解图,并按照控制流来逐步合并静态划分区域.文献[7]针对计算和数据分解问题,提出了一套关于数据空间融合的理论框架,利用数据融合技术优化数据分布.文献[8]为了优先划分主导数组,优先划分与主导数组相关的循环;并通过设置其他循环为串行增加冗余并行的定义点,以减少划分的约束.

Zhang 提出了一种基于代表元的划分算法[9],用于解决由非紧密嵌套循环、不同语句对数组元素访问步长的不同以及一条语句对数组的多次引用存在重叠而无法求出循环无通信划分方向的问题.Wang 等人基于分布存储模型提出了一种面向 Cell Broadband Engine 芯片的数据分布算法,并能为 Cell 生成带消息传递原语的 SPMD 程序[10].文献[11]则提出了一个基于整数线性规划的全局部分重复计算划分框架,以提高并行程序节点间局部性.Shih 基于超平面提出了一种语句级的无通信划分原则,将每个语句都当作一个调度单位,即每个语句都一个独立的迭代空间,并对其分别进行划分[12].

Chen 和 Chang 提出了一种扭曲的数据划分和对准技术,相比传统的维对准方法,能够处理更复杂的情况[13].Fresno 研究了如何使用自动数据划分技术对 NPB(NAS parallel benchmark)基准测试集中的 MG(multi grid)程序进行处理[14].文献[15]提出了一种混合编译时-运行时的翻译方案,能够将共享内存地址的 OpenMP 程序翻译到集群上,使用了一种运行时的数据流分析技术来增加数据亲和性和减小通信.Zhong 针对 MPI 程序在异构多核平台提出了两种数据划分技术,主要处理异构多核结点间和结点内部核间的负载均衡问题[16].文献[17]根据嵌套循环的结构和 MPI 并行程序的特点,提出了一种基于嵌套循环分类的并行识别技术.文献[18]则为了挖掘循环的并行性,根据二次规划提出了一种非线性数组下标的依赖测试方法.

数据的生命期就是其“定义-引用(define-use)”周期.由于数据的生命期是以其定义开始的,旧的定义必定在新的生命期中被注销,所以不会存在一个数据在某个生命期中定义,又在另一个生命期被引用.而在分布存储结构下,通信产生的原因是引用的数据不在本地.这意味着,只有在同一生命期中的定义和引用才会产生数据交换.数组作为一种数据集合,当然也有生命期.而数据分解描述的是数组的数据空间到处理器空间的映射关系,也继承了数组生命期的特性,即数组的数据分解在不同的生命期中不一致,并不需要数组的重分布来满足数据的一致性.

```
L0  for (k=1; k<=n; k++){        //doacorss
      L1  for (i=1; i<=n; i++){
              sum=0.0;
              for (j=1; j<=n; j++)
                  sum=sum+p[j];
              w[i]=sum;
          }
      L2  for (i=1; i<=n; i++)
              q[i]=w[i];
      L3  for (i=1; i<=n; i++){      //doacorss
              w[i]=0;
              p[i]=p[i−1];
          }
    }
L4  for (i=1; i<=n; i++){
        d=0.0;
        for (j=1; j<=n; j++)
            d=d+p[j];
        w[i]=sum;
    }
L5  for (i=1; i<=n; i++)
        r[i]=w[i];
```

Fig.1 Example to illustrate life cycle

图 1 用于说明生命期的示例

以往的研究很少从生命期的角度考虑划分问题,用一个缺少生命期属性的数据分解涵盖了数组所有的定义-引用周期.这样在寻找数据分解的过程中,数组的定义要与其活跃范围之外的引用对齐.这不仅会增加无效的通信,还会影响代价评估模型对并行收益的判断,减少数据分解的选择.例如在文献[1,2,5,6]中,并不区分数组的定义和引用.在图 1 中,如果按照以往的划分方法,因为数组 w 在 $L3$ 中受到 p 的约束而无法分布,所以数据分解无法保持一致,从而引起了 w 在 $L2$ 和 $L3$ 之间的数据重组通信.我们在图 1 为 w 数组的每个生命期标记了不同的颜色,可以发现,$L2$ 和 $L3$ 间没有定义-引用关系,通信是冗余的.而这部分多余的通信代价除了会降低程序的并行收益外,还可能会使编译器在判断 $L2$ 是否值得划分和 w 数组是否值得分布时,做出错误的选择.

针对这样的问题,本文提出了一种新的数据分解算法,通过定义-引用图来表示数组的数据流信息,并使用

分解映射表为数组不同的生命期建立各自的数据分解.经过对矩阵求逆等 9 个实际用例的实验和分析,表明了本文所提方法的有效性.

本文第 1 节对数组生命期的定义进行讨论,并对定义-引用图进行介绍.第 2 节介绍如何在定义-引用图的基础上建立分解映射表;通过映射表,能够将生命期信息引入到数据分解中.第 3 节根据定义-引用图和分解映射图,改进数据分解算法.第 4 节是实例、实验与分析.最后是结束语.

1 定义-引用图

1.1 数组的生命期

笼统的说,数组的生命期就是其“定义-引用”的周期.但由于数组是数据的集合,而这些数据的生命期并不会整齐划一.对数组而言,一个循环可能只定义了部分元素,也可能会读取来自不同循环定义的数据.这说明,生命期的划分不可能像标量那样泾渭分明,无法简单地套用标量的生命期定义.在自动数据分解算法中,我们关注的是数组在循环中定义了哪些元素,又在哪些循环中被引用.即使定义-引用的双方是数组的部分元素,依然需要保持一致的分布来避免通信.所以,我们从数据分解的角度重新给出数组生命期的定义,以满足划分算法的需求.

定义 1. 假设数组 x 在循环 v 中定义了数据集合 $defout[v]^x$,那么对 $defout[v]^x$ 的定值以及对集合中数据元素的引用周期,就是数组 x 的一次生命期.

这个定义强调的是,数组一个生命期的开始,并不一定是全部元素的更新,而是在一个循环中对某些数据进行了定值.按照这样的定义,可能会出现的情况是:数组不同生命期的数据在同一个循环中被引用,如图 3 所示.在文中,这样的情况称为生命期重叠.

定义 2. 假设数组 x 存在数据 $x[i]$和 $x[j]$分别在循环 v 和 v'中定义,又都在结点 v''中引用.如果满足 $i\neq j$ 且 $v\neq v'$,则说明 x 在 v 和 v'中分别定义的生命期重叠于 v''.v''就是重叠点.

1.2 定义-引用图

数组生命期的定义强调了其元素集合的建立和使用.为了便于区分数组的生命期和寻找数据分解,我们建立定义-引用图 $G(V,E)$来表示数组的数据集合是如何在循环之间流动的.

定义-引用图中的结点集表示为 V,由程序中的循环、过程调用以及循环和过程调用以外的其他语句构成.在建立 V 时,假设使用文献[17,18]中的方法已经对所有循环做过了并行性分析,并根据 Open64 中依赖测试的能力,将循环分为两类:

- N 层嵌套循环的第 $k(0<k\leqslant N)$层循环是单一嵌套循环,当且仅当该循环的循环体满足:
  (1) 不存在非规则数组区域访问;
  (2) 无跳转、过程调用和过程返回语句;
- N 层嵌套循环的第 $k(0<k\leqslant N)$层循环是复合嵌套循环,当且仅当该循环的循环体无跳转指令.

如果一个复合嵌套循环的外层循环可以并行,或者不包含可并行循环,那就将其加入 V;否则,如果复合嵌套循环的内层可有并行循环,则将其打开,将内层循环依次加入到结点集中.

$L0$ 在例 1 中是复合嵌套循环,并且 $L1$ 和 $L2$ 都是可并行循环.由于 $L0$ 的外层循环携带了真依赖,我们将其忽略,而是把 $L1$,$L2$ 和 $L3$ 加入到节点集中.

定义-引用图中的边集表示为 E.图中的边是有向的,代表了数组的定义-引用关系.E 的一个边(u,v,a)表示:数组 a 有元素在结点 u 定义,并在结点 v 中被引用.

如图 2 所示,$L3$ 携带了 p 数组的流依赖,所以存在一条 p 指向自身的定义-引用边.虽然 w 数组在 $L3$ 也有定义,但是缺少引用,所以没有边加入 E.

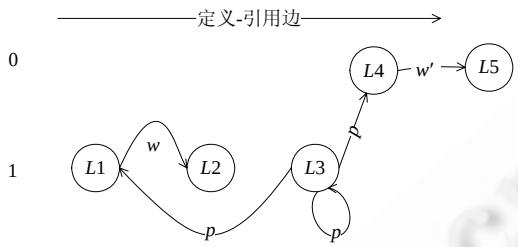


Fig.2 Define-Use graph

图 2 定义-引用图

1.3 定义-引用边

建立定义-引用图的关键在于添加定义-引用边,这需要判断数组在一个循环中定义的数据是否在另一个循环中被引用,而这离不开数组在循环中的定义、引用等数据集合.我们将这些数据集合作为建立定义-引用图的输入,并在本节进行简单介绍.

- $defout[v]^x$:数组 x 在定义-引用图的结点 v 中产生的定义集合,这些定义在 v 中未被消除.换句话说,它们是 x 在 v 中产生且能够到达 v 以外其他结点的定义;
- $use[v]^x$:数组 x 在定义-引用图的结点 v 中引用的数据集合,在 v 中,这些变量没有预先定义.换言之,它们在 v 中是“不满足”的引用,对从前面结点到达 v 的任何定义,都是向上暴露的;
- $kill[v]^x_{v'}$:数组 x 在结点 v 中定义,又在 v'中消除的定义区域集合.对于 $defout[v]^x$ 集合,其在 v'中重新定义的部分就是 $kill[v]^x_{v'}$.这部分数据和 $defout[v]^x$ 属于不同的生命期;
- $livein[v]^x_{v'}$:数组 x 在结点 v 中定义,且能够到达 v'的数据集合.即它们从 v 流向 v'的过程中没有被消除.

对于嵌套循环中的数组引用来说,其所定义或引用的数组元素空间就是由访问函数和外层索引空间决定的.因此,大部分数组的 $defout[v]$和 $use[v]$可以简单求出.在示例 1 的 $L3$ 中,w 数组的 $defout[l3]^w=\{1,n\}$.

但是当循环中存在流依赖时,读引用在其流依赖迭代空间中访问的数组元素都在循环内部定义,而其他数据则是向上暴露的.事实上,我们关心的是数组的数据在循环结点之间是如何流动的,只需要向上暴露的读数据.此时,可以使用基于值的依赖测试[20]为读-写引用对建立一棵 lwt(last write tree)二叉树.读引用访问的数据中不满足写引用循环边界的部分,就是数组的 $use[v]$.在 $L3$ 中,p 数组的 $use[v]^q=\{0\}$.

由于数组的定义集合在从一个结点流向一个引用结点时,集合中的数据元素可能会被途经的其他结点重新定义,开始另外的生命期.假设按照控制流顺序,从结点 v 到达结点 v'需要途径结点 v_1~v_n.$defout[v]^x$在到达一个结点 v_k 时,依然没有被 $\sum_{i=1}^{k} kill[v]^x_{v_i}$ 消除的数据集合就是 $livein[v]^x_{v_k}$.当 $livein[v]^x_{v'}$ 中的数据在 v'中被引用,就表明 x 有元素在 v 中定义并在 v'中引用.

$$livein[v]^x_{v'} \cap use[v']^x \neq \varnothing \tag{1}$$

对定义-引用图的两个不同结点 v 和 v',如果数组满足条件(1),则添加一条从 v 指向 v'的有向边(v,v',x)到边集 E.

2 数据分解映射表

本节将在定义-引用图的基础上对数组的不同生命期建立各自独立的数据分解,并通过映射表来一一对应.数组的每次定义都有其固定的活跃范围,数据分解用于描述数组空间到处理器的映射关系,自然也应该继承了数组的生命期属性.

在划分时,虽然可以使用数组的静态单赋值来区分不同的生命期[20],但是代价太大.通过借鉴这样的思想,

可以使用数据分解的重命名来标记不同的生命期,以此来避免分布一致的限制和冗余通信.首先,寻找被定义-引用边连接在一起的同名数组引用,然后,为它们的数据分解重命名,再通过一个映射表,将数据分解、数组引用以及所在的循环结点关联在一起.

2.1 生命期未重叠

在建立分解映射表时,数组未重叠的生命期是一种最普通、也是最希望遇见的情况.例如在图 1 中,*w* 数组的生命周期界限分明,丝毫没有重叠的部分.处理的方法就是直接为数组不同的生命期建立各自的数据分解,并通过映射表与各自区域内的数组引用一一对应.*w* 在 *L*3 中也属于生命期未重叠的情况,但是由于其定义没有被引用,为 *w* 寻找数据分解并无太大意义,只要划分的循环层没有携带真依赖即可.

在分解映射表中,我们添加了 *region* 项来代表通信区域.当 *region* 对应的数据分解无法满足时,处理器需要交换 *region* 中的数据集合.表 1 是图 9 中代码的分解映射图.对于循环中的写引用,通信的最大区域就是该数组的 *defout* 集合.而对于读引用来说,通信的最大区域就是该数组的向上暴露的 *expose* 集合.

$expose[v']_v^x$:数组 *x* 在定义-引用图的结点 *v* 中定义,且又在 *v*′中引用的数据集合.即它们是 *x* 从 *v*′向上暴露于 *v* 的数据集合. $expose[v']_v^x$ 可以依据等式(2)进行求解.在生命期未重叠的情况下,$expose[v']^x$ 就等于 $use[v']^x$.

$$expose[v']_v^x = livein[v]_{v'}^x \cap use[v']^x \tag{2}$$

虽然数据分解强调的是对数组空间整体的一种分割方式,但通过指定通信区域,就能够细化到了其生命期中定义-引用的数据集合.这样,在分解不一致时,交换的只会是其生命期内定义的数据.在映射表中,*Opr*=*W*/*R* 则代表数组的写/读引用.只有区分数组的读写引用,才能完成区分其不同的生命期.

2.2 生命期重叠

由于数组自身的特殊性,可能会出现生命期重叠的问题.数组的生命期重叠存在 3 种情况,图 3 是这 3 种情况的示例,并分别用 *w*,*w*′和 *w*″代表了 *w* 数组不同的生命期.其中,图 3(a)代表的情况是数组的多个生命期重叠于某个循环,但未引用数组在此循环的定义;图 3(b)则代表了发生生命期重叠时,数组引用了其在循环中的定义;图 3(c)则是图 3(a)和图 3(b)的联合.

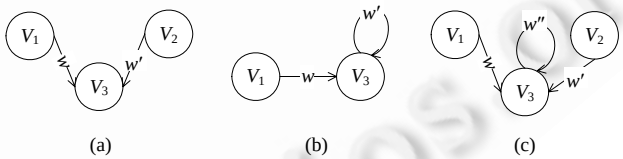


Fig.3 Life cycle overlap

图 3 生命期重叠

生命期重叠的特点是,某个数组在一个循环的 *use* 集合包含了来自不同循环的定义.划分算法要在切割计算和数据空间的同时,尽量避免通信.因此,可以采取严格的处理方法,将这些重叠的生命期看做一个且只有一个数据分解版本.这样就保留了这些数组在重叠的生命期中划分一致的可能性,也与以往的不区分生命期的划分算法是一致的.但缺点是没有挖掘生命期的特性,重叠点以外的循环在尝试合并到一个划分的静态子集时,必然受彼此约束的影响.而这些约束违背了定义-引用的基本要求,是冗余的.例如在图 3(a)中,在合并 V_1 和 V_2 到一个静态子集时,却会受到 *w* 数组两次定义的划分约束.但 *w* 的生命期在 V_1 和 V_2 中并没有重叠,这些约束没有意义.

为了体现定义-引用的特点,也可以将重叠的生命期都充分细化加以区分,每个都有自己独立的数据分解,这样就消除了划分时的冗余约束.但问题在于重叠点的归属,武断地将其划入到某个生命期并不合适.因为重叠点的 *use* 集合也需要一致的分布,一旦切断了数据分解的联系,那么在重叠点就无法通过不同生命期的划分约

束来限制解空间.例如在图 3(b)中,w 在 V_1 中定义产生约束,并通过定义-引用边传播 V_3.将 V_3 直接划入到 w'的生命期,就消除了这个划分限制,使得在 V_3 划分 w 时无法兼顾 V_1 的定义,可能得出不同的数据分解,引起了 w 在两者之间的通信.

由以上的分析可以看出,在为数据分解添加生命期信息时,矛盾集中在重叠点.我们的处理方法是上述两种方法的折衷,首先为每个生命期建立独立的数据分解,然后为重叠点的 *use* 集合建立多个数据分解版本,并与重叠的生命期一一对应.这些数据分解都是为重叠点中的读引用服务,划分约束也都必须保持一致,区别在于分解映射图的 *region* 项.我们将每个定义-引用边流动的 *expose* 集合添加到数据分解对应的 *region* 中.

这样,在重叠点的数据分解映射图中,虽然看起来数组对应了多个分解,但实际上通过 *region* 中数据集合,将划分从对整个数组空间的映射细化到了数据元素.这样就避免了划分约束在重叠点以外的无效传播,又保证了其在重叠点适当限制了划分的解空间.而且,无论哪个数据分解不能满足,都只会交换其 *region* 中的数据,减少了冗余的通信.

以图 3(c)为例,在 V_3 中,根据定义-引用边为 w 建立 3 个数据分解 d_{w1},d_{w2} 和 d_{w3},分别对应每个传播到此的生命期.然后,根据等式(2)分别计算 $expose[v_3]^w_{v_1}$,$expose[v_3]^w_{v_2}$ 和 $expose[v_3]^w_{v_3}$,并记录在 d_{w1},d_{w2} 和 d_{w3} 的通信区域中.其中,d_{w1} 和 d_{w2} 消除了 w 在 V_1 和 V_2 之间无效的定义-引用.而 d_{w3} 同时对应了 w 在 V_3 中读和写引用,确保它们分解一致.在划分 V_3 时,无论哪个数据分解不能满足,交换的数据都限定在其通信区域中.根据第 2.1 节和第 2.2 节的处理方法,建立数据分解映射表的算法如图 4 所示,其中,$lhs(x)$是指数组 x 的写引用,而 $rhs(x)$则是指数组 x 的读引用.

```
01  //功能:建立数据分解映射表
02  //输入:定义-引用图G(V,E);
03  //输出:数据分解映射表Decomp_Map.
04  procedure Create_DecompMap_Algorithm
05    foreach v in V do begin
06      初始化映射表,令Decomp_Map[v]=Null;
07      foreach array x in v do begin
08        if 不存在e(v,v',x) then continue;   //x没有定义
09        建立x的写引用在v'中的数据分解 d_x^v';
10        添加 ⟨v,lhs(x)_v,d_x^v',defout[v]^x⟩ 到Decomp_Map[v];
11      end
12    end
13    foreach v in V do begin
14      foreach array x in v do begin
15        foreach e(v',v,x) in E do   //v是e的终点
16          计算x的 expose[v]_v^x,等于 livein[v']_v^x ∩ use[v]^x;
17          搜索Decomp_Map[v'],找到lhs(x)_v'的数据分解 d_x^v';
18          添加 ⟨v,rhs(x)_v,d_x^v',defout[v]_v'^x⟩ 到Decomp_Map[v];
19        end
20      end
21    end
22    return Decomp_Map;
23  end Create_DecompMap_Algorithm
```

Fig.4 Algorithm of data decomposition mapping table

图 4 数据分解映射表算法

3 数据分解算法

本节是基于生命期对数据分解算法进行改进.为了方便后端代码生成,数据分解算法使用了仿射划分.在定义-引用图和分解映射表的基础上,首先对分解核空间的计算进行了改进,以保证在重叠点添加的数据分解保持一致;然后建立了开销评估模型,通过精确的通信代价,来考察一个嵌套循环的并行是否具有收益;最后给出了

完整的数据分解算法.

3.1 仿射分解

仿射分解是一种有效表示、求解计算和数据分解的方法,广泛应用于划分研究中[1−3,5−9].数据分解刻画的是 m 维数组空间到 n 维处理器空间的映射,表示为 $\vec{d}:A\to P, d(\vec{a})=D\vec{a}+\vec{\delta}$,其中,$D$ 是一个 $n\times m$ 的线性转换矩阵,$\vec{d}$ 是偏移常向量,$\vec{a}$ 是数组索引向量.计算分解刻画的是 l 维迭代空间到 n 维处理器空间的映射,表示为 $\vec{c}:I\to P, c(\vec{i})=C\vec{i}+\vec{\gamma}$,其中,$C$ 是一个 $n\times l$ 的线性转换矩阵,$\vec{\gamma}$ 是偏移常向量,$\vec{i}$ 是迭代向量[1,2].

$$D_x(\vec{f}_{xj}^{\,k}(\vec{i}))+\vec{\delta}_x=C_j(\vec{i})+\vec{\gamma}_j \tag{3}$$

如果处理引用的数据就在本地,那就不需要通信,等式(1)描述的就是数据和计算之间的这种关系.如果能够为每个嵌套循环 j 定义一个计算分解,并为每个数组 x 定义一个数据分解,使它们满足等式(3),那么就没有通信发生.

3.2 分解核空间的计算

为了增加划分的选择,优先保证没有代价较大的数据重组通信,一般都暂时舍弃等式(3)的偏移部分,使用等式(4)来求解.一个循环的无通信划分的通用解是设置计算分解 C=0 和数据分解 D=0,所以,C 的核空间 $N(C)$就生成了整个迭代空间 I,D 的核空间 $N(D)$生成了整个数组空间 A[1,2].$N(C)$和 $N(D)$的物理意义是计算和数据需要被指定到相同处理器的空间约束.

$$D_xF_{xj}^{k}(\vec{i})=C_j(\vec{i}) \tag{4}$$

划分算法首先根据数组的依赖信息和循环的可并行性,在每个循环为计算和数组计算分解的核空间;然后通过合并核空间,尽量将所有的循环都加入一个划分的静态子集.静态子集由循环结点组成,数组在子集内只能有一种分布方式.

当数组在两个循环中的核空间合并时,就对划分的约束进行了传播.但是,以往的研究不区分数组引用的读和写,这就导致数组一次的定义所产生的核空间会传播到其定义-引用范围之外的循环,约束到不同生命期的引用(甚至定义)的并行.例如在图 1 中,w 数组在 $L3$ 循环的 $N(D_w)$=$span${(1)},在 $L1$ 和 $L2$ 的 $N(D_w)'$=∅.如果合并 $N(D_w)$和 $N(D_w)'$,会导致 w 在 $L1$ 和 $L2$ 中也无法并行.实际上,w 在 $L3$ 的定义没有被 $L1$ 和 $L2$ 引用.

通过分解映射表,能够为不同周期的定义-引用建立各自的数据分解,也将生命期引入到了分解的核空间.重叠点一个数组的读引用,可能要对应多个数据分解.所以在划分的静态子集中,不仅要计算每个数据分解的核空间,还要保证重叠点的读引用对应的多个核空间保持一致.这里的处理方法是,为每个数据分解的核空间添加过约束后,再合并重叠点的读引用对应的多个核空间,使其保持一致.计算核空间的算法如图 5 所示,其中,S 是一个静态子集;*Propagate_Nullspace* 是传播函数,用于计算和数据分解之间的核空间传播,函数详细的算法见文献[1,2].

```
01  //功能:计算静态划分区的核空间
02  //输入:G(V,E),静态子集S;
03  //输出:分解的核空间集合(Γ,Δ).
04  procedure Calc_Nullspaces
05    foreach v∈S do
06      N(C_v)=/*添加Doacorss约束*/;
07      Γ=Γ∪N(C_v);   //x是任意数组
08      foreach d_x^v in Decomp_Map[v] do
09        N(D_x^v)=/*添加x数组的依赖约束*/;
10        Δ=Δ∪N(D_x^v);
11    end
12    Propagate_Nullspace(Γ,Δ);
13    foreach v∈S do   //x是任意数组
14      foreach rhs(x) in Decomp_Map[v] do
15        合并rhs(x)每个数据分解的核空间;
16        更新回每个核空间;
17    end
18    return (Γ,Δ);
19  end Calc_Nullspaces
```

Fig.5 Algorithm of calculating null-space

图5 计算核空间的算法

3.3 开销评估

数据分解算法需要建立开销评估模型来考察一个嵌套循环的并行是否具有收益,并通过对比并行前后的开销来决定是否提交并行.而通信代价则是开销评估中重要的依据,其计算是否合理和精确,直接影响并行收益的评估.通信的产生是由于对数据的定义和引用没有分布在同一处理器上,在忽略定义-引用信息的情况下,一旦数据分解不一致,就要通信整个数组空间.而在分解映射表的基础上,可以根据定义-引用边和通信区域来计

算通信代价,以此增加开销评估的精度.开销由计算开销和通信开销构成.

计算开销的求解过程是,算法对静态子集中包含的指令数目做出静态的评估.一个循环 v 串行执行时占用的时钟周期数 $seq(v)$,由指令数目 $instr(v)$、一条指令的时钟周期数 cpi 以及循环迭代总数 $iter(v)$三者的乘积获得;而并行执行占用的时钟周期数 $par(v)$,则由 $seq(v)$除以处理器个数 np 获得.计算公式见等式(5).

$$\begin{cases} seq(v) = iter(v) \times instr(v) \times cpi \\ par(v) = seq(v) / np \end{cases} \tag{5}$$

通信开销的求解过程则是算法对静态子集的通信代价进行静态评估.一条定义-引用边 $e(v,v',x)$带来的通信代价 $w(e)$,由 $\psi(e)$和 $\omega(|region|)$的积获得.$\psi(e)$是数组 x 从 v 到 v'路径的执行频率,$\omega(k)$是在目标机器上远程交换 k 个数据所需的时间,$|region|$则是 e 引用的数据总量.计算公式见等式(6).

$$w(e)=\psi(e)\times\omega(|region|) \tag{6}$$

由两个由定义-引用边连接的静态子集,如果它们的核空间合并后未满秩,说明依然存在着一致的分解,就可以忽略这个定义-引用边.我们编译器生成的是 SPMD 程序,并遵循冗余并行执行模型[21],即串行数据在各个处理器冗余执行.如果一个数组的定义是串行的,那么对其引用就不会引起通信,自然划分的限制也可以忽略.图 6 是开销评估的算法.

```
01  //功能:计算静态划分区的通信代价
02  //输入:G,S,Decomp_Map,Δ;
03  //输出:cost.
04  procedure Calc_Cost
05     cost=0;
06     foreach v∈S do   //计算开销
07        sequ_cycles(v)=iter(v)×instr(v)×cpi;
08        par_cycles(v)=sequ_cycles(v)/np;
09        cost=cost+par_cycles(v);
10     end
11     foreach e(v,v',x),v或v'∈S do
12        if v∈S then
13           Δ'=v'所在静态子集的核空间;
14        else
15           Δ'=v所在静态子集的核空间;
16        从Decomp_Map中找到v的数据分解d_x;
17        if Δ和Δ'的N(D_x)合并后满秩 then
18           if v循环串行 then contine;
19           if v & v'∈S then
20              cost=∞;
21              continue;
22           从Decomp_Map[v']取出region;
23           w(e)=ω(|region|)×ψ(e);
24           cost=cost+w(e);
25     end
26     return cost;
27  end Calc_Cost
```

Fig.6 Algorithm of cost evaluation

图 6 开销评估的算法

假设在循环结点 v 和 v'合并到一个静态子集时,某个数组 x 的 $N(D_x)$合并后满秩,这说明 x 无法在子集内分解一致,合并是失败的.此时,将 $cost$ 设置为无穷大.

3.4 寻找计算和数据分解

数组在一个划分的静态子集内没有数据重组通信,所以在寻找动态的计算和数据分解时,要将尽可能多的循环合并到一个静态划分区内.为了优先划分核心循环,文献[1,2]中采用了贪婪算法,优先合并通信代价最大的循环.但是这会将不相邻的循环划入同一静态划分区,可能会引起更多的重分布.文献[5,6]则为了解决这个问题,将通信图改为融合控制流的层次分解图,并按照控制流逐步合并静态子集(如图 7 所示).

并行程序依然遵循串行的执行序,所以本文按照控制流的合并顺序来寻找数据分解.根据定义-引用图的结点集 V,我们建立了控制流图 $G_c(V,E_c)$.数据分解算法按照 G_c 自底向上的顺序,先对最内层的嵌套循环进行划分,以减少被循环包围的通信.在同层的嵌套循环间,算法按照向前流的顺序,依次合并由 E_c 中的控制流边连接在一起的循环结点.

假设按照 $e_c(v,v')$合并 v'到 v 的划分静态子集.在寻找 v 的分解时,已经对其子集的开销进行了计算,不妨设为 $init\_cost$.此时的合并操作主要考察的是 v'的串并行开销,因此,合并后的新开销 $new\_cost=init\_cost+par(v')+w(v')$,合并前的总开销为 $old\_cost=init\_cost+seq(v')$.合并前后开销的计算公式见等式(7).当开销能够满足不等式(8),即 $new\_cost<old\_cost$ 时,就说明对 v'的划分是值得的,算法提交 v'的并行.这意味着循环合并是有收益的,可以提交循环合并到一个静态子集;否则就撤销合并操作,为待合并循环创建新的静态子集.

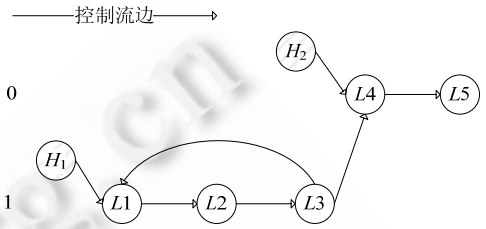


Fig.7 Control flow graph of example 1

图 7 示例 1 的控制流图

$$\begin{cases} new\_cost = init\_cost + par(v') + w(v') \\ old\_cost = init\_cost + seq(v') \end{cases} \tag{7}$$

$$new\_cost<old\_cost \tag{8}$$

为了能够正确判断每层的第 1 个循环是否值得划分,我们在其之前添加了虚结点.虚结点的计算为空,只有指向第 1 个结点的控制流边,没有定义-引用边,默认是并行的.在寻找数据分解的过程中,一旦出现合并失败的情况,也在处理的结点前添加一个虚结点,以便于为这个循环的寻找合适的数据分解.数据分解算法如图 8 所示,图中没有列出计算分解矩阵和偏移的详细算法,可以在文献[1,2]中找到.

```
01  //功能:寻找动态计算和数据分解
02  //输入:串行程序P;
03  procedure Dynamic_Decomposition_Algorithm
04    为P建立定义-引用图G(V, E)
05    根据P和G建立控制流图Gc(V,Ec);
06    Decomp_Map=Create_DecompMap(G);
07    foreach v∈Vd do
08      初始化v到静态子集Sv;
09      (Γv,Δv)是Sv的分解核空间集合;
10      (Γv,Δv)=Calc_Nullspaces(G,Sv,Decomp_Map);
11    end
12    foreach level p in G, in bottom-up order do
13      foreach e(v,v')∈Ec at p, in forward-flow order do
14        (Γv,Δv)是Sv的分解核空间集合;
15        old_cost=Calc_Cost(G,Sv,Decomp_Map,Δv);
16        old_cost=old_cost+iter(v')*instr(v')*cpi
17        (Γv',Δv')是Sv'的分解核空间集合;
18        合并Sv'到Sv,重新计算(Γv,Δv);
19        if Calc_Cost(Sv,Δv)<old_cost then
20          确认合并;
21        else
22          放弃合并,回滚;
23          在v'前添加一个的虚结点Hv';
24          添加e'(Hv',v')到Ec,用于下次迭代处理v'的划分;
25    end
26    foreach Sv∈G的静态子集 do
27      根据公式(2)计算分解矩阵;
28      根据公式(1)计算偏移;
29    end
30  end Dynamic_Decomposition_Algorithm
```

Fig.8 Algorithm of data decomposition

图 8 数据分解算法

4 实例、实验与分析

4.1 实例与分析

图 9 是矩阵运算中的求逆算法,本节对其进行实例分析,介绍本文数据分解算法的划分过程.

```
for i=1 to n do                    //doacorss
    a[i,i]=1/a[i,i];
L1  for j=1 to i−1,i+1 to n do
        a[i, j]=a[i,j]*a[i,i]
L2  for k=1 to i−1 n do
        for j=1 to n do
            a[k,j]=a[k,j]−a[k,i]*a[i,j]
    for j=1 to i−1,i+1 to n do
        a[i,j]=a[i,j]−a[i,i]*a[i,j]
    for k=i+1 to n do
        for j=1 to n do
            a[k,j]=a[k,j]−a[k,i]*a[i,j]
L3  for k=1 to i−1,i+1 to n do
        a[k,i]=−a[k,i]*a[i,i];
end
```

Fig.9 Matrix inversion

图 9 矩阵求逆

1. 建立例 3 的定义-引用图.

- 首先,建立 3 个循环结点,对应 *L*1,*L*2 和 *L*3,其中,*L*1 和 *L*2 分别包含了两个循环,*L*3 则包含了 4 个循环.为了便于说明问题,依然将其各自看作一个循环结点;
- 然后,建立一个语句结点,对应 *a*[*i*,*i*]的计算语句 *L*0.由于最外层循环无法并行,所以 *a*[*i*,*i*]的计算实际上是在各个处理器上冗余并行执行的,并且只引用在 *L*2 定义的一个数据,在这里暂时忽略其生命期和划分问题;
- 最后,根据输入的 *defout*,*use*,*kill* 以及 *livein* 集合,为循环结点添加定义-引用边.我们在图 10 中通过 a_1, a_2 和 a_3 来标明 *a* 数组的 3 个生命期.

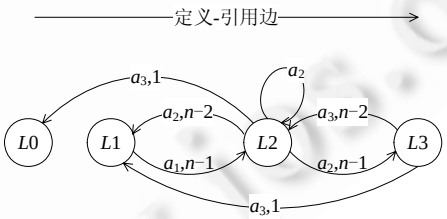


Fig.10 Define-Use graph of matrix inversion

图 10 矩阵求逆的定义-引用图

2. 建立数据分解映射表.

- 首先,为 *a* 在 *L*1,*L*2 和 *L*3 中的定义建立各自的数据分解 d_{a_1},d_{a_2} 和 d_{a_3},添加到分解映射表中,*Opr* 为 *W*,*Region* 为各自的 *defout* 集合;
- 然后,根据指向 *L*1 的两条定义-引用边,为 *a* 数组添加两个数据分解 d_{a_2} 和 d_{a_3},*Opr* 为 *R*,*Region* 为根据等式(2)求出的 *expose* 集合;
- 最后,再根据以 *L*2 和 *L*3 为终点的边,添加分解表项.

表 1 就是矩阵求逆的数据分解映射表.

Table 1 Data decomposition mapping table

表 1 数据分解映射表

循环	数组名	Opr	数据分解	Region
*L*1	*a*	*W*	d_{a_1}	[*i*,1~*i*−1]∪[*i*,*i*+1~*n*]
*L*1	*a*	*R*	d_{a_2}	[*i*+1,1~*i*−1]∪[*i*+1,*i*+2~*n*]
			d_{a_3}	[*i*+1,*i*]
*L*2	*a*	*W*	d_{a_2}	[1~*i*−1,1~*n*]∪[*i*,1~*i*−1]∪[*i*,*i*+1~*n*]∪[*i*+1~*n*,1~*n*]
*L*2	*a*	*R*	d_{a_1}	[*i*,1~*i*−1]∪[*i*,*i*+1~*n*]
			d_{a_2}	*defout*[*L*2]−[1~*i*−1,*i*]∪[*i*+2~*n*,*i*]−[*i*+1,1~*i*]∪[*i*+1,*i*+2~*n*]
			d_{a_3}	[1~*i*−1,*i*]∪[*i*+2~*n*,*i*]
*L*3	*a*	*W*	d_{a_3}	[1~*i*−1,*i*]∪[*i*+1~*n*,*i*]
*L*3	*a*	*R*	d_{a_2}	[1~*i*−1,*i*]∪[*i*+1~*n*,*i*]

3. 动态划分算法.计算每个循环的核空间,并默认串行执行.

首先,按照控制流边取出 *H* 和 *L*1,尝试合并 *L*1 到 *H* 的静态子集.*H* 是虚结点,初始开销显然为 0.所以,算法对比的就是 *L*1 循环串行执行与并行执行的开销.*L*1 和 *L*2 的核空间合并不满秩,而 *L*3 默认串行,所以通信代价为 0.根据等式(5)~等式(7),求出了合并前开销 *old_cost* 和合并后开销 *new_cost*.合并前开销的计算方法为 *old_cost*=*init_cost*+*seq*(*L*1),其中,*seq*(*L*1)=*n*(*n*−1)*cpi*,*init_cost*=0;合并后开销的计算方法为 *new_cost*=*init_cost*+*seq*(*L*1)/*np*+*w*(*L*1),其中,*w*(*L*1)=*init_cost*=0,*seq*(*L*1)=*n*(*n*−1)*cpi*.当满足 *new_cost*<*old_cost* 时,即 *n*(*n*−1)*cpi*/*np*<*n*(*n*−1)*cpi* 时,说明 *L*1 的并行有收益,可以提交对 *L*1 的划分.由于并行程序的进程数 *np*≥2,条件能够满足,*H* 和 *L*1 被合并到一个静态子集,以下称为 *L*1 静态子集.

然后,尝试合并 *L*2 到 *L*1 的静态子集,初始为 *L*1 的并行后开销.*L*2 和 *L*1 的核空间合并不满秩,*L*3 依然串行,所以只需要通信定义-引用边(*L*2,*L*3,*a*)对应的 *region*.根据等式(5)~等式(7),对合并前后的开销进行了计算.合并前开销的计算方法为 *old_cost*=*init_cost*+*seq*(*L*2),其中,*init_cost*=*n*(*n*−1)*cpi*/*np*,*seq*(*L*2)=*n*(n^2−1)*cpi*;合并后开销 *new_cost*=*init_cost*+*seq*(*L*2)/*np*+*w*(*L*2),其中,*w*(*L*2)=*n*×*ω*(*n*−1).设循环迭代次数 *n* 为 1 024,进程数 *np*为 4,带入公式(8)为(1024+10242−2)*cpi*/4+*ω*(1024−1)<((1024−1)/4+1024^2−1)*cpi*.即当*ω*(1)<769×*cpi* 时,*L*2 的并行被提交.假设在这里完成了合并,以下称为 *L*1-*L*2 静态子集.

随后,尝试合并 *L*3 到 *L*1-*L*2 静态子集中.由于 *L*3 和 *L*1-*L*2 静态子集合并后 *a* 数组的数据分解核空间满秩,合并失败.因此,在 *L*3 前添加了虚结点 H_1,尝试合并 *L*3 到 H_1 静态子集中,初始代价为 0.合并后存在两条需要通信的定义-引用边,|*region*|的总量为 *n*−1.根据等式(5)~等式(7),求出了合并前后开销.合并前开销 *old_cost*=*init_cost*+*seq*(*L*3),其中,*init_cost*=0,*seq*(*L*3)=*n*(*n*−1)*cpi*;合并后开销 *new_cost*=*init_cost*+*seq*(*L*3)/*np*+*w*(*L*3),其中,*w*(*L*3)=*n*×*ω*(*n*−1).将 *n*=1024 和 *np*=4 代入不等式(8)可得:(1024−1)*cpi*/4+*ω*(1024−1)<1024(1024−1)*cpi*.即当*ω*(1)<3/4×*cpi* 时,才会提交 *L*3 的并行.考虑到分布存储结构下通信能力和计算能力的不匹配,只有在通信速度非常快时,才会划分 *L*3.

最终结果为 *L*1 和 *L*2 划分后合并至一个静态子集,*a* 数组在子集分布一致;*L*3 没有被划分,*a* 数组在 *L*3 的定义中串行执行.

4. 在 IBM System x3850 M2 对矩阵求逆的 3 种并行程序进行了正确性测试和性能模拟.

M2 配置了两颗 4 核 CPU,型号是 Intel Xeon E7420,核心频率为 2.13GHz,内存为 4GB.3 个并行程序中,CloseL3 代表了本文算法的划分结果,由编译器自动生成,没有划分 *L*3.OpenL3 则是在 CloseL3 基础上手工打开了 *L*3 的并行.A&H 代表了文献[1,2,5,6]中的仿射分解算法,根据论文手工编写.三者测试结果如图 11 所示,横坐标轴是进程数,纵坐标轴是加速比.

矩阵求逆的特点是外层循环无法并行,内层循环的计算量不多,对通信特别敏感.*a* 数组需要在两个静态子集间做两次重分布.在不区分生命期的情况下,通信的是数组的整体空间,即|*region*|=n^2,根据公式(6)计算出的通

信量为 $n^3\omega(1)$.过多的冗余通信抵消了并行的收益,导致 A&H 的加速比不明显,甚至为负.在为数据分解添加了生命期属性后,根据公式(6)计算出通信的数据量由 $n^3\omega(1)$减少为 $n(n-1)\omega(1)$,即从 $O(n^3)$减少到了 $O(n^2)$.测试取得了较好的加速比,达到了预期的效果.而 OpenL3 虽然打开了 *L*3 的并行,通信量也增加到 $2n(n-1)\omega(1)$,但是并未取得更好的并行效果,加速比反而有所降低.这是由于 M2 的计算能力很强,但是访存速度却不能与其很好地匹配.

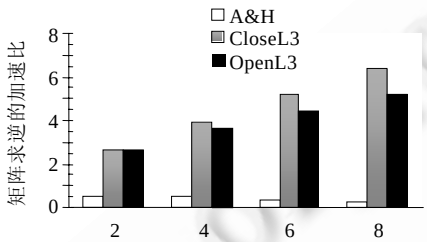


Fig.11 Experimental results of matrix on M2

图 11 矩阵求逆的 M2 实验结果

4.2 实验与分析

为了进一步验证引入生命期后数据分解算法的有效性,对基准测试集 NPB 中的 5 个示例以及 4 个常见的科学计算程序进行了测试.

本文的数据分解算法是基于 Open64 编译器实现的,通过自动并行化,生成了测试用例的并行版本.实验的测试结果与 Anderson[1,2]和 Han[5,6]的算法进行了对比.Anderson 的算法使用了仿射分解和通信图,并通过贪婪算法寻找计算和数据分解,具有代表性.Han 的算法是对 Anderson 研究的改进,将通信图改为融合控制流的层次分解图.我们根据两者的论文对测试用例进行了手工并行.手工并行严格依照两者的算法中对 NPB 程序的核心循环进行了划分,但抛弃了一些无收益小循环.而对其他测试用例则进行了全并行.

4.2.1 矩阵运算和 ADI 方法

基于区分生命期的数据分解算法实现,我们首先在超微服务器上对 4 个科学计算程序进行了加速比测试,以验证算法的适用性和正确性.超微服务器配置了 4 颗 6 核 CPU,型号是 Intel Xeon X5670,主频为 2.93GHz,内存为 36GB.测试用例包括了矩阵运算中的矩阵求逆和矩阵乘两个程序、ADI 运算以及基于 ADI 的 2D-Heat 程序.测试结果如图 12 和图 13 所示,横坐标轴是进程数,纵坐标轴是加速比.

矩阵运算是数值计算中最重要的一类运算,特别是在线性代数和数值分析中,它是一种最基本的运算[23].测试选取了矩阵求逆和矩阵相乘两个程序.

实例分析中详细列出了矩阵求逆寻找分解的过程,并进行了性能模拟.矩阵求逆存在着生命期重叠现象,经过本文算法的优化,通信量从 $O(n^3)$减少到了 $O(n^2)$,在超微上的测试也取得了预期的加速效果.矩阵相乘在并行后只带来了少量的规约通信,十分适合并行.3 种划分方法对矩阵相乘给出了一样的数据分解结果,加速效果基本一致.两个测试程序的源代码见文献[22].

交替方向迭代 ADI 通常用于求解偏微分方程,核心代码包含了 6 个可并行的嵌套循环.由于真依赖的影响,前 3 个循环只能进行行划分,后 3 个循环只能进行列划分.ADI 中存在着复杂的生命期重叠现象,但是由于依赖限制了划分选择,区分生命期后也无法减少两个静态子集间的通信.3 种划分方法给出了一样的数据分解结果,加速比都不好;甚至在两个进程的测试中,还取得负加速比.在早期对 ADI 的划分研究中,有些给出了只做行划分,并且其他循环串行的数据分解结果.通过手工实现了这样的并行,测试结果虽然在进程 2 和进程 4 时取得了稍好的加速比,但趋势与 3 种并行程序类似.

2D-Heat 使用了 ADI 方法处理 2D 的热公式(heat equation),即偏微分方程 $u_t=b_1u_{xx}+b_2u_{yy}$,并将两维问题简化成一连串的一维问题.2D-Heat 的核心代码分为行扫描区域和列扫描区域,两个区域因为真依赖的存在,不能合

并成一个划分的静态子集.主导数组 *u* 和 *v* 在区域间存在生命期重叠现象,但由于引用的数据量较大,区分生命期后并未减少通信量.工作数组 *p* 和 *q* 在两个区域中都存在不重叠的生命期,本文算法在优化了 *p* 和 *q* 的冗余通信后,提升了加速比.2D-Heat 的源代码见文献[4].

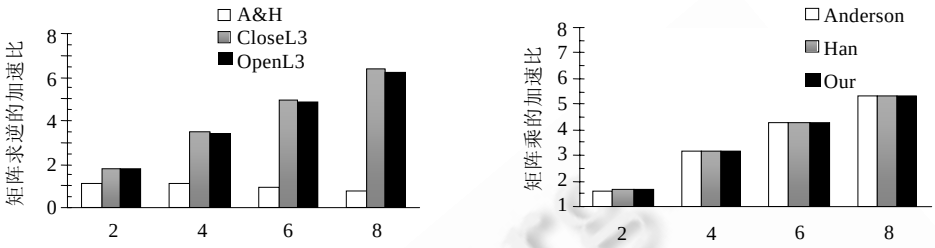


Fig.12 Experimental results of matrix inversion and multiplication on Supermicro

图 12 矩阵求逆和矩阵乘的超微实验结果

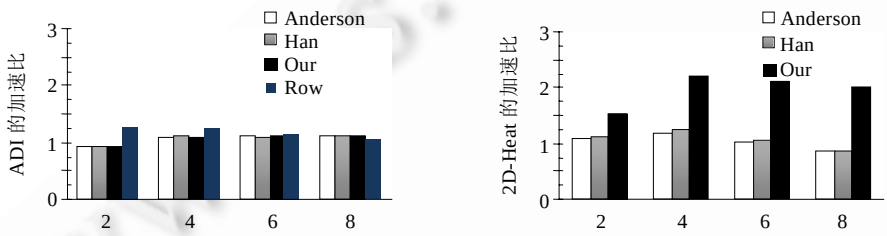


Fig.13 Experimental results of ADI and 2D-Heat

图 13 ADI 和 2D-Heat 的实验结果

4.2.2 NPB 测试

为了验证算法在较大规模时的并行性能,我们在神威蓝光(Sunway Blue-Light)集群下对 NASA(National Aeronautics and Space Administration)提供的 NPB 基准测试集进行了测试.神威蓝光是由我国自主研发的高性能计算机系统,CPU 为 16 核 64 位处理器的 SW1600,单核最高频率为 1.1GHz.NPB 基准测试集目前广泛应用于并行计算机性能的评价,从中选取了 BT(块状三角)、SP(五对角方程)、CG(共轭梯度)、LU(稀疏矩阵分解)和 FT(快速傅里叶变换)这 5 个用例进行测试,测试规模为 A 规模,结果如图 14~图 16 所示.

BT 用于模拟计算流体动力学.*rhs* 是程序的主导数组,并主要通过 3 个程序段对其进行计算.3 个程序段都包含了大量的循环,但由于真依赖的存在,只能加入两个不同的划分静态子集.而 *fjac*,*njac*,*lhs* 等 3 个等数组主要为 *rhs* 的计算提供数据,每个程序段都将之前的定义注销.所以,这 3 个数组实际有 3 个未重叠的生命期.由于 Anderson 和 Han 的算法未区分数组的定义和引用,所以在程序段之间多产生了 *fjac*,*njac*,*lhs* 的数据重组通信.本文算法通过为数据分解添加生命期信息,消除了这些冗余通信,取得了预期的并行效果.而其他两个版本的并行程序虽然也取得了加速比,但是过多的通信降低了并行的收益.SP 的代码在细节上与 BT 十分类似,区别是通信相对计算的比例更高,所以加速比略逊.

CG 用于求解大型稀疏对称正定矩阵的最小特征值的近似值,最外层循环使用反幂法,次外层循环则使用了共轭梯度法,但这两层循环均不能并行.共轭梯度法内部的循环多是做向量计算,对通信比较敏感.*w* 和 *r* 等数组多次被清零,并存在多个未重叠的生命期.这些生命期之间的冗余通信降低了 Anderson 和 Han 两个版本程序的并行性能,本文算法则通过为数据分解添加生命期信息取得了较好的加速效果.

LU 的核心循环存在真依赖,因此无法被算法划分.虽然函数 *rhs* 也包含了许多的循环并被多次调用,但在整体计算中的比重不大.对其 *rhs* 中的循环进行划分后取得了少量的加速效果.Andeson 和 Han 的并行版本存在数组 *flux* 不同生命期间的冗余通信,但由于 *rhs* 函数没有在计算中占据主导地位,所以 3 种并行程序的加速比差别

并不十分明显.FT 的计算相对较小,通信量大,因此自动划分后的并行收益不高.数组 *y*0 则在核心循环中存在 3 个不重叠的生命期,在消除了冗余通信后,加速比得到了少量提升.

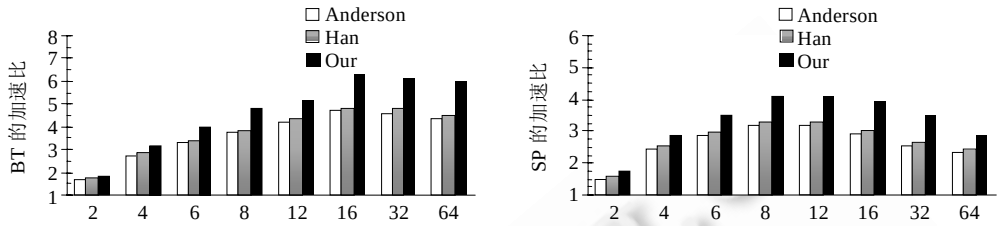


Fig.14 Experimental results of BT and SP

图 14 BT 和 SP 的实验结果

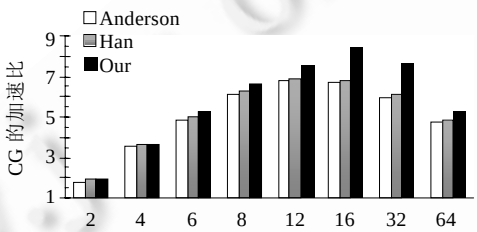


Fig.15 Experimental results of CG

图 15 CG 的实验结果

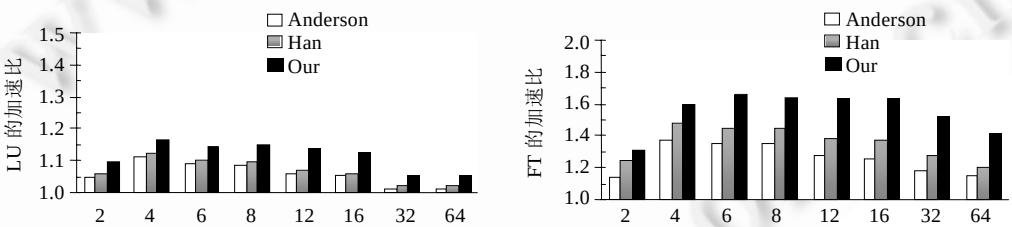


Fig.16 Experimental results of LU and FT

图 16 LU 和 FT 的实验结果

在测试的 5 个 NPB 程序中,主导数组都存在生命期重叠现象,而且基本都是图 3(b)所示的情况.由于重叠点引用数据大都是向上暴露,这导致在分解不一致时,是否区分生命期的通信量几乎一样多.所以,本文的数据分解算法未在 NPB 测试中对此类现象取得较好的优化效果.同时,5 个 NPB 程序也存在了较多的生命期不重叠情况.数据分解算法通过区分生命期对此类情况进行了优化,提升了加速比.

5 结束语

相对于共享存储结构和短向量部件而言,实现分布存储结构下的自动并行化更加困难.并行化编译器不仅要识别出程序中的并行性及安排并行调度,同时要管理物理上分布的存储空间.这意味着程序中的数据要在不同的存储器之间合理分布,并且在并行执行过程中还要安排消息通信以保证数据在各个进程间的一致性.

数据分解反映了数组与处理器间的映射关系,也相应继承了数组的生命期.根据数组定义-引用周期来建立数据分解,不仅在划分时减少了不必要的约束、更准确地评估并行的收益,还减少了冗余通信,提升了程序的并行性能.

但通过测试我们也发现,总有一些通信在程序自动并行后是无法避免的.而传统的代码生成研究往往通过

冗余复制技术来生成通信,也就是将一个处理器局部定义的所有数据都保守地发送到其他每个处理器.这种保守的通信代码生成方法易于实现,但也意味着处理器会接收到计算不需要的数据.下一步研究的重点将是根据数据分解在数组重分布前后的转换,确定每个数组元素迁移的源处理器和目标处理器,通过减少通信冗余来提高这些通信的效率.

References:

[1] Anderson JM, Lam MS. Global optimizations for parallelism and locality on scalable parallel machines. In: Cartwright R, ed. Proc. of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation. New York: ACM Press, 1993. 112−125. [doi: 10.1145/155090.155101]

[2] Anderson JM. Automatic computation and data decomposition for multiprocessors [Ph.D. Thesis]. Stanford: Stanford University, 1997.

[3] Lim AW. Improve parallelism and data locality with affine partitioning [Ph.D. Thesis]. Stanford: Stanford University, 2001.

[4] Lee PZ, Kedem ZM. Automatic data and computation decomposition on distributed memory parallel computers. ACM Trans. on Programming Languages and Systems, 2002,24(1):1−50. [doi: 10.1145/509705.509706]

[5] Han L, Zhao RC, Pang JM. Dynamic decomposition algorithm merging control flow analysis. In: Arabniaed HR, ed. Proc. of the 2007 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications. CSREA Press, 2007. 245−250.

[6] Han L, Zhao RC, Pang JM. A consistency combination algorithm for global dynamic computation and data decompositions. In: Proc. of the IEEE 2nd Int'l Conf. on Complex, Intelligent and Software Intensive Systems. Washington: IEEE Computer Society, 2008. 148−154. [doi: 10.1109/CISIS.2008.89]

[7] Xia J, Yang XJ. A data space fusion based approach for global computation and data decompositions. Ruan Jian Xue Bao/Journal of Software, 2004,15(9):1311−1327 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1311.htm

[8] Ding R, Zhao RC, Han L. An automatic computation and data decomposition algorithm based on dominant value. Computer Science, 2012,39(3):290−294 (in Chinese with English abstract).

[9] Zhang WH, Wang P, Zang BY. An affine partition algorithm based on representative element. Chinese Journal of Computers, 2008, 31(3):400−410 (in Chinese with English abstract).

[10] Wang M, Bodin F, Matz S. Automatic data distribution for improving data locality on the cell BE architecture. In: Gao GR, Pollock LL, Cavazos J, Li XM, eds. Proc. of the 22th Int'l Workshop on Languages and Compilers for Parallel Computing. Berlin, Heidelberg: Springer-Verlag, 2009. 247−262. [doi: 10.1007/978-3-642-13374-9_17]

[11] Wang YR, Chen L, Feng XB, Zhang ZQ. Global partial replicate computation partitioning. Journal of Computer Research and Development, 2006,43(12):2158−2165 (in Chinese with English abstract).

[12] Shih KP, Sheu JP, Huang CH. Statement-Level communication-free partitioning technique for parallelizing compilers. The Journal of Supercomputing, 2000,15(3):243−269. [doi: 10.1007/BFb0017265]

[13] Chen TS, Chang CY. Skewed data partition and alignment techniques for compiling programs on distributed memory multicomputer. The Journal of Supercomputing, 2002,21(2):191−211. [doi: 10.1007/3-540-39999-2_10]

[14] Fresno J, González-Escribano A, Llanos DR. Automatic data partitioning applied to multigrid PDE solvers. In: Proc. of the 19th Euromicro Int'l Conf. on Parallel, Distributed and Network-Based Processing. Washington: IEEE Computer Society, 2011. 239−246. [doi: 10.1109/PDP.2011.38]

[15] Kwon O, Jubair F, Eigenmann R, Midkiff S. A hybrid approach of OpenMP for clusters. In: Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2012. 75−84. [doi:10.1145/2370036.2145827]

[16] Zhong Z, Rychkov V, Lastovetsky A. Data partitioning on heterogeneous multicore platforms. In: Proc. of the 2011 IEEE Int'l Conf. on Cluster Computing. Washington: IEEE Computer Society, 2011. 580−584. [doi: 10.1109/CLUSTER.2011.64]

[17] Zhao J, Zhao RC, Ding R, Huang PF. Parallelism recognition technology based on nested loops classifying. Ruan Jian Xue Bao/Journal of Software, 2012,23(10):2695−2704 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4178.htm [doi: 10.3724/SP.J.1001.2012.04178]

[18] Zhao J, Zhao RC, Han L. A nonlinear array subscripts dependence test. In: Proc. of the 14th IEEE Int'l Conf. on High Performance Computing and Communications. Washington: IEEE Computer Society, 2012. 764−771. [doi: 10.1109/HPCC.2012.108]

[19] Gu JJ, Li ZY. Efficient interprocedural array data-flow analysis for automatic program parallelization. IEEE Trans. on Software Engineering, 2000,26(3):244−261. [doi: 10.1109/32.842950]

[20] Rus S, He G, Alias C, Rauchwerger L. Region array SSA. In: Proc. of the 15th Int'l Conf. on Parallel Architectures and Compilation Techniques. New York: ACM Press, 2006. 43−52. [doi: 10.1145/1152154.1152165]

[21] Zhong HT, Shu JW, Wen DC, Zheng WM. A communication optimization algorithm based on data-flow analysis of region graph. Ruan Jian Xue Bao/Journal of Software, 2003,14(2):175−182 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/175.htm

[22] Chen GL, An H, Chen L, Zheng QL, Dan JL. Parallel Algorithm Practice. Beijing: Higher Education Press, 2004. 459−486 (in Chinese).

附中文参考文献:

[7] 夏军,杨学军.基于数据空间融合的全局计算与数据划分方法.软件学报,2004,15(9):1311−1327. http://www.jos.org.cn/1000-9825/15/1311.htm

[8] 丁锐,赵荣彩,韩林.基于主导值的计算和数据划分算法.计算机科学,2012,39(3):290−294.

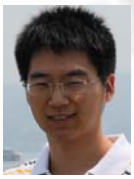
[9] 张为华,王鹏,臧斌宇.一种基于代表元的划分算法.计算机学报,2008,31(3):400−410.

[11] 王轶然,陈莉,冯晓兵,张兆庆.全局部分重复计算划分.计算机研究与发展,2006,43(12):2158−2165.

[17] 赵捷,赵荣彩,丁锐,黄品丰.基于嵌套循环分类的并行识别技术.软件学报,2012,23(10):2695−2704. http://www.jos.org.cn/1000-9825/4178.htm [doi: 10.3724/SP.J.1001.2012.04178]

[21] 钟洪涛,舒继武,温冬婵,郑纬民.基于区域图数据流分析的通信优化算法.软件学报,2003,14(2):175−182. http://www.jos.org.cn/1000-9825/14/175.htm

[22] 陈国良,安虹,陈峻,郑启龙,单久龙.并行算法实践.北京:高等教育出版社,2004.459−486.



丁锐(1984－),男,河南滑县人,博士,讲师,CCF 会员,主要研究领域为并行编译.
E-mail: dr2012earth@gmail.com

赵荣彩(1957－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为体系结构,先进编译.
E-mail: rczhao126@126.com

韩林(1978－),男,博士,副教授,CCF 会员,主要研究领域为并行编译.
E-mail: strollerlin@163.com