ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.12, December 2010, pp.3011−3028
doi: 10.3724/SP.J.1001.2010.03696

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

面向OpenMP和OpenTM应用的并行数据重用理论*

吴俊杰[1+], 杨学军[1], 刘光辉[1], 唐玉华[2]

[1](国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

[2](国防科学技术大学 计算机学院 计算机科学与技术系,湖南 长沙 410073)

Parallel Data Reuse Theories for OpenMP and OpenTM Applications

WU Jun-Jie[1+], YANG Xue-Jun[1], LIU Guang-Hui[1], TANG Yu-Hua[2]

[1](National Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](Department of Computer Science and Technology, School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: junjiewu@nudt.edu.cn

Wu JJ, Yang XJ, Liu GH, Tang YH. Parallel data reuse theories for OpenMP and OpenTM applications. *Journal of Software*, 2010,21(12):3011−3028. http://www.jos.org.cn/1000-9825/3696.htm

Abstract: This paper extends the famous data reuse theory to a parallel domain and proposes parallel data reuse theories for OpenMP and OpenTM applications, respectively. Through studying the relationships between threads and transactions, the parallel data reuse theories systemically discuss how to classify, judge, and compute reuses in parallel programs. Meanwhile, the optimization framework for reducing OpenTM transactions rolled back is studied. Finally, the data reuses in SPEComp2001 benchmarks are analyzed. The parallel data reuse theories can be used to direct the analysis of parallel applications and the research of compiler optimization techniques on multi-core shared memory architecture.

Key words: parallel application; reuse; locality; multi-core; OpenMP; OpenTM

摘 要: 将经典的数据重用理论扩充到并行领域,分别提出了面向OpenMP和OpenTM应用的并行数据重用理论.针对重用在线程、事务中的关系,系统地讨论了并行应用中重用的分类、判定和求解方法.同时,应用这一理论研究了 OpenTM 循环的优化技术,以降低事务被回退的风险.最后,使用并行数据重用理论分析和统计了 SPEComp2001 中的数据重用.并行数据重用理论可以用于指导面向多核存储共享结构的并行程序分析和编译优化技术研究.

关键词: 并行应用;重用;局部性;多核;OpenMP;OpenTM

中图法分类号: TP314 **文献标识码**: A

处理器和存储器之间的速度差异严重制约了处理器性能的发挥,因此,存储墙问题的研究在计算机系统设计中始终占有非常重要的地位.局部性优化技术通过将经常访问的数据放在距离处理器较近的存储层次中(寄

* Supported by the National Natural Science Foundation of China under Grant Nos.60921062, 60873014, 61003082 (国家自然科学基金)

Received 2009-03-31; Accepted 2009-07-07

存器或Cache中),能够有效减少处理器对存储器的直接访问,为计算机系统的性能发挥起到了重要作用.局部性优化技术的基本原理在于程序的访存操作中存在数据重用.Wolf等人曾提出数据重用理论[1],该理论面向循环中的数组重用问题进行了深入研究,给出了重用的分类和求解方法,对于推动局部性优化技术的研究起到了重要作用.到 2009 年 3 月为止,根据ACM数字图书馆的统计,Wolf等人提出的数据重用理论已经被引用超过310 次[2].

另一方面,随着微电子技术的不断发展,芯片上可以集成的晶体管数目越来越多.为了更有效地利用片上的晶体管资源,多核处理器技术逐渐成为学术界和产业界关注的焦点.在多核技术发展的推动下,并行技术开始从原先的高性能计算等高端应用领域开始向普通用户计算机,甚至是桌面计算机迁移,并行技术研究变得更加广泛.然而多核技术并不能解决存储墙问题,特别是对追求执行速度的应用,并行带来的处理速度的提升更加剧了CPU 与主存之间的速度差异.因此,对于多核并行领域,局部性优化技术的研究甚至更加重要.然而,Wolf 等人提出的数据重用理论只面向串行程序,缺少对数据重用并行性质的刻画和描述,不能有效指导当前并行技术背景下的局部性优化技术.

在并行编程领域,OpenMP是一种基于共享主存技术的并行应用编程接口[3],通过在串行程序中添加编译指导语句,OpenMP程序具有编写简洁、执行高效的特点,在并行程序设计中被广泛应用[4−6].多核处理器结构中,不仅主存共享,往往末级Cache结构也是逻辑共享的,因此,OpenMP也非常适合多核环境下的并行程序设计.另一方面,面对传统并行程序设计中的锁问题,一种新的并行技术——事务存储技术正在逐渐兴起[7].事务存储技术兼有传统并行程序中粗粒度锁编程简单和细粒度锁性能高效的优点,备受研究人员关注.Stanford大学的Baek等人扩展了OpenMP,提出了一种面向事务存储的OpenTM应用编程接口[8].OpenTM继承了OpenMP的优点,为开发事务并行、识别关键变量以及指导调度和冲突管理提供了简洁、高级的编程界面,OpenTM程序可以有效映射到硬件事务存储、软件事务存储和混合式事务存储等多种体系结构上.

面向 OpenMP 和 OpenTM 应用,本文将 Wolf 等人提出的串行数据重用理论扩展到并行领域,系统地提出并行数据重用理论.据我们所知,这项工作第一次系统地研究了 OpenMP 和 OpenTM 应用中数据重用的分类和求解理论,为并行环境下的局部性优化技术提供了理论基础.本文的主要贡献包括:

- 提出了基于迭代的并行数据重用理论.基于迭代的并行数据重用理论描述了并行数据重用理论的最基本框架,是面向 OpenMP 和 OpenTM 应用的并行数据重用理论的基础.
- 提出了面向 OpenMP 应用的并行数据重用理论.研究了 OpenMP 应用中的重用分类和重用求解.
- 提出了面向 OpenTM 应用的并行数据重用理论.在研究 OpenTM 应用重用关系的同时,还研究了降低事务回退风险的 OpenTM 程序优化框架.
- 通过我们提出的理论分析,统计了SPEComp2001 基准测试程序[9]中的数据重用.

本文第 1 节简要介绍 Wolf 等人提出的串行数据重用理论.第 2 节研究基于迭代的并行数据重用理论,作为面向 OpenMP 和 OpenTM 的并行数据重用理论研究的基础.第 3 节详细介绍面向 OpenMP 应用的并行数据重用理论.第 4 节介绍面向 OpenTM 应用的并行数据重用理论.第 5 节通过我们提出的并行数据重用理论进行案例分析和对 SPEComp2001 基准测试程序做数据重用统计.最后,我们在第 6 节讨论相关工作,并在第 7 节给出全文的总结.

1 串行数据重用理论概述[1]

在 Wolf 的理论中,数据的重用性和局部性被区别对待.重用性描述的是程序中数据访问的一种固有属性,与考察的存储层次无关;而局部性是程序中数据重用性的一种具体体现.比如,有两次访存操作同时访问了数据 A,我们说数据 A 具有重用性;但可能在 A 的第 2 次访问发生前,它已经被替换出 Cache,那么数据 A 就不具备在 Cache 中的局部性.

串行数据重用理论关注循环中的数组对象.深度为n的循环对应迭代空间Z^n(iteration space)中的一个有穷凸多面体,多面体的边界由循环的边界决定.迭代空间以循环变量为坐标,其中的每一个空间点对应循环中的一

个迭代,表示为迭代向量 $\vec{i}=(i_1,i_2,...,i_n)$,其中,$i_x$是第$x$层循环的循环变量,$x$由 1~$n$取值时从最外层循环向最内层计数.而对于数组,关注的则是具有一致生成访问的数据重用.一致生成访问的定义如下:

定义 1(一致生成访问). 如果循环嵌套的深度为n,数组A的维数为d, $\vec{f}$ 和 $\vec{g}$ 是A的下标函数:$Z^n\to Z^d$.当满足下列等式时,数组A的两次访问 $A[\vec{f}(\vec{i})]$ 和 $A[\vec{g}(\vec{i})]$ 称为一致生成访问(uniformly generated referenced).

$$\vec{f}(\vec{i})=H\vec{i}+\vec{c}_f \text{ 且 } \vec{g}(\vec{i})=H\vec{i}+\vec{c}_g,$$

其中,H 称为访问矩阵(access matrix), $\vec{c}_f$ 和 $\vec{c}_g$ 称为偏移向量(offset vector)或常数向量(constant vector).

容易看出,具有一致生成访问的数据重用才可能在多组迭代间反复出现.因此,对这一类重用的研究更有意义.

Wolf 将面向 Cache 的串行数据重用按照两个维度分成 4 个类别,分别是自时间重用、自空间重用、组时间重用和组空间重用.其中,“自”与“组”的差别在于是否是同一次访问引起的数据重用.如果是,则是自重用,比如循环体中同一条访存语句在两次迭代时访问同一主存地址引起的重用;否则,即为组重用.显然,对组重用的考察,重点聚焦于一致生成访问集,即对同一数组的具有相同访问矩阵 H 的访问集合.如果循环的两次迭代 $\vec{i}_1$ 和 $\vec{i}_2$ 访问了相同的数据单元,那么这次重用的重用向量 $\vec{r}=\vec{i}_1-\vec{i}_2$.重用向量之间满足封闭率、结合率、交换率,有单位元、零元和逆元构成向量空间,记为R^n.

- **自时间重用(self-temporal reuse)**

数组 $A[H\vec{i}+\vec{c}]$ 在两次迭代 $\vec{i}_1$ 和 $\vec{i}_2$ 中访问了相同的数据单元,当且仅当 $H\vec{i}_1+\vec{c}=H\vec{i}_2+\vec{c}$,即

$$H(\vec{i}_1-\vec{i}_2)=H\vec{r}=\vec{0}.$$

我们称数组 $A[H\vec{i}+\vec{c}]$ 在循环中存在自时间重用,自时间重用向量 $\vec{r}$ 为方程 $H\vec{r}=\vec{0}$ 的解.显然,这个方程的解集合为ker$H$,构成重用空间R^n的子空间,称为自时间重用向量空间R_{ST},即R_{ST}=ker$H$.

- **自空间重用(self-spatial reuse)**

根据Wolf的定义,循环的两次迭代中,如果数组 $A[H\vec{i}+\vec{c}]$ 的访问满足除数组最低维外其他各维下标相等,则称这两次访问具有自空间重用.若H_S为将访问矩阵H的最后一行替换为全 0 的矩阵,数组 $A[H\vec{i}+\vec{c}]$ 具有自空间重用的充要条件为 $H_S\vec{i}_1+\vec{c}=H_S\vec{i}_2+\vec{c}$,因此,数组$A$访问的自空间重用向量空间$R_{SS}$=ker$H_S$.从$H_S$和$H$的关系可以看出,ker$H\subseteq$ker$H_S$,自时间重用是自空间重用的一种特例.

- **组时间重用(group-temporal reuse)**

若数组 A 的两次访问 $A[H\vec{i}_1+\vec{c}_1]$ 和 $A[H\vec{i}_2+\vec{c}_2]$ 具有组时间重用,当且仅当 $H\vec{i}_1+\vec{c}_1=H\vec{i}_2+\vec{c}_2$,即 $H(\vec{i}_1-\vec{i}_2)=H\vec{r}=\vec{c}_2-\vec{c}_1$.如果 $\vec{r}_p$ 是方程的一个特解,则方程 $H\vec{r}=\vec{c}_2-\vec{c}_1$ 的通解是 $\ker H+\vec{r}_p$.

一般地,对于一致生成集 $\{A[H\vec{i}+\vec{c}_1],...,A[H\vec{i}+\vec{c}_g]\}$,组时间重用向量空间 $R_{GT}=span\{\vec{r}_2,...,\vec{r}_g\}+\ker H$.其中,$\vec{r}_k$ 是方程 $H\vec{r}=\vec{c}_1-\vec{c}_k$ 的特解,k=2,…,g.

- **组空间重用(group-spatial reuse)**

与自空间重用类似,令H_S是将访问矩阵H的最后一行替换为全 0 的矩阵,则组空间重用向量空间:

$$R_{GS}=span\{\vec{r}_2,...,\vec{r}_g\}+\ker H_S,$$

其中,$\vec{r}_k$ 是方程 $H\vec{r}=\vec{c}_{S,1}-\vec{c}_{S,k}$ 的特解, k=2,…,g, $\vec{c}_{S,i}$ 是将 $\vec{c}_i$ 最后一个元素替换为 0 的向量.

2 基于迭代的并行数据重用理论

并行数据重用理论研究的本质是揭示并行环境下数据重用的特性.一般地,数据重用的两次访问可能位于同一个并行执行体(线程或事务)内,也可能分别处于两个不同的并行执行体.然而,不论并行的执行体是线程还是事务,OpenMP 或 OpenTM 中的调度单位都是并行后迭代(parallelized sub-loop,又称并行后子循环),因此我们首先研究基于迭代的并行数据重用.基于迭代的并行数据重用理论是OpenMP 和OpenTM 并行数据重用理论的基础,它们三者的关系可以从图 1 中看出.基于迭代的并行数据重用理论研究了重用在不同并行后子循环之间的关系:在 OpenMP 程序中,并行后子循环被调度到不同的线程上,因此,面向 OpenMP 应用的并行数据重用理论

研究重用在线程之间的关系;而 OpenTM 程序中,并行后迭代先被调度到事务上,事务又被调度在线程上,因此,面向 OpenTM 应用的并行数据理论还关注重用在事务之间的关系.本节介绍基于迭代的并行数据重用理论,面向 OpenMP 和 OpenTM 的并行数据重用理论将在第 3 节和第 4 节进行介绍.

为了描述清楚基于迭代的并行数据重用理论,我们首先定义连接向量、迭代内/迭代间等概念.

定义 2(连接向量). 如果 $\vec{i}_1$ 和 $\vec{i}_2$ 是循环对应的两个迭代向量,那么 $\vec{l}=\vec{i}_1-\vec{i}_2$ 称为迭代连接向量,简称连接向量(link vector).

显然,重用向量是连接向量的一种特例.连接向量之间构成连接向量空间,而重用向量空间是连接向量空间的一个子空间.

定义 3(迭代内连接向量和迭代间连接向量). 一个循环在第$p_1,p_2,\ldots,p_s$层被并行化,$\vec{l}=\vec{i}_1-\vec{i}_2$ 是这个并行化循环的连接向量.如果 $\vec{i}_1$ 和 $\vec{i}_2$ 中第$p_1,p_2,\ldots,p_s$个分量的值均相等,则 $\vec{l}$ 为并行迭代内连接向量,简称迭代内连接向量(intra-iteration link vector);否则,$\vec{l}$ 为并行迭代间连接向量,简称迭代间连接向量(inter-iteration link vector).

根据定义 3,可以将迭代空间中的迭代连接向量分为两个集合:迭代内连接向量集合L_{AI}和迭代间连接向量集合L_{EI}.相应地,我们定义迭代内重用向量和迭代间重用向量.

定义 4(迭代内重用向量和迭代间重用向量). 一个循环在第$p_1,p_2,\ldots,p_s$层被并行化,$\vec{r}=\vec{i}_1-\vec{i}_2$ 是这个并行化循环中数组A的重用向量.如果 $\vec{i}_1$ 和 $\vec{i}_2$ 中第$p_1,p_2,\ldots,p_s$个分量的值均相等,则 $\vec{r}$ 为数组A的并行迭代内重用向量,简称迭代内重用向量(intra-iteration reuse vector);否则,$\vec{r}$ 为数组A的并行迭代间重用向量,简称迭代间重用向量(inter-iteration reuse vector).

迭代内重用向量和迭代间重用向量分别是迭代内连接向量和迭代间连接向量的特例,并且它们分别构成迭代内重用向量集合R_{AI}和迭代间重用向量集合R_{EI}.

通过迭代内和迭代间重用的划分,基于迭代的并行数据重用被划分成 8 类,在原先串行数据重用理论分类的基础上又扩充一维,如图 2 所示.

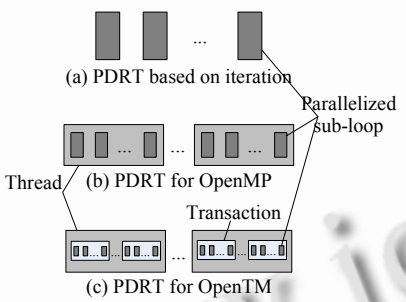


Fig.1 Three parallel data reuse theory (PDRT)

图 1 3 种并行数据重用理论(PDRT)

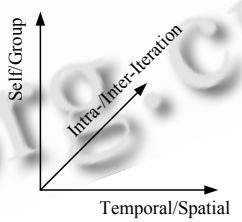


Fig.2 Classifying dimension in the iteration-distinguished parallel data reuse theory

图 2 基于迭代的并行数据重用理论分类维度

下面引入一个例子,详细介绍基于迭代的并行数据重用理论中各种类别数据重用集合的求解办法.

```
for (i1=0; i1<4; i1++)
    for (i2=0; i2<4; i2++) {
        c[i1]+=a[i1][i2]*b[i2];
    }
```

Fig.3 Matrix vector multiplication

图3 矩阵向量乘法

图 3 是 C 语言编写的矩阵向量乘法程序片段,包含两层嵌套循环,迭代向量 $\vec{i}=(i_1,i_2)^T$.如果我们对其最外层循环进行并行化,将得到 4 个并行后的子循环,如图 4 所示.基于迭代的并行数据重用理论考察的关键就在于,重用是其中某一个并行后子循环的迭代内重用还是迭代间重用.由图 4 容易看出,如果 $\vec{r}=\vec{i}_1-\vec{i}_2$ 是图 3 中的一个重用向量,迭代向量 $\vec{i}_1$ 和 $\vec{i}_2$ 中的第 1 个元素决定了 $\vec{r}$ 的前后两次访问是否位于同一个并

行化子循环.一般地,如果一个循环的第p层被并行化,迭代向量$\vec{i}_1$和$\vec{i}_2$属于同一被并行化后的子循环当且仅当它们中的第p个元素相等,即$i_{p,1}=i_{p,2}$(本文中假设循环都以最外层开始并行化,内层循环并行化的情况可以只看并行化开始的内层循环部分).比如,在迭代向量$\vec{i}_1=(0,0)^T$和$\vec{i}_2=(0,1)^T$的两次迭代中,$c[i_1]$访问了相同的元素$c[0]$,因为$\vec{i}_1$和$\vec{i}_2$中对应的第 1 个元素相等,所以这次重用$\vec{r}_c=(0,1)^T$属于迭代内重用.而迭代向量$\vec{i}_1=(0,0)^T$和$\vec{i}_2=(0,1)^T$的两次迭代中,$b[i_2]$访问了相同的元素$b[0]$,$\vec{i}_1$和$\vec{i}_2$中对应的第 1 个元素不等.因此,数组b的重用向量$\vec{r}_b=(1,0)^T$是迭代间重用.为了进一步求解迭代内和迭代间重用,我们定义循环并行化矩阵.

```
for (i2=0; i2<4; i2++) {
    c[0]+=a[0][i2]*b[i2];
}
```
(a) i_1=0

```
for (i2=0; i2<4; i2++) {
    c[1]+=a[1][i2]*b[i2];
}
```
(b) i_1=1

```
for (i2=0; i2<4; i2++) {
    c[2]+=a[2][i2]*b[i2];
}
```
(c) i_1=2

```
for (i2=0; i2<4; i2++) {
    c[3]+=a[3][i2]*b[i2];
}
```
(d) i_1=3

Fig.4 Four sub-loops from parallelizing matrix vector multiplication

图 4 矩阵向量乘法程序并行化后的 4 个子循环

定义 5(循环并行化矩阵). 一个循环的深度为n,当这个循环在第$p_1,p_2,\ldots,p_s$层被并行化时,它对应的循环并行化矩阵LPM(loop-parallelized matrix)是一个$n\times n$的对角方阵,并且其对角线上元素满足下式:

$$LPM(i,i)=\begin{cases}1, & i\in\{p_1,\ldots,p_s\}\\ 0, & \text{otherwise}\end{cases}.$$

迭代向量$\vec{i}_1$和$\vec{i}_2$属于同一个并行后迭代当且仅当它们满足$LPM\vec{i}_1=LPM\vec{i}_2$.假设连接向量$\vec{l}=\vec{i}_1-\vec{i}_2$,则$\vec{l}$为迭代内连接向量的充要条件为$LPM\vec{l}=LPM(\vec{i}_1-\vec{i}_2)=0$,这个方程的解集合为 ker$LPM$.因此,迭代内连接向量集合$L_{AI}$=ker$LPM$,并且$L_{AI}$构成连接向量空间的子空间.

关于迭代内连接向量和迭代间连接向量,存在定理 1(文中所有定理和推论的证明见附录).

定理 1. 给定并行化的循环,一个连接向量只能是迭代内连接向量或迭代间连接向量中的一种,即

$$L_{AI}\cap L_{EI}=\varnothing.$$

根据定理 1,对于给定的重用向量集合R,迭代内重用向量集合$R_{AI}=R\cap L_{AI}$,迭代间重用向量集合$R_{EI}=R-L_{AI}$.从而我们可以得到基于迭代的并行数据重用分类,见表 1.

Table 1 Reuse categories in iteration-distinguished parallel data reuse theory

表 1 基于迭代的并行数据重用分类

	Self-Temporal	Self-Spatial	Group-Temporal	Group-Spatial
Intra-Iteration	$R_{ST}\cap L_{AI}$	$R_{SS}\cap L_{AI}$	$R_{GT}\cap L_{AI}$	$R_{GS}\cap L_{AI}$
Inter-Iteration	$R_{ST}-L_{AI}$	$R_{SS}-L_{AI}$	$R_{GT}-L_{AI}$	$R_{GS}-L_{AI}$

3 面向 OpenMP 应用的并行数据重用理论

如图 1 所示,OpenMP 程序中并行后子循环被调度在不同的线程上.在以往的并行观念中,重用的考察往往与处理器关系结合起来,见表 2.处理器间的时间重用就是真共享,而处理器间非时间重用的空间重用就可能是伪共享.然而实际系统中,处理器调度线程执行,影响线程调度包含调度时处理器的上下文信息,属于动态不确定因素,因此,面向 OpenMP 应用的并行数据重用理论关注重用在线程之间的不同特性.这种由处理器关系向线程关系的转变是一种保守研究,因为并行环境下最特别的处理器间重用也一定是线程间重用.

与定理 1 揭示的重用在并行后子循环中的关系不同,线程内/线程间的重用关系更加复杂.具体地说,一个重用可能是线程内重用,又是线程间重用,图 5 给出了一个例子.图 5(a)是矩阵向量乘法程序的一个 OpenMP 版本,图 5(b)是其迭代在两个线程上的划分结果,深色圆角矩形代表线程占有的迭代.由图中可以看出,$\vec{r}$是关于数组b的一个线程内空间重用向量,$\vec{r}'$是一个线程间重用向量.但是$\vec{r}=(1,2)^T-(0,1)^T=(1,1)^T=(2,2)^T-(1,1)^T=\vec{r}'$,因

此,重用向量$(1,1)^T$既是线程内重用,又是线程间重用.为此,面向OpenMP应用的并行数据重用理论将重用分为 3 大类:纯线程内重用、纯线程间重用和混合线程重用.这一维分类依然与串行数据重用理论中的分类正交,将得到 12 种类别的重用.

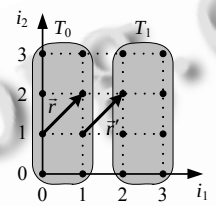
Table 2 Parallel data reuse regarding processor relationship

表 2 考虑处理器关系的并行数据重用

	Self-Temporal	Self-Spatial	Group-Temporal	Group-Spatial
Intra-Processor	Similar to serial data reuse on a single-core platform			
Inter-Processor	True sharing	Inter-Processor self-temporal+ false sharing	True sharing	Inter-Processor group-temporal+ false sharing

```
#pragma omp parallel for default (none)\
        private(i1,i2) shared(a,b,c)\
        schedule(static)
for (i1=0; i1<4; i1++)
      for (i2=0; i2<4; i2++) {
          c[i1]+=a[i1][i2]*b[i2];
      }
```

(a) A matrix vector multiplication program in OpenMP
(a) 矩阵向量乘法程序的一个 OpenMP 版本



(b) Iteration scheduling on threads
(b) 线程上的迭代调度

Fig.5 A hybrid-thread reuse

图 5 混合线程重用

定义 6(纯线程内重用、纯线程间重用和混合线程重用). 给定一个 OpenMP 程序和其中数组 A 的重用向量 $\vec{r}$ 以及确定的调度方式和线程数目:(1) 如果 $\forall \vec{i}_1,\vec{i}_2 \in Z^n : \vec{r}=\vec{i}_1-\vec{i}_2, \vec{i}_1$ 和 $\vec{i}_2$ 都属于同一线程,那么重用 $\vec{r}$ 是数组 A 的纯线程内重用;(2) 如果 $\forall \vec{i}_1,\vec{i}_2 \in Z^n : \vec{r}=\vec{i}_1-\vec{i}_2, \vec{i}_1$ 和 $\vec{i}_2$ 属于不同的线程,那么重用 $\vec{r}$ 是数组 A 的纯线程间重用;(3) 如果 $\exists \vec{i}_1,\vec{i}_2 \in Z^n : \vec{r}=\vec{i}_1-\vec{i}_2, \vec{i}_1$ 和 $\vec{i}_2$ 属于同一线程,并且 $\exists \vec{i}_3,\vec{i}_4 \in Z^n : \vec{r}=\vec{i}_3-\vec{i}_4, \vec{i}_3$ 和 $\vec{i}_4$ 属于不同的线程,那么重用 $\vec{r}$ 是数组 A 的混合线程重用.

根据定义 6,我们可以得到相应的重用向量集合:纯线程内重用向量集合R_{AT}、纯线程间重用向量集合R_{ET}和混合线程重用集合R_{HT}.同理,我们可以定义纯线程内连接向量集合L_{AT}、纯线程间连接向量集合L_{ET}和混合线程连接向量集合L_{HT},这里不再赘述.由于OpenMP程序中以并行后子循环为线程上的调度单位,迭代内连接向量一定是纯线程内连接向量,即$L_{AI}\subseteq L_{AT}$;而迭代间连接向量则可能是任何一种线程关系的连接向量.

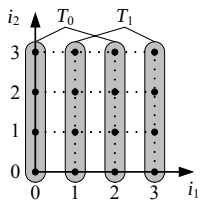
给定一个连接向量,影响其在线程之间关系的因素包括:这个连接向量在并行后子循环之间的关系、线程数目以及迭代在线程上的调度方式.OpenMP 程序的调度子句包含 static,dynamic,guided 和 runtime 等 4 种调度模式和 *chunk* 参数,chunk 的值决定了每个线程上调度的迭代数.Dynamic 和 guided 属于动态调度模式,连接向量在线程之间的关系受到系统动态信息的影响,无法通过静态分析得到.Runtime 方式则是在运行时根据系统的环境变量在 static,dynamic 和 guided 中选择调度方式,没有引入新的调度方法.在动态调度方式中,迭代内连接向量一定是纯线程内连接向量,而迭代间连接向量则可能受循环形式和运行时迭代在线程上调度方式的影响.因此,对于 dynamic,guided 等动态调度模式下的连接向量,只能判断得到迭代内连接向量一定是纯线程内连接向量,而迭代间连接向量是不确定线程关系的连接向量.因此,我们只分析 static 调度模式下 OpenMP 程序中的连接向量求解方法.

3.1 Static调度模式,*chunk*为1

图 6 展示了矩阵向量乘法程序在static,*chunk*=1 模式下的迭代调度.给定迭代向量 $\vec{i}$,这个迭代被分配在线程T_q上当且仅当$i_j \bmod p=q$.其中,i_j是被并行化那层循环的循环变量正规化后的值(正规化使得循环变量下界为0,跨步为 1),p是线程数目,假设线程由 0 开始顺序编号.图 6 中对应的i_j为i_1,p=2.

```
#pragma omp parallel for default (none)\
        private(i1,i2) shared(a,b,c)\
        schedule(static,1)
for (i1=0; i1<8; i1++)
      for (i2=0; i2<4; i2++) {
          c[i1]+=a[i1][i2]*b[i2];
      }
```

(a) Matrix vector multiplication program
(a) 矩阵向量乘法程序



(b) Iteration scheduling on threads
(b) 线程上的迭代调度

Fig.6 Matrix vector multiplication program in static and *chunk*=1 mode
图 6 Static,*chunk*=1 模式下的矩阵向量乘法程序

定理 2. 在 OpenMP 程序 static 且 *chunk*=1 调度模式下,一个连接向量不是纯线程内连接向量,就是纯线程间连接向量,即$L_{HT}=\varnothing$.同时,如果$\vec{l}$是连接向量,下列两式成立:

$$\vec{l}\in L_{AT} \Leftrightarrow \exists n\in \mathbf{Z}: l_j = np,$$

$$\vec{l}\in L_{ET} \Leftrightarrow \neg\exists n\in \mathbf{Z}: l_j = np.$$

其中,p是线程数目,第j层循环被并行化,l_j是连接向量$\vec{l}$的第j个元素.

根据定理 2,在OpenMP程序static且*chunk*=1 的调度模式下,连接向量空间仅包含两个子集合:纯线程内连接向量集合L_{AT}和纯线程间连接向量集合L_{ET},它们满足:

$$L_{AT}\cup L_{ET}=L, L_{AT}\cap L_{ET}=\varnothing.$$

因此,我们只需求解纯线程内连接向量集合L_{AT},然后通过公式$L_{ET}=L-L_{AT}$求解纯线程间连接向量集合L_{ET}.

从定理 2 可以发现,连接向量$\vec{l}\in L_{AT}$当且仅当它满足下式:

$$LPM\vec{l} = (0,...,np,...,0)^T_{e_j=np}, n\in\mathbf{Z} \tag{1}$$

所以,线程内连接向量集合是式(1)所有解集合的并集.方程(1)的通解为$\ker LPM + (0,...,np,...,0)^T_{e_j=np}$,从而有

$$L_{AT}=\bigcup_{n\in\mathbf{Z}}\left[\begin{pmatrix}0\\ \vdots\\ np\\ \vdots\\ 0\end{pmatrix}_{e_j=np} + \ker LPM\right].$$

此外,通过定理 2 可以得到推论 1.

推论 1. OpenMP 程序中 static 调度模式且 *chunk*=1 情况下,线程内连接向量集合构成连接向量空间的一个子空间,且它满足:

$$L_{AT}=span\left\{\begin{pmatrix}0\\ \vdots\\ p\\ \vdots\\ 0\end{pmatrix}_{e_j=p} + \ker LPM\right\} = span\left\{\begin{pmatrix}0\\ \vdots\\ p\\ \vdots\\ 0\end{pmatrix}_{e_j=p}\right\} + \ker LPM.$$

3.2 Static调度模式,*chunk*不为1

图 7 是矩阵向量乘法程序在 static,*chunk*=2 情况下的 OpenMP 代码及其迭代空间在双线程上的调度结果.

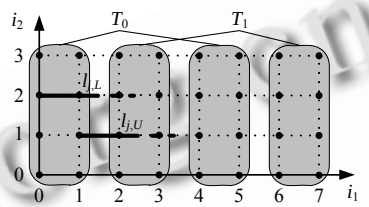
为便于考察迭代在线程上的调度情况,我们将向量长度改为 8.对于迭代向量 $\vec{i}$,设i_j是其中并行化对应的循环变量,线程数目为p,则 $\vec{i}$ 被分配在线程T_q上当且仅当 $\left\lfloor \frac{i_j}{2} \right\rfloor \bmod p = q$.

若$chunk=k$,则 $\vec{i}$ 被分配在线程T_q上当且仅当 $\left\lfloor \frac{i_j}{k} \right\rfloor \bmod p = q$.

从上式可以看出,迭代在线程上的调度方式只与并行化的那层循环变量有关.为此,我们只需关注这个循环变量.记迭代向量中并行化对应的元素为i_j,连接向量中并行化对应的元素为l_j.在下面的研究中,假设循环已经正规化.

```
#pragma omp parallel for default (none)\
        private(i1,i2) shared(a,b,c)\
        schedule(static,2)
for (i1=0; i1<8; i1++)
      for (i2=0; i2<4; i2++) {
          c[i1]+=a[i1][i2]*b[i2];
      }
```

(a) Matrix vector multiplication program
(a) 矩阵向量乘法程序



(b) Iteration scheduling on threads
(b) 线程上的迭代调度

Fig.7 Matrix vector multiplication program in static and $chunk\neq 1$ mode
图 7 Static,$chunk\neq 1$ 模式下的矩阵向量乘法程序

对于i_j和l_j,定义它们的表示方式:

$$|i_j| = (a_1, a_2, a_3)\begin{pmatrix} pk \\ k \\ 1 \end{pmatrix}, |l_j| = (b_1, b_2, b_3)\begin{pmatrix} pk \\ k \\ 1 \end{pmatrix}.$$

其中,p为线程数目,k为$chunk$的值,系数$a_1,a_2,a_3,b_1,b_2,b_3\in\square$,且$a_1\geqslant 0, 0\leqslant a_2<p, 0\leqslant a_3<k, b_1\geqslant 0, 0\leqslant b_2<p, 0\leqslant b_3<k$.

关于连接向量的线程关系,我们只需关注两种类型的迭代点:一种是处于块(chunk)下界的迭代点,比如图7(b)中迭代向量$(i_1,i_2)^T=(0,2)^T$对应的迭代点;另一种是处于块上界的迭代点,比如迭代向量$(i_1,i_2)^T=(1,1)^T$对应的迭代点.容易发现,块下界迭代点的向量表示中$a_3=0$,块上界迭代点的向量表示中$a_3=k-1$.定理 3 揭示了连接向量分别以块的下界和上界为起点时对其线程关系的影响.

定理 3. 在 OpenMP 程序 static 调度模式下,给定 $chunk=k$ 的块,连接向量 $\vec{l}$ 的起点分别连接块的上界和下界:(1) 该连接向量的终点指向的两个迭代点属于同一个块,当且仅当该连接向量并行化分量l_j在向量表示法中$b_3=0$,即$|l_j|=b_1pk+b_2k$;(2) 若连接向量 $\vec{l}$ 的终点指向的两个迭代点属于不同的块,则这两个不同的块必定相邻.

根据定理 3,我们可以得到推论 2 和推论 3.

推论 2. 在 OpenMP 程序 static 且 $chunk=k\neq 1$ 的调度模式下,连接向量 $\vec{l}\in L_{AT}$ 当且仅当对应的并行化分量$|l_j|=b_1pk$.

推论 3. 在 OpenMP 程序 static 且 $chunk=k\neq 1$ 的调度模式下,连接向量 $\vec{l}\in L_{HT}$ 当且仅当对应的并行化分量$|l_j|=b_1pk+b_3$或$|l_j|=b_1pk+(p-1)k+b_3$.其中,$0<b_3<k$.

由推论 2 和推论 3 可以得出,在 static,$chunk$ 不为 1 的 OpenMP 程序中:

$$L_{AT} = span\left\{\begin{pmatrix} 0 \\ \vdots \\ kp \\ \vdots \\ 0 \end{pmatrix}_{e_j=kp} + \ker LPM\right\} = span\left\{\begin{pmatrix} 0 \\ \vdots \\ kp \\ \vdots \\ 0 \end{pmatrix}_{e_j=kp}\right\} + \ker LPM,$$

$$L_{HT}=\bigcup_{\substack{0<m<k,m\in\mathbf{Z}\\ n\in\{0,1\}}}\left\{\begin{pmatrix}0\\ \vdots\\ m+nk(p-1)\\ \vdots\\ 0\end{pmatrix}+span\left\{\begin{pmatrix}0\\ \vdots\\ kp\\ \vdots\\ 0\end{pmatrix}_{e_j=kp}+\ker LPM\right\}\right\}.$$

从而,再使用公式$L_{ET}=L-L_{AT}-L_{HT}$得出纯线程间连接向量集合.

4 面向 OpenTM 应用的并行数据重用理论

事务存储(transactional memory)是借鉴了数据库中事务思想的一种并行执行模型.通过事务的原子执行,事务存储可以避免锁的使用,同时获得粗粒度锁编程的简单性和细粒度锁程序的性能.在事务提交前,事务通过对比自己与其他事务的读写集合完成冲突检测.当发生事务冲突时,冲突事务被回退(rollback),系统状态回到事务没有执行的时刻[7].OpenTM扩展了OpenMP应用编程接口,使用编译指导语句完成支持事务的并行编程.对于了解OpenMP编程的用户来说,OpenTM提供了熟悉、简洁的编程接口,统一了并行和事务存储的表述,为编译器的优化提供了支持,实现高性能和可扩展性[8].

4.1 OpenTM应用中的重用分类和重用求解

与面向 OpenMP 的程序类似,在 OpenTM 程序中,存在 3 种不同类型的事务关系重用:纯事务内重用、纯事务间重用和混合事务重用.与 OpenMP 程序不同,重用不仅影响 OpenTM 程序的局部性性能,还可能影响事务的执行过程.因为含有写操作的时间重用实际是一种数据依赖(dependency),而事务之间的数据依赖可能导致事务的回退.也就是说,事务之间含有写操作的时间重用在执行时必须满足一定的序关系,否则可能引起事务回退.为此,我们除了定义纯事务内、纯事务间和混合重用外,还定义安全重用和危险重用的概念.

定义 7(纯事务内重用、纯事务间重用和混合事务重用). 给定一个 OpenTM 程序和其中数组 A 的重用向量 $\vec{r}$ 以及确定的事务调度方式:

(1) 如果 $\forall\vec{i}_1,\vec{i}_2\in Z^n:\vec{r}=\vec{i}_1-\vec{i}_2,\vec{i}_1$ 和 $\vec{i}_2$ 都属于同一事务,那么重用 $\vec{r}$ 是数组 A 的纯事务内重用;

(2) 如果 $\forall\vec{i}_1,\vec{i}_2\in Z^n:\vec{r}=\vec{i}_1-\vec{i}_2,\vec{i}_1$ 和 $\vec{i}_2$ 属于不同的事务,那么重用 $\vec{r}$ 是数组 A 的纯事务间重用;

(3) 如果 $\exists\vec{i}_1,\vec{i}_2\in Z^n:\vec{r}=\vec{i}_1-\vec{i}_2,\vec{i}_1$ 和 $\vec{i}_2$ 属于同一事务,并且 $\exists\vec{i}_3,\vec{i}_4\in Z^n:\vec{r}=\vec{i}_3-\vec{i}_4,\vec{i}_3$ 和 $\vec{i}_4$ 属于不同的事务,那么重用 $\vec{r}$ 是数组 A 的混合事务重用.

定义 8(安全重用和危险重用). 给定一个 OpenTM 程序和其中数组 A 的重用向量 $\vec{r}$ 以及确定的事务调度方式,如果重用向量 $\vec{r}$ 是数组 A 的纯事务间时间重用或混合事务时间重用,并且 $\vec{r}$ 的两次访问中包含写操作,则重用 $\vec{r}$ 为数组 A 的危险重用;否则,$\vec{r}$ 为数组 A 的安全重用.

同理,我们可以定义OpenTM程序中相应的连接向量以及相应的向量集合:纯事务内连接向量集合L_{AX}、纯事务间连接向量集合L_{EX}、混合事务连接向量集合L_{HX}、安全连接向量集合L_{SX}、危险连接向量集合L_{DX}等.这里的下标"X"代表事务(transaction),本文不再赘述.

从定义 8 可以看出,危险重用是跨越了多个事务并包含写操作的重用,可能导致事务回退;而安全重用一定不会引起事务的回退.我们首先介绍不同类别重用集合的求解,然后,在第 4.2 节中介绍如何优化 OpenTM 程序,以尽可能减少程序中的危险重用,降低程序被回退的风险.

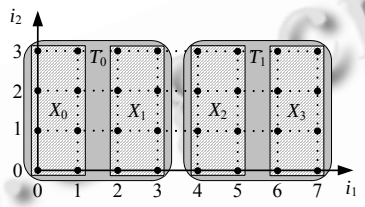
OpenTM 通过扩展 OpenMP 应用编程接口实现对事务的支持.比如,OpenTM 程序中使用 transfor 结构继承 OpenMP 程序中的 for 结构,完成 for 循环上对事务的支持.原本带有依赖的 for 循环在 OpenMP 程序中不能被并行化,因为迭代的不同执行顺序可能导致不一致的程序结果.然而,在面向事务存储的 OpenTM 程序中,用户可以通过 ordered 语句指定事务的提交顺序,即使某些事务"违反"依赖关系前瞻执行了,在提交时通过冲突检测机制也能够回退该事务,保证程序运行结果的正确性.如果不使用 ordered 子句,transfor 将产生无序事务,冲突检测机制不维护事务提交顺序.Transfor 结构的 schedule 子句中最多包括 3 个参数.与 OpenMP 中的 for 结构类似,第 1

个参数指示迭代在线程上的调度方式,可以为 static,dynamic,guided 或 runtime;第 2 个参数是线程上调度的迭代数目(chunk size);最后一个参数是单位事务上分配的并行后子循环数,记为 IPX(iterations per transaction).与 OpenMP 程序线程意义上的块(chunk)对应,调度在每个事务上迭代构成的集合称为事务块(transactional chunk).迭代、事务和线程之间的关系如图 1(c)所示,迭代被调度在事务上,事务被调度在线程上.

图 8 展示了 transfor 结构的 OpenTM 程序,其中,每个事务包含两个并行后子循环,每个线程上包含两个事务.对比传统的 OpenMP 程序容易发现,OpenTM 程序中事务的地位类似 OpenMP 中的线程.不同的是,事务打包成“组”再被调度到 OpenTM 程序中的线程上.为此,我们容易扩展 OpenMP 程序中的很多结论得到重用在 OpenTM 程序中的关系.

```
#pragma omp transfor\
        schedule(static,4,2)
for (i1=0; i1<8; i1++)
      for (i2=0; i2<4; i2++) {
          c[i1]+=a[i1][i2]*b[i2];
      }
```

(a) Matrix vector multiplication program
(a) 矩阵向量乘法程序



(b) Iteration scheduling on iterations and threads
(b) 事务(*X*)和线程(*T*)上的迭代调度

Fig.8 Matrix vector multiplication program in OpenTM
图 8 OpenTM 编程的矩阵向量乘法程序

定理 4. OpenTM 程序中 static 调度模式且单位事务包含并行后子循环数 *IPX* 为 1 时,一个连接向量不是纯事务内连接向量,就是纯事务间连接向量.即混合事务连接向量集合$L_{HX}=\varnothing$.同时,如果 $\vec{l}$ 是连接向量,下列两式成立:

$$\vec{l} \in L_{AX} \Leftrightarrow \exists n \in \mathbf{Z} : l_j = np,$$

$$\vec{l} \in L_{EX} \Leftrightarrow \neg\exists n \in \mathbf{Z} : l_j = np,$$

其中,p是事务数目,第j层循环被并行化,l_j是连接向量 $\vec{l}$ 的第j个元素.

定理 5. OpenTM 程序中 static 调度模式,给定 $IPX=k$ 的事务块,连接向量 $\vec{l}$ 的起点分别连接事务块的上界和下界:

(1) 该连接向量的终点指向的两个迭代点属于同一个事务块,当且仅当该连接向量并行化分量l_j在向量表示法中$b_3=0$,即$|l_j|=b_1pk+b_2k$.其中,p为系统中事务的数目,k为事务块的大小;

(2) 若连接向量 $\vec{l}$ 的终点指向的两个迭代点属于不同的事务块,则这两个不同的事务块必定相邻.

另外,给出定理 5 的如下两个推论:

推论 4. OpenTM程序static,$IPX=k\neq1$ 的情况下,连接向量 $\vec{l} \in L_{AX}$ 当且仅当对应的并行化分量$|l_j|=b_1pk$.

推论 5. OpenTM程序static,$IPX=k\neq1$ 的情况下,连接向量 $\vec{l} \in L_{HX}$ 当且仅当对应的并行化分量$|l_j|=b_1pk+b_3$或$|l_j|=b_1pk+(p-1)k+b_3$.其中,$0<b_3<k$.

根据定理 4,我们可以得到 static,$IPX=1$ 时的纯事务内连接向量集合:

$$L_{AX} = span\left\{\begin{pmatrix} 0 \\ \vdots \\ p \\ \vdots \\ 0 \end{pmatrix}_{e_j=p}\right\} + \ker LPM.$$

根据推论 4 和推论 5,我们可以得到 static,$IPX=k\neq 1$ 时的纯事务内连接向量集合和混合事务连接向量集合:

$$L_{AX}=span\left\{\begin{pmatrix}0\\ \vdots\\ kp\\ \vdots\\ 0\end{pmatrix}_{e_j=p}\right\}+\ker LPM,\ L_{HX}=\bigcup_{\substack{0<m<k,m\in\mathbf{Z}\\ n\in\{0,1\}}}\left(\begin{pmatrix}0\\ \vdots\\ m+nk(p-1)\\ \vdots\\ 0\end{pmatrix}+span\left\{\begin{pmatrix}0\\ \vdots\\ kp\\ \vdots\\ 0\end{pmatrix}+\ker LPM\right\}\right).$$

然后根据定义 8,通过区分循环中的读写操作,我们很容易得到 OpenTM 程序的安全连接向量集合和危险连接向量集合.再通过连接向量集合和重用集合的交集与差集,我们可以得到对应的重用向量集合.其中,安全重用向量一定不会引起事务回退,而危险重用向量可能导致事务回退.

4.2 优化OpenTM循环以减少事务回退

这里,我们研究如何通过循环变换将 OpenTM 程序中的危险重用转换为安全重用.从上文的研究中可以看出,OpenTM 程序中线程的调度方式以及事务块的大小对重用向量在事务间的关系有着重要影响.简单起见,采用保守的优化策略研究如何通过循环交换(loop interchange)、循环倾斜(loop skewing)、循环反转(loop reversing)等循环变换将危险重用向量转换为迭代内重用向量.

在向量空间的循环表示中,循环交换、循环倾斜和循环反转等循环变换等价于幺正矩阵变换,幺正矩阵 U 是满足下列数学性质的矩阵:

- 幺正矩阵是方阵,且其中的元素都是整数;
- 幺正矩阵的行列式的绝对值等于 1;
- 两个幺正矩阵的乘积仍然是幺正矩阵;
- 幺正矩阵的逆也是幺正矩阵.

比如,两重嵌套循环上的循环交换等价于一个 2×2 的幺正矩阵 $U=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$.

给定循环和其中的某个连接向量 $\vec{l}=\vec{i}_1-\vec{i}_2$,如果使用循环变换 U 对该循环进行变换,则变换后迭代向量 $\vec{i}_1$ 和 $\vec{i}_2$ 分别变为 $U\vec{i}_1$ 和 $U\vec{i}_2$,因此,连接向量 $\vec{l}$ 变为 $U\vec{l}$.变换前,连接向量 $\vec{l}$ 是迭代内连接向量当且仅当 $LPM\vec{l}=0$,迭代内连接向量空间为 $\ker LPM$;变换后,连接向量 $U\vec{l}$ 是迭代内连接向量当且仅当 $LPM(U\vec{l})=0$.向量乘法满足结合律,因此 $(LPM\cdot U)\vec{l}=0,\vec{l}$ 的解集合为 $\ker(LPM\cdot U)$.

假设重用向量 $\vec{r}$ 是某OpenTM循环的危险连接向量,循环优化的目标是求解循环变换U,使得 $\vec{r}\in\ker(LPM\cdot U)$.即,$\vec{r}$ 是变换后循环的迭代内重用向量($\vec{r}\in R_{AI}$).由于$R_{AI}\subseteq R_{AX}\subseteq R_{SX}$,因此 $\vec{r}\in R_{SX}$.循环变换后,重用 $\vec{r}$ 一定不会引起事务回退.在第 5 节中,我们将通过一个具体例子解释OpenTM程序的优化过程.

5 实验分析与统计

本节使用并行数据重用理论分析和统计实际程序中的重用情况.第 5.2 节介绍面向 OpenMP 和 OpenTM 程序的并行数据重用理论的具体案例,第 5.3 节分析和统计 SPEComp2001 基准测试程序中的并行数据重用.

5.1 基准测试程序

通过在串行程序中增加编译指导语句,OpenMP应用编程接口简洁、高效,被广泛应用.在OpenMP的案例分析和统计中,我们使用SPEComp2001 中的基准测试程序:312.swim_m,314.mgrid_m,320.equake_m,324.apsi_m和330.art_m,具体描述见表 3 [9].然而另一方面,OpenTM是 2007 年由Stanford大学研究组提出的事务编程接口,目前还没有OpenTM的基准测试程序.在第 5.2 节中,我们构造一个简单的OpenTM循环说明面向事务并行数据重用理论的应用.

Table 3 Data reuse in SPEComp2001 benchmarks

表 3 SPEComp2001 基准测试程序中的并行数据重用

Benchmark	Remarks	Language	OMP loop nests Num.	Intra-Iteration reuse loops Num.	Inter-Iteration reuse loops Num.
312.swim_m	Shallow water modeling	Fortran	8	8	7
314.mgrid_m	Multi-Grid solver in 3D potential field	Fortran	12	11	6
320.equake_m	Finite element simulation; earthquake modeling	C	11	10	8
324.apsi_m	Solves problems regarding temperature, wind, velocity and distribution of pollutants	Fortran	28	17	16
330.art_m	Neural network simulation; adaptive resonance theory	C	5	3	4

5.2 案例分析

5.2.1 面向 OpenMP 的并行数据重用理论

图 9 展示了测试程序 312.swim_m 中的一段 OpenMP 循环.循环体内共有 4 次对数组 PSI 的访问,这 4 次访问的下标向量表示形式为

$$PSI(I+1,J+1):\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}J\\ I\end{pmatrix}+\begin{pmatrix}1\\ 1\end{pmatrix},\ PSI(I+1,J):\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}J\\ I\end{pmatrix}+\begin{pmatrix}0\\ 1\end{pmatrix},\ PSI(I,J+1):\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}J\\ I\end{pmatrix}+\begin{pmatrix}1\\ 0\end{pmatrix}.$$

注意,图 9 中的程序使用 Fortran 语言编写,数组元素按照列优先方式存储.显然,数组 4 次访问对应的访问矩阵相同,属于一致生成访问.下面分别讨论第 1 个 $PSI(I+1,J+1)$访问的自重用以及 $PSI(I+1,J+1)$和 $PSI(I+1,J)$之间的组重用.

对于$PSI(I+1,J+1)$来说, $H=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$, $H_S=\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}$.因此,$\ker H=span\{(0,0)^T\}$,$\ker H_S=span\{(0,1)^T\}$.即$PSI(I+1,J+1)$没有自空间重用,但存在时间重用.另外,图 9 的最外层循环被并行化, $LPM=\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}$, $L_{AI}=\ker LPM=span\{(0,1)^T\}$.从而,$R_{AI}=R\cap L_{AI}=span\{(0,1)^T\}$, $R_{EI}=R-L_{AI}=\varnothing$.即,$PSI(I+1,J+1)$包含的自空间重用属于迭代内重用.又因为$L_{AI}\subseteq L_{AT}$,$PSI(I+1,J+1)$包含的是纯线程内自空间重用,不会引起数据在Cache块中的伪共享.

```
!$OMP PARALLEL DO
      DO 60 J=1,N
      DO 60 I=1,M
      U(I+1,J)=-(PSI(I+1,J+1)-PSI(I+1,J))/DY
      V(I,J+1)=(PSI(I+1,J+1)-PSI(I,J+1))/DX
60    CONTINUE
```

Fig.9 A loop from 312.swim_m

图9 312.swim_m中的一段循环

对于$PSI(I+1,J+1)$和$PSI(I+1,J)$,根据数据重用理论可知,它们之间的组时间重用向量空间$R_{GT}=span\{(1,0)^T\}$.而$L_{AI}=span\{(0,1)^T\}$,因此,这些重用属于迭代间重用.在图 9 所示的OpenMP程序中,循环并没有指定调度方式.又因为组重用向量空间R_{GT}以 $1,0)^T$为基,所以,$PSI(I+1,J+1)$和$PSI(I+1,J)$之间的组重用一定包含线程间重用,可能包含线程内重用.

5.2.2 面向 OpenTM 的并行数据重用理论

本节讨论通过面向 OpenTM 的并行数据重用理论分析事务程序中的重用以及优化 OpenTM 程序,从而减少事务回退的风险.由于目前尚没有标准的 OpenTM 基准测试程序,我们构造一个简单的例子进行介绍.

图 10 是 OpenTM 程序的一段循环,其中,数组 a 的自增包含写操作.下面我们运用面向 OpenTM 的并行数据重用理论分析数组 a 在事务之间的重用关系.

数组a的访问矩阵$H=(0\ \ 1)$,因此根据第 2 节可知,数组a的自时间重用向量空间$R_{ST}=\ker H=span\{(1,0)^T\}$;同时,循环的并行化访问矩阵 $LPM=\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}$.因此,循环的迭代内连接向量$L_{AI}=\ker LPM=span\{(0,1)^T\}$,数组$a$自时间

重用向量集合中的所有元素均属于迭代间重用向量(向量$(0,0)^T$连接同一个访问,在重用理论中不考虑).根据推论 4 可知,数组a的纯事务内连接向量集合$L_{AX}=span\{(N_1,0)^T\}+span\{(0,1)^T\}$.

自时间重用向量集合R_{GT}的子集合$span\{(N_1,0)^T\}$中的元素一定是纯事务内重用向量;而另一个子集合$span\{(0,1)^T\}-span\{(N_1,0)^T\}=\{(x,0)|x \bmod N_1\neq 0\}$中的元素是纯事务间重用向量,并且属于危险重用,这些重用可能导致事务之间的冲突,造成事务回退.

下面我们对图 10 中的循环进行优化,目的是将危险重用转换为安全重用,从而避免事务回退的风险.根据第 4 节,问题归结为:

问题:求解二维幺正矩阵U,使得$R_{ST}\subseteq\ker(LPM\cdot U)$.

重用向量$(1,0)^T$是张成自时间重用向量空间R_{ST}的基,且$\ker(LPM\cdot U)$是封闭的向量空间.

因此,$R_{ST}\subseteq\ker(LPM\cdot U)$当且仅当$(1,0)^T\in\ker(LPM\cdot U)$,即 $(LPM\cdot U)(1,0)^T=\vec{0}$.

设 $U=\begin{pmatrix}u_1 & u_2\\ u_3 & u_4\end{pmatrix}$,则$u_i(i=1,2,3,4)$满足:

1) $|u_1u_4-u_2u_3|=1$;

2) $u_1,u_2,u_3,u_4\in$□;

3) $\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}u_1 & u_2\\ u_3 & u_4\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix}=\begin{pmatrix}0\\ 0\end{pmatrix}$.

容易发现,$U=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$是上述问题的一个解.而$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$对应循环交换变换,也就是说,循环交换能够将图 10 中的危险重用转换为安全重用.

根据文献[10]中循环交换的合法性定理,图 10 中循环可以进行循环交换.因此,优化后的循环如图 11 所示.容易验证,图 11 中数组a的自时间重用向量空间$R_{ST}=span\{(0,1)^T\}$,且数组a的所有自时间重用均为纯事务内重用,属于安全重用,不会引起事务回退.

```
#pragma omp transfor\
        schedule(static,8,1)
for (i1=0; i1<N1; i1++)
      for (i2=0; i2<N2; i2++) {
          a[i2]++;
      }
```

Fig.10 An OpenTM loop nest for optimization

图 10 待优化的 OpenTM 循环

```
#pragma omp transfor\
        schedule(static,8,1)
for (i2=0; i2<N2; i2++)
      for (i1=0; i1<N1; i1++) {
          a[i2]++;
      }
```

Fig.11 OpenTM loop nest optimized

图 11 优化后的 OpenTM 循环

5.3 SPEComp2001基准测试程序中的重用统计

表 3 使用并行数据重用理论分析了 SPEComp2001 中的 5 个 OpenMP 程序.由于重用的线程关系受到运行时信息的影响,这里我们主要分析和统计重用在并行后子循环之间关系的属性.表 3 分析的 5 个程序中,320.equake_m 和 330.art_m 两个程序使用 C 语言编写,而其他 3 个程序使用 Fortran 语言编写.表 3 中统计了每个程序中 OpenMP 循环的数目、包含迭代内重用的循环数目以及包含迭代间重用的循环数目.可以看出,大多数的 OpenMP 循环既包含迭代内重用,又包含迭代间重用.这是因为程序中的大多数 OpenMP 循环都包含对多个变量的访问操作,导致循环中重用关系多且复杂.比如,第 5.2.1 节 312.swim_m 的循环,$PSI(I+1, J+1)$的自重用是迭代内重用,而 $PSI(I+1,J+1)$和 $PSI(I+1,J)$之间的组重用属于迭代间重用.

6 相关工作

并行数据重用理论系统地研究了并行应用下数据重用的分类和求解方法.本节从重用研究、相关性分析理论和编译变换技术 3 个方面讨论相关研究工作.

- **数据重用研究**

1991 年,Wolf等人首次提出面向串行程序的数据重用理论[1],这一理论对局部性优化技术的很多领域都产生了重要影响,包括编译优化技术[10−16]、Cache体系结构技术[17−21]等多个研究领域.截至 2009 年 3 月,根据ACM数字图书馆的统计,串行数据重用理论的论文已经被引用超过 310 次[2].在并行应用中,Kandemir等人曾提出基于并行化向量判定伪共享数据的方法[22],与基于迭代的数据重用理论中使用循环并行化矩阵的方法类似.但是,Kandemir等人的方法只是简单地研究了重用在并行化循环之间的关系,没有研究并行程序中重用的分类和求解,更没有研究具体应用中的数据重用.本文提出的并行数据重用理论第一次系统地将串行数据重用理论扩充至并行领域,研究了OpenMP和OpenTM应用中的数据重用.

- **相关性分析理论**

Lamport等人最早开发和描述了相关性分析理论[23].到目前为止,相关性分析理论是现代编译技术中的重要理论.数据相关(dependency,又译为依赖)也是一种重用,因此,相关性分析理论与重用理论存在一些共同点.然而,一方面,依赖关系只是重用关系的一种特例,即包含写操作的时间重用,重用理论研究的内容比相关性分析理论广;另一方面,相关性分析理论只关注重用的两次数据访问发生的时间顺序,而重用理论更多地关注重用的条件、重用的类别等各个方面.在实际的程序优化过程中,相关性分析理论往往和数据重用理论紧密结合,比如第 4 节讨论的OpenTM程序的优化,通过面向OpenTM程序的并行数据重用理论确定循环变换U之后,要运用相关性分析理论验证该变换的合法性,然后才能应用相应的循环变换.

- **编译变换技术**

编译变换技术包括循环变换、数据变换以及组合变换等,编译变换能够在不增加程序员负担的情况下完成对程序的优化,从而将程序并行化、向量化,或者提高程序中的数据局部性[10,11].本文第 5 节中讨论了并行数据重用理论与循环交换结合减少事务回退风险的例子.事实上,本文介绍的并行数据重用理论在实际应用中往往会和各种编译优化技术结合,可以是循环变换类的循环分块、循环交换、软件流水等,也可以是数据变换,甚至是组合变换.可以说,通过更精细地刻画并行程序中的数据重用,并行数据重用理论为编译器变换优化程序的局部性进一步提供了理论基础和手段.

7 结 论

在并行计算机系统中,存储墙问题依然是系统的性能瓶颈,局部性优化技术仍然十分重要.本文将经典的数据重用理论扩展至并行领域,进一步刻画了数据重用在并行环境下的特点和求解方法,提出面向 OpenMP 和面向 OpenTM 的并行数据重用理论.文中先研究了基于迭代的并行数据重用理论,再分别研究了 OpenMP 和OpenTM 应用中重用的分类、判定和求解方法.最后,文中分析和统计了 SPEComp2001 基准测试程序中的数据重用.并行数据重用模型研究了重用与处理器间的关系,可以用于指导面向多核存储共享结构的并行程序分析和编译优化技术研究.

References:

[1] Wolf ME, Lam MS. A data locality optimizing algorithm. In: Proc. of the ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation (PLDI'91). New York: ACM, 1991. 30−44. http://portal.acm.org/citation.cfm?id=113449

[2] The ACM Digital Library. A data locality optimizing algorithm. Association for Computing Machinery, 2009-3-28. http://portal.acm.org

[3] OpenMP Architecture Review Board. OpenMP application program interface version 3.0. 2008. http://www.openmp.org

[4] Ayguadé E, Copty N, Duran A, Hoeflinger J, Lin Y, Massaioli F, Teruel X, Unnikrishnan P, Zhang G. The design of OpenMP tasks. IEEE Trans. on Parallel Distrib. System, 2009,20(3):404−418. [doi: 10.1109/TPDS.2008.105]

[5] Jeun W, Kee Y, Ha S, Kee C. Overcoming performance bottlenecks in using OpenMP on SMP clusters. Parallel Computer, 2008, 34(10):570−592. [doi: 10.1016/j.parco.2008.06.002]

[6] Chen Y, Shu J, Li J, Wang D. Static analysis of OpenMP directive nesting types and its application. Journal of Software, 2005,16(2):184−204 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/184.htm [doi: 10.1360/jos160184]

[7] Herlihy M, Moss JE. Transactional memory: Architectural support for lock-free data structures. SIGARCH Computer Architecture News, 1993,21(2):289−300. [doi: 10.1145/173682.165164]

[8] Baek W, Minh CC, Trautmann M, Kozyrakis C, Olukotun K. The OpenTM transactional application programming interface. In: Proc. of the 16th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2007). Washington: IEEE Computer Society, 2007. 376−387.

[9] Aslot V, Domeika MJ, Eigenmann R, Gaertner G, Jones WB, Parady B. SPEComp: A new benchmark suite for measuring parallel computer performance. In: Proc. of the Int'l Workshop on OpenMP Applications and Tools (WOMPAT 2001). London: Springer-Verlag, 2001. 1−10.

[10] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures. San Francisco: Morgan Kaufmann Publishers, 2002. 213−217.

[11] McKinley KS, Carr S, Tseng CW. Improving data locality with loop transformations. ACM Trans. on Programming Language System, 1996,18(4):424−453. [doi: 10.1145/233561.233564]

[12] Kandemir M, Banerjee P, Choudhary A, Ramanujam J, Ayguadé E. Static and dynamic locality optimizations using integer linear programming. IEEE Trans. on Parallel Distributed System, 2001,12(9):922−941. [doi: 10.1109/TPDS.2001.1184186]

[13] Loechner V, Meister B, Clauss P. Precise data locality optimization of nested loops. Journal of Supercomputing, 2002,21(1):37−76. [doi: 10.1023/A:1013535431127]

[14] Bondhugula U, Hartono A, Ramanujam J, Sadayappan P. A practical automatic polyhedral parallelizer and locality optimizer. SIGPLAN Notices, 2008,43(6):101−113.

[15] Xia J, Yang X, Zeng L, Zhou H. A projection-delamination based approach to optimizing spatial locality in loop nests. Chinese Journal of Computers, 2003,26(5):539−551 (in Chinese with English abstract).

[16] Zeng L, Yang X, Xia J, Chen J. Improving data locality and reducing false-sharing based on data fusion. Chinese Journal of Computers, 2004,27(1):32−41 (in Chinese with English abstract).

[17] Vera X, Lisper B, Xue J. Data cache locking for tight timing calculations. ACM Trans. on Embedded Computing System, 2007,7(1): 1−38.

[18] McFarling S. Cache replacement with dynamic exclusion. In: Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92). New York: ACM, 1992. 191−200. http://portal.acm.org/citation.cfm?id=139727

[19] Chen TF. An effective programmable prefetch engine for on-chip caches. In: Proc. of the 28th Annual Int'l Symp. on Microarchitecture (MICRO 28). Los Alamitos: IEEE Computer Society Press, 1995. 237−242. http://portal.acm.org/citation.cfm?id=225198

[20] Petrov P, Orailğlu A. Towards effective embedded processors in codesigns: Customizable partitioned caches. In: Proc. of the 9th Int'l Symp. on Hardware/Software Codesign (CODES 2001). New York: ACM, 2001. 79−84. http://portal.acm.org/citation.cfm?id=371687

[21] Wu J, Yang X, Zeng K, Zhang B, Feng Q, Liu G, Tang Y. DOOC: A software/hardware co-managed cache architecture for reducing cache thrashing. Journal of Computer Research and Development, 2008,45(12):2020−2032 (in Chinese with English abstract).

[22] Kandemir M, Choudhary A, Ramanujam J, Banerjee P. Reducing false sharing and improving spatial locality in a unified compilation framework. IEEE Trans. on Parallel Distributed System, 2003,14(4):337−354. [doi: 10.1109/TPDS.2003.1195407]

[23] Lamport L. The parallel execution of DO loops. Communications of the ACM, 1974,17(2):83−93. [doi: 10.1145/360827.360844]

附中文参考文献:

[6] 陈永健,舒继武,李建江,王鼎兴.OpenMP 指导语句全局嵌套类型的静态分析及应用.软件学报,2005,16(2):194−204. http://www.jos.org.cn/1000-9825/16/184.htm [doi: 10.1360/jos160184]

[15] 夏军,杨学军,曾丽芳,周海芳.基于投影分层技术的嵌套循环空间局部性优化方法.计算机学报,2003,26(5):539−551.

[16] 曾丽芳,杨学军,夏军,陈娟.一种利用数据融合来提高局部性和减少伪共享的方法.计算机学报,2004,27(1):32−41.

[21] 吴俊杰,杨学军,曾坤,张百达,冯权友,刘光辉,唐玉华.DOOC:一种能够有效消除抖动的软硬件合作管理 Cache.计算机研究与发展,2008,45(12):2020−2032.

附录

证明(定理 1):反证法.假设存在连接向量 $\vec{l}$ 同时满足 $\vec{l} \in L_{AI}$ 和 $\vec{l} \in L_{EI}$.

根据假设,存在 $\vec{i}_1,\vec{i}_2,\vec{i}_3$ 和 $\vec{i}_4$ 使得 $\vec{l}=\vec{i}_1-\vec{i}_2=\vec{i}_3-\vec{i}_4$.其中,$\vec{i}_1$ 和 $\vec{i}_2$ 属于同一个并行后子循环,$\vec{i}_3$ 和 $\vec{i}_4$ 属于不同的并行后子循环.

如果 LPM 是这个并行化循环对应的循环并行化矩阵,那么 $\vec{i}_1$ 和 $\vec{i}_2$ 属于同一个并行后子循环当且仅当 $LPM\vec{i}_1=LPM\vec{i}_2$,即 $LPM(\vec{i}_1-\vec{i}_2)=0$.所以 $LPM(\vec{i}_3-\vec{i}_4)=0$,从而 $LPM\vec{i}_3=LPM\vec{i}_4$.与 $\vec{i}_3$ 和 $\vec{i}_4$ 属于不同并行后子循环的假设矛盾.假设错误,定理成立,证毕. □

证明(定理 2):我们首先证明L_{EH}=∅,再证 $\vec{l} \in L_{AT} \Leftrightarrow \exists n \in \mathbf{Z}: l_j = np$,从而 $\vec{l} \in L_{ET} \Leftrightarrow \neg\exists n \in \mathbf{Z}: l_j = np$ 就自然成立了.

(证L_{EH}=∅)反证法.假设 $\exists\vec{l} \in L_{HT}$,即 $\exists\vec{i}_1,\vec{i}_2 \in Z^n: \vec{l}=\vec{i}_1-\vec{i}_2$,$\vec{i}_1$ 和 $\vec{i}_2$ 都属于同一线程,并且 $\exists\vec{i}_3,\vec{i}_4 \in Z^n: \vec{l}=\vec{i}_3-\vec{i}_4$,$\vec{i}_3$ 和 $\vec{i}_4$ 属于不同的线程.

设循环在第j层被并行化,p是线程数目,则$i_{j,1}\equiv i_{j,2}(\mathrm{mod}\ p)$.所以$i_{j,3}-i_{j,4}=l_j=i_{j,1}-i_{j,2}\equiv 0(\mathrm{mod}\ p)$,意味着 $\vec{i}_3$ 和 $\vec{i}_4$ 属于同一个线程.与假设矛盾,L_{EH}=∅成立.

(证 $\vec{l} \in L_{AT} \Leftrightarrow \exists n \in \mathbf{Z}: l_j = np$)设连接向量 $\vec{l}$ 连接的两个迭代分别为 $\vec{i}_1$ 和 $\vec{i}_2$.

⇐)设n∈□:l_j=np,那么$i_{j,1}-i_{j,2}=l_j=np$.所以$i_{j,1}-i_{j,2}\equiv 0(\mathrm{mod}\ p)$,$i_{j,1}\equiv i_{j,2}(\mathrm{mod}\ p)$,从而 $\vec{i}_1$ 和 $\vec{i}_2$ 属于同一线程.又因为 L_{EH}=∅,所以 $\vec{l} \in L_{AT}$.

⇒)设 $\vec{i}_1$ 和 $\vec{i}_2$ 都属于线程T_q,则∃m_1∈□:$i_{j,1}$=m_1p+q且∃m_2∈□:$i_{j,2}$=m_2p+q.令n=m_1−m_2,则n∈□且l_j=$i_{j,1}$−$i_{j,2}$=np.

证毕. □

证明(推论 1):设 $\vec{l}_1,\vec{l}_2 \in L_{AT}$,则 $\exists m,n\in\square$, $\exists\vec{l}_1^{LPM},\vec{l}_2^{LPM} \in \ker LPM$,且 $\vec{l}_1=(0,...,mp,...,0)^T_{e_j=mp}+\vec{l}_1^{LPM}$,$\vec{l}_2=(0,...,np,...,0)^T_{e_j=np}+\vec{l}_2^{LPM}$.因为 $m+n\in\square$, $\vec{l}_1^{LPM}+\vec{l}_2^{LPM} \in \ker LPM$,所以,

$$\vec{l}_1+\vec{l}_2=\begin{pmatrix}0\\ \vdots\\ (m+n)p\\ \vdots\\ 0\end{pmatrix}_{e_j=(m+n)p}+(\vec{l}_1^{LPM}+\vec{l}_2^{LPM}) \in L_{AT}.$$

因此,L_{AT}满足封闭率,构成连接向量空间的子空间.同时,由于 $(0,...,p,...,0)^T_{e_j=p}$ 是向量空间 $(0,...,np,...,0)^T_{e_j=np}$,$n\in\square$的基,ker$LPM$是封闭的向量空间,并且根据LPM的定义,LPM对角的第j个元素为1,kerLPM的第j个元素一定为 0.从而,$(0,...,p,...,0)^T_{e_j=p}$ 和kerLPM正交.所以,

$$L_{AT}=span\left\{\begin{pmatrix}0\\ \vdots\\ p\\ \vdots\\ 0\end{pmatrix}_{e_j=p}+\ker LPM\right\}=span\left\{\begin{pmatrix}0\\ \vdots\\ p\\ \vdots\\ 0\end{pmatrix}_{e_j=p}\right\}+\ker LPM.$$ □

证明(定理 3):这里,我们只证明l_j≥0 的连接向量.因为面向OpenMP重用理论中并不关注重用向量的方向,因此,下面的证明也可以自然扩展到l_j≤0 的连接向量.

设某个块的某下界迭代点$i_{j,L}$=a_1pk+a_2k,则这个块的上界迭代点满足$i_{j,U}$=a_1pk+a_2k+k−1.若连接向量l_j=b_1pk+b_2k+b_3的起点分别连接上述两点,则连接向量的终点分别是

$$i_{j,L}+l_j=(a_1+b_1)pk+(a_2+b_2)k+b_3,\ i_{j,U}+l_j=(a_1+b_1)pk+(a_2+b_2)k+k-1+b_3.$$

由于 $0\leqslant b_3<k$, $\left\lfloor \frac{i_{j,L}+l_j}{k} \right\rfloor \equiv a_2+b_2 (\mathrm{mod}\ p)$, $\left\lfloor \frac{i_{j,U}+l_j}{k} \right\rfloor \equiv a_2+b_2+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p)$.

(1a) 若连接向量 $\vec{l}$ 的终点指向的两个迭代点属于同一个块,则 $a_2+b_2 \equiv a_2+b_2+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p)$,从而

$$\left\lfloor \frac{k-1+b_3}{k} \right\rfloor \equiv 0(\mathrm{mod}\ p).$$

设 $\left\lfloor \frac{k-1+b_3}{k} \right\rfloor = np, n\in \mathbf{Z}$,因为 $0\leqslant b_3<k$,所以 $k-1\leqslant k-1+b_3<2k-1$.从而,$\frac{k-1}{k}\leqslant \frac{k-1+b_3}{k}<1+\frac{k-1}{k}$,$0\leqslant \left\lfloor \frac{k-1+b_3}{k} \right\rfloor \leqslant 1$,$0\leqslant np\leqslant 1$.又因为线程数 $p>1$,所以 $n=0$,即 $\left\lfloor \frac{k-1+b_3}{k} \right\rfloor = 0$,$k-1+b_3<k$,$b_3<1$.因此,$b_3=0$.

(1b) 若连接向量 $l_j=b_1pk+b_2k$,即 $b_3=0$,则 $\left\lfloor \frac{k-1+b_3}{k} \right\rfloor \equiv 0(\mathrm{mod}\ p)$, $\left\lfloor \frac{i_{j,L}+l_j}{k} \right\rfloor \equiv \left\lfloor \frac{i_{j,U}+l_j}{k} \right\rfloor (\mathrm{mod}\ p)$.因此,连接向量 $\vec{l}$ 的终点指向的两个迭代点属于同一个块.

(2) 若连接向量 $\vec{l}$ 的终点指向的两个迭代点属于不同的块,则 $a_2+b_2 \neq a_2+b_2+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p)$,从而 $\left\lfloor \frac{k-1+b_3}{k} \right\rfloor \neq 0(\mathrm{mod}\ p)$.又因为 $0\leqslant b_3<k$,所以 $k-1\leqslant k-1+b_3<2k-1$.从而,$k\leqslant k-1+b_3<2k-1$,$1\leqslant b_3<k$并且 $\left\lfloor \frac{k-1+b_3}{k} \right\rfloor \equiv 1(\mathrm{mod}\ p)$, $a_2+b_2+1 \equiv a_2+b_2+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p)$.因此,连接向量 $\vec{l}$ 的终点指向的两个迭代点属于相邻的块.证毕. □

证明(推论 2):将连接向量 $\vec{l}$ 的起点分别连接某个块的上界和下界,那么 $\vec{l}\in L_{AT}$ 当且仅当 $\vec{l}$ 的终点同时指向这个块.设 $\vec{l}$ 的起点和终点分别为 $\vec{i}_1$ 和 $\vec{i}_2$,则 $\left\lfloor \frac{i_{j,1}}{k} \right\rfloor \equiv \left\lfloor \frac{i_{j,2}}{k} \right\rfloor (\mathrm{mod}\ p)$;并且根据定理 3,$l_j=b_1pk+b_2k$.因为 $l_j=i_{j,2}-i_{j,1}$,所以 $\left\lfloor \frac{i_{j,1}}{k} \right\rfloor \equiv \left\lfloor \frac{i_{j,1}+b_1pk+b_2k}{k} \right\rfloor (\mathrm{mod}\ p)$,即 $b_1p+b_2\equiv 0(\mathrm{mod}\ p)$.又因为 $0\leqslant b_2<p$,所以 $b_2=0$.以上推导步步可逆,因此推论 2 成立. □

证明(推论 3):设某个块的某上界迭代点 $i_{j,L}=a_1pk+a_2k$,则这个块的下界迭代点满足 $i_{j,U}=a_1pk+a_2k+k-1$.若连接向量 $l_j=b_1pk+b_2k+b_3$ 的起点分别连接上述两点,则连接向量的终点分别是

$$i_{j,L}+l_j=(a_1+b_1)pk+(a_2+b_2)k+b_3,\ i_{j,U}+l_j=(a_1+b_1)pk+(a_2+b_2)k+k-1+b_3.$$

根据定理 3,$\vec{l}\in L_{HT}$ 当且仅当存在如图 12 所示的两种情况之一.其中,$i_{j,L}$ 和 $i_{j,U}$ 是对应块中的下界和上界点.

(a) $\vec{l}$ points to the same block from the lower bound (b) $\vec{l}$ points to the same block from the upper bound

(a) $\vec{l}$ 的起点对齐块的下界时,终点指向同一个块 (b) $\vec{l}$ 的起点对齐块的上界时,终点指向同一个块

Fig.12 Possible situations when $\vec{l}\in L_{HT}$ in static and *chunk*≠1 mode

图 12 Static 且 *chunk*≠1 时,$\vec{l}\in L_{HT}$ 可能的情况

(1) 首先,不论是图 12(a)还是图 12(b)的情况,$\left\lfloor \frac{i_{j,L}+l_j}{k} \right\rfloor +1 \equiv \left\lfloor \frac{i_{j,U}+l_j}{k} \right\rfloor (\mathrm{mod}\ p)$ 始终成立,即

$$\left\lfloor \frac{(a_1+b_1)pk+(a_2+b_2)k+b_3}{k} \right\rfloor \equiv \left\lfloor \frac{(a_1+b_1)pk+(a_2+b_2)k+k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p).$$

从而 $(a_1+b_1)p+(a_2+b_2)+1\equiv(a_1+b_1)p+(a_2+b_2)+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p), \left\lfloor \frac{k-1+b_3}{k} \right\rfloor \equiv 1(\mathrm{mod}\ p)$,所以 $b_3\neq 0$.又因为 $0\leqslant b_3<k$,所以 $0<b_3<k$.

(2) 在图 12(a)所示情况下,$\left\lfloor \frac{i_{j,L}}{k} \right\rfloor \equiv \left\lfloor \frac{i_{j,L}+l_j}{k} \right\rfloor (\mathrm{mod}\ p)$,即

$$\left\lfloor \frac{a_1pk+a_2k}{k} \right\rfloor \equiv \left\lfloor \frac{(a_1+b_1)pk+(a_2+b_2)k+b_3}{k} \right\rfloor (\mathrm{mod}\ p).$$

从而 $a_1p+a_2\equiv(a_1+b_1)p+(a_2+b_2)(\mathrm{mod}p), a_2\equiv a_2+b_2(\mathrm{mod}p), b_2=0(\mathrm{mod}p)$.又因为 $0\leqslant b_2<p$,所以 $b_2=0$,即 $|l_j|=b_1pk+b_3$.

(3) 在图 12(b)所示情况下,$\left\lfloor \frac{i_{j,U}}{k} \right\rfloor \equiv \left\lfloor \frac{i_{j,U}+l_j}{k} \right\rfloor (\mathrm{mod}\ p)$,即

$$\left\lfloor \frac{a_1pk+a_2k+k-1}{k} \right\rfloor \equiv \left\lfloor \frac{(a_1+b_1)pk+(a_2+b_2)k+k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p).$$

从而 $a_1p+a_2\equiv(a_1+b_1)p+(a_2+b_2)+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor (\mathrm{mod}\ p), b_2+\left\lfloor \frac{k-1+b_3}{k} \right\rfloor \equiv 0(\mathrm{mod}\ p)$.

由情况(1)可知,$\left\lfloor \frac{k-1+b_3}{k} \right\rfloor \equiv 1(\mathrm{mod}\ p)$,所以 $b_2+1\equiv 0(\mathrm{mod}\ p)$.又因为 $0\leqslant b_2<p$,所以 $b_2=p-1$,即 $|l_j|=b_1pk+(p-1)k+b_3$.证毕. □

定理 4、定理 5 以及推论 4 和推论 5 的证明分别与定理 2、定理 3 以及推论 2、推论 3 类似,不再赘述.

吴俊杰(1981－),男,安徽蚌埠人,博士,助理研究员,主要研究领域为计算机体系结构与编译.

刘光辉(1982－),男,博士生,主要研究领域为计算机体系结构,容错技术.

杨学军(1963－),男,博士,教授,博士生导师,主要研究领域为并行计算机体系结构与编译,容错并行算法,流处理器体系结构与编译.

唐玉华(1962－),女,研究员,主要研究领域为计算机体系结构,计算机网络.