1000-9825/2002/13(06)1178-09 ©2002 Journal of Software 软 件 学 报 Vol.13, No.6

一类不规则问题的自动并行性识别*

李 靖, 臧斌宇, 朱传琪

(复旦大学 并行处理研究所,上海 200433)

E-mail: lijing_nj@hotmail.com

http://www.fudan.edu.cn

摘要: 传统的数据相关性分析主要针对线性数组下标表达式,并不适用于不规则计算中的循环并行性识别.利用间接数组特征分析和基于严格数组私有化定义的运行时动态测试技术来解决包含间接数组下标引用的循环并行性判断问题,给出性能估测,并与相关工作进行了比较.

关 键 词: 并行识别;不规则;数组私有化;运行时测试

中图法分类号: TP311 文献标识码: A

循环并行性识别是并行编译中不可缺少的重要组成部分.不规则问题中的并行性识别比规则计算面临更多的困难.由于不规则问题存在大量编译时未知的信息,且循环界限和数组引用下标不再是循环控制变量的线形表达式,以求丢番图方程整数解为代表的原有相关性分析方法已不再适用.因此有必要寻求新的并行循环识别技术去完善现有的分析方法,为数据分布、计算负载平衡和通信优化等工作奠定基础.

稠密矩阵计算程序的数组访问下标是外部循环控制变量的线性表达式,现有的自动并行编译器对这类问题能较好地判断语句之间的相关性.但是,另外一些计算问题如有限元分析、分子动力学计算等涉及到稀疏矩阵,从实现效率角度出发,其主要数据结构不再使用连续数组,而是用链表、树、图等数据结构组成程序的基本内部表示框架,我们称其为不规则问题.不规则问题计算中的非标准循环结构、多级数组访问和指针运算是自动并行化分析时遇到的主要障碍[1].不解决这些问题,并行编译器就难以推广应用.本文主要讨论其中如图 1 所示的具有间接数组下标形式的循环并行性识别问题.此时,主数组的访存行为由间接数组的取值决定.

```
Do I=1, n
   …
   Home(index1(I))=…
   …
   …=Home(index2(I+a))
End
```

Fig.1
图 1

传统的并行编译器主要针对规则的计算程序,从安全性考虑,对不规则的数组访问区域只做保守估计,如对读操作取超集、写操作取子集.由于无法获知间接数组的具体值,通常并行编译器把主数组存取集合取为整个数组定义区间或空集合.这样的保守估计显然会错过一些将不规则循环并行化的机会.

对实际应用程序的观察表明,不规则计算中确实存在大量可并行循环,并且通过编译时或运行时分析能够将其并行性挖掘出来.我们首先在第 1 节给出判断循环并行性的一般规则,并由此得到严格的数组私有化定义,然后在第 2 节从间接数组特征分析和动态测试两方面分别提出不规则计算中的循环并行性识别方法.

1 循环并行化判定规则

记 L 为所考察的循环,L 的循环控制变量在边界约束下构成迭代空间 I,记 $READ_i$ 为迭代 $i(i\in I)$读取的变量

---

* 收稿日期: 2000-09-14; 修改日期: 2001-03-08

基金项目: 国防科技重点实验室基金资助项目(JS94.6.1JW0703);国家教育部博士点基金资助项目;上海市青年科技启明星计划资助项目(99QD14043)

作者简介: 李靖(1972-),男,安徽合肥人,博士,主要研究领域为并行编译;臧斌宇(1965-),男,上海人,博士,教授,主要研究领域为并行处理;朱传琪(1943-),男,上海人,教授,博士生导师,主要研究领域为并行处理.

© 中国科学院软件研究所 http://www.jos.org.cn

集合,$WRITE_i$ 为迭代 i 的赋值变量集合,$UPEXPOSE_i$ 为迭代 i 的向上暴露变量集合.向上暴露变量是指在某块程序代码段中先引用后赋值的变量.

计算密集型程序将大多数时间消耗在循环上.因此,串行程序的自动并行化目标集中在如何能使循环并行执行上.近 10 年来的实践与研究表明,可直接并行循环,又称独立(independet)循环,是程序中循环并行性的主要来源.如果循环可直接并行执行,其不同迭代之间的任意两条访问相同变量的语句(至少有一个是写操作)必须没有任何形式的数据相关,包括流相关、反相关和输出相关.这种方法需要考察循环中所有访问相同变量的语句对.循环直接并行化定义为[2]

$$\left(READ_i \cap WRITE_j = \varnothing\right) \wedge \left(WRITE_i \cap WRITE_j = \varnothing\right),\quad \forall i,j \in I, \text{且} i \neq j. \tag{1}$$

循环可直接并行化意味着各迭代可按任意分配方式交给各节点同时执行,相互之间没有同步或数据通信操作.然而,实际应用程序中存在许多非直接并行循环.我们可以通过程序变换方法消除其中的存储关联数据相关,即反相关和输出相关,这是程序自动并行化工具的主要工作之一.数组私有化是目前所采用的最主要的并行变换方法.

1.1 数组私有化

数组私有化不仅广泛应用于共享存储系统的并行编译,在分布存储系统上也可发挥作用.分布存储系统为了提高存储空间使用效率,通常要对全局数组做局部化处理,而数组私有化是数组局部化的逆过程,通过重用部分存储空间,起到消除跨处理机假相关的作用.数组私有化的极端情况是每个节点拥有所有数组的私有拷贝,节点程序数据分配与单机情况相同,这正是许多 MPI(message-passing interface)和 PVM(parallel virtual machine)等消息传递类应用程序实现时采用的方法.此时可以在最大程度上消除跨循环假相关,但同时也会带来冗余数据分配.因此,在进行数组私有化时应该选择恰当的时机,既要在最大程度上使循环并行化,又不致给系统带来不必要的开销.

数组私有化无法沿用判断循环直接并行化时所使用的成对(pair-wise)方法.首先,这是因为一些具有相关性的语句之间并不存在真正的数据流关系,如反相关和输出相关;另外,Pair-wise 方法将所有访问相同地址数据的写读语句对标记为流相关,而实际上两条语句实例之间可能存在离读操作更近的写操作,因此前一个写操作对于读操作是过期的,真正的数据流发生在后两条读写语句之间.因此,要采用基于值(value-based)的数据流分析技术对数组进行私有化.

对数组私有化的认识有一个逐步深化的过程.最初,文献[3]中认为若循环在私有化后可并行执行,则每个迭代引用的变量必须在本次迭代开始时被重新定义,即所有迭代中的读操作被写操作所覆盖.

$$UPEXPOSE_i = \varnothing,\quad \forall i \in I. \tag{2}$$

随后,这一定义被放松为任一迭代的向上暴露变量集合与其他迭代的赋值变量集合之间不存在相关[2],此时,为保持串行语义,循环再做并行化需要把原数组拷贝至私有数组.

$$WRITE_i \cap UPEXPOSE_j = \varnothing,\quad \forall i,j \in I, \text{且} i \neq j. \tag{3}$$

由于跨迭代反相关被私有数组消除,只有迭代间流相关才是阻碍数组私有化的真正原因,式(3)可优化为[4]

$$WRITE_i \cap UPEXPOSE_j = \varnothing,\quad \forall i,j \in I, \text{且} i < j. \tag{4}$$

循环满足上式意味着它在数组私有化后能够并行执行.但是我们可以发现,在此条件下数组私有化不一定是必须的.循环可能未经数组私有化就可以直接并行执行.由于 $READ_i$ 可以分解为它与写变量集合 $WRITE_i$ 的交集(记作 RW_i)和只读变量集(记作 RO_i),从式(1)可得:

$$\begin{aligned}
&\neg\left(\forall i \forall j\left(\left(READ_i \cap WRITE_j = \varnothing\right) \wedge \left(WRITE_i \cap WRITE_j = \varnothing\right)\right)\right)\\
&\Rightarrow \exists i \exists j\left(\left(READ_i \cap WRITE_j \neq \varnothing\right) \vee \left(WRITE_i \cap WRITE_j \neq \varnothing\right)\right)\\
&\Rightarrow \exists i \exists j\left(\left(\left(RO_i \cup RW_i\right) \cap WRITE_j \neq \varnothing\right) \vee \left(WRITE_i \cap WRITE_j \neq \varnothing\right)\right)\\
&\Rightarrow \exists i \exists j\left(\left(RO_i \cap WRITE_j \neq \varnothing\right) \vee \left(RW_i \cap WRITE_j \neq \varnothing\right) \vee \left(WRITE_i \cap WRITE_j \neq \varnothing\right)\right)\\
&\Rightarrow \exists i \exists j\left(\left(RO_i \cap WRITE_j \neq \varnothing\right) \vee \left(\left(RW_i \cup WRITE_i\right) \cap WRITE_j \neq \varnothing\right)\right)\\
&\Rightarrow \exists i \exists j\left(\left(RO_i \cap WRITE_j \neq \varnothing\right) \vee \left(WRITE_i \cap WRITE_j \neq \varnothing\right)\right).
\end{aligned} \tag{5}$$

© 中国科学院软件研究所 http://www.jos.org.cn

其中 i, j 满足 $i\neq j$.因为 $READ_i$, RO_i 与本次迭代的向上暴露集合 $UPEXPOSE_i$ 具有包容关系 $RO_i \subset UPEXPOSE_i \subset READ_i$,由式(4)可知,当需要进行数组私有化以获取循环并行性时,在 $i<j$ 范围内 $WRITE_i \cap RO_j$ 为空,所以式(5)中 $WRITE_i \cap RO_j \neq \varnothing$ 仅在 $i>j$ 时成立.完整的数组私有化条件应该由式(4)和式(5)共同组成.

式(4)和式(5)里存在两种形式的读集合:只读变量集和向上暴露集.事实上,式(5)中的只读变量集可以放大成向上暴露集而不影响并行性判断的正确性.因为 $READ_i$ 又可当作由向上暴露集 $UPEXPOSE_i$ 和被写操作覆盖的部分变量集合(记作 S_i^D)所组成,而 S_i^D 同样是 $WRITE_i$ 的子集.把 $READ_i$ 的这种分解方式代入式(5)的证明过程中,得到的结果是将结论里的只读变量集 RO_i 替换成向上暴露集 $UPEXPOSE_i$.这样做可能会使部分迭代间反相关判断与输出相关的判断相互重叠,但优化后的测试算法不会因此而引入冗余计算,而且少收集一种数据访问信息有利于提高算法效率和存储空间利用率.综上所述,最终我们把数组私有化定义如下:

定义 1(数组私有化). 多处理机可通过分配数组的私有拷贝获取循环级并行性,当且仅当循环各个迭代中的数组访问集合满足下式:

$$\begin{cases} WRITE_i \cap UPEXPOSE_j = \varnothing, \quad \forall i,j \in I, i<j, & (6.1) \\ (WRITE_i \cap WRITE_j \neq \varnothing) \vee (WRITE_k \cap UPEXPOSE_l \neq \varnothing), \quad \exists i,j,k,l \in I, i \neq j, k>l. & (6.2) \end{cases}$$

在第 2.2 节中将利用该数组私有化定义公式进行循环并行性的动态测试.

2 自动识别不规则计算中的并行性

如果经过符号关系等式传播以后,数组访问下标表达式仍然是间接数组形式,我们就可以依据从定义点获知的间接数组取值的有关特征,提高循环并行化的机率.间接数组的取值通常具有一定的规律性,但是这些数据访问模式信息仅用循环内数据流分析无法得到.例如,间接数组在程序中可能有若干个定义点,因此,需要沿不同控制流路径跨越若干循环与过程才能够确定间接数组在使用点所具有的特征.

程序的间接数组取值特征并非总是能够静态地获取,此时只能借助于运行时分析获取循环并行化所需要的未知信息.尽管运行时分析的开销较大,但作为程序并行化的最后手段,它能够在最大程度上揭示程序中潜在的循环并行化机会,且有助于改进现有的静态分析技术.

2.1 基于间接数组特征的并行性识别

间接数组在程序中的作用类似于指针,主数组的访问模式完全由间接数组的取值来决定.对循环并行化有利的间接数组特征是数组取值的惟一性及其特例单调性.

```
bt=10
Do I=1,99 {                 …l1
   If (I<bt)   A(I)=2I      …s
   Else   A(I)=2I−5         …r
}
Do I=1,99 {                 …l2
   If (c)   B(A(I))=…       …t1
   …=B(A(I))                …t2
}
```

Fig.2
图 2

- 惟一性

$$\forall i \neq j, index(i) \neq index(j) .$$

- 单调性

$$\forall i<j, index(i)<index(j) \quad \text{or} \quad \forall i<j, index(j)<index(i) .$$

例如,在图 2 给出的这段代码中,循环 $l1$ 是间接数组 A 的定义点.A 在[1,bt]区间和[bt,99]区间分别满足单调性,但在整个[1,99]区间不满足单调性.为了证明数组 A 在[1,99]内满足惟一性,令:

$$Rs = \langle i_s < bt \mid A = 2i_s, 1<i_s<99, bt=10 \rangle,\ Rr = \langle i_r > bt \mid A = 2i_r - 5, 1<i_r<99, bt=10 \rangle.$$

由于条件谓词和数组访问区间都由整数线形不等式表示,把谓词嵌入[5]数组访问区间并利用 Fourier-Motzkin 消元[6]可得出数组取值惟一性结论.

$$\begin{aligned} Rs \cap Rr &= \langle A=2i_s, 1<i_s<99, bt=10, i_s<bt \rangle \cap \langle A=2i_r-5, 1<i_r<99, bt=10, i_r>bt \rangle \\ &= \langle A=2i_s, 1<i_s<99, i_s<10 \rangle \cap \langle A=2i_r-5, 1<i_r<99, i_r>10 \rangle \\ &= \langle A=2i_s, 1<i_s<10 \rangle \cap \langle A=2i_r-5, 10<i_r<99 \rangle \\ &= \varnothing. \end{aligned}$$

采用传统方法分析循环 $l2$ 的并行性时,由于存在非线性的数组下标表达式且向上暴露集非空,因此认为不同迭代的 $t1$,$t2$ 语句对之间具有相关性.而根据间接数组的取值惟一性可以直接判断循环 $l2$ 的主数组访问无跨迭代

© 中国科学院软件研究所 http://www.jos.org.cn

相关,从而循环能够并行执行.

间接数组的特征通常在循环语句结构中定义.程序中的条件控制流使得间接数组在不同定义域区间上具有不同的特征表达式,这种多样性给数组在整个定义域上的综合特征判断带来了困难.

嵌套在循环结构里的条件分支语句把间接数组定义域分为若干子区域,如上例所示.因为间接数组在各个子区域上的取值不一定具有连续性,仅仅根据各个子区域上的特征作简单累加将会得出错误的结论,所以在合并子区域上的特征时不能只做交、差等集合运算,而应使用间接数组在各子区域上的取值表达式重新作一次完整的判断.

若定义间接数组的循环结构出现在条件语句的不同分支里,则意味着间接数组特征将依赖于程序当前执行环境.若编译时无法确定控制流向,则只能假设存在所有可能的特征组合.此时,间接数组特征集将成倍扩大.

程序中的过程调用也会使多种特征组合经不同的控制流到达主数组引用点.而间接数组沿其中任意一条路径若不具备必要的特征都将使得主数组引用点所在的循环无法并行执行.若记 P_{reach} 为所有从程序起始点到主数组引用点的控制流集合,我们需要对其中每一条路径检查最近的间接数组定义是否可使当前循环并行化.函数 satisfiable 的返回值表示间接数组是否满足预定义的特征.

$$\bigcap_{p\in P_{\text{reach}}} satisfiable(p, L_{\text{def}}(index)) = \text{true}\ .$$

数组特征作为数组的本质属性沿控制流在整个程序范围内传播.为避免计算与当前数组使用点无关的间接数组特征定义,可以采用需求驱动方式从主数组访问点逆控制流程反向查询间接数组特征.在查询进行之前,先进行常数和符号关系传播,然后利于程序切片技术在保留程序控制结构的前提下压缩与间接数组定义无关的代码.由于间接数组特征可能由一条控制流路径上的若干个定义点共同决定,我们在逆向查询过程中逐步缩小间接数组特征的未知定义域区间.若间接数组在某段定义域区间上不具备所需要的特征,即可中断进一步的查询,从而判定访问主数组的循环不可并行化.循环可并行化当且仅当间接数组特征沿所有可达路径都可满足.一个完整的数组特征分析框架可以用下面的算法来表示.每个定义点获得的间接数组特征片段$\langle K,p\rangle$表示间接数组在定义域子区间 K 上满足特征 p.在特征分析进行之前,程序已得到结构化处理.

```
ArrayPropertyChech(){
  输入:Ω为已知可满足间接数组特征集合;Current 为当前函数;L 为待测定义域区间;
  输出:如果间接数组在整个定义域上满足并行特征,函数返回值为真,否则为假.
  Traverse the IR in backward order;
  Case node-type{
    Def-point:
      If (reach program head) return (false);
      Ω′←0;
      For (each possible pair ⟨K, p⟩ in Def-point){
      /* K is sub-region of index array which own property p */
      For (every ω∈Ω){       /* ω is one path reach Use-point */
        KK←∪K′, for all ⟨K′, p′⟩∈ω;
        If (L∩(K-KK)≠0) {       /* index array now has not any property in L∩(K-KK) */
          ω←ω∪⟨L∩(K-KK), p⟩;
          KK←KK∪K;
          ω′←Reduce(ω);
          If (ω′ has not satisfied property)
            return (false);
        }
        If (L-KK≠0) Ω′←Ω′∪{ω′};
      }}
      Ω←Ω′;
      If (Ω=0) return (true);
    Function-Header:
      For (each function caller_i which call Current in call-graph)
        Pred_i←ArrayPropertyChech(caller_i,entry_i,Ω);
      Return (∩Pred_i);
  }
}
```

© 中国科学院软件研究所 http://www.jos.org.cn

算法有利于递归函数调用检查数组特征沿各条过程调用路径的完整性.reduce 函数则是借助于消元法化简由不同间接数组定义域区段组成的特征集.当把一个新获得的间接数组特征$\langle K,p\rangle$加入对应于某一条控制流路径的特征项$\omega=\{\langle K_1,p_1\rangle,\langle K_2,p_2\rangle,\ldots,\langle K_n,p_n\rangle\}$时,分别检查 n 组特征对$(\langle K,p\rangle,\langle K_1,p_1\rangle),\ldots,(\langle K,p,\langle K_n,p_n\rangle)$是否满足间接数组的取值惟一性条件,并做必要的特征集合并.只有当 n 组检查都通过后,才能把$\langle K,p\rangle$加入到ω里,并最终构成新的间接数组特征集合Ω.

循环中的主数组下标可能不完全是间接数组形式,其他形式的数组下标访问同样对数据相关性产生影响,这时,单纯地分析间接数组的特征不足以证明循环能否并行.因此,间接数组特征分析只适用于主数组的所有访问都通过间接数组的情形.即使符合这一约束条件,使用静态方法也并不是总能够确定间接数组特征.例如,程序可能用外部文件初始化间接数组,此时需要利用动态测试技术才能够判断循环能否并行化.

2.2 基于动态剖析技术判断循环并行性

尽管目前编译阶段对数据相关性分析做了大量工作,仍有相当一部分循环无法判断能否并行执行.影响静态分析效果的因素包括符号计算、变量输入依赖性、控制相关以及其他在编译时无法确定的信息.作为静态分析方法的补充,运行时分析技术在自动并行化中起着重要作用.运行时分析可确定哪些循环是本质上可并行的,这有助于我们事后分析分析阻碍并行编译器发现可并行循环的原因,并找出相应对策,提高并行编译器的性能.

因此,如果仅用静态分析方法不足以明确间接数组是否具备使外部循环直接并行化的必要特征,那么只能通过在程序运行时收集到的间接数组访问信息判断外部循环的并行性以及是否需要数组私有化.程序运行时已不存在任何控制流和变量值的不确定性.依据间接数组的数据依赖性,可以采用不同的动态测试方法.

首先考虑较为简单的情形,即主数组具有固定的访存行为模式,或者说主数组间接数组具有值不变性.程序每次运行时,间接数组的初始值都来自惟一的外部文件或不受输入数据影响而由初始赋值语句惟一指定.循环并行性和数组私有化信息可直接从串行程序在单机上的一次运行中得到,我们称其为剖析(profile)测试.如果间接数组的取值随输入数据或程序中间结果而变化,则需要采用第 2.3 节的 inspector-executor 方法去判断循环能否并行化.

剖析技术常被用于程序的行为统计与性能分析,例如,函数的调用次数统计、循环与函数的执行时间占程序总运行时间的比例等等.这些结果是通过附加测试代码在程序运行过程中采集有关信息而获得的.这里,我们将插入代码用于含间接数组引用的循环的并行性识别.

先讨论单层循环,我们以每个迭代为单位,记录迭代内读写的实际数组元素,并将其依次放入临时队列中.临时队列由三元组⟨数组名,位移量,读写形式⟩构成.在迭代的结尾处,按先进先出顺序逐个取出队列中的三元组,判断是否存在跨迭代的流相关.若流相关存在,则停止进一步工作,标识该循环不可并行化.若直到最后一个迭代执行结束仍未发现迭代间流相关,即可以断定其为并行循环.在计算过程中同时判断变量私有化是否为循环并行化的必要条件.由于标量可以看作是数组的特例,因此,算法中没有单独列出.算法中用到的辅助变量的数据结构和初始值定义如下:整型数组 W_l 记录每个数组元素最后一次写操作所在迭代;如果某个数组元素第一次读操作向上暴露于整个循环,则记整型数组 R_f 中的对应元素为当次迭代.布尔变量 $parallel$ 和 $privatize$ 分别用来标识循环是否可以并行执行以及数组私有化是否为并行化的必要条件.布尔变量 $Cpin$ 表示循环运行前是否需要拷贝原数组到私有数组.记迭代空间为$[1{:}n]$,$curit$ 为当前迭代,II 为当前数组访问下标值.图 3 中的两列分别是读、写计算所对应的附加工作.

对于每次迭代中的暴露读首先判断先前迭代是否有对同一数组元素的写操作,如果存在,则表明并行化失败;如果不存在,则说明该读操作被本次迭代里的写操作所覆盖,或者相对于整个循环都是暴露的,若是后者就意味着循环对该数组元素的首次存取为读操作,将首次读操作所在迭代号记录在辅助数组 R_f 中.在处理写操作时,先看先前迭代中有没有对同一数组元素的写操作,然后看它是否属于前面某个迭代的向上暴露集,若发现两者之一,则数组应私有化,同时更新辅助数组 W_l.这些判断只需在数组私有化未定的情况下进行.由于我们是在迭代间无输出相关性的前提下才去判断迭代间的反相关性,所以,虽然所测试的是向上暴露读集合,但实际上被测试的数组元素在所在迭代内是只读的,并没有出现冗余的判断.可以看出,算法在做读操作时完成式(6.1)的判

© 中国科学院软件研究所 http://www.jos.org.cn

断工作;而在做写操作时完成式(6.2)的判断工作.若测试中途退出 *parallel* 为 false,说明循环不可并行化.若测试正常结束后,布尔变量 *Privatize* 和 *parallel* 都为 true,则循环需要进行数组私有化才能够并行;若 *parallel* 为 true 而 *Privatize* 为 false,则循环可直接并行化.

```
Wl[ ]:int=0                         Rf[ ]:int=0;
Cpin:boolean=flase                  Privatize:boolean=false
Parallel:boolean=true

Read :
  If (Wl[II]>0){
    If (curit<Wl[II]){
        Parallel←false
        Exit the test
    }
  } Else {
     If (Rf[II]=0)
        Rf[II]←curit
  }

Write:
  If (¬ Privatize){
    If (Wl[II]≠curit){
      If (Wl[II]≠0){
        Privatize←true
      } Else {
         If (Rf[II]>0)
           If (Rf[II]<curit){
             Privatize←true
             Cpin←true;
      }   }
      Wl[II]←curit
  } }
  Wl[II]←curit
```

Fig.3 Additional judgement in runtime test

图 3 运行时测试中的附加判断

将该算法扩展到多重循环,待测数组在每层循环拥有两个辅助数组.先按照程序从内向外执行顺序,考虑相邻两层嵌套循环 L_j,L_k 的测试算法.$curit^j$ 代表外层循环 L_j 的当前迭代.内层循环 L_k 执行完后得到的辅助数组 W_l^k 和 R_f^k 对循环 L_j 的辅助数组 W_l^j 和 R_f^j 的影响是

$$R_f^j[II]=\begin{cases} curit^j & \text{if } R_f^j[II]=0,\ R_f^k[II]\neq 0,\ W_l^j[II]=0 \\ R_f^j[II] & \text{otherwise}\end{cases},\quad W_l^j[II]=\begin{cases} curit^j & \text{if } W_l^j[II]<curit^j,\ W_l^j[II]\neq 0 \\ W_l^j[II] & \text{otherwise}\end{cases}.$$

外层循环的每个迭代在内部循环 L_k 执行完之后,取出它的辅助数组 R_f^k 和 W_l^k 中所有非零元素,分别当作循环 L_j 的读、写操作,与 R_f^j, W_l^j 中的对应元素一起进行外层循环的并行与私有化判断,注意,必须先处理完 $R_f^j[II]$ 之后才能接着处理 $W_l^j[II]$.具体判断方法与一般单循环读写情况类似,下面仅指出需要注意之处.对于读操作,分析 R_f^k 和 R_f^j 的关系式,可以排除大多数条件组合,只有当 $R_f^j[II]=0$, $R_f^k[II]>0$ 且 $W_l^j[II]=0$ 时,内层循环 L_k 中对下标为 i 的数组元素的暴露读操作才是循环 L_j 的首次向上暴露读.当内层循环 L_k 在运行过程中发现跨迭代流相关时,不应立即中止该层循环的测试,而是继续作数据存取记录,为外层循环的并行性测试提供完整的信息,同时,记 *privatize* 为 true 且 $R_f^k[II]$值保持不变.当多重嵌套循环运行时,测试方法可依此类推.

当循环中出现过程调用语句时,需要将被调用过程中的读写序列记录在当前迭代的访存临时队列里,就像把过程体嵌入到循环中一样.但是,FORTRAN 程序在过程调用时按地址传递参数,存在数组形态变迁(reshape)现象,即被调用过程中的数组结构定义与调用过程定义不同,或者数组首地址之间存在偏移量.数组结构定义的作用域局限在过程体内,因此,在程序退出被调用过程之前,需要把数组访问下标地址投射到调用过程定义的数组结构上.或者,当参数里出现待测数组地址时,把两个辅助数组的对应地址作为参数的一部分传入被调用过程,并按待测数组在被调用过程中的结构重新定义这两个辅助数组.在进入被调用过程之前,对所有已经完成的读写操作进行判断,随后清空临时队列.被调用过程依据自身定义的辅助数组结构作并行与私有化判断,在返回调用过程之前同样要清空临时队列,这样才能够保证过程调用前后数组访问地址的正确性.

程序中的跳转语句可能会使控制流一次跨越若干嵌套循环.对于向外跳转的情形($L_n \to L_m$,m 和 n 为循环层次编号且 $m<n$),在把控制权交给循环 L_m 之前,按嵌套层次的逆序对每一个循环 L,$L_n<L<L_m$ 进行各自辅助数组的更新与并行判断.

当测试单重循环时,剖析算法仍有进一步改进的余地.注意到在这个串行算法中,向上暴露集仅在其刚出现时才具有与其他读操作不同的作用,因此,在向辅助数组 R_f 赋值时没有必要区分读操作是向上暴露还是被写操作所覆盖,R_f 只管记录每个数组元素的第 1 次读操作所在的迭代,写操作将检查式(1)是否成立.这样做虽然使辅

© 中国科学院软件研究所 http://www.jos.org.cn

助数组 R_f 包含了冗余信息,但却减少了测试工作量.

2.3 动态测试的并行优化

在编译时,间接数组的值的不确定性使得我们必须采用动态测试方法进行不规则计算的并行性识别.但是,程序的外部因素,如输入数据的变化,可能会使间接数组在程序多次运行时具有不同的初始值,并且将因此影响循环在不同时刻的并行性.其他导致间接数组的值发生变化的原因包括用随机数发生器初始化间接数组和程序在运行过程中使用重新赋值过的间接数组值作为下标再次访问主数组.在这些情况下,一次性地获得循环并行性的剖析测试方法就不再适用了.

对于上述几种情况,程序在并行机上运行时可以先测试预定循环,如果整个循环执行过程中不存在跨迭代流相关,则可以继续并行执行该循环,否则执行循环的串行化版本,这就是所谓的 inspector-executor 方法.另外,还可以通过推测执行方式动态地实现循环的并行化.推测并行是指对于被认为具有并行前景的待测循环,让各个处理机并行执行分配给它的部分迭代,在运行过程中,各处理机分别记录读写操作并判断是否有违背相关性原则的情况发生.如果能正常执行完整个循环,则提交计算所作的变量修改.如果在并行执行过程中或结束后发现无法满足迭代间相关性约束,则立即取消先前所做并行操作,程序执行点卷回至循环起始位置,然后串行执行本次循环.

对比以上两种不同的动态测试形式可以发现,推测式并行化要做数据备份和恢复工作,所需系统开销较大,只有在事先预知循环具有比较大的并行化机率时才会取得理想效果.而 inspector-executor 方法是在做循环并行执行之前识别其并行性,当待测主数组下标和控制主数组存取的分支条件不受循环中被赋值变量影响的时候,不规则计算问题更适合于使用预测方法判断循环并行性.根据循环直接并行与私有化判断规则,我们可以将剖析算法用作 inspector 阶段或推测并行中的相关测试部分.但是考虑到执行效率,有必要对其进行并行优化.

共享存储对称多处理机(SMP)在执行并行计算时采用主从模式.程序中的串行部分都由主处理机执行,而并行循环按照处理机编号以分块方式将迭代均分给所有处理机共同完成.利用多处理机计算划分的这一特点,我们将分析分配到各处理机的迭代组之间的并行性,而不仅仅是单个迭代之间的并行性.

Inspector 阶段分为两个部分:

(1) 采集过程(collection phase):按并行循环划分方式收集在每个处理机上执行的迭代组的访存信息.这一过程可以并行执行.相当于把第 2.2 节中的辅助数组进行了私有化,每个处理机 p 有两个对应于待测主数组的辅助数组 R_p 和 W_p,其中 R_p 用于记录当前处理机所分得的迭代组中的暴露读集合,W_p 用于记录当前迭代组中的写集合.辅助数组各元素的值为 0 或当前处理机编号.

(2) 决策过程(decision phase):进行循环直接并行化与数组私有化判断.对多处理机的并行计算产生影响的只是迭代组之间的相关性.例如,数组私有化时只需考虑迭代组之间的流相关,迭代组内的流相关可以忽略不计.这一过程按照处理机编号顺序依次分析相邻迭代组之间的数据相关性,在合并两个辅助数组时同样采用并行判断以缩短运行时间.

记采集过程用时为 t_c,决策过程用时为 t_d,若待测循环可以并行执行,其并行执行时间记作 t_e,inspector-executor 的全部计算时间 t_p 等于 $t_c+t_d+t_e$.假设一段代码的相对运算时间大小可以由数组访存次数大致决定,同时设处理机总数为 n,待测数组的元素总数为 m,循环中的访存总次数为 l,其中对每个待测数组元素平均访问了 k 次,那么 t_p 的最大值可预估为$(3km+6m(n-1)+l)/n$.因为循环级并行主要适用于粗粒度循环,所以,在一般情况下 $l >> m*k$.图 4(a)和图 4(b)分别表示 4 处理机和 8 处理机上 t_p 相对于串行执行时间的加速比.其中横坐标为 $a=l\backslash(m*k)$,数组存取占全部访存操作的比重在很大程度上反映了循环计算粒度的大小;纵坐标为并行加速比.图表显示,并行加速比随着循环粒度的增大而上升.如果能较早地发现数组需私有化并且此后不再进行有关判断,加速比将随着 t_d 的缩短而进一步得到提高.

程序状态空间随程序运行发生的变化使循环每一次执行时的环境很不相同,造成不同时刻得出不同的并行与私有化结果.记录下每次条件的差异,例如间接数组值是否发生变化,并且在相同条件重复出现时直接利用上次判断的结论,可以大大减少动态测试的工作量.

© 中国科学院软件研究所 http://www.jos.org.cn

辅助数组在并行测试算法中仅起到标识作用,因此可以按位分配.Inspector 部分的辅助数组空间可以用来存放动态分配给每个处理机的私有化数组.根据读写范围,在一定条件下只需对数组的部分区段进行私有化.两个辅助数组的存储分配空间可以合并在一起,数组元素定义为具有读、写两个数据域的结构.这些措施能够提高存储空间利用率和数据访问局部性.

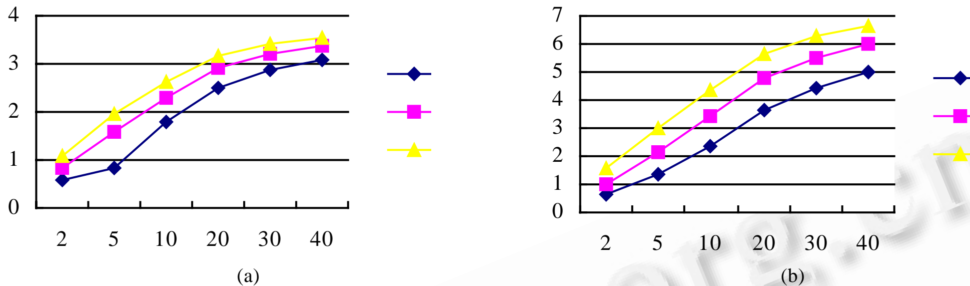

Fig.4 Speedup of parallel loop with runtime test
图 4 并行循环的动态测试加速比

当推测执行并行循环时,待测数组通常已经做了私有化,只需在暴露读处判断式(4)是否成立;若待测数组未做私有化,写操作所附加的私有化判断也一并用作循环直接并行性测试.通常,多重循环中只能有一个循环被指定并行执行,因此,不存在内层并行循环的信息向外传播过程.为了及早发现处理机之间的流相关,避免做无用的计算,也可以在循环推测并行执行过程中同时对数据相关性进行测试[7].除了应用于间接数组下标访问,动态测试技术同样可以用于解决其他不规则计算中的循环并行性与数组私有化判断的问题.

2.4 相关工作比较

Rauchwerger 等人最早提出了动态测试数组私有化的方法[8,9].Moon 的 ELPD(extended-LPD)测试[2]在LPD(lazy privatizing doall test)算法基础上扩展了处理多重循环的功能.ELPD 测试先收集所有迭代里的读写信息,然后根据式(3)和式(5)定义的私有化条件进行综合判断.由于难以在有限空间内记录向上暴露集与写集的全部迭代次序关系,该方法无法具备处理式(4)的功能,在识别私有化数组的能力上弱于本文提出的测试方法.其数组私有化的测试并不完整,当一个变量的访问仅出现在某个迭代且先后进行了读、写操作时,循环将被误判为不可并行化.而我们实现的是第 2.1 节中式(6)定义的数组私有化判定规则,测试算法把向上暴露集和数组的 3 类定义点[3]的计算融合进数组访问信息的记录过程中,通过每个读写操作的附加测试逐步排除不可直接并行化和非并行的情况,实现过程比较直观.以迭代为单位进行判断能够及早排除不可并行化的循环,而不是在整个循环执行结束后才可获得结果.另外,同 Moon 测试算法比较,这种方法少用了两个辅助数组,大约节省了 50%的测试数据空间,因此能够应用在内存较小的环境里.

如第 1.1 节所述,依据读操作相对于写操作的先后次序,迭代 i 的读写交集 RW_i 可分解为 S_i^U 和 S_i^D .LPD 测试里的辅助数组 S_{np} 仅指 S_i^U .ELPD 测试把 S_{np} 恢复成 $RO_i \cup S_i^U$,即本文中的向上暴露集 $UPEXPOSE_i$,但取值为布尔类型并且仍使用 RO_i 测试循环的直接并行性,为此,不得不引入新的辅助数组 S_{rf} 来实现内层循环信息向外层循环的传播.从式(5)的推演过程可以看出,读写交集 RW_i 对循环直接并行性的影响已经在输出相关中得到体现,只读变量集是判断迭代间读写相关的最精简的集合.但是在数组私有化的识别框架内应避免出现新的属性集,因此,我们在算法性能所受到的影响可忽略不计的情况下统一使用向上暴露集.同时,在进行了式(6.1)的判断之后,向上暴露集只是在当整个循环对数组元素的首次访问为读操作的情况下才记录变量出现的位置,因此,在处理多重循环时无须引入其他辅助数组.由于消除了冗余信息,算法结构由得复杂变得清晰,表现为读操作的相关辅助数组由 3 个减少为 1 个,直接并行与私有化的判断过程也得以简化.

为了提高运行效率,SPNT(speculative parallelization with new technology)测试[9]提出了嵌入在推测执行过程中的动态测试方法.在数组已私有化的前提下,各处理机一边推测执行,一边检查是否存在跨处理机流相关.由于每次访问全局辅助数组时需要进行同步操作,SPNT 测试难以取得预期效果.如何减少推测并行测试中的

© 中国科学院软件研究所 http://www.jos.org.cn

共享数据访问开销,是今后的工作方向之一.

3 结 论

不规则计算给自动识别程序并行性带来了新的问题,传统的相关性分析技术不足以判断拥有间接数组下标的语句对之间的数据依赖关系.为了充分利用数组私有化技术发掘程序中潜在的并行性,本文对数组私有化作出严格的定义,并从间接数组特征和动态测试两个方面提出相应的改进措施,所总结出的具体算法有助于完善与提高现有的并行编译优化技术.

References:

[1] Patterson, D., Hennessy, J. Computer Architecture: a Quantitative Approach. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1996.

[2] Moon, S., So, Byoungro, Hall, M.W. Evaluating automatic parallelization in SUIF. IEEE Transactions on Parallel and Distributed Systems, 2000,11(1):36~49.

[3] Banerjee, U., Eigenmann, R., Nicolau, A., *et al*. Automatic program parallelization. Proceedings of the IEEE, 1993,81(2):211~240.

[4] Zang Bin-yu, Chen Tong, Zhang Yu, *et al*. Dependence-Overlay——an efficient solution for array privatization. Chinese Journal of Computers, 2000,23(1):1~8 (in Chinese).

[5] Moon, S., Hall, M.W. Evaluation of predicated array data flow analysis for automatic parallelization. In: Proceedings of the ACM SIGPLAN Conference on Principles and Practices of Parallel Programming. 1999. 84~95.

[6] Pugh, W. A practical algorithm for exact array dependence analysis. Communications of the ACM, 1992,35(8):102~114.

[7] Huang, T.C., Hsu, P.H. The SPNT test: a new technology for run-time speculative parallelization of loops. In: Proceedings of the 10th International Workshop on Languages for Parallel Computing. 1997. 177~191.

[8] Rauchwerger, L., Padua, D. The LRPD test: speculative run-time parallelization of loops with privatization and reduction parallelization. In: Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation. 1995. 218~232.

[9] Lawrence, T. Implementation of run time techniques in the Polaris Fortran restructurer [MS. Thesis]. Department of Computer Science, UIUC, 1996.

附中文参考文献:

[4] 臧斌宇,陈彤,张瑜,等.相关-覆盖方法——有效的数组私有化方法.计算机学报,2000,23(1):1~8.

Automatic Parallelism Detection for One Kind of Irregular Problems*

LI Jing, ZANG Bin-yu, ZHU Chuan-qi

(*Institute of Parallel Processing*, *Fudan University*, *Shanghai* 200433, *China*)

E-mail: lijing_nj@hotmail.com

http://www.fudan.edu.cn

Abstract: Traditional data dependence analysis focuses on affine subscript, which is not applicable to detect parallelism in irregular problem. In this paper, two analysis techniques for subscripted subscripts are presented. One takes the property of indirect array into accounts, the other uses runtime test based on the strict array privatization definition. Comparison between the existing methods and the new techniques is also given.

Key words: parallelism detection; irregular; array privatization; runtime test

* Received September 14, 2000; accepted March 8, 2001

Supported by the Denfence Science-Technology Key Laboratory Foundation of China under Grant No.JS94.6.1JW0703; the National Research Foundation for the Doctoral Program of Higher Education of China; the Youth Science-Technology Program of Shanghai of China under Grant No.99QD14043