文章编号:1001 - 2486(2004)01 - 0029 - 07

一种新的数组逻辑结构猜想方法*

夏 军,戴华东,杨学军
(国防科技大学计算机学院,湖南 长沙 410073)

摘 要:许多编译优化技术都依赖于数组的逻辑结构,然而在实际的应用中,有相当多的数组是无结构的一维数组,从而妨碍了编译器的优化工作。提出了一种新的数组逻辑结构猜想算法,它能将无结构的一维数组自动变换成具有多维逻辑结构的数组,从而使编译器的优化工作成为可能。首先给出一个引理,指出猜想后的多维数组应满足的基本性质,然后基于该引理给出了猜想数组的逻辑结构应遵循的两条基本规则,最后基于这两条基本规则给出了猜想数组逻辑结构的算法。实验结果验证了所提出的数组逻辑结构猜想算法的有效性。

关键词:数组逻辑结构;访问矩阵;下标表达式向量;偏移向量;维长向量
中图分类号:TP311 文献标识码:A

A New Approach of Guessing Logical Structures of Arrays

XIA Jun, DAI Hua-dong, YANG Xue-jun
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Many powerful parallelizing compiling techniques rely on the logic structures of arrays. However, in real applications, lots of arrays are flat one-dimensional arrays and hence may disable some compiling optimizations. This paper presents a new framework and algorithms for guessing the logical multi-dimensional array structures from the flat one-dimensional arrays automatically, and makes powerful compiling optimizations possible. We first give a lemma that points out the basic property which the guessed multi-dimensional arrays should satisfy, then based on this lemma we give two basic rules that should be followed in guessing logic structures of arrays, and finally based on the two rules we present the algorithms for guessing logic structures of arrays. The experimental results show the algorithms for guessing the logical multi-dimensional array structures are effective.

Key words: logical structures of arrays; access matrix; subscript expression vectors; offset vectors; subscript range vectors

当前,许多编译优化技术都依赖于数组的逻辑结构(如数组的多维结构),并且对程序中数组的逻辑结构信息知道得越多,编译器对程序的优化可能性就越大。对于分布式共享存储的机器,经常需要通过改变程序中数组的存储布局来优化程序的空间局部性[1,2],然而如果数组不具有多维结构,一般则不能应用数据变换来对其进行空间局部性优化。对于消息传递的分布式存储机器,并行编译器需要决定数据的划分方法,即对于每一个数组,编译器需要根据某种算法计算出将数据空间映射到处理器空间的映射函数[3,4],然而这些算法都依赖于数组的多维结构。另外,如果一维数组的下标表达式比较复杂,那么对其进行相关性分析则需要求解一个带限定条件的较复杂的等式,但如果该数组能被变换成多维数组,那么对其进行的相关性分析则转化为求解多个带限定条件的较简单的等式,从而简化了相关性分析[5]。在许多实际应用中[6,7],有相当一部分数组并不具有多维结构而只是线性的一维数组,因而编译器很难对它们进行优化。但通过数组逻辑结构猜想,可以将之变换为具有多维逻辑结构的数组,从而使编译器可以对它们进行局部性优化,决定较优的数据分布以及简化相关性分析等优化步骤。

一维数组所隐含的多维逻辑结构的多样性,以及变换后多维数组所必须满足的一些限制条件,使得要想找到一个通用的算法能对任何数组进行逻辑结构猜想是很困难的,国外在这方面的相关研究也很

* 收稿日期:2003 - 08 - 21
基金项目:国家自然科学基金重点资助项目(69933030);国家杰出青年科学基金资助项目(69825104)
作者简介:夏军(1976—),男,博士生。

少。Maslov[5] 提出了一种有效简洁的相关性测试方法,即将一个带限定条件的复杂等式拆分为多个带限定条件的简单等式,并通过对这多个简单等式的求解来完成相关性测试。Maslov 的方法虽然对猜想数组的逻辑结构具有指导和借鉴作用,但他并未给出如何确定猜想后多维数组的下标表达式以及新数组定义的具体方法,从而未真正给出将一维数组变换成多维数组的方法。Cierniak 和 Li[8] 提出了一种将无结构的一维数组变换成多维数组的数组逻辑结构猜想算法,该算法是通过求解步长向量*、映射向量**、访问矩阵和偏移向量的顺序来确定猜想后多维数组的下标表达式和数组定义的,但是该算法限定了原数组下界为0,并假定步长向量中各个非零元素的绝对值都不相同。另外为了算法应用成功,该算法还要求步长向量必须满足 在去掉其中的零元素并对剩余的非零元素按绝对值的大小从小到大排序后,新的步长向量中前一个元素能整除后一个元素(第一个元素除外)。所以上述限制条件使得 Cierniak 和 Li 所提出的数组逻辑结构猜想算法可以处理的数组种类有限。例如,他们的方法对图1(a) 中的数组 B 不适用,因为在数组 B 的步长向量中,存在非零元素的绝对值相同的情况。本文提出了一种新的去除了上述限制条件的数组逻辑结构猜想算法。由于本文所提出的数组逻辑结构猜想算法能处理数组下界不为零的情况,步长向量中存在绝对值相同的非零元素的情况,以及前面所提到的新的步长向量中前一个元素不能整除后一个元素的情况,从而使得该算法的适用面更广。最后,通过对一组基准测试程序的测试,验证了本文提出的数组逻辑结构猜想算法的有效性。

```
REAL B(1:3300)
DO i1 = 0,7
  DO i2 = 0,2
    DO i3 = 0,6
      DO i4 = 0,i2
        B(360i1 + 80i2 - 3i3 + 80i4 + 400) = …
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

(a) 数组逻辑结构猜想前的代码

(a) The code before array logical structure guess

```
REAL B(0:39,0:8,0:8)
DO i1 = 0,7
  DO i2 = 0,2
    DO i3 = 0,6
      DO i4 = 0,i2
        B(39 - 3i3,2i2 + 2i4,i1 + 1) = …
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

(b) 数组逻辑结构猜想后的代码

(b) The code after array logical structure guess

图1 代码段示例

Fig.1 Example code fragments

1 基本术语

考虑在一个 n 重嵌套循环中对一个 m 维的数组进行访问(在后面,约定用 n 表示循环嵌套重数,m 表示数组维数)。假设数组下标为整仿射下标,迭代向量用 $\boldsymbol{I} = (i_1, \cdots, i_n)^{\mathrm{T}}$ 表示,且 $i_1, \cdots, i_n$ 从左至右分别代表最外层循环索引变量直至最内层循环索引变量。循环上下界为包含该层循环的外层循环索引变量和常量的整仿射函数。基于上述假设,数组访问可以表示为 $\boldsymbol{AI} + \boldsymbol{o}$,$m \times n$ 维矩阵 $\boldsymbol{A}$ 被称作访问矩阵,具有 m 个元素的向量 $\boldsymbol{o}$ 被称作偏移向量[9]。另外,在猜想数组逻辑结构的过程中还用到以下一些基本术语,它们都取自文献[8]。

- 下标表达式向量 $\boldsymbol{S}$:由数组每维的下标表达式组成。下标表达式向量 $\boldsymbol{S}$、访问矩阵 $\boldsymbol{A}$、迭代向量 $\boldsymbol{I}$ 和偏移向量 $\boldsymbol{o}$ 之间的关系为:$\boldsymbol{S} = \boldsymbol{AI} + \boldsymbol{o}$,其中 $\boldsymbol{S} = (s_1, s_2, \cdots, s_m)^{\mathrm{T}}$,$\boldsymbol{A} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mm} \end{pmatrix}$,$\boldsymbol{o} = (o_1, o_2, \cdots, o_m)^{\mathrm{T}}$。
- 维长向量 $\boldsymbol{w}$:定义了数组中各维的维长。

本文假设编译器缺省的存储布局为按列存储(按行存储的讨论是类似的)。我们对嵌套循环进行归一化(normalized)[10],以使每层循环的索引变量都以1为步长从0变化到其上界。如果在归一化后的嵌套

* 步长向量由嵌套循环中每个循环所产生的步长组成。某个循环所产生的步长是指该循环连续两个迭代所访问的数组元素间的距离。例如在图1(a) 中,数组 B 的步长向量为 $(360 \quad 80 \quad -3 \quad 80)^{\mathrm{T}}$。

** 映射向量定义了数组的存储布局,具体定义见文献[8]。

循环中,某层循环的上界为常量,那么它应大于等于 0,否则归一化前的嵌套循环的迭代空间将为空,另外还假设该上界大于等于 1,否则该层循环对应的循环索引变量的取值会始终为 0,因此可以在整个嵌套循环中用常量 0 来替换出现该循环索引变量的地方,从而消去该层循环。如果在归一化后的嵌套循环中,某层循环的上界不为常量,那么它是包含该层循环的外层循环索引变量的函数。设该层循环是第 $j(2 \leqslant j \leqslant n)$ 重循环(由外至内),其上界为 $f_j(i_1,\cdots,i_{j-1})$。令由最外面 $j-1$ 重循环所构成的 $j-1$ 维迭代空间为 G(迭代空间表示迭代向量的变化范围),那么假设该层循环的上界满足 $\min\limits_{(i_1,\cdots,i_{j-1})^{\mathrm{T}}\in G} f_j(i_1,\cdots,i_{j-1}) \geqslant 0$, $\max\limits_{(i_1,\cdots,i_{j-1})^{\mathrm{T}}\in G} f_j(i_1,\cdots,i_{j-1}) \geqslant 1$。

2　猜想数组的逻辑结构

本文所考虑的是具有整仿射下标数组的逻辑结构猜想方法,即将具有整仿射下标的合法的一维数组变换成具有整仿射下标的合法的多维数组,且变换后的多维数组的每一维的下标表达式至少包含一个循环索引变量,每个循环索引变量最多出现在变换后多维数组的其中一个下标表达式中。所谓合法即是指在迭代向量的整个变化范围内,数组都没有越界。对于变换后的多维数组,它不仅应是合法的,而且它也应满足其线性化后的下标表达式与变换前的原一维数组线性化后的下标表达式相同,即:设原一维数组的下标表达式为 s,数组下界为 c,变换后的多维数组从第一维直至最后一维(即从左至右),其下标表达式分别为 $s_1,\cdots,s_m$,数组下界分别为 $L_1,\cdots,L_m$,数组各维的维长分别为 $w_1,\cdots,w_m$,那么则有 $s-c=\sum\limits_{j=2}^{m}\left[(s_j-L_j)\prod\limits_{k=1}^{j-1}w_k\right]+s_1-L_1$。在下面的讨论中,不失一般性地假设猜想前原一维数组的下界为 c,猜想后的多维数组的下界都为 0。

2.1　算法准备

引理 1　给定一个满足本文第 1 节假设条件的 n 重嵌套循环以及其中被访问的一个合法的 m 维数组 B,并且数组 B 的下界都为 0,那么该数组任意一维的下标表达式中的任何一个循环索引变量前的系数的绝对值必小于该维的维长。

基于上述引理,给出以下两条基本规则。首先给定一个满足本文第 1 节假设条件的 n 重嵌套循环以及其中被访问的一个合法的一维数组 B,并假设数组 B 的下标表达式为 $s=\sum\limits_{j=1}^{n}a_ji_j+d$。还假设数组 B 能被猜想成合法的多维数组,并且猜想后的合法的多维数组的下界都为 0,每个循环索引变量最多出现在猜想后多维数组的其中一个下标表达式中。

规则 1　如果在数组 B 的下标表达式中,$\exists k,l,1\leqslant k\leqslant n,1\leqslant l\leqslant n,k\neq l$,有 $|a_k|=|a_l|\neq 0$,那么 i_k 与 i_l 必出现在猜想后的合法的多维数组的同一数组维中。

规则 2　如果在数组 B 的下标表达式中,$\exists k,l,1\leqslant k\leqslant n,1\leqslant l\leqslant n,k\neq l$,有 $|a_k|>|a_l|>0$,且在猜想后的合法的多维数组中,i_k 出现在第 p 维,i_l 出现在第 q 维,那么必有 $p\geqslant q$。

2.2　算法第一部分

给定待猜想的一维数组的下标表达式 $s=\sum\limits_{j=1}^{n}a_ji_j+d$(其中 $a_1,\cdots,a_n$ 不全为零)以及各循环索引变量变化范围的上下界(如果循环索引变量所处循环的上界不是常量,那么可以保守估计其可能的最大变化范围,以求出其下界和上界),为了将其猜想成多维数组,需要确定猜想后多维数组的维数,各维的下标表达式以及各维的维长,并保证各维的下标表达式的变化范围在其维长限制的范围内,以及猜想后多维数组线性化后的下标表达式与猜想前的原一维数组线性化后的下标表达式相同。由于本文假设在猜想后的多维数组中,每一维的下标表达式至少包含一个循环索引变量,并且每个循环索引变量最多出现在一个下标表达式中,所以猜想后的多维数组的维数不会超过原一维数组所含的循环索引变量个数,并且根据规则 1,系数绝对值相同的循环索引变量应位于猜想后多维数组的同一维中,同时根据规则 2,系

数绝对值较大的循环索引变量应位于猜想后多维数组的高维中,所以可以根据循环索引变量前系数绝对值的大小从小到大地对它们排序,并按该顺序依次确定猜想后多维数组第一维至最后一维的下标表达式中的非常量部分。由于系数为零的循环索引变量不在原数组中也不会在猜想后的数组中出现,所以只需对 $a_1,\cdots,a_n$ 中的非零元素按其绝对值的大小从小到大地排序即可。图2给出了将一维数组猜想成多维数组算法的第一部分,即计算猜想后多维数组的维数,各维下标表达式的非常量部分,以及除最后一维外各维的维长。

由于 Cierniak 和 Li[8] 限制了猜想后的多维数组必须满足简单下标条件,即每个下标表达式最多含有一个循环索引变量,且每个循环索引变量最多出现在一个下标表达式中,所以他们的方法对步长向量有额外的限制条件。由于我们的方法没有限制猜想后的多维数组必须满足简单下标条件,所以我们的方法能去除 Cierniak 和 Li[8] 对步长向量所设的限制条件。从图2中的算法可看出,我们尽量使得猜想后的多维数组满足简单下标条件(这能使猜想后的多维数组的维数尽可能地多),但若不能满足,就通过使下标表达式可以含多个循环索引变量的方法来尽量使得原一维数组能被成功地猜想成多维数组。

输入:待猜想的一维数组的下标表达式的非常量部分 $s'=\sum_{j=1}^{n}a_ji_j$($a_1,\cdots,a_n$ 不全为零),循环索引变量 i_j 变化范围的下界 l_j 和上界 u_j($1\leqslant j\leqslant n$)。

输出:猜想后的多维数组的维数 m,第1维至第 m 维下标表达式的非常量部分 $s'_1,\cdots,s'_m$,以及第1维至第 $m-1$ 维的维长 $w_1,\cdots,w_{m-1}$。

将 $a_1,\cdots,a_n$ 中的非零元素按其绝对值的大小从小到大地排序,设非零元素的个数为 f,即求下标 p_j,以使 $\forall 1\leqslant j\leqslant f-1$,有 $0<|a_{p_j}|\leqslant|a_{p_{j+1}}|$。

```
m = 0; accu_max = accu_min = 0; k = h = 1; accu_w = 1;
DO WHILE k ≤ f
    temp_a = |a_{p_k}|;
    DO WHILE k ≤ f AND temp_a = |a_{p_k}|
        IF a_{p_k} > 0 THEN
            accu_min = accu_min + a_{p_k} l_{p_k}; accu_max = accu_max + a_{p_k} u_{p_k};
        ELSE
            accu_min = accu_min + a_{p_k} u_{p_k}; accu_max = accu_max + a_{p_k} l_{p_k};
        ENDIF
        k = k + 1;
    ENDDO
    IF k > f THEN
        s'_{m+1} = Σ_{j=h}^{k-1} a_{p_j} i_{p_j} / accu_w; m = m + 1; RETURN m, s'_1,...,s'_m, w_1,...,w_{m-1};
    ELSE
        g = gcd(a_{p_k},...,a_{p_f}) / accu_w;
    ENDIF
    IF (accu_max - accu_min)/accu_w + 1 ≤ g THEN
        s'_{m+1} = Σ_{j=h}^{k-1} a_{p_j} i_{p_j} / accu_w; w_{m+1} = g; accu_w = accu_w × w_{m+1};
        m = m + 1; h = k; accu_max = accu_min = 0;
    ENDIF
ENDDO
```

图2 数组逻辑结构猜想算法第一部分

Fig.2 The first part of the array logical structure guess algorithm

2.3 算法第二部分

本节将给出数组逻辑结构猜想算法的第二部分,确定猜想后多维数组各维的偏移常量,以使各维下标表达式的变化范围在其所在维限制的范围内,同时也使猜想前和猜想后的数组线性化后的下标表达式的常量部分相同。另外,算法的第二部分还确定了猜想后多维数组最后一维的维长,以使其为满足最后一维下标表达式变化范围的最短维长。图3给出了求解猜想后多维数组各维偏移量以及最后一维维

长的算法。

输入:待猜想的一维数组的数组下界 c,下标表达式的偏移量(即常量部分)d,猜想后多维数组的维数 $m(m \geqslant 2)$,前 $m-1$ 维的维长 $w_1,\cdots,w_{m-1}$,各维下标表达式的非常量部分 s'_j 的变化范围的下界 y_j 和上界 $z_j(1 \leqslant j \leqslant m)$。

输出:猜想后多维数组各维下标表达式的偏移量 $o_1,\cdots,o_m$,最后一维的维长 w_m。

$temp\_l_1 = temp\_u_1 = d - c$;$w_0 = 1$;

DO　$p = 1, m-1, 1$

　$temp\_l_{p+1} = temp\_l_p / w_{p-1} - w_p + z_p + 1$;$temp\_u_{p+1} = temp\_u_p / w_{p-1} + y_p$;

　IF $[temp\_l_{p+1}/w_p, temp\_u_{p+1}/w_p]$ 不含整数 THEN RETURN FALSE;

ENDDO

IF $[temp\_l_m/w_{m-1}, temp\_u_m/w_{m-1}] \cap [-y_m, +\infty] = \varnothing$ THEN RETURN FALSE;

ELSE

　任取某个整数 $\alpha \in [temp\_l_m/w_{m-1}, temp\_u_m/w_{m-1}] \cap [-y_m, +\infty]$;$o_m = \alpha$;

　DO $p = m-1, 2, -1$

　　任取某个整数 $\alpha \in [temp\_l_p/w_{p-1} - \sum_{j=p+1}^{m}\left(o_j \prod_{k=p}^{j-1} w_k\right)$;

　　$temp\_u_p/w_{p-1} - \sum_{j=p+1}^{m}\left(o_j \prod_{k=p}^{j-1} w_k\right)] \cap [-y_p, w_p - z_p - 1]$;$o_p = \alpha$;

　ENDDO

　$o_1 = d - c - \sum_{j=2}^{m}\left(o_j \prod_{k=1}^{j-1} w_k\right)$;

ENDIF

$w_m = z_m + o_m + 1$;

RETURN $o_1,\cdots,o_m,w_m$;

图 3　数组逻辑结构猜想算法第二部分

Fig.3　The second part of the array logical structure guess algorithm

如果数组逻辑结构猜想算法第一部分所返回的猜想后多维数组的维数大于 1,那么再应用数组逻辑结构猜想算法第二部分。如果应用成功,就能将原合法的一维数组猜想成合法的多维数组,并能保证猜想前后数组线性化后的下标表达式相同。

2.4　示例

考虑图 1(a) 中的例子,该嵌套循环已经过归一化处理。根据图 2 给出的数组逻辑结构猜想算法的第一部分,求得图 1(a) 中数组 B 猜想后多维数组的维数 $m = 3$;第一维至第三维的下标表达式的非常量部分分别为 $s'_1 = -3i_3, s'_2 = 2i_2 + 2i_4, s'_3 = i_1$;第一维和第二维的维长分别为 $w_1 = 40$ 和 $w_2 = 9$。因为猜想后多维数组的维数大于 1,所以根据图 3 给出的数组逻辑结构猜想算法的第二部分,求得猜想后多维数组第一、二、三维的偏移量分别为 $o_1 = 39, o_2 = 0, o_3 = 1$,最后一维的维长 $w_3 = 9$。所以猜想后数组 B 的下标表达式变为 $B(39 - 3i_3, 2i_2 + 2i_4, i_1 + 1)$,其新的数组定义为 $B(0:39, 0:8, 0:8)$。

猜想前数组 B 包含 3300 个数组元素,但猜想后数组 B 只包含 3240 个数组元素,这是因为原数组的最后 60 个元素不会在嵌套循环中被访问,所以新数组将不再包含这 60 个元素。数组逻辑结构猜想后的代码如图 1(b) 所示。

前面所讨论的是如何将一维数组猜想成多维数组,实际上前面所讨论的方法同样也能应用于多维数组,即将多维数组的其中某一维猜想成多维。

3　数组多次访问的逻辑结构猜想

同一数组可能在程序中被多次访问,如果分别单独对它们应用逻辑结构猜想算法,那么最后猜想出

来的逻辑结构可能会不一致,这时,必须考虑如何使这些不同的逻辑结构相容。目前对该问题的解决方法是让它们都使用相同的逻辑结构。下面,通过一个例子来阐述该方法的应用过程。考虑图4(a)中的例子。图4(a)是数组逻辑结构猜想前的代码,其中数组 B 在两个嵌套循环中都被访问。如果分别对数组 B 的这两次访问应用猜想算法,那么可以求得数组 B 第一次访问猜想后的数组定义为 $B(0:9,0:9,0:9)$,下标表达式为 $B(i_3,i_2,i_1)$;第二次访问猜想后数组的定义为 $B(0:99,0:9)$,下标表达式为 $B(i_2,i_1)$。从上面可以看出,两次访问猜想后的数组逻辑结构不相同,为了使它们具有相容的数组逻辑结构,有两种可供选择的解决方法:第一种方法是让它们都使用数组 B 第二次访问猜想后的逻辑结构,这样最终新数组的定义将为 $B(0:99,0:9)$,第一次访问的下标表达式将变为 $B(10i_2+i_3,i_1)$,第二次访问的下标表达式保持不变,即为 $B(i_2,i_1)$,猜想后的代码如图4(b)所示;第二种方法是让它们都使用数组 B 第一次访问猜想后的逻辑结构,这样最终新数组的定义将为 $B(0:9,0:9,0:9)$,第一次访问的下标表达式保持不变,即为 $B(i_3,i_2,i_1)$,第二次访问的下标表达式将变为 $B(i_2\%10,i_2/10,i_1)$,猜想后的代码如图4(c)所示。系统地解决逻辑结构相容问题将是我们的下一步工作。

```
REAL B(0:999)
DO i1 = 0,9
  DO i2 = 0,9
    DO i3 = 0,9
      B(100i1 + 10i2 + i3) = …
    ENDDO
  ENDDO
ENDDO
DO i1 = 0,9
  DO i2 = 0,99
    B(100i1 + i2) = …
  ENDDO
ENDDO
```

(a)

```
REAL B(0:99,0:9)
DO i1 = 0,9
  DO i2 = 0,9
    DO i3 = 0,9
      B(10i2 + i3,i1) = …
    ENDDO
  ENDDO
ENDDO
DO i1 = 0,9
  DO i2 = 0,99
    B(i2,i1) = …
  ENDDO
ENDDO
```

(b)

```
REAL B(0:9,0:9,0:9)
DO i1 = 0,9
  DO i2 = 0,9
    DO i3 = 0,9
      B(i3,i2,i1) = …
    ENDDO
  ENDDO
ENDDO
DO i1 = 0,9
  DO i2 = 0,99
    B(i2%10,i2/10,i1) =
…ENDDO
ENDDO
```

(c)

图4 同一数组多次访问的相容逻辑结构猜想

Fig.4 The consistent logical structure guess for multiple accesses of the same array

4 实验结果

对 PERFECT、SPLASH 和 BLAS 基准测试程序包中的16个程序进行了测试验证,表1给出了测试程序中原一维线性数组的个数(用 num1 表示)以及能经猜想成功变换为多维数组的一维线性数组的个数(用 num2 表示)。从表1中可看出,利用本文所提出的算法,测试程序中的绝大部分一维数组能被变换为多维数组,因此我们的算法是有效的。在对 advc、advt、advv 和 advu 中的一维数组进行猜想时,某些数组最后猜想出来的逻辑结构有不一致的情况,但利用第3节介绍的方法使这些数组都使用相同的逻辑结构,就能将它们成功地变换为具有相同逻辑结构的多维数组。

表1 原一维数组以及能被成功猜想的一维数组个数比较

Tab.1 Comparison of the numbers of the original one-dimensional arrays and the one-dimensional arrays that can be guessed successfully

程序名	版本	来源	num1	num2	程序名	版本	来源	num1	num2
advc	Fortran	PERFECT	7	5	cher2k	C	BLAS	3	3
advt	Fortran	PERFECT	6	4	csymm	C	BLAS	3	3
advv	Fortran	PERFECT	8	4	dtrmm	C	BLAS	2	2
advu	Fortran	PERFECT	10	6	dgemm	C	BLAS	3	3
luo	C	SPLASH	1	1	ssyr2k	C	BLAS	3	3
bdiv	C	SPLASH	2	1	strsm	C	BLAS	2	2
bmodd	C	SPLASH	2	2	zher2k	C	BLAS	3	3
toucharray	C	SPLASH	4	3	zsymm	C	BLAS	3	3

为了表明经数组逻辑结构猜想,将一维数组变换为多维数组所带来的好处,我们在一个由4个结点

(每个结点有两个 CPU,结点内部采用共享存储结构,结点间采用分布存储结构) 所构成的集群系统上对程序 advu、bmodd 和 csymm 进行了性能测试,每个测试程序有两个不同的版本:第一个是原程序版本(用 original 表示),第二个是用本文所提出的算法将测试程序中的一维数组变换为多维数组后得到的版本(用 original + ag 表示)。表 2 给出了上述两个版本程序的执行时间。从表 2 中可看出,第二个版本的测试程序的执行效果优于第一个版本的测试程序,这是因为:第一个版本程序中的许多一维数组含有多个循环索引变量,所以编译器很难对其进行数据分布,因此其运行时有较多的远地访存开销;而第二个版本程序中含有多个循环索引变量的一维数组都被变换成了多维数组,所以编译器能很容易对其进行数据分布,从而使得运行时的远地访存开销尽可能地少,并且对于程序 bmodd,还能对猜想后的多维数组进行数据变换,以使其局部性得到优化。

表 2　数组逻辑结构猜想前和猜想后的测试程序执行时间比较

Tab.2　Comparison of executing times for test programs before and after array logical structure guess

处理器数	advu		bmodd		csymm	
	original	original + ag	original	original + ag	original	original + ag
2	1.419	0.978	10.2414	4.838	1.198	0.8389
4	10.08	0.5018	47.53	2.449	0.77	0.4792
8	27.249	0.2514	134.43	1.249	0.79	0.2102

注:表 2 中执行时间的单位是 s。advu、bmodd、csymm 的问题规模参数分别为 128、512、256。

5　结束语

数组逻辑结构猜想使得并行编译器对程序进行局部性优化,决定较优的数据分布的可能性增加,并且它也能简化相关性分析。本文提出了一种新的比 Cierniak 和 Li 所提出的猜想算法适应面更广的数组逻辑结构猜想算法,同时也对同一数组的多次访问的逻辑结构猜想方法进行了讨论,最后通过实验验证了本文所提出的数组逻辑结构猜想算法的有效性。

参 考 文 献:

[1] Leung S A. Array Restructuring for Cache Locality[R]. Technical Report UW-CSE-96-08-01, Dept. Computer Science and Engineering, University of Washington, 1996.

[2] Kandemir M, Choudhary A, Shenoy N, Banerjee P, Ramanujam J. A Hyperplane Based Approach Foroptimizing Spatial Locality in Loop Nests[C]. In: Proc. 1998 ACM International Conference on Super-computing (ICS '98), Melbourne, Australia, 1998: 69 – 76.

[3] Anderson J M, Lam M S. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines[C]. In Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation, Albuquerque, NM, 1993: 112 – 125.

[4] Bau D, Kodukula I, Kotlyar V, Pingali K, Stodghill P. Solving Alignment Using Elementary Linear Algebra[C]. In Proceedings of the Seventh Annual Workshops on Languages and Compilers for Parallel Computing, Ithaca, NY, 1994: 46 – 60.

[5] Maslov V. Delinearization: An Efficient Way to Break Multiloop Dependence Equations[C]. In the Proceedings of SIGPLAN '92 Conference on Programming Language Design and Implementation, 1992: 152 – 161.

[6] Berry M. The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers[J]. International Journal of Supercomputer Applications, 1989, 3(3): 9 – 40.

[7] Callahan D, Porterfield A. Data Cache Performance of Supercomputer Applications[C]. In Proc. of Supercomputer '90, New York, NY, 1990: 564 – 572.

[8] Cierniak M, Li W. Recovering Logical Data and Code Structures[R]. Technical Report 591, Department of Computer Science, University of Rochester, 1995.

[9] Wolf M, Lam M. A data Locality Optimizing Algorithm[C]. In: Proc. ACM SIGPLAN Conf. Prog. Lang. Des.& Impl., 1991: 30 – 44.

[10] Wolf M. High Performance Compilers for Parallel Computing[M]. Redwood City: Addison-Wesley Publishing Company, 1996.