文章编号:1001 - 9081(2007)05 - 1251 - 03

基于线性不等式的数据划分方法的优化

董春丽,赵荣彩,杜 澎,王 峥
(信息工程大学 信息工程学院,河南 郑州 450002)
(dclldh@sina.com)

摘 要:计算和数据划分是串行程序并行化时所要解决的一个重要问题,如何对程序中引用的数据进行合理的分布以最大限度的发现程序的并行性减少数据重分布的通信开销,是并行编译优化的重点。给出的数据和计算的优化分解方法是基于 Anderson-Lam 的分解算法上改进得到的。根据 Anderson-Lam 的算法得到数据和计算划分后,以线性不等式的形式表示,然后通过分析循环嵌套中能够进行边界冗余的只读数组,重新构造数据划分不等式,根据此不等式进行数据分布,实现具有边界冗余的只读数组的数据划分,有效地减少了数据收发的通信量。

关键词:并行编译;数据分布;计算划分;线性不等式;边界冗余
中图分类号: TP301 **文献标识码**:A

An optimizing data decomposition method based on linear inequalities

DONG Chun-li, ZHAO Rong-cai, DU Peng, WANG Zheng
(*College of Information Engineering, University of Information Engineering, Zhengzhou Henan* 450002, *China*)

Abstract: Maximize parallelism and minimizing communication by increasing the locality of data references are important issues for achieving high performance on large-scale parallel machines. This paper presented an optimizing method for data calculation and computation decomposition. The authors have improved the method by Anderson and Lam. Use linear inequality system to describe data and computation decomposition and reconstruct linear inequality system after adding read-only array with boundary replication. Communication load can be reduced effectively.

Key words: paralleling compiler; data decomposition; computation decomposition; linear inequalities; boundary replication

0 引言

使用先进的并行化编译技术,自动将串行程序转换为等价的能在并行计算机上高效运行的并行程序,是克服并行计算机编程困难、软件移植困难的主要手段,是充分发挥高性能并行计算机系统潜能的有效途径[1]。并行化编译器可以从多方面对并行程序的效率进行优化。在本文将讨论如何对能够进行边界冗余的只读数组进行合理分布以减少通信提高程序的并行度。

论文的主要工作已经在我们所开发的并行化编译器 SW-KAP 上实现的,串行程序经过并行识别后,基于符号系数不等式理论,自动实现具有边界冗余的只读数组[2]的分布,并以注释的形式提供给后端代码生成遍,生成具有边界冗余的 SPMD 程序。实验结果表明,该方法能显著提高并行化程序的执行性能,减少边界通信。

1 线性不等式系统及傅立叶消元

并行化编译需要对循环迭代空间、数组空间和处理器空间进行复杂的分析、变换和优化。通过把循环嵌套的迭代、数组的数据元素和处理器之间的关系表示为多维整数空间[3]的映射关系就可以很方便地对其实施数学运算从而进行分析与优化。由于这些多维整数空间往往呈现为多维凸多面性质的空间,因此就可以以凸多面体来表示迭代、数据和处理器空间[3]。

定义 1[3]

对于 n 层嵌套循环:

Do $i_1 = l_1(v_1,\cdots,v_m), h_1(v_1,\cdots,v_m)$
  Do $i_2 = l_2(v_1,\cdots,v_m,i_1), h_2(v_1,\cdots,v_m,i_1)$
  …
    Do $i_n = l_n(v_1,\cdots,v_m,i_{n-1}), h_n(v_1,\cdots,v_m,i_{n-1})$

迭代 $I^n(v_1,\cdots,v_m)$ 可以以凸多面体来表示:

$$I^n(v_1,\cdots,v_m) = \left\{(i_1,\cdots,i_n)\in\Im \,\middle|\, \bigwedge_{k=1,\cdots,n}^{\ } \begin{matrix} i_k \geqslant l_k(v_1,\cdots,v_m,i_1,\cdots,i_{k-1}) \\ i_k \leqslant h_k(v_1,\cdots,v_m,i_1,\cdots,i_{k-1}) \end{matrix}\right\}$$

其中 $v_1,\cdots,v_m$ 是整型常量,$i_1,\cdots,i_n$ 是循环索引变量,l_k,h_k 是线性函数。

定义 2[3]

m 维数组 $a[u_1]\cdots[u_m]$ 构成其数组空间可以形式化定义为:

$$A = \{\vec{a} = (a_1,\cdots,a_m)\in A \mid \forall k = 1,\cdots,m;\ 0 \leqslant a_k \leqslant u_k\}$$

数组下标空间 A 是一个 m 维的空间,用于表示 m 维数组的数据分布空间,其空间中的任意一点为一个数组元素。

定义 3[3]

q 维处理器,每一维的最大编号大小为 $r_i(0 \leqslant i \leqslant q)$。该空间可以形式化定义为:

收稿日期:2006 - 11 - 24;**修订日期**:2007 - 01 - 16 **基金项目**:河南省杰出人才创新基金资助项目(0521000200)

作者简介:董春丽(1970 -),女,河南郑州人,博士研究生,主要研究方向:计算机软件、并行编译; 赵荣彩(1957 -),男,河南洛阳人,教授,博士生导师,主要研究方向:计算机软件、并行编译; 杜澎(1976 -),男,山西人,硕士研究生,主要研究方向:计算机软件、并行编译; 王峥(1985 -),女,北京人,主要研究方向:计算机软件、并行编译.

$$P = \left\{ \vec{p} = (p_1, \cdots, p_q) \in P \mid \forall k = 1, \cdots, q; 0 \leqslant p_k \leqslant r_k \right\}$$

处理器空间 P 是一个 q 维的空间,用于表示 q 维的处理器,其空间中的任意一点表示一个处理器的编号。

不论是迭代空间、数组空间还是处理器空间,在进行并行化分析和优化时都要求得到每一维变量的上下界范围。投影是求得每一维变量上下界范围的主要方法之一。不等式系统 S^n 消去第 n 维变量 v_n 等价于投影凸多面体 S^n 到变量 v_n 所代表的坐标轴。投影 n 维凸多面体到 $n-1$ 维凸多面体可以通过傅立叶消元的方法得到。在具体实现过程中,基于不等式的傅立叶消元已经作为基本库函数添加到 SW-KAP 中。

在 SW-KAP 中,线性不等式系统表示的不论是迭代、数组还是处理器空间都由两部分构成:1) 名字符号表(*columns*)表示变量名;2) 抽象的线性不等式类(*lin_ineq*)用来表示变量的范围。

对于例 1 中表示的简单循环,下面给出如何将迭代空间表示为线性不等式并对线性不等式消元得到迭代变量的范围。

例 1:
```
for i1 = 0 to 3
    for i2 = 0 to min(i1,6)
        X[i1] = X[i2] + 1;
```

例 1 中表示的循环迭代空间可以表示为线性不等式(1):

$$\begin{cases} 0 \leqslant i_1 \leqslant 3 \\ 0 \leqslant i_2 \leqslant \min(i_1, 6) \end{cases} \tag{1}$$

将不等式(1)转换为不等式(2):

$$\begin{cases} i_1 \geqslant 0 \\ 3 - i_1 \geqslant 0 \\ i_2 \geqslant 0 \\ i_1 - i_2 \geqslant 0 \\ 6 - i_2 \geqslant 0 \end{cases} \tag{2}$$

将不等式(2)写成矩阵式形式 $\boldsymbol{M}\vec{i} + \vec{c} \geqslant \vec{0}$,得到不等式:

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 1 & -1 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 3 \\ 0 \\ 0 \\ 6 \end{bmatrix} \geqslant \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

其中 $\vec{c}$ 是常数向量,$\vec{i}$ 是迭代向量,矩阵 $\boldsymbol{M}$ 是 $\vec{i}$ 的系数。

将不等式(3)进一步变换为不等式(4):

$$\begin{bmatrix} 0 & 1 \\ 3 & -1 \\ 0 & 0 \\ 0 & 1 \\ 6 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ i_1 \\ i_2 \end{bmatrix} \geqslant \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

将不等式(4)表示为 SW-KAP 规定的格式,得到线性不等式(5):

$$\begin{array}{lccc} & 1 & i_1 & i_2 \\ columns: & 0 & 1 & 0 \\ & 3 & -1 & 0 \\ & 0 & 0 & 1 \\ lin\_ineq: & 0 & 1 & -1 \\ & 6 & 0 & -1 \end{array} \tag{5}$$

其中名字符号表(*columns*)是由变量 i_1,i_2 构成的向量。

调用傅立叶消元(FME)可以从不等式系统里得到某一变量的上下界约束范围。在具体实现中,设计 *filter* 函数(*filter_thru*, *filter_away*)从不等式系统中得到其子集。

2 具有边界冗余的只读数组的划分方法

2.1 问题的提出

分析测试集 ppopp 中的 sor.f 程序片断见图 1(将数据规模变小为 4)。

```
      DO 11 I = 1,4
         DO 10 J = 1,4
            a(J,I) = b(J,I)
10       CONTINUE
11    CONTINUE
      DO 21 I = 2,4
         DO 20 J = 2,4
            b(J,I)=a(J,I)+a(J-1,I)+a(J+1,I)+a(J,I-1)+a(J,I+1)
20       CONTINUE
21    CONTINUE
```

图 1 sor 程序片段

a11 a12 a21 a22 P00	a13 a14 a23 a24 P01
a31 a32 a41 a42 P10	a33 a34 a43 a44 P11

图 2 未作边界冗余的数据分布

图 1 例子中的嵌套循环 I、J 两层循环都可以并行执行,如果将计算空间按照两维划分进行块划分,假定物理处理器为 2×2 的结构,则数组 a 的数据存放如图 2 所示,此时对数组 a 的访问存在对不齐现象,我们无法简单的通过引入偏移或错位等方式来完全消除通讯,如果想减少通信,必须引入数组边界冗余。引入边界冗余后,每个节点上需存储比简单的划分略多的数据,从所有的处理机整体来看,数据存在着重叠部分。每个节点中的存贮的数组 a 的数据可以理解为如图 3 中的形式。

a11 a12 a13 a21 a22 a23 a31 a32 a33 P00	a12 a13 a14 a22 a23 a24 a32 a33 a34 P01
a21 a22 a23 a31 a32 a33 a41 a42 a43 P10	a22 a23 a24 a23 a33 a34 a42 a43 a44 P11

图 3 具有边界冗余的数据分布

2.2 只读数组的数据划分方法

只读数组的冗余是常用的用于提高并行机性能的方法,引入此方法是在保证原有读写数据的并行度并且不引入额外的数据重分布通信开销的基础上进行的只读数组的冗余。本文中对只读数组进行优化是在原有 Anderson-Lam[1] 数据和计算划分算法基础上放宽了对只读数组的约束,即在求解基本数据和计算划分方式时只考虑写数组对划分的影响,算法的改进部分是对数组存在如图 1 中所示读写对不齐的情况时,通过对数组的边界部分进行少量冗余来保证程序执行结果的正确,冗余数据的范围通过分析只读数组的输入依赖来获得。

只读数组是指在某个循环中对此数组只存在读操作,但在其他循环中允许存在对此数组的写操作。图 1 中的数组 a 就是这样一种只读数组。对于只读数组如何通过边界冗余的方法来保证程序执行出正确的结果,我们从分析数据的依赖关系出发来解决这类问题。

基本的依赖关系分析问题是:给定两个程序变量,判定它们在程序执行中是否会访问相同的存储单元。

定义 循环嵌套 L 中的语句 T 的一个实例 $T(j)$ 和语句 S

的一个实例 $S(i)$,如果存在一个存储单元 M,且满足下述条件,则称语句 T 的实例 $T(j)$ 依赖于语句 S 的实例 $S(i)$:

(1) $S(i)$ 和 $T(j)$ 都引用(读或写)M;

(2) 在程序串行执行时,$S(i)$ 在 $T(j)$ 之前执行;

(3) 在程序串行执行时,从 $S(i)$ 执行结束到 $T(j)$ 开始执行前,没有其他对 M 的写操作。

语句 S 和 T 不一定要是不同的语句,但条件(2)要求实例 $S(i)$ 和 $T(j)$ 是不同的。

根据读/写的不同情况,通常定义四类数据依赖关系:

(1) 如果 $S(i)$ 写 M 而 $T(j)$ 读 M,则 $T(j)$ 流依赖于 $S(i)$;

(2) 如果 $S(i)$ 读 M 而 $T(j)$ 写 M,则 $T(j)$ 反依赖于 $S(i)$;

(3) 如果 $S(i)$ 和 $T(j)$ 都写 M,则 $T(j)$ 输出依赖于 $S(i)$;

(4) 如果 $S(i)$ 和 $T(j)$ 都读 M,则 $T(j)$ 输入依赖于 $S(i)$。

只读数组的边界冗余所处理的依赖关系是上述定义中所描述的第四种依赖关系即输入依赖。

定义　边界通信

对于数据 A 的一个划分,在循环嵌套 L 中,会出现处理机需要相邻处理机的数据的情况,需要进行处理器之间的数据段边界的传递,称之为边界通信。

对于存在边界通信的只读数组,我们可以通过数组的边界冗余来解决,数据冗余的范围,通过只读数组的输入依赖关系来求出。

当程序中出现输入依赖时,我们将数组的多个读访问函数 $f(\vec{i}) = F\vec{i} + \vec{k}$ 记录下来,分析数组的读访问函数中的常向量 $\vec{k}$,边界冗余中冗余宽度的选取由只读数组访问函数中的常向量 $\vec{k}$ 来决定。对问题的分析过程中,我们假定数组在每一维上向上和向下冗余的宽度分别存于向量 $\overrightarrow{up}$ 和 $\overrightarrow{low}$ 中(初始值为零向量),向量的长度由只读数组的维数确定。假定缺省情况下数组的划分方向为正向,分析读数组访问函数中的常向量 $\vec{k}$,取各维中最小的负数作为向下冗余的宽度保存于向量 $\overrightarrow{low}$ 中,各维中最大的正数作为向上冗余的宽度保存于向量 $\overrightarrow{up}$ 中。根据向上向下的冗余宽度和数组划分函数,在对应的数组的可划分的各个维上对边界数据进行冗余。在虚拟处理器到物理处理器的块映射不等式中表现为将分到物理处理器的数组片段的上下界范围扩大。

我们所做的并行化编译器是基于斯坦福大学的 SUIF[4] 基础上实现的,在程序实现时首先要在数据和计算划分遍中分析哪些数组在循环中只进行了读操作,对只读数组加只读注释,根据所加的只读注释,取出只读数组的访问函数,从访问函数中,求出数组冗余的上下界宽度,将此值作为进行冗余带来的对数组划分上下边界的扩大合并到对应的数据划分不等式中,后端代码生成遍根据数据和计算划分不等式对数据和计算进行划分生成 MPI 并行代码。

3　一个具体例子的分析

在图 1 中,数组 a 为只读数组,可进行边界冗余,由于 I、J 两维均可并行,所以数组 a 在行和列上均需进行冗余,由访问函数可知向上和向下的冗余宽度值均为 1,将此值合并到数据划分不等式中,对数组划分的上下界进行扩大,即可完成边界冗余,即在数组的行和列的边界上分别冗余一行和一列数组元素实现边界冗余如图 3 中所示。具体实现步骤如下所示:

第一步:从程序的循环中找到只读数组,并增加只读注释。

所加注释形式如下:

["read_replicated": <a, 0>]

第二步:找到数组的读访问函数,计算出边界冗余中向上向下的冗余宽度。

在求数据划分 $d(\vec{a}) = D\vec{a} + \vec{\delta}$ 时,判断数组是否为只读数组,如果是只读数组,则求只读数组访问函数的偏移向量 $\vec{k}$,根据 N 个读访问函数的偏移量 $\vec{k}$,分别找到数组在每一维上正向和负向的最大偏移。

只读数组 a 的 5 次读访问函数的偏移向量 $\vec{k}$:

(0　0),(0　-1),(0　1),(-1　0),(1　0)

在偏移向量 $\vec{k}$ 的每一维正向和负向上分别求最大偏移保存于向量 $\overrightarrow{up}$ 和 $\overrightarrow{low}$ 中,即数组在每一维上可进行的最大冗余量:

$\overrightarrow{low} = (-1　-1)$

$\overrightarrow{up} = (1　1)$

第三步:在虚拟处理器到物理处理器映射的不等式中,实现数组的边界冗余。

数据划分的不等式系统可表示如下:

$$BLK(i) * PROC(i) \leqslant LINEAR(i) + OFFSET(i) < BLK(i) * PROC(i) + BLK(i) \tag{6}$$

不等式中 $LINEAR(i)$ 表示数据分布函数的第 i 行,$OFFSET(i)$ 表示数据分布函数第 i 行的偏移值,$BLK(i)$ 表示进行块分布时的第 i 维虚拟处理器到物理处理器映射时的块大小,$PROC(i)$ 为第 i 维上的处理器。

为了实现边界冗余,对上述不等式上下界进行放大,对不等式的上下界按前面求得的冗余量 $\overrightarrow{up}$ 和 $\overrightarrow{low}$ 进行放大。为如下形式所述:

$$BLK(i) * PROC(i) + \overrightarrow{low} \leqslant LINEAR(i) + OFFSET(i) < BLK(i) * PROC(i) + BLK(i) + \overrightarrow{up} \tag{7}$$

将此不等式表示为第 2 节中线性不等式所需的形式,可以等价为下述三个不等式的表示形式:

$$(BLK(i) - OFF - 1 + \overrightarrow{up}) + BLK(i) * PROC(i) + (-1) * LINEAR >= 0 \tag{8}$$

$$(OFF + \overrightarrow{low}) + (-1) * BLK(i) * PROC(i) + LINEAR >= 0 \tag{9}$$

$$BLK(i) * PROC(i) >= 0 \tag{10}$$

在用此方法对只读数组进行边界冗余时,可以引出的一个问题就是数组的边界会发生越界,解决此问题的方法就是在数组划分不等式中增加数组上下界不等式对其边界进行约束来解决越界问题。

$$LINEAR(i) + OFFSET(i) < ub(i) \tag{11}$$

$$LINEAR(i) + OFFSET(i) \geqslant lb(i) \tag{12}$$

式(11)、(12)中的 $ub(i)$、$lb(i)$ 分别为只读数组在第 i 维上的上下界。

第四步:按修改后的约束不等式用 SW-KAP 中的命名符号系数不等式表示出来,并将不等式表示的结果以注释中的立即数链的形式输出给后端,后端代码生成遍根据此注释中的数组划分边界值生成数据划分代码。

进行边界冗余后进行数据划分时,划分的上下界根据冗余宽度向量 $\overrightarrow{low}$ 和 $\overrightarrow{up}$ 进行了扩大。根据对只读数组进行边界冗余划分所生成的并行代码执行结果与串行结果一致。

只读数组的边界冗余算法减少了数据重分布引起的通信,提高了并行化程序的执行性能,减少了处理器间的通信开销,图 4 给出了未作边界冗余的算法和论文中算法所产生的并行化程序在 cluster 机上的运行时间,横坐标为处理器个数,竖坐标为程序执行时间(单位 ms)。(下转第 1263 页)

千兆网。

该试验共对四组文件对进行了测试:

表 1　试验测试结果

	处理器数				
	1(串行)	2	4	6	8
hhctrl.ocx(s)	14 160	8 479	5 057	3 316	2 372
Msdtcprx.dll(s)	10 560	6 104	3 667	2 439	1 740
tcpip.sys(s)	9 300	5 962	3 536	2 263	1 162
comctl32.dll (s)	14 780	9 067	4 910	3 230	2 243

表 1 记录了对这四组文件进行串行比较和并行比较算法所耗费的时间。这四组文件分别为:

(hhctrl. ocx Windows XP SP2 514kB,
hhtctrl. ocx(MS05-001) Windows XP SP2 527kB)
(msdtcprx. dll Windows 2000 SP4 690kB,
msdtcprx. dll(MS06-018) Windows 2000 SP4 709kB)
(tcpip. sys Windows XP SP2 350kB,
tcpip. sys(MS06-032) Windows XP SP2 351kB)
(comctl32. dll Windows 2000 SP4 537kB,
comctl32. dll(MS06-057) Windows 2000 SP4 517kB)

试验中,分别提取了四组补丁前后的文件,并在不同处理器数的机群上测试运行并行的结构化比较程序所用的时间。结果显示,与串行算法相比(在 1 个节点上运行),采用并行算法后大大减少了程序的运行时间。

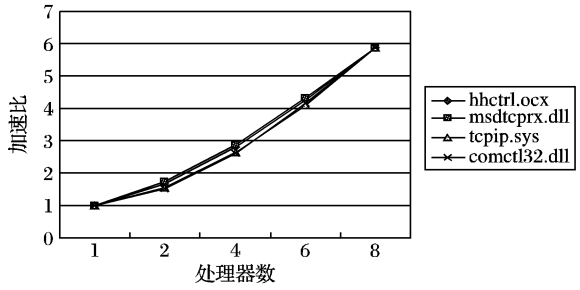

图 2　并行算法的加速比曲线

图 2 和图 3 是其对应的加速比 S_P 和并行效率 E_P。加速比的定义为 $S_P = T_S/P_S$,其中 T_S 为单 CPU 串行执行时间,P_S 为并行执行时间。并行效率E_P定义为$E_P = S_P/P * 100\%$,P是处理器个数。从以上结果可见,随着处理器台数的增加,算法求解所需要的执行时间就变短,在一定的范围内,并行加速比也越大。采用并行的结构化比较算法以后,程序的执行效率得到了很大的提高。

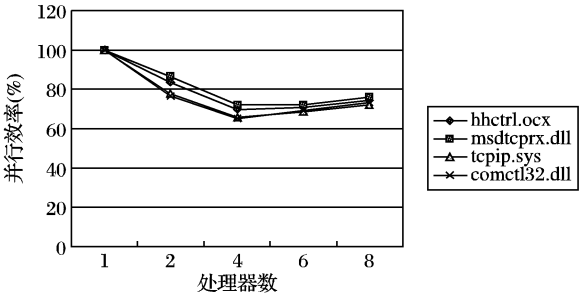

图 3　并行算法的并行效率曲线

4　结语

补丁比对用于对病毒或者木马的变种进行分析,以及分析闭源软件中存在的安全漏洞,同时对于利用其他产品的未公开特性也很有帮助。本文提出了一种基于全局地址空间编程模型实现的并行结构化比较算法,在该算法的实现过程中,使共享数据尽量由其亲缘进程上进行计算,减少因访问非本地数据而进行的通信开销,同时为了提高处理器的利用率,提出一种动态负载平衡策略,减少了处理器空闲情况的出现。

参考文献:

[1] WANG Z, PIERCE K, MCFARLING S. Bmat-a binary matching tool [A]. 2nd ACM Workshop on Feedback-Directed Optimization[C]. 1999.

[2] FLAKE H. Structural comparison of executable object[EB/OL]. http://www. sabre-security. com/files/dimva_paper2. pdf, 2004 - 10 - 10.

[3] 陈国良. 并行算法的设计与分析[M]. 北京: 高等教育出版社, 1994.

[4] ZHU YC. Communication Optimizations for Parallel C Programs[EB/OL]. http://www. sable. mcgill. ca/ ~ hendren/ftp/pldi98. ps. gz, 1998 - 10 - 10.

[5] CHEN WY, BONACHEA D. A Performance Analysis of the Berkeley UPC Compiler[EB/OL]. http://www. gwu. edu/ ~ upc/publications/performance. pdf, 2003 - 10 - 10.

(上接第 1253 页)

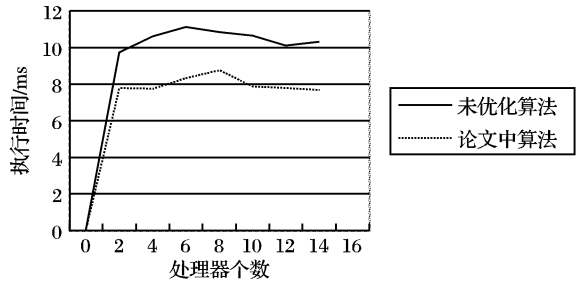

图 4　执行时间比较

4　结语

在自动并行化编译中,数据和计算的划分问题是影响程序并行化的重要因素,如何对数据和计算进行合理的划分以减少通信量是并行化编译中需要重点解决的问题。本文中所介绍的优化算法是在基于斯坦福大学的 SUIF[4] 基础上实现的。通过本文中的边界冗余,可以减少边界数据的通信,减少并行程序执行时间。通过对基准测试程序 NAS 进行测试发现,经过对只读数组进行冗余的优化,可以使得程序的并行度更大,程序并行执行时间更短,提高了串行程序并行化的效率。

参考文献:

[1] ANDERSON JM, LAM MS. Global optimizations for parallelism and locality on scalable parallel machines[A]. Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation[C]. 1993. 112 - 125.

[2] BECKMANN O, KELLY PHJ. A linear algebra formulation for optimizing replication in data parallel programs[A]. Proceedings of the 12th International Workshop on languages and Compilers for Parallel Computing[C]. Yorktown Heights, NY, USA, 2000. 100 - 116.

[3] SAMAN PA. Parallelizing Compiler Techniques Based on Linear Inequalities[D]. PhD thesis. California: Stanford University, 1997.

[4] AMARASINGHE SP, ANDERSON JM, LAM MS, *et al.* An overview of the SUIF compiler for scalable parallel machines[A]. Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing[C]. San Francisco, CA, 1995. 662 - 667.