第35卷 第11期 计 算 机 工 程 2009年6月
Vol.35 No.11 Computer Engineering June 2009
·软件技术与数据库· 文章编号：1000—3428(2009)11—0035—03 文献标识码：A 中图分类号：TP314

消除冗余通信的暴露集求解算法

刘晓娴，赵荣彩，梁 玲
(解放军信息工程大学信息工程学院，郑州 450002)

摘 要：针对分布存储结构计算机系统在并行编译过程中存在的问题，提出一种消除冗余通信的暴露集求解算法，分别采用数组数据流分析和自干扰分析技术对嵌套循环中的流依赖和输入依赖进行分析，从而得到暴露集空间。仿真实验结果表明，将该算法所得结果作为后端生成数据分布通信代码的依据，可有效消除冗余通信，提高系统整体性能。

关键词：并行编译；冗余通信；暴露集

Exposed-set Calculating Algorithm for Eliminating Redundant Communication

LIU Xiao-xian, ZHAO Rong-cai, LIANG Ling
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】Aiming at problems existed in process of parallel compilation for distribution of storage structure in computer system, an exposed-set calculating algorithm to eliminate redundant communication is proposed, which finds exposed-set by analyzing flow dependences and input dependences with array data-flow analysis and self-interfere analysis technologies. Simulation experimental results show that using the results of the algorithm to generate data distribution communication code can eliminate the redundant communication effectively, and improve the performance of the whole system.

【Key words】parallel compilation; redundant communication; exposed-set

1 概述

传统的依赖关系识别只能确定需要通信的位置，不能确定所需通信数组空间的范围，因此，在进行数据分布时只能保守地分布整个数组，称为全数组分布。这样做的优点是不会漏掉任何数据，保证正确性，但全数组空间庞大而数组的实际访问空间较小，使用其生成的通信循环会产生大量冗余通信。文献[1]使用计算划分信息与计算循环结合所得迭代空间加上循环中数组读访问函数来生成数据分布循环，只分布那些被使用的数组元素，简称计算循环分布。与全数组分布相比，这种方法能在一定程度上减少通信，然而当循环中数组引用之间存在某些依赖关系时，采用计算循环分布会造成部分数组元素重复收发，造成冗余的通信开销，在某些情况下效率甚至比全数组分布还差。要解决这个问题，就要将循环中在读引用前已被修改的元素从其访问的数组元素中去除，这样得到的数组元素集合才是没有冗余的数据分布空间。

本文以线性不等式系统为基础，提出并实现一种基于数组数据流分析技术[2]、自干扰分析技术[2]和傅里叶消元法[3]求解嵌套循环暴露集[4]的算法，以消除冗余通信。

2 冗余通信的消除

2.1 相关概念

(1)线性不等式系统及其消元

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，由于用来表示它们的多维整数空间往往呈现为多维凸多面体性质的空间，因此可以凸多面体来表示迭代、数据和处理器空间[5]，这样就可以通过对凸多面体投影进行消维来获得每一维变量的上下界范围。线性不等式系统 S^n 消去第 n 维变量 v_n 就等价于投影凸多面体 S^n 到变量 v_n 所代表的坐标轴，而投影操作可以通过傅里叶消元(Fourier-Motzkin-Elimination)来完成，因此，通过傅里叶消元就可以实现线性不等式系统的消元，从而得到每一维变量的上下界范围，这就可以进行并行化分析和优化。

(2)自干扰分析

自干扰分析用来处理读数组引用的自反输入依赖关系。存在自反输入依赖关系的读引用为自干扰读引用；不存在自反输入依赖关系的读引用为非自干扰读引用。读引用对所有数组数据的最后一次访问迭代可构成其非自干扰迭代空间。

(3)数组数据流分析

数组数据流分析是对数组引用的精确依赖关系分析，该分析方法为读写引用建立一颗 LWT 二叉树。该树根据 R_r 与 R_w 的访问函数构建依赖关系等式，结合循环迭代空间可以给出两者之间存在流依赖关系的迭代空间。本文使用 LWT 算法[2]实现数组数据流的分析。

2.2 求解暴露集进行冗余通信消除

为消除数据分布时的冗余通信，需要求解在嵌套循环内部使用但在外部赋值的那些数组元素的集合，即暴露集。为便于后端生成数据分布通信代码时使用，本文用一个二元组

基金项目：国家“863”计划基金资助项目(2006AA01Z408)
作者简介：刘晓娴(1985－)，女，硕士研究生，主研方向：计算机软件与并行编译；赵荣彩，教授、博士生导师；梁 玲，副教授
收稿日期：2008-12-20 **E-mail：**liuxiaoxian0410@sina.com

(A_x, I_{ue})表示单个读引用的暴露集空间，其中，A_x 是读引用的访问函数；I_{ue} 是其暴露迭代空间，使用不等式组来表示。所有读引用在其暴露迭代空间中访问的数组元素的集合构成整个循环的暴露集空间，后端在生成通信代码时将暴露迭代空间作为通信代码循环索引，并使用读引用的访问函数来进行数据分布。求解暴露集需要对每个读引用分别进行分析，求出其暴露迭代空间：(1)当循环中存在流依赖时，读引用在其流依赖迭代空间中访问的数组元素都在循环内部赋值，不属于暴露集，因此，将其去除。使用 LWT 算法对流依赖进行分析可得到该读引用流依赖于写引用的迭代空间，将其从暴露迭代空间中去除。(2)当多个读引用访问同一外部赋值的数组元素时，若将这个元素同时添加到访问它的所有读引用暴露集中，在进行数据分布时，就会出现对同一数组元素的多次分布，冗余仍然存在。要消除这种冗余，就必须保证不同读引用的暴露集之间交集为空，这需要对读引用之间的输入依赖进行分析。与流依赖相比，读引用之间的输入依赖除引用位置不同之外，其他属性均相同，完全满足 LWT 算法的输入条件，利用 LWT 算法可以得到 R_r^1 输入依赖于 R_r^2 的迭代空间。因此，将使用 LWT 算法得到的某一读引用输入依赖于其他读引用的迭代空间从其暴露迭代空间中去除，就能保证不同读引用的暴露集交集为空。(3)当单个读引用自身对同一数组元素进行多次访问时，会导致对同一数组元素进行多次分布。当出现这种情况时，称这个读引用有自反输入依赖。使用自干扰分析可以得到读引用的非自干扰空间，在非自干扰空间中，读引用对每个数组元素的访问只出现一次。

经过上面的分析可知，求解暴露集需要对自反输入依赖关系、不同读引用之间的输入依赖关系和流依赖关系分别进行分析。本文提出一种新的求解暴露集的算法，算法流程如图 1 所示。

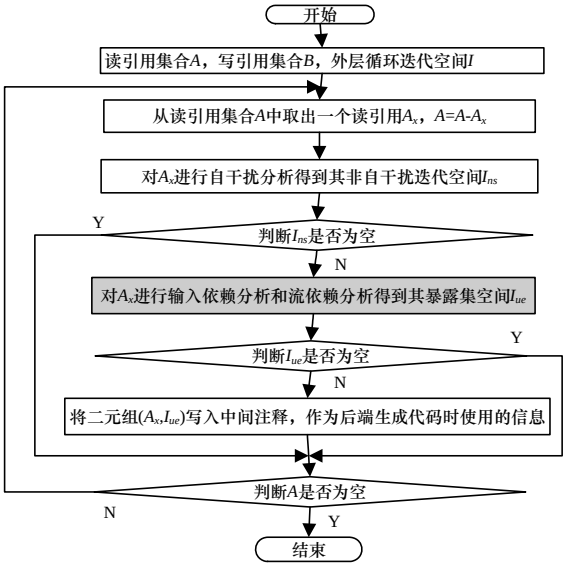

图 1　暴露集求解算法流程

在图 1 中，“输入依赖分析和流依赖分析”是求解暴露集算法中的关键部分，在该部分的代码实现中所涉及的主要数据结构包括命名线性不等式(*named_lin_ineq*)、命名线性不等式链(*iter_list_nli*)和引用链(*reference_list*)。该算法实现过程如下所示：

算法 1 输入依赖分析和流依赖分析

输入：(1)读引用 A_x 的下标表达式，用命名线性不等式存储及读引用集合 $A=\{A_i|0\leqslant I\leqslant m\}$和写引用集合 $B=\{B_j|0\leqslant j\leqslant n\}$，用引用链存储。(2)$n$ 层嵌套循环迭代空间 $\vec{I}=(i_1,i_2,\cdots,i_l)$，用命名线性不等式存储。(3)$A_x$ 的非自干扰迭代空间 I_{ns}，用命名线性不等式链 $\vec{I}_{ns}$ 存储。

输出：读引用 A_x 的暴露集空间

步骤：

(1)若 I_{ns} 为空，转步骤(3)；若 I_{ns} 不为空，建立命名线性不等链 I_{lw}，以存储 A_x 的暴露集空间 I_{ue}，初始化 $I_{ue}=I_{ns}$。

(2)对所有读引用 A_i，$0\leqslant I\leqslant m$ 执行以下分析过程：

1)使用传统依赖分析测试 A_x 是否依赖于 A_i；

2)若 A_x 依赖于 A_i 则使用 LWT 算法分析 A_x 和 A_i 之间的输入依赖关系，得到依赖关系不等式 I_{i0}；

3)对 I_{i0} 使用消元算法消去其中 A_i 索引变量对应的列，得到 A_x 依赖于 A_i 的迭代空间的线性不等式表示 I_{rr}；

4)$I_{ue}=I_{ue}-I_{rr}$，若 $I_{ue}=\Phi$，则转步骤(4)。

(3)对所有写引用 B_j，$0\leqslant j\leqslant n$ 执行以下分析过程：

1)使用传统依赖分析测试 A_x 是否依赖于 B_j；

2)若 A_x 依赖于 B_j 则使用 LWT 算法分析 A_x 和 B_j 之间的流依赖关系，得到依赖关系不等式 I_{i0}；

3)对 I_{i0} 使用消元算法消去其中 B_j 索引变量对应的列，得到 A_x 依赖于 B_j 的迭代空间的线性不等式表示 I_{rw}；

4)$I_{ue}=I_{ue}-I_{rw}$，若 $I_{ue}=\varnothing$，则转步骤(4)。

(4)算法结束。

3　实例分析

示例代码段如下：

```
#define N 1024
#define M 512
int i,j,k;
int a[M][N][5];
int b[N][M];
for(i=0;i<M;i++)
for(j=0;j<N;j++)
for(k=0;k<5;k++)  {
a[i][j][k]=i+j+k;
b[j][i]=b[j][i]+i-j+k;  }
for(i=0;i<N;i++)
for(j=1;j<M;j++)
for(k=1;k<5;k++)  {
b[i][j]=b[i][j-1]+a[1][i][k-1]+a[1][i][k];  }
```

其中，第 2 个循环的计算使用了第 1 个循环计算产生的 *a* 和 *b* 2 个数组的数据，但 2 个循环的数据划分无法对齐，因此，需要在 2 个循环间进行数据分布。如果采用全数组分布，则数组 *a*,*b* 的通信空间分别为 $\begin{cases}0\leqslant i<M\\0\leqslant j<N\\0\leqslant k<5\end{cases}$ 和 $\begin{cases}0\leqslant i<N\\0\leqslant j<M\end{cases}$，未使用的元素也被分布，有冗余；如果采用计算循环分布，则 3 个数组引用的通信空间都是 $\begin{cases}0\leqslant i<N\\1\leqslant j<M\\1\leqslant k<5\end{cases}$，其中，数组 *b* 的部分元素在引用前已被修改，有冗余，且嵌套循环中未使用索引 *j* 和 *k* 产生的自反输入依赖会导致数组 *a*,*b* 的大量元素重复分布，另外，*a*[1][*i*][*k*-1]和 *a*[1][*i*][*k*] 2 个读引用之间存在输入依赖，分布的数组元素也会有重复。

使用图 1 所示的算法流程可求解第 2 个循环的暴露集空间，以消除数据分布时的冗余通信。其中，读引用 *b*[*i*][*j*-1]的暴露集空间为 $\begin{cases}0\leqslant i<N\\1\leqslant j<2\\4\leqslant k<5\end{cases}$，读引用 *a*[1][*i*][*k*-1]和 *a*[1][*i*][*k*]

的暴露集空间分别为 $\begin{cases} 0 \leqslant i < N \\ M-1 \leqslant j < M \\ 1 \leqslant k < 5 \end{cases}$ 和 $\begin{cases} 0 \leqslant i < N \\ M-1 \leqslant j < M \\ 1 \leqslant k < 2 \end{cases}$ 。

表 1 中对该例的全数组分布、计算循环分布和本文提出的暴露集数据分布进行比较。从表中可以看出，当使用暴露集进行数据分布时，通信空间最小。

表 1　3 种数据分布方式的比较

	通信空间大小		
	全数组分布	计算循环分布	暴露集分布
数组 *a*	$5MN$	$8N(M-1)$	$5N$
数组 *b*	MN	$4N(M-1)$	N

4　性能测试

4.1　测试平台

测试目标机是 SunWay 集群，由 20 个节点组成，每个节点配置 2 个主频为 2.8 GHz 的 Xeon 处理器、4 GB 内存、16 MB 二级缓存、128 KB 一级缓存。节点间通过百兆交换机相连。操作系统为 Red Hat Linux 7.2 2.96-118.7.2 smp，MPI 编译器为 MPICH 1.2.6。

4.2　测试方法

为对比 3 种不同的数据分布方式给并行程序性能带来的影响，在测试平台上分别对采用这 3 种分布方式的并行程序进行测试。为更好地测试本文所提算法消除冗余通信的能力，在程序运行时，应确保并行的每个进程运行在不同节点上。SunWay 集群每个节点配置 4 个处理器，因此，在测试时对节点做了指定，使每个节点只运行一个进程。

4.3　测试结果和性能分析

测试结果见图 2。从图中可看出，使用本文的算法能够消除全数组分布和计算循环分布的冗余通信，加速比显著提升。对其他循环内数组引用间有流依赖和输入依赖的并行程序的测试结果表明本文算法对该类程序的性能都有所提升。

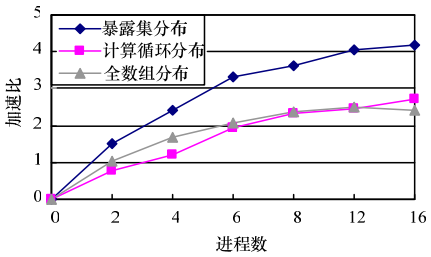

图 2　优化前后的加速比

5　结束语

冗余通信的开销是影响并行程序执行效率的关键因素。本文提出一种消除数据分布中冗余通信的暴露集求解算法，实验结果表明，该算法能有效提高循环内数组引用间有流依赖和输入依赖的并行程序的性能，但目前对数组引用间依赖关系的分析局限于循环内部，如何使用全局的数组数据流信息对通信进行优化是下一步工作。

参考文献

[1] 杜　彭, 赵荣彩, 董春丽. MPI 通信代码自动生成算法[J]. 计算机应用, 2007, 27(3): 759-761.

[2] Maydan D E, Amarasinghe S P. Array Data-flow Analysis and Its Use in Array Privatization[C]//Proc. of ACM Conf. on Principles of Programming Languages. [S. 1.]: ACM Press, 1993.

[3] Schrijiver A. Theory of Linear and Integer Programming[M]. [S. 1.]: Wiley Chichester Press, 1986.

[4] Gu Junjie, Li Zhiyuan. Efficient Interprocedural Array Data-flow Analysis for Automatic Program Parallelization[J]. IEEE Trans. on Software Engineering, 2000, 26(3): 244-261.

[5] Saman P A. Parallelizing Compiler Techniques Based on Linear Inequalities[D]. California, USA: Stanford University, 1997.

编辑　陈　文

（上接第 34 页）

表 2　3 种中间件通信延迟比较表　　ms

		最大值			最小值			平均值		
		RTI	EVSM	ACE	RTI	EVSM	ACE	RTI	EVSM	ACE
响应速度测试	第 1 组	29.212	6.091	24.338	1.004	2.411	0.981	6.475	2.710	1.853
	第 2 组	22.976	6.806	20.987	1.134	2.449	0.986	8.662	3.161	2.185
	第 3 组	25.576	13.134	24.325	1.167	2.371	0.982	7.505	2.689	2.261
	第 4 组	166.959	5.176	28.987	1.731	2.484	0.998	43.606	2.808	2.214
负载测试	第 1 组	26.814	77.093	20.495	1.341	2.216	1.003	8.966	10.664	7.233
	第 2 组	26.253	97.410	28.313	1.508	2.235	1.031	9.141	10.992	2.192
	第 3 组	27.319	99.278	43.213	1.597	2.239	1.987	9.661	16.746	8.341
	第 4 组	31.152	111.649	57.912	4.235	2.404	1.997	10.585	19.709	9.345
延迟抖动	第 1 组	19.532	13.994	50.495	0.934	0.385	1.003	5.966	4.619	5.233
	第 2 组	21.149	12.312	29.413	1.054	0.096	1.031	5.614	4.397	2.692
	第 3 组	21.063	15.894	44.414	1.149	0.286	0.987	7.452	3.607	5.341
	第 4 组	23.138	19.907	67.900	1.338	0.115	0.957	7.815	2.371	5.345

从表 2 可以看出：(1)在网络空闲情况下，小信息量通信延迟的最大值可能比大信息量通信延迟的最大值还大，而且也存在它们最小值的倒位问题，这主要是由于操作系统处理的不确定性造成的。(2)在数据包大小相同情况下，ACE 和 EVSM 的延迟较 RTI 小，这是由于 RTI 除了提供数据传输外还维护了庞大的其他服务降低通信效率，因此如果虚拟试验未用到 RTI 的其他服务，没有必要以 RTI 作为底层中间件。(3)EVSM 在三者中延迟抖动最小，可见其采用的精简通信协议实时性较好。(4)在负载环境下，统计数据普遍较同数据包空闲情况下的数据有所增大，但 RTI 和 ACE 较 EVSM 增大不多，所以，在负载很大的情况下，EVSM 的稳定性较差。

6　结束语

中间件技术作为一种解决应用之间的互联和互操作的技术，目前有很多种，而在分布式交互仿真系统中如何选择恰当的中间件技术作为不同仿真平台之间交换数据并且能够满足一定的实时性方面，目前还没有一个合适的选择标准。本文通过对 3 种中间件在网络响应等性能方面进行测试和分析，得出它们在处理网络数据方面的性能差异，希望能为分布式交互仿真系统在选择中间件时做一定参考。

参考文献

[1] 黄柯棣. 系统仿真技术[M]. 长沙: 国防科学技术大学出版社, 1998.

[2] 张亚崇, 孙国基, 严海蓉, 等. 基于 HLA/RTI 的分布式交互仿真中数据分发管理的研究[J]. 系统仿真学报, 2004, 16(6): 1284-1287.

[3] 陆艳洪, 马捷中, 杜承烈, 等. 基于VSM中间件的分布式试验与测试系统的通信技术[J]. 计算机应用研究, 2003, 20(7): 26-28.

[4] 崔桂香, 丁晓明. ACE 框架在网络通讯软件设计中的应用研究[J]. 北京电子科技学院学报, 2004, 12(4): 55-59.

编辑　陈　文