

一种数据并行中的群通信优化策略

王 珏 胡长军 张纪林 李建江

(北京科技大学信息工程学院 北京 100083)

摘 要 群通信是影响大规模数据并行系统效率的关键因素,其主要发生在程序不同阶段间的数组重分布与循环划分后的数组重映射这两种情况.在一次通信中显著影响群通信效率常被忽视的因素是消息冲突和消息长度不一致.因为它们会导致进程间大量的空闲等待时间.然而以前的研究要么不能完全避免消息冲突,要么针对某些特殊情况.对此,提出了在数组分布为 Block_Cyclic(k)情况下的一种更具有普遍适用性的通信调度策略 CSS.通过证明表明该策略能使一个通信步内的消息互不冲突且消息长度尽量相等.从而最小化通信调度生成时间和实际通信时间.最后的测试结果也表明,与传统的通信优化算法和 MPI_Alltoallv 实现相比,CSS 策略使得通信效率得以明显提高.

关键词 并行编译;数据并行;组通信;数组重分布;分布内存

中图法分类号 TP311

An Optimized Strategy for Collective Communication in Data Parallelism

WANG Jue HU Chang-Jun ZHANG Ji-Lin LI Jian-Jiang

(School of Information Engineering, University of Science and Technology Beijing, Beijing 100083)

Abstract Collective communication significantly influences the performance of data parallel applications. It is required often in two situations: One is array redistribution from phase to phase; another is data remapping after loop partition. Nevertheless, an important factor that influences the efficiency of collective communication is often neglected: When there is node contention and difference among message lengths during one particular communication step, a larger communication idle time may occur. In previous works, researchers can't completely avoid communication conflict and focus on some special cases. This paper is devoted to develop an universal and efficient communication scheduling strategy (CSS) concerning with the situation where array distributions are Block_Cyclic(k). Base on the proof for the recursive theorem of communication table elements, this strategy generates a communication scheduling table so that each column is a permutation of receiving node number in each communication step. And the messages with the close size are put into a communication step as near as possible. This indicates that the strategy not only avoids inter-processor contention, but it also minimizes real communication cost in each communication step. Finally, experimental results show that CSS has better performance than the general method and the implementation of MPI_Alltoallv.

Keywords parallel compiling; data parallelism; collective communication; array redistribution; distributed memory

收稿日期:2006-10-23;最终修改稿收到日期:2007-10-09. 本课题得到国家“八六三”高技术研究发展计划项目基金(2006AA01Z105)、国家自然科学基金(60373008)和教育部重点基金(106019)资助. 王 珏,男,1981年生,博士研究生,主要研究方向为并行计算与并行编译技术. E-mail: ncepu5@163.com. 胡长军,男,1963年生,博士,教授,博士生导师,主要研究方向为并行计算与并行编译技术、并行软件工程、网络存储体系结构、数据工程与软件工程. 张纪林,男,1980年生,博士研究生,主要研究方向为并行计算与并行算法. 李建江,男,1971年生,博士,副教授,主要研究方向为并行计算、并行编译与多线程技术.

1 引言

当前大多数并行应用系统都是采用基于分布式内存的数据并行范例. 群通信在数据并行程序中普遍存在并且是影响其执行效率的关键因素. 群通信主要发生在以下两种情况: 程序的不同阶段间的数组重分布和循环划分后的数组重映射. 许多数据并行编程语言, 例如 HPF^[1]、Vienna Fortran^[2] 和 p-HPF^[3] 等, 都是通过在分布内存下提供数组分布指导支持全局命名空间(global name space). 在这些语言中, 程序员通过指导语句来指定数组在处理器间的分布模式. 根据“拥有者计算”原则, 编译器利用这些分布信息和循环中数组赋值语句来产生通信集. 许多时候由于通信代价过高而导致并行效率很低, 甚至出现负加速比. 针对这些典型问题, 越来越多的研究者将通信优化的研究重点放在如何最小化通信集生成的时间^[4-5]、重叠通信与计算^[6] 以及重分布数组上, 不过他们却忽略了由于结点间消息冲突和一次通信中消息长度不一样所带来的大量通信开销. 另外, 通过对消息进行调度减轻通信代价的研究有: Park 等人^[7] 在考虑数据传输、通信调度和索引计算的基础上提出了一个针对某些特例减少通信总时间的算法, 不过它只是针对某些特例. Desprez 等人^[8] 给出了一个最小化消息冲突的算法, 但该算法仍然不能避免通信冲突. Yuan 等人^[9-10] 提供了一个基于 MPI_Alltoallv 实现的通信调度算法, 但是其运行时的开销很大. Guo 等人在以前工作的基础上^[11] 给出了关于一维数组的重分布通信调度策略, 其中目标数组在发送处理器组 Q 和接收处理器组 P 上具有不同分布形式^[12], 然而该研究只是着眼于并行程序不同阶段之间数组的重分布问题, 并不能解决循环划分之后的数组重映射问题. 与以往的研究相比, 本文的主要贡献有以下 3 点:

(1) 本文提出的策略首次解决了循环划分后在数组重映射时出现的消息冲突问题. 它避免了数组重映射时结点间的消息冲突, 同时也最小化了每步通信所花费的时间.

(2) 本文提出的策略更具有普遍性, 也适用于不同阶段的数组重分布.

(3) 本文提出的策略引入关于通信表的周期性理论, 大幅度减低通信调度生成的时间. 因此有利于该通信调度算法在编译阶段实现, 不会带来额外的运行时间开销.

综上所述, 本文给出了一种不仅具有普遍适用性, 而且空间和时间花费很小, 可以用于编译时完成的通信调度策略 CSS (Communication Scheduling Strategy). 为了更好地描述问题, 文中使用通信表 COM (COMmunication) table 和调度表 CS (Com-munication Scheduling) table (表的定义在第二部分给出) 来分别描述处理器间的消息交互模式和消息的调度情况. 首先引入通信表的周期性理论来缩短通信调度表的生成时间, 然后给出与通信表内元素递推关系相关的定理和推论的证明. 在以上定理和推论的基础上, 构造出有效的调度算法使调度表的每一列变成接收处理器序号的枚举, 并尽量把长度相近的消息放到同一个通信步内. CSS 策略既避免了处理器间的通信冲突, 同时也最小化了调度表的生成时间和实际通信的总时间. 为了便于算法描述, 本文主要是以一维数组为例进行说明, 该策略很容易扩展到多维数组的情况.

本文第 2 节介绍所要解决的问题并举例进行说明; 第 3 节是本文的核心内容, 论述处理器间的通信调度策略; 第 4 节给出在分布式内存机群上的测试结果并进行分析; 最后是结论及进一步的研究工作.

2 问题的描述

本节首先给出一段代码抽象, 然后在此基础上给出了 MPI_Alltoallv 和传统方法的实现. 以一段 HPF 代码抽象为例 (如图 1 所示). 数组 A 和 B 分别以 $cyclic(x)$ 和 $cyclic(y)$ 分布到不同的处理器组 P 和 Q 上. $a_1 \times i_s + b_1$ 和 $a_2 \times i_s + b_2$ 分别是数组 A 和 B 的访问函数, 且二者均为线性函数. F 是一个数组函数或者数组固有函数 (intrinsic function) (如数组转置操作^[13]). 编译器可以通过数组分布和 FORALL 语句所提供的信息来产生通信集和通信代码^[14]. 图 1 中代码抽象的符号说明详见表 1. 图 2(a) 为图 1 代码抽象的一个实例, 其中数组 A 以 $cyclic(4)$ 分布到 5 个处理器上, 数组 B 以 $cyclic(3)$ 分布到另外的 5 个处理器上, 即 $n_A = 182, n_B = 120, n_P = n_Q = 5, n_s = 60, x = 4, y = 3, a_1 = 3, a_2 = 2, b_1 = 4, b_2 = 1$.

```

1. real A(0:(nA-1)), B(0:(nB-1))
2. !processor P(0:(nP-1))
3. !processor Q(0:(nQ-1))
4. !distribute A(cyclic(x)) onto P
5. !distribute B(cyclic(y)) onto Q
6. FORALL(is=0:(ns-1))
   A(a1 × is + b1) = F(B(a2 × is + b2))

```

图 1 HPF 代码示例

表 1 图 1 中代码抽象的符号说明

符号	符号说明	符号	符号说明
n_A	数组 A 中元素的个数	b_1	数组 A 中访问函数的参数
n_B	数组 B 中元素的个数	a_2	数组 B 中访问函数的参数
n_P	处理器组 P 中处理器的个数	b_2	数组 B 中访问函数的参数
n_Q	处理器组 Q 中处理器的个数	x	数组 A 以 <i>cyclic</i> 分布的参数
n_s	FORALL 语句的循环上限	y	数组 B 以 <i>cyclic</i> 分布的参数
a_1	数组 A 中访问函数的参数	i_s	循环变量

2.1 通信表、通信调度表和通信冲突的定义

下面给出关于通信表、通信调度表和消息冲突

```
real A(0:181),B(0:119)
!processor P(0:4)
!processor Q(0:4)
!distribute A(cyclic(4)) onto P
!distribute B(cyclic(3)) onto Q
FORALL (i_s=0:59)
  A[3i_s+4]=B[2i_s+1];
```

(a) 例1

	COM	P_j				
		0	1	2	3	4
Q_i	0	2	2	3	3	2
	1	3	3	2	2	2
	2	2	2	2	3	3
	3	2	3	3	2	2
	4	3	2	2	2	3

(b) 由图2(a)所产生的通信表

	COM	P_j				
		0	1	2	3	4
Q_i	0	3	3	2	2	2
	1	2	2	3	3	2
	2	3	2	2	2	3
	3	2	3	3	2	2
	4	2	2	2	3	3

(c) *cyclic*(3)到*cyclic*(4)分布的通信表

图 2 例 1 及其通信表和 *cyclic*(3)到 *cyclic*(4)分布的通信表

定义 2 . 调度表(Communication Scheduling Table). 用来描述处理器间通信调度的结果. 如图 4 (a)和图 4(c), 其中 $CS[i][k]=j$ 表示发送处理器 Q_i 在第 k 步发送消息到接收处理器 P_j .

```
接收阶段:
for p=0 to  $n_P-1$  do
   $src=(myid+p)\bmod n_P$ ;
  non-block receive & unpack messages into target local array
  from processor " $src$ ";
end do
发送阶段:
for q=0 to  $n_Q-1$  do
   $dest=(myid+q)\bmod n_Q$ ;
  pack & non-block send messages to processor " $dest$ ";
end do
waitall;
```

图 3 传统通信代码实现

定义 3. 消息冲突(也称通信冲突). 当几个发送处理器同时发送消息到同一个接收处理器时, 接收处理器端便会产生瓶颈. 本文所提出的定理对消除消息冲突具有普遍意义.

本文的通信调度策略对于不同阶段的数据分布(例如, *cyclic*(3)到 *cyclic*(4)分布)同样适用. 图 2(c)给出了 *cyclic*(3)到 *cyclic*(4)重分布的通信表 COM. 其中, 发送处理器组和接收处理器组分别包括有 5 个处理器, 并且数组分别以 *cyclic*(3)和 *cyclic*(4)分布在上面. 只要使图 2(a)中 $a_1=a_2=1$ 和 $b_1=b_2=0$, 即让循环中的赋值语句变为 $A[i]=B[i]$, 其所得的通信表也为图 2(c). 因此, 本文提出的通信调度策略同样可应用于不同阶段的数组重分布, 数组重分布是该策略所解决问题的一个特殊形式.

的定义.

定义 1. 通信表(Communication Table). 用来描述源处理器组与目标处理器组之间的通信模式. 例如, 图 2(b)为图 2(a)中的循环语句根据“拥有者计算”原则划分后所产生的通信表, 表中 $COM(i,j)$ 表示处理器 Q_i 发送到处理器 P_j 的消息大小, 有 $COM(i,j)\neq 0$.

2.2 MPI_Alltoallv 和传统方法的通信调度实现

当发送处理器和接受处理器个数不同时(即 Many_to_Many 通信方式), MPI_Alltoallv 实现虽然有一些处理器的发送数据大小被设置成零, 但是仍然有启动时间上的浪费. 为了解决这一问题, 图 3 中的代码^[11]可以是在实现 Many_to_Many 通信时对 MPI_Alltoallv 的一个简单而实用的替换. 其中 *myid* 是当前执行的进程号. 在发送阶段和接收阶段中, 当前进程采用非阻塞方式进行消息传递. 该实现使得发送处理器在一开始以不同的处理器作为目的处理器, 这在某种程度上能够简单地减少消息冲突.

采用上述方法, 由例 1 的代码, 可以得到图 4(a)中的通信调度表. 图 4(b)是各处理器在图 4(a)调度下的通信状态图. 其中水平长条编有数字 j , 代表发送处理器发送消息到接收处理器 P_j (处理器组 P 中的第 j 个处理器). 一个块是一个消息传输的单元, 连续的一段带有相同数字 j 的块数代表发送处理器发送给接收处理器 P_j 的消息长度. 可以看到图 4(b)中的灰色块代表存在消息冲突的部分, 它们会大幅度降低通信效率. 本文的目的之一就是通过设计有效的算法, 消除这些冲突, 图 4(c)是经过改进后的通信调度表, 图 4(d)是其相应的通信状态图, 可以看出改进后的通信调度有效避免了消息冲突.

为了得到上述改进后的通信调度表, 下一节将给出一个更具有普遍适用性的通信优化策略. 该策略在避免处理器间通信冲突的前提下, 能够最小化调度表的生成时间和实际通信时间.

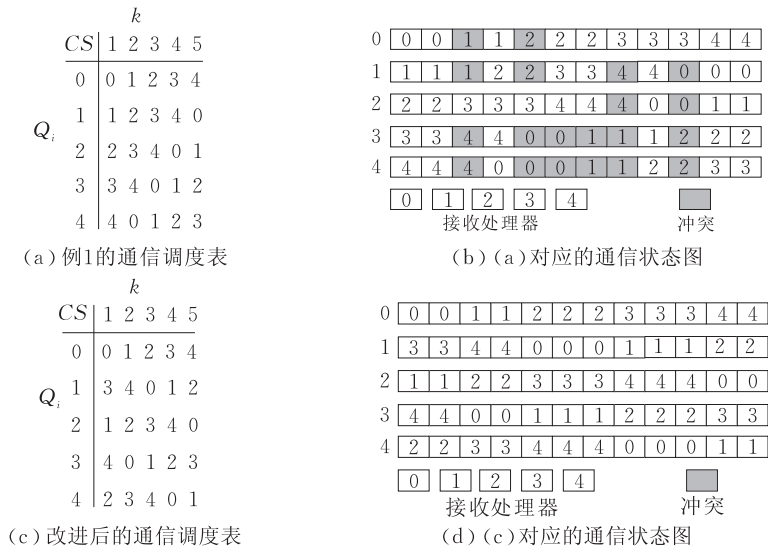


图 4 与图 2(c)对应的通信调度表和通信状态图

3 CSS 通信调度策略

本节仍旧针对图 1 的代码抽象,首先给出其访问模式的周期性公式,然后论述通信表的周期性引理和在一个周期内通信表内元素递推关系的定理及推论,最后根据这些定理和推论分情况给出通信调度算法。

3.1 访问模式的周期性公式

在图 1 的代码抽象的基础上(本节中一些变量,如 n_A, n_B, n_P, n_Q 等的定义可以参考第 2 节中的表 1),假设 $g \in \{a_2 \times c + b_2 \mid 0 \leq c \leq n_s - 1\}$ 是数组 B 中某个元素的全局地址,并且该元素在处理器 Q_i (处理器组 Q 中的第 i 个处理器)上,其相应的数组 A 的元素在处理器 P_j (处理器组 P 中的第 j 个处理器)上,则 i 和 j 由如下公式给出:

i_s	$3 \times i_s + 4$	$2 \times i_s + 1$	Dest	Src	i_s	$3 \times i_s + 4$	$2 \times i_s + 1$	Dest	Src	i_s	$3 \times i_s + 4$	$2 \times i_s + 1$	Dest	Src
0	4	1	P_1	Q_0	23	73	47	P_3	Q_0	46	142	93	P_0	Q_1
1	7	3	P_1	Q_1	24	76	49	P_4	Q_1	47	145	95	P_1	Q_1
2	10	5	P_2	Q_1	25	79	51	P_4	Q_2	48	148	97	P_2	Q_2
3	13	7	P_3	Q_2	26	82	53	P_0	Q_2	49	151	99	P_2	Q_3
4	16	9	P_4	Q_3	27	85	55	P_1	Q_3	50	154	101	P_3	Q_3
5	19	11	P_4	Q_3	28	88	57	P_2	Q_4	51	157	103	P_4	Q_4
6	22	13	P_0	Q_4	29	91	59	P_2	Q_4	52	160	105	P_0	Q_0
7	25	15	P_1	Q_0	30	94	61	P_3	Q_0	53	163	107	P_0	Q_0
8	28	17	P_2	Q_0	31	97	63	P_4	Q_1	54	166	109	P_1	Q_1
9	31	19	P_2	Q_1	32	100	65	P_0	Q_1	55	169	111	P_2	Q_2
10	34	21	P_3	Q_2	33	103	67	P_0	Q_2	56	172	113	P_3	Q_2
11	37	23	P_4	Q_2	34	106	69	P_1	Q_3	57	175	115	P_3	Q_3
12	40	25	P_0	Q_3	35	109	71	P_2	Q_3	58	178	117	P_4	Q_4
13	43	27	P_0	Q_4	36	112	73	P_3	Q_4	59	181	119	P_0	Q_4
14	46	29	P_1	Q_4	37	115	75	P_3	Q_0					
15	49	31	P_2	Q_0	38	118	77	P_4	Q_0	60	184	121	P_1	Q_0
16	52	33	P_3	Q_1	39	121	79	P_0	Q_1	61	187	123	P_1	Q_1
17	55	35	P_3	Q_1	40	124	81	P_1	Q_2	62	190	125	P_2	Q_1
18	58	37	P_4	Q_2	41	127	83	P_1	Q_2	63	193	127	P_3	Q_2
19	61	39	P_0	Q_3	42	130	85	P_2	Q_3	64	196	129	P_4	Q_3
20	64	41	P_1	Q_3	43	133	87	P_3	Q_4	65	199	131	P_4	Q_3
21	67	43	P_1	Q_4	44	136	89	P_4	Q_4	66	202	133	P_0	Q_4
22	70	45	P_2	Q_0	45	139	91	P_4	Q_0			

图 5 图 2 中代码的数组访问模式

$$i = (g \operatorname{div} y) \bmod n_Q \quad (1)$$

$$j = \left[\left(\frac{g - b_2}{a_2} \times a_1 + b_1 \right) \operatorname{div} x \right] \bmod n_P \quad (2)$$

为了最小化 CSS 策略的时间,下面给出了数组分布和赋值语句的周期公式:

$$\operatorname{period}_e^A = [\operatorname{lcm}(n_P \times x, a_1)] / a_1 \quad (3)$$

$$\operatorname{period}_s = \operatorname{lcm}(n_Q \times y, a_2 \times \operatorname{period}_e^A) \quad (4)$$

$$\operatorname{period}_{\text{iter}} = \operatorname{period}_s / a_2 \quad (5)$$

式(3)中的 $\operatorname{period}_e^A$ 表示数组 A 迭代访问模式的周期. 式(4)中的 period_s 是数组 B 索引形式的周期. 式(5)中 $\operatorname{period}_{\text{iter}}$ 是整个循环迭代模式的周期. div 代表整除, \bmod 代表取余, lcm 代表最小公倍数. 通过图 2 中的代码来说明上述周期公式. 数组 A 和 B 分别以 $\operatorname{cyclic}(4)$ 和 $\operatorname{cyclic}(3)$ 分布到处理器组 P 和 Q 上. 图 5 给出了图 2 中代码的数组访问模式. 在图 5 中, 对于赋值语句 $A(3 \times i_s + 4) = B(2 \times i_s + 1)$, 当 $i_s = 0$ 时, $B(1)$ 在 Q_0 上并发送给 P_1 ($A(4)$ 所在的处理器). 在小长方形竖条中的数据(个数为 $\operatorname{period}_e^A = 20$)即代表数组 A 的一个迭代访问模式. $\operatorname{period}_{\text{iter}} = 60 (0 \leq i_s \leq 59)$ 是整个循环迭代模式的周期, 即 Q_i 与 P_j 组合的周期.

3.2 通信表的周期性引理和表内元素递推关系的定理及推论

为了最小化通信调度算法的计算量,本节利用上述数组访问模式的周期性特点给出通信表的周期性引理.

引理 1. 在 COM 表中,对于任意的 (Q_i, P_j) 有 $\operatorname{COM}'(i, j) = u \neq 0$, 如果数组访问正好是一个周期, 那么当数组访问是周期的 k (k 是正整数) 倍时, 即 $k \times \operatorname{period}_{\text{iter}}$, 有 $\operatorname{COM}(i, j) = k \times u$.

证明. 见附录.

由引理 1, 可以知道 COM 表和 CS 表都是周期性变化的. 此处仍然以图 2 中的代码为例, 当 $n_s = \operatorname{period}_{\text{iter}} = 60$ 则有 $\operatorname{COM}(0, 2) = 3$, 进一步如果 $n_s = k \times \operatorname{period}_{\text{iter}} = k \times 60$, 则有 $\operatorname{COM}(0, 2) = k \times 3$. 为了便于说明, 本文将只讨论一个周期的情况 (也就是 $n_s = \operatorname{period}_{\text{iter}}$). 对于多个周期的情况, 调度算法同样适用.

下面将给出通信表内元素递推关系定理和推论, 本文的通信调度算法主要是基于该定理和推论构建的. 为了帮助理解该定理 1 和推论 1, 2, 继续以图 2 中的代码为例进行说明. 当 $\operatorname{COM}(0, 2) = 3$ (相应的 COM 表在图 2(b) 中), 对于数组 B 在一个周期里存在 3 个不同的全局地址: 17, 31, 45, 它们都是

在处理器 Q_0 上并且需要被发到处理器 P_2 . 上述 3 个元素在图 5 中用椭圆虚线标出. 一定存在 $M = 24$ 使得 B 数组的另外 3 个元素 $41 (= 17 + M)$, $55 (= 31 + M)$, $69 (= 45 + M)$ 在处理器 Q_3 上并需要发送给 P_1 , 也就是 $\operatorname{COM}(3, 1) = 3$. 这 3 个元素在图 5 上用实线椭圆标出. 依次类推, 有如下关于数组 B 的全局地址:

$\{(17 + 24 \times k) \bmod \operatorname{period}_s, (31 + 24 \times k) \bmod \operatorname{period}_s, \text{ and } (45 + 24 \times k) \bmod \operatorname{period}_s \mid 2 \leq k < \operatorname{period}_s / M, \operatorname{period}_s = 120, M = 24\}$ 是分别在处理器 $Q_i (i = 1, 4, 2)$ 上并且需要发给处理器 $P_j (j = 0, 4, 3)$, 也就是 $\operatorname{COM}(1, 0) = \operatorname{COM}(4, 4) = \operatorname{COM}(2, 3) = 3$. 通过上述分析, 可以知道通信表内存在元素的递推关系, 下面给出相关定理和推论的证明. 其中部分符号的说明可以参考第 2 节中的表 1.

定理 1. 假设数组 A 和数组 B 分别以 $\operatorname{cyclic}(x)$ 和 $\operatorname{cyclic}(y)$ 分布到 n_P 和 n_Q 个处理器上, 那么 $\operatorname{COM}(i, j) = u \Rightarrow \operatorname{COM}((i + k \times m) \bmod n_Q, (j + k \times n) \bmod n_P) = u, 0 \leq k < \operatorname{lcm}(s, t)$, 其中 $m = \frac{M}{y}, n = \frac{m \times a_1}{a_2 \times x}, M = \frac{\operatorname{lcm}(a_1 \times \operatorname{lcm}(a_2, y), a_2 \times x)}{a_1}, s = n_Q / \operatorname{gcd}(m, n_Q), t = n_P / \operatorname{gcd}(n, n_P)$, 同时有 $\operatorname{COM}(i, j) = u \Rightarrow \operatorname{COM}((i + k \times m') \bmod n_Q, (j + k \times n') \bmod n_P) = u, 0 \leq k < \operatorname{lcm}(s', t')$, 其中 $m' = \frac{m' \times a_2}{a_1 \times y}, n' = \frac{M'}{x}, M' = \frac{\operatorname{lcm}(a_2 \times \operatorname{lcm}(a_1, x), a_1 \times y)}{a_2}, s' = n_Q / \operatorname{gcd}(m', n_Q), t' = n_P / \operatorname{gcd}(n', n_P)$.

证明. 见附录.

在定理 1 中, 给出了两个通信表中元素的递推公式, 其中 gcd 代表最大公约数, $m, n, M, s, t, m', n', M', s', t'$ 均为系数, 其余符号的说明可以在第 2 节中的表 1 中获得. 由递推公式得到的通信表中的元素都相等. 为了更好地说明算法, 下面给出定理 1 的两个推论.

推论 1. $m = \frac{M}{y}, n = \frac{M \times a_1}{a_2 \times x},$
 $M = \frac{\operatorname{lcm}(a_1 \times \operatorname{lcm}(a_2, y), a_2 \times x)}{a_1},$
 $m' = \frac{M' \times a_2}{a_1 \times y}, n' = \frac{M'}{x},$
 $M' = \frac{\operatorname{lcm}(a_2 \times \operatorname{lcm}(a_1, x), a_1 \times y)}{a_2}.$
 $\Rightarrow m = m', n = n'.$

证明. 见附录.

其中的 a_1, a_2, x 和 y 可以参考第 2 节中的表 1.

通过推论 1,可以得到定理 1 中两个递推公式中系数的关系. 由于 $m=m', n=n'$, 所以在通信调度算法中只需要使用第一个递推公式来计算 m 和 n .

推论 2. $n_p \times n_Q = \text{lcm}(s, t) \Rightarrow \text{COM}$ 中的所有元素相等.

证明. 见附录.

$n_p \times n_Q$ 为整个通信表中元素的总个数. 由于其与定理 1 中所能递推到的所有 $\text{lcm}(s, t)$ 个元素个数相同, 所以 COM 中所有元素相等, 此时采用传统的通信调度算法就可以得到好的效果.

3.3 通信调度算法

本节根据 $(\text{gcd}(m, n_Q) = 1) \wedge (n_p / \text{gcd}(n, n_p) \geq n_Q)$ 是否成立将通信调度策略分成算法 1 和算法 2 两部分. 根据定理 1, 当 $(\text{gcd}(m, n_Q) = 1) \wedge (n_p / \text{gcd}(n, n_p) \geq n_Q)$ 成立时, 每一个发送处理器都会在同一步内发送相同长度的消息到不同的接收处理器. 由此给出下面算法.

算法 1. 当 $(\text{gcd}(m, n_Q) = 1) \wedge (n_p / \text{gcd}(n, n_p) \geq n_Q)$ 成立时,

输入:

(1) COM 表.

(2) 数组分布模式和循环变量(例如如图 1 中的 n_A, n_B ,

```
real A(0:181), B(0:119)
!processor P(0:8)
!processor Q(0:2)
!distribute A(cyclic(4)) onto P
!distribute B(cyclic(5)) onto Q
FORALL (i_s=0:59)
  A[3×i_s+4]=B[2×i_s+1];
```

(a) 例 2

COM	0	1	2	3	4	5	6	7	8
0	2	3	2	2	3	2	1	4	1
1	2	3	2	1	4	1	2	3	2
2	1	4	1	2	3	2	2	3	2

(b) 例 2 的 COM 表

CS	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8
1	6	7	8	0	1	2	3	4	5
2	3	4	5	6	7	8	0	1	2

(c) 例 2 的 CS 表

图 6 例 2 及其相应的 COM 表和 CS 表

例 2 的循环如图 6(a) 所示. 数组 A 以 $\text{cyclic}(4)$ 分布到 9 个处理器上, 数组 B 以 $\text{cyclic}(5)$ 分布到另外的 3 个处理器上, 即 $a_1=3, a_2=4, b_1=4, b_2=1, x=4, y=5, n_p=9, n_Q=3$, 其通信表 COM 如图 6(b) 所示. 由 $\text{period}_s=240, m=16, n=15$ 可以得到 $\text{COM}((i+k \times 16) \bmod 3, (j+k \times 15) \bmod 9) = u, 0 \leq k < 3$. 因为 $\text{gcd}(16, 3)=1 \wedge \text{gcd}(15, 9)=3$, 通过算法 1 可以得到图 6(c) 所示的通信调度表.

从定理 1, 能够看出 $s(=n_Q / \text{gcd}(m, n_Q))$ 和 $t(=n_p / \text{gcd}(n, n_p))$ 分别是 $(i+k \times m) \bmod n_Q$ 和 $(j+k \times n) \bmod n_p$ 的周期. COM 表(例如如图 7(b)) 能够被划分成 $s \times t$ 个子矩阵, 每个子矩阵有相同的大小, 即 $n_Q/s \times n_p/t$, 该 COM 表如下所示:

$n_p, n_Q, n_s, x, y, a_1, a_2, b_1$ 和 b_2 , 符号说明参见表 1 和定理 1).

输出:

(1) CS 表.

1. 对于某一个发送处理器 Q_{i_0} (以 Q_0 为例, 即 $i_0=0$), 它所对应 CS 表的第 i_0 行设为接收处理器号的一个任意排列(以 $\text{CS}[0][\] = \{\langle 0 \rangle, \langle 1 \rangle, \dots, \langle n_p-1 \rangle\}$ 为例).

2. 对于 CS 表 i_0 行所对应的每一个接收处理器, 都有 $\text{COM}(i, j) = u \Rightarrow \text{COM}((i+k \times m) \bmod n_Q, (j+k \times n) \bmod n_p) = u, 0 \leq k < \text{lcm}(s, t)$. 通过 CS 表中的第一行元素就可以递推得到整个改进后的 CS 表.

本节通过两个有代表性的例子(图 2 例 1 和图 6 例 2)来说明算法 1 的具体实现细节. 对于例 1, 有 $\text{period}_s=120, m=8, n=9 \Rightarrow \text{COM}((i+k \times 8) \bmod 5, (j+k \times 9) \bmod 5) = u, 0 \leq k < 5$. 由于 $\text{gcd}(8, 5)=1 \wedge \text{gcd}(9, 5)=1$, 初始化 $\text{CS}[0][\] = \{\langle 0 \rangle, \langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 4 \rangle\}$, $\text{CS}[0][1]=0$ 表示在第 1 步中 Q_0 发送消息到 P_0 上, 通过定理 1 可以得到 $\text{COM}(0, 0) = \text{COM}(3, 4) = \text{COM}(1, 3) = \text{COM}(4, 2) = \text{COM}(2, 1)$, 即 $\text{CS}[\] [1] = \{\langle 0 \rangle, \langle 3 \rangle, \langle 1 \rangle, \langle 4 \rangle, \langle 2 \rangle\}$. 同理, CS 的其余列可以被得到, 即 $\text{CS}[\] [2] = \{\langle 1 \rangle, \langle 4 \rangle, \langle 2 \rangle, \langle 0 \rangle, \langle 3 \rangle\}$, $\text{CS}[\] [3] = \{\langle 2 \rangle, \langle 0 \rangle, \langle 3 \rangle, \langle 1 \rangle, \langle 4 \rangle\}$, $\text{CS}[\] [4] = \{\langle 3 \rangle, \langle 1 \rangle, \langle 4 \rangle, \langle 2 \rangle, \langle 0 \rangle\}$, $\text{CS}[\] [5] = \{\langle 4 \rangle, \langle 2 \rangle, \langle 0 \rangle, \langle 3 \rangle, \langle 1 \rangle\}$.

$$\text{COM} = \begin{bmatrix} C_{0,0} & C_{0,1} & \cdots & C_{0,t-1} \\ C_{1,0} & C_{1,1} & \cdots & C_{1,t-1} \\ \cdots & \cdots & \cdots & \cdots \\ C_{s-1,0} & C_{s-1,1} & \cdots & C_{s-1,t-1} \end{bmatrix},$$

$$C_{i,j} = \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n_p/t-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n_p/t-1} \\ \cdots & \cdots & \cdots & \cdots \\ c_{n_Q/s-1,0} & c_{n_Q/s-1,1} & \cdots & c_{n_Q/s-1,n_p/t-1} \end{bmatrix}.$$

这些子矩阵 $C_{i,j}$ 可以被划分成 $f(=s \times t / \text{lcm}(s, t))$ 组, 每个组有 $\text{lcm}(s, t)$ 个相同的子矩阵. 当 $(s=n_Q / \text{gcd}(m, n_Q) \neq n_Q) \vee (t=n_p / \text{gcd}(n, n_p) < n_Q)$ 为真, 也就是 $(\text{gcd}(m, n_Q) \neq 1) \vee (n_p / \text{gcd}(n, n_p) < n_Q)$. 下面给出一种启发式的算法 2 来对该情况进行通信

调度. 算法 2 首先对每个组进行遍历, 随后遍历组内的每个子矩阵, 最后遍历子矩阵内的每个元素, 具体的通信调度算法如算法 2 所示.

算法 2. 当 $(\gcd(m, n_Q) \neq 1) \vee (n_p / \gcd(n, n_p) < n_Q)$ 时 (算法 2 中的符号请参见表 1 和定理 1 中的参数说明),

输入:
(1) COM 表.
(2) 数组分布模式和循环变量.
输出:
(1) CS 表.
1. 初始化阶段
1.1. 消去 COM 表中全零的行和列.
1.2. 将 COM 表中所有的子矩阵 $C_{i,j} (0 \leq i \leq s-1, 0 \leq j \leq t-1)$ 分成 $f = s \times t / \text{lcm}(s, t)$ 个组, 即 $D^h (0 \leq h \leq f-1)$. 每个组 D^h 有 $\text{lcm}(s, t)$ 个相同的子矩阵. 然后将每个组 D^h 内的消息大小按降序排列, 即 $m_0^h \geq m_1^h \geq \dots \geq m_{e-1}^h$.
1.3. 按照 $m_0^h (0 \leq h \leq f-1)$ 的大小将 D^h 按降序排列, 即 $D^0 \geq D^1 \geq \dots \geq D^{f-1}$.
1.4. 为了不失普遍性, 将 D^h 内的 $\text{lcm}(s, t)$ 个子矩阵按定理 1 中的 k 进行排序. 各步按正序和逆序交替遍历 D^h 中的子矩阵 $C_{i,j}$.
2. 生成 CS 表, 使得 CS 表内每一列是接收处理号的枚举, 并且相似大小的消息尽可能地放入一个通信步内.
while (COM 中所有的元素被标记) {
 $nstep = 1$; /* $nstep$ 为步数 */
 for ($h = 0; h < f; h++$)
 /* 遍历所有的组 $D^h (0 \leq h \leq f-1)$ */
 for ($hh = 0; hh < \text{lcm}(s, t); hh++$)

```
real A(0:71), B(0:109)
!processor P(0:5)
!processor Q(0:5)
!distribute A(cyclic(3)) onto P
!distribute B(cyclic(2)) onto Q
FORALL (i_s=0:35)
    A[2 * i_s + 1] = B[3 * i_s + 4];
```

(a) 例3

COM	P_j					
	0	1	2	3	4	5
Q_i	0	1	2	1	2	1
	2	1	2	1	2	1
	3	1	2	1	2	1
	5	1	2	1	2	1

(c) 消去全零行和列的COM表

/* 遍历每个组内 D^h 的子矩阵 $C_{i,j}$ (总共 $\text{lcm}(s, t)$ 个). 对该 $C_{i,j}$ 序列, 当 $nstep$ 为奇数时采用正序遍历, 当 $nstep$ 为偶数时采用逆序遍历. */
for ($r = 0; r < e; r++$) { /* 对每个子矩阵中的元素 (总共有 e 个元素) 进行遍历 */
 $\forall c_{a,\beta} = m_r^h \in D^h$
 u 被赋值为 $c_{a,\beta}$ 在 COM 表中的行坐标;
 v 被赋值为 $c_{a,\beta}$ 在 COM 表中的列坐标;
 $CS[u][nstep] += \langle v \rangle$;
 /* 注意, 在这里允许一个发送处理器在一个通信步中连续发送多个消息, 以确保每个大步内各个处理器所发消息总大小相近. 也就是说, 如果每个大步内各个处理器发送消息的长度不相等, 则将相邻的两个大步用“同步”隔开. */
 标记 $COM[u][v] = \langle X \rangle$, 并消去该元素和该元素所在的行和列;
 如果当前 CS 的第 $nstep$ 列的各消息大小已经对齐, 则中止;
}

恢复消去的元素, 被标记的元素不再参与下次计算.

$nstep++$;

}

通过求例 3 的 CS 表来对算法 2 进行说明. 例 3 的循环与通信表分别如图 7(a) 和图 7(b) 所示. A 以 $cyclic(3)$ 分布到 6 个处理器上, B 以 $cyclic(2)$ 分布到另外的 6 个处理器上, 即 $a_1 = 2, a_2 = 3, x = 3, y = 2, n_p = 6, n_Q = 6, period_e^A = 9, period_s = 108$, 依据定理 1 可以得到 $m = 9, n = 4, s = 2, t = 3$.

COM	P_j					
	0	1	2	3	4	5
Q_i	0	1	2	1	2	1
	1	0	0	0	0	0
	2	1	2	1	2	1
	3	1	2	1	2	1
	4	0	0	0	0	0
	5	1	2	1	2	1

(b) 例3的COM表

CS	k					
	1	2	3	4	5	6
Q_i	0	1	5	3	2,4	0
	2	3	0,4	1	5	2
	3	5	3	0,4	1	2
	5	0,2	1	5	3	4

(d) 例3的CS表

图 7 例 3 及其对应的 COM 表和 CS 表

按算法 2 有:
步骤 1. 首先消去 COM 中全为零的第 2 和第 5 行, 则得到图 7(c). COM 被划分成 6 个子块, 即 $COM =$

$\begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \end{bmatrix}, C_{i,j} = \begin{bmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{bmatrix}$. 进而得到 $f = 1$, $m_0^0 = 2, m_1^0 = 1, D^0 = \{C_{0,0}, C_{1,2}, C_{0,1}, C_{1,0}, C_{0,2}, C_{1,1}\}$.

步骤 2.

如图 8(a)所示,当 $nstep=1$ 时,首先按序列 $\{C_{0,0}, C_{1,2}, C_{0,1}, C_{1,0}, C_{0,2}, C_{1,1}\}$ 执行,从中选出 $m_0^0=2=COM[0][1]=COM[2][3]=COM[3][5]$ 进行标记.然后选择 $m_1^0=1=COM[5][0]=COM[5][2]$ 进行标记.在该步最后得到 $CS[1]=\langle 1 \rangle, \langle 3 \rangle, \langle 5 \rangle, \langle 0, 2 \rangle$.当 $nstep=2$ 时,按逆列

$\{C_{1,1}, C_{0,2}, C_{1,0}, C_{0,1}, C_{1,2}, C_{0,0}\}$ 执行,得到 $CS[2]=\langle 5 \rangle, \langle 0, 4 \rangle, \langle 3 \rangle, \langle 1 \rangle$.依次类推,可以得到:

当 $nstep=3$ 时, $CS[3]=\langle \langle 3 \rangle, \langle 1 \rangle, \langle 0, 4 \rangle, \langle 5 \rangle \rangle$;当 $nstep=4$ 时, $CS[4]=\langle \langle 2, 4 \rangle, \langle 5 \rangle, \langle 1 \rangle, \langle 3 \rangle \rangle$;当 $nstep=5$ 时, $CS[5]=\langle \langle 0 \rangle, \langle \rangle, \langle 2 \rangle, \langle 4 \rangle \rangle$;当 $nstep=6$ 时, $CS[6]=\langle \langle \rangle, \langle 2 \rangle, \langle \rangle, \langle \rangle \rangle$,即为图 8(d)所示.

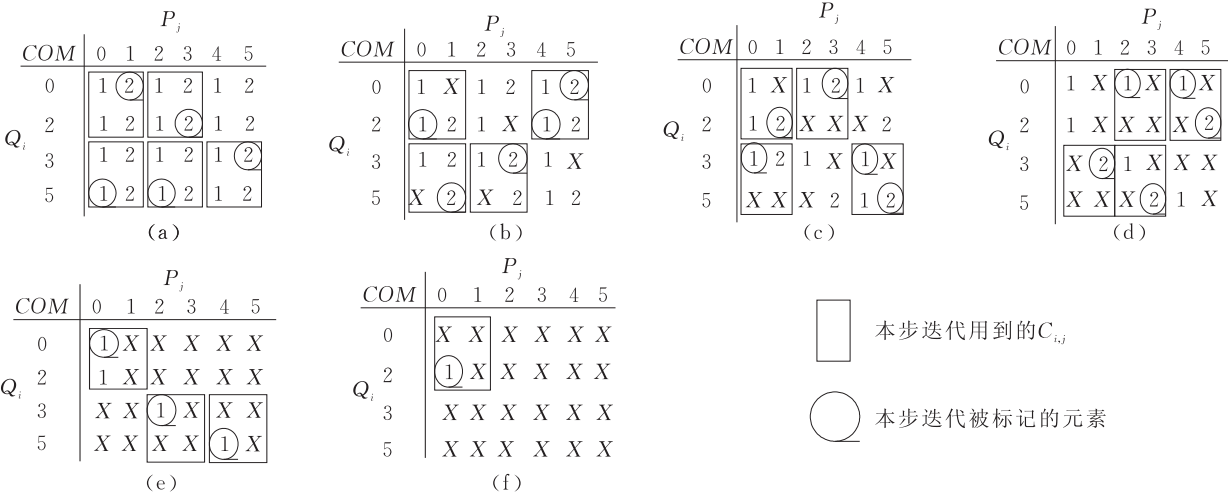


图 8 通信表迭代过程

从上述证明以及算法的分析,可看出算法 1 和算法 2 使调度表的每一列变成接收处理器号的枚举,即避免了处理器间的通信冲突,最小化了调度表的生成时间和实际通信的总时间.

4 测试结果与分析

所有的测试都是基于一个 20 节点的 SMP-Cluster,每节点配置两个 Intel Xeon 3.0GHz/1024KB L2 Cache,2GB 内存,结点通过一个 24 口千兆以太网交换机互联.安装操作系统 Redhat Linux 9.0,内核版本为 2.4.21-smp,串行编译器为 gcc 3.3.2,MPI 运行环境为 Argonne 实验室开发的 mpich-1.2.7,使用 ch_p4 设备.结点程序是在 C 和 MPI 基础上手工编写完成的,并由 MPI_Wtime()来测试执行时间.测试用的数组均为浮点型.测试为了突出分布内存的特点,在每个结点只起一个 MPI 进程.

对于前面的例 1、例 2 和例 3 分别给出下面 3 组实验,其中实验 1 和 2 使用算法 1,测试 3 使用算法 2.当通信量很大时,计算 CS 表所花费的时间可以忽略不计.实验中的 3 条曲线分别代表 3 种不同的通信实现,即 MPI_Alltoallv 实现、传统方式实现 (general)和算法 1 或 2 的实现 (optimized scheduling).从以下测试,可以看出随着消息长度的增加,

算法 1 和 2 的通信效果提高得越来越显著.在测试所用的 mpich-1.2.7 中,MPI_Alltoallv 的实现是各结点采用非阻塞发送 MPI_Isend 一次将所有的消息都发送到网络中,然后接收处理器采用非阻塞接收 MPI_Irecv 和 MPI_Watallv 来将网络中相应的消息接收到本地.对于图 9 和图 10,当发送处理器组和接收处理器组不同时,MPI_Alltoallv 与传统方法和算法 1 相比将会有部分非阻塞发送和接收的启动时间被浪费.因此传统方式最好时要比 MPI_Alltoallv 在效率上提高 20%,但是传统方法并不能很好地避免消息冲突所带来的通信代价.由于算法 1 将发送消息的队列进行重排,使得在 CS 表中每一步的接收处理器号各不相同且所发送消息的长度尽量相等,从而在避免消息冲突的前提下,最小化了通信总时间,因此通信效率能够比传统方法提高 5 %

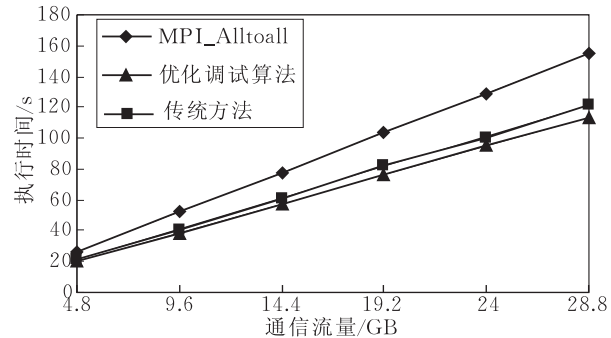


图 9 针对例 1 的通信时间对比

左右。在图 11 中,由于采用传统方式导致消息冲突现象更为严重,因此算法 2 使得通信效率在最好时比传统方式提高了将近 10%,比 MPI_Alltoallv 提高了将近 30%。

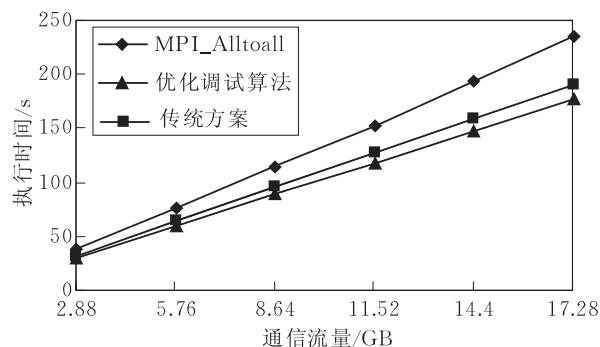


图 10 针对例 2 的通信时间对比

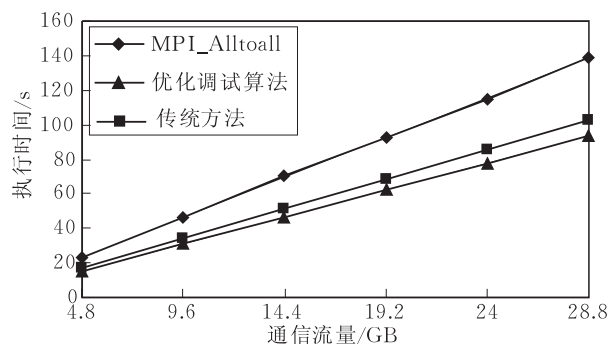


图 11 针对例 3 的通信时间对比

从上述测试和分析,不难看出算法 1 和算法 2 使得结点间的通信效率得到了显著的提高。

5 结论及进一步的研究工作

根据数组分布和 FORALL 语句所提供的信息,给出并证明了通信表的周期性和表中元素对应关系的定理和推论。并在此基础上给出了关于一维数组的处理器间通信的调度优化策略。该策略具有以下特点:在避免消息冲突的前提下,最小化了通信表生成的时间和整个通信所用的时间;应用范围更广,既适用于并行程序各个阶段之间数组重分布的调度优化,而且还能应用于循环划分后数据重映射的优化。通过与传统方式和 MPI_Alltoallv 实现相比,它在很大程度上改善了处理器间的通信效率。此外,该方法还对并行程序局部数组最优分布的确定、编译时处理器程序的生成和 MPI_Alltoallv 的实现提供参考。下一步的工作包括:面向不同网络拓扑的处理器间通信调度策略以及针对核外、非规则问题的通信调度策略进行研究。

参 考 文 献

- [1] HPF Forum. High Performance Fortran Language Specification, version 2.0 edition. Rice University, Houston, Texas, November 1996
- [2] Benkner S. VFC: The vienna fortran compiler. Scientific Programming, 1999, 7(1): 67-81
- [3] Yu H-S, Hu C-J, Huang Q-J, Ding W-K, Xu Z-Q. A time-slicing optimization framework of computation partitioning for data-parallel languages. Journal of Software, 2001, 12(10): 1434-1446
- [4] Guo M, Yi Pan, Liu Z. A symbolic analysis method of communication set generation for irregular array reference. Journal of Supercomputing, 2004, 25(3):199-214
- [5] Hwang G-H. An efficient algorithm for communication set generation of data parallel programs with block-cyclic distribution. Parallel Computing, 2004, 30(4): 473-501
- [6] Basumallik A, Eigenmann R. Optimizing irregular shared-memory application for distribution memory systems. Principles and Practice of Parallel Programming, Manhattan, NY, USA, 2006: 119-128
- [7] Park N, Prasanna V-K, Raghavendra C-S. Efficient algorithms for block-cyclic array redistribution between processor sets. IEEE Transactions on Parallel Distributed Systems, 1999, 10(12): 1217-1239
- [8] Desprez F, Dongarra J, Petitet A, Randriamaro C, Robert Y. Scheduling block-cyclic array redistribution. IEEE Transactions on Parallel Distributed Systems, 1998, 9(2): 192-205
- [9] Faraj A, Yuan X. An empirical approach for efficient all-to-all personalized communication on Ethernet switched clusters//Proceedings of the 34th International Conference on Parallel Processing 2005. Oslo, Norway, 2005: 321-328
- [10] Faraj A, Yuan X. Message scheduling for all-to-all personalized communication on Ethernet switched clusters. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(2): 264-276
- [11] Guo M, Nakata I, Yamashita Y. Contention-free communication scheduling for array redistribution. Parallel Computing, 2000, 26(10): 1325-1343
- [12] Guo M, Pan Y. Improving communication scheduling for array redistribution. Journal of Parallel Distributed Computing, 2005, 65(5): 553-563
- [13] Adams J-C, Brainerd W-S, Martin J-T, Smith B-T, Wagener J-L. Fortran 90 Handbook Complete Ansi/iso Reference. New York: McGraw-Hill, 1992
- [14] Hu C-J, Li J, Wang J, Li Y-H, Ding L, Li J-J. Communication generation for irregular parallel applications. IEEE International Symposium on Parallel Computing in Electrical Engineering 2006. Bialystok, Poland, 2006: 263-270

附录

下面的符号在表 1 中已说明。

引理 1. 在 COM 表中, 对于任意的 (Q_i, P_j) 有 $COM'(i, j) = u \neq 0$, 如果数组访问正好是一个周期, 那么当数组访问是周期的 k (k 是正整数) 倍时, 即 $k \times period_{iter}$, 有 $COM(i, j) = k \times u$.

证明. 当在一个周期的情况下, $COM'(i, j) = u \Rightarrow \forall g \in \{g_1, g_2, \dots, g_u\}$, 有 $i = (g \div y) \bmod n_Q \Rightarrow i' = [(g + k \times preiod_s) \div y] \bmod n_Q = i$ 和 $j = \left[\left(\frac{g - b_2}{a_2} \times a_1 + b_1 \right) \div x \right] \bmod n_p \Rightarrow j' = \left[\left(\frac{g + k \times preiod_s - b_2}{a_2} \times a_1 + b_1 \right) \div x \right] \bmod n_p = j$, 其中 k 是正整数. 所以, 当数组访问是周期的 k 倍时, $COM(i, j) = k \times u$. 证毕.

定理 1. 假设数组 A 和数组 B 分别以 $cyclic(x)$ 和 $cyclic(y)$ 分布到 n_p 和 n_Q 个处理器上, 那么 $COM(i, j) = u \Rightarrow COM((i + k \times m) \bmod n_Q, (j + k \times n) \bmod n_p) = u, 0 \leq k < \text{lcm}(s, t)$, 其中 $m = \frac{M}{y}, n = \frac{M \times a_1}{a_2 \times x}, M = \frac{\text{lcm}(a_1 \times \text{lcm}(a_2, y), a_2 \times x)}{a_1}$, $s = n_Q / \text{gcd}(m, n_Q), t = n_p / \text{gcd}(n, n_p)$, 同时有 $COM(i, j) = u \Rightarrow COM((i + k \times m') \bmod n_Q, (j + k \times n') \bmod n_p) = u, 0 \leq k < \text{lcm}(s', t')$, 其中 $m' = \frac{M' \times a_2}{a_1 \times y}, n' = \frac{M'}{x}, M' = \frac{\text{lcm}(a_2 \times \text{lcm}(a_1, x), a_1 \times y)}{a_2}, s' = n_Q / \text{gcd}(m', n_Q), t' = n_p / \text{gcd}(n', n_p)$.

证明. $COM(i, j) = u$ 表示数组有 u 个元素 (假设其全局地址为 g_1, g_2, \dots, g_u), 其相应的 B 数组元素 ($B[g_n]$) 在 Q_i 上, 相应的 A 数组元素 ($A\left[\frac{g_n - b_2}{a_2} \times a_1 + b_1\right]$) 在 P_j 上, 这时 $i = (g_n \div y) \bmod n_Q, j = \left[\left(\frac{g_n - b_2}{a_2} \times a_1 + b_1 \right) \div x \right] \bmod n_p$. 令 $M = \frac{\text{lcm}(a_1 \times \text{lcm}(a_2, y), a_2 \times x)}{a_1}$, 则 $B[g_n + M]$ 在处理器 $Q_{i'}$ 上, 相应的 $A\left[\frac{g_n + M - b_2}{a_2} \times a_1 + b_1\right]$ 在处理器 $P_{j'}$ 上. 由于 M 和 $a_1 \times M$ 分别是 y 和 $a_2 \times x$ 的整数倍, 即 $m = \frac{M}{y}, n = \frac{a_1 \times M}{a_2 \times x}$ 均为正整数. 为了使 $g_n + M \in \{a_2 \times i + b_2 \mid i \geq 0, i \in Z\}$, M 还能够被 a_2 整除, 所以有 $i' = [(g_n + M) \div y] \bmod n_Q = [(g_n \div y) \bmod n_Q + \frac{M}{y}] \bmod n_Q = (i + m) \bmod n_Q$, 和 $j' = \left[\left(\frac{g_n + M - b_2}{a_2} \times a_1 + b_1 \right) \div x \right] \bmod n_p = \left\{ \left[\left(\frac{g_n - b_2}{a_2} \times a_1 + b_1 \right) \div x \right] \bmod n_p + \frac{a_1 \times M}{a_2 \times x} \right\} \bmod n_p = (j + n) \bmod n_p$, 故 $B[g_n + M]$ 在处理器 $Q_{(i+m) \bmod n_Q}$ 上, 相应的 $A\left[\frac{g_n + M - b_2}{a_2} \times a_1 + b_1\right]$ 在处理器 $P_{(j+n) \bmod n_p}$ 上. 同理可得, $B[g_n + k \times M]$ 在处理器 $Q_{(i+k \times m) \bmod n_Q}$ 上, 相应的 $A\left[\frac{g_n + k \times M - b_2}{a_2} \times a_1 + b_1\right]$ 在

$P_{(j+k \times n) \bmod n_p}$ 上. 假设 $g_{n1}, g_{n2} \in \{g_1, g_2, \dots, g_u\}$ 和 $g_{n1} \neq g_{n2}$, 则 $\frac{g_{n1} - b_2}{a_2} \times a_1 + b_1 \neq \frac{g_{n2} - b_2}{a_2} \times a_1 + b_1$. 由此可以得到 $(g_{n1} + k \times M) \bmod period_s \neq (g_{n2} + k \times M) \bmod period_s$ 和 $\left(\frac{g_{n1} + k \times M - b_2}{a_2} \times a_1 - b_1 \right) \bmod (a_1 \times period_s / a_2) \neq \left(\frac{g_{n2} + k \times M - b_2}{a_2} \times a_1 - b_1 \right) \bmod (a_1 \times period_s / a_2)$, 所以 $COM((i + k \times m) \bmod n_Q, (j + k \times n) \bmod n_p) = u, 0 \leq k < \text{lcm}(s, t)$, 其中 $s = n_Q / \text{gcd}(m, n_Q), t = n_p / \text{gcd}(n, n_p)$. 同理可证:

$COM(i, j) = u \Rightarrow COM((i + k \times m') \bmod n_Q, (j + k \times n') \bmod n_p) = u, 0 \leq k < \text{lcm}(s', t')$, 其中 $m' = \frac{M' \times a_2}{a_1 \times y}, n' = \frac{M'}{x}, M' = \frac{\text{lcm}(a_2 \times \text{lcm}(a_1, x), a_1 \times y)}{a_2}, s' = n_Q / \text{gcd}(m', n_Q), t' = n_p / \text{gcd}(n', n_p)$. 证毕.

推论 1.

$$m = \frac{M}{y}, n = \frac{M \times a_1}{a_2 \times x}, M = \frac{\text{lcm}(a_1 \times \text{lcm}(a_2, y), a_2 \times x)}{a_1},$$

$$m' = \frac{M' \times a_2}{a_1 \times y}, n' = \frac{M'}{x},$$

$$M' = \frac{\text{lcm}(a_2 \times \text{lcm}(a_1, x), a_1 \times y)}{a_2}.$$

$$\Rightarrow m = m', n = n'.$$

证明.

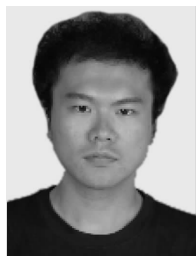
$$m = \frac{M}{y} = \frac{\text{lcm}(a_1 \times \text{lcm}(a_2, y), a_2 \times x)}{a_1 \times y} = \frac{\frac{a_2 \times x}{\text{gcd}(a_1 \times y, x \times \text{gcd}(a_2, y))}}{\frac{a_2 \times x}{\text{gcd}(a_2, y) \times \text{gcd}\left(\frac{a_1 \times y}{\text{gcd}(a_2, y)}, x\right)}}.$$

$$m' = \frac{M' \times a_2}{a_1 \times y} = \frac{\text{lcm}(a_2 \times \text{lcm}(a_1, x), a_1 \times y)}{a_1 \times y} = \frac{\frac{a_2 \times x}{\text{gcd}(a_2 \times x, y \times \text{gcd}(a_1, x))}}{\frac{a_2 \times x}{\text{gcd}(a_1, x) \times \text{gcd}\left(\frac{a_2 \times x}{\text{gcd}(a_1, x)}, y\right)}}.$$

令 $u = \frac{y}{\text{gcd}(a_2, y)}, v = \frac{x}{\text{gcd}(a_1, x)}$, 代入上式 $m = \frac{\frac{a_2 \times x}{\text{gcd}(a_2, y) \times \text{gcd}(a_1 \times u, x)}}{\frac{a_2 \times x}{\text{gcd}(a_2, y) \times \text{gcd}(a_1, x) \times \text{gcd}(u, v)}}$, $m' = \frac{\frac{a_2 \times x}{\text{gcd}(a_1, x) \times \text{gcd}(a_2 \times v, y)}}{\frac{a_2 \times x}{\text{gcd}(a_1, x) \times \text{gcd}(a_2, y) \times \text{gcd}(v, u)}}$, 因此 $m = m'$, 同理可证 $n = n'$. 证毕.

推论 2. $n_p \times n_Q = \text{lcm}(s, t) \Rightarrow COM$ 中的所有元素相等.

证明. 由于 $COM(i, j) = u \Rightarrow COM((i + k \times m) \bmod n_Q, (j + k \times n) \bmod n_p) = u, 0 \leq k < n_p \times n_Q$, 故 COM 中的所有元素相等. 证毕.



WANG Jue, born in 1981, Ph. D. candidate. His research interests include parallel computing and parallel compilation technology.

HU Chang-Jun, born in 1963, Ph. D. , professor, Ph. D. supervisor. His main research interests include parallel com-

puting model, parallel compilation technology, parallel software engineering, network storage system, data engineering and software engineering.

ZHANG Ji-Lin, born in 1980, Ph. D. candidate. His research interests include parallel computing and parallel algorithm.

LI Jian-Jiang, born in 1971, Ph. D. , associate professor. His main research interests include parallel computation, parallel compilation and multi-threaded technology.

Background

The work in this paper belongs to communication optimization of parallel compilation. Recently, many researchers have concentrated on the problem of collective communication. Some studies have pay attention to efficiently generate communication message using compilation technique, while some have addressed the communication optimization. Many researches on communication scheduling, concentrate mainly on some special cases or dynamic redistribution from phase to phase. However, little researchers concern with communication optimization of array remapping by using array distribution pattern and assignment statements in loop. The authors pay attention to messages scheduling by making use of the compilation information provided by array subscripts, array distribution pattern and array access period from parallel application on single-layer switch network. Comparing the previous researches, the main contributions of the work are as follow;

(1) Proposing the approach to schedule messages for data remapping after loop partition.

(2) It can be applied to data redistribution from phase to phase.

(3) Minimizing the overhead of communication schedule generation by the period theory of array access. It can be

completely generated at compile-time phase, so there is no overhead at run-time phase.

This work is supported by the Hi-Tech Research and Development Program (863) of China under grant No. 2006AA01Z105, the National Natural Science Foundation of China under grant No. 60373008 and the Ministry of Education of the People's Republic of China Major Foundation under grant No. 106019. The objective of the Hi-Tech Research and Development Program (863) is to improve the productivity of parallel programmer using parallel compiling techniques. The National Natural Science Foundation of China pays attentions to OpenMP extension on clusters. The Ministry of Education of the People's Republic of China Major Foundation focuses on a difficult problem in parallel computing, i. e. , out-of-core irregular problem. In communication optimization area, we have proposed a communication generation approach, which integrates the benefits from the algebra method and the integer lattice method, to derive an algebraic solution of communication set enumeration. The work in this paper focuses on improving efficiency of collective communication using compiling techniques. The authors will extend this technique to their prototype compiler for irregular applications.