负载平衡无关的并行程序最适处理器网格选择*

张云泉[1], 施巍松[2]

[1](中国科学院 软件研究所 并行软件研究开发中心,北京 100080)

[2](中国科学院 计算技术研究所 系统结构研究室,北京 100080)

E-mail: zyq@mail.rdcps.ac.cn; wsshi@water.chpc.ict.ac.cn

http://www.rdcps.ac.cn

摘要: 用户在编写并行程序时,通常是把物理处理器看成逻辑的处理器(进程)网格,以便于算法的实现.随着用户可用处理器的不断增多,可选择的网格形状也随之增加,如何为基于消息传递的并行程序选择合适的、能发挥出并行机潜在性能的处理器网格形状,是一个迫切需要解决的问题.在提出基于通信点概念的最小度数通信点集合法之后,通过对并行程序通信模式的分析,试图解决与负载平衡无关的并行程序的最适处理器网格选择问题.通过对 ScaLAPACK 软件包中的一个并行测试程序——并行 Cholesky(对称正定矩阵分解)通信点集合度的分析,此方法成功地选择了最适处理器网格形状,并与实验结果相一致.

关键词: 处理器网格;并行程序;通信点;数据分布;负载平衡

中图法分类号: TP 文献标识码: A

如何高效地利用 MPP 中的大量处理器是高性能计算领域所面临的最严重的挑战.目前,在并行语言开发(如 HPF[4])、并行数值软件包设计(如 ScaLAPACK[2])和消息传递库设计(MPI[3])中,有一个共同的趋势是把底层大量物理处理器抽象成一个逻辑处理器网格(当进程与处理器之间不一一对应时,也叫做过程网格)提供给编程者,这样,用户不必关心实际的物理连接而只需关心处理器间的逻辑位置关系,为并行算法设计和程序实现提供了极大的方便.另外,处理器网格的采用,使用户可以更好地控制数据和计算的分配,较方便地实现从一维网格到多维网格的转变,保证了并行程序在大规模机器上的可扩展性.

众所周知,基于消息传递的并行程序设计的最大困难就在于其数据划分和显式的数据移动.数据划分对程序的负载平衡和通信花费有很大的影响,从而也极大地影响了程序的最后性能.通常,并行数据划分包含两步:数据对齐和数据分布.其中,在数据对齐阶段解决数组之间的相互对齐关系,而数据分布则决定分布数组的哪一维和如何把数组分布到一个处理器网格上[4].在关于数据划分的文献中,一般都预先假定一个一维的处理器网格或任意选一个网格形状,虽然处理器网格的选择也会影响程序的最终性能,但却没有引起足够的重视.基于这一事实,我们提出了把确定一个并行程序适用的处理器网格作为并行程序数据划分的第 3 步来考虑的观点,并给出了具体可行的基于通信点的通信开销计算模型和分析算法.实际上,关于处理器网格形状的选择问题,是由 Blackford 和 Demmel 等人在文献[5]中首次提出的,他们认为,基于 ScaLAPACK 的应用程序的处理器网格形状选择可以通过把并行程序的执行时间表示为并行机器和具体问题参数的函数,用分析的方法来解决.但是,这需要用户对机器和算法很熟悉,而且所得到的函数会很复杂,难以分析,用户在分析时会因为对参数的假定过于理想化而无法直接得到有用的网格选择.另外,他们也提出了建立依赖于具体机器和问题参数的花费模型来进行精确选择的想法,但并未提出切实可行的方法.本文提出的方法,通过利用 ScaLAPACK 软件包中并行程序负

* 收稿日期: 1999-06-18; 修改日期: 1999-09-28

基金项目: 国家攀登计划 B 资助项目;国家 863 高科技项目基金资助项目(863-306-ZT06-02-01;863-306-ZD01-03-2);国家自然科学基金资助项目(NSF69883006)

作者简介: 张云泉(1973—),男,山东聊城人,博士,主要研究领域为高性能数值计算,网络并行计算;施巍松(1974—),男,安徽巢湖人,博士,主要研究领域为分布式共享存储系统,网络并行计算.

载均衡分配与处理器网格形状无关的特点,把对计算时间和通信时间的分析分离开来,用户不必考虑并行程序的计算行为与处理器网格的关系,只需分析并行程序的通信点的分布,提取必要的通信点相关信息,就可以通过对程序通信行为的精确的量的分析,较容易地确定一个并行程序的最适处理器网格,从而解决了基于 ScaLAPACK 的应用程序最适处理器网格的选择问题.

本文第 1 节先给出最小度数通信点集合法的基本原理和方法.第 2 节进行实例分析.最后总结全文.

1 最小度数通信点集合法

1.1 通信点定义

我们给出的通信点定义是对文献[5]中通信点概念的扩充.在文献[5]中,Yan Yong 等人通过把并行程序分成计算片段和通信点,建立了一个实用的并行程序性能预测模型.当然,通过这种对并行程序总体的性能建模,我们相信也能够选出最适处理器网格.但基于我们的观察,这是不必要的,而且会浪费很多时间.只要对并行程序的通信模式进行分析,就足以选出最适处理器网格形状.另外,在文献[5]中的通信点只包含非堵塞发送和堵塞接收.我们扩充它可以包含堵塞发送和集合(collective)通信,如 Broadcast,Scatter 和 Gather 等,如图 1 所示.图中的 g 表示通信点的通信开销,l 表示消息长度,f 表示通信点被调用的次数,即后面将要给出的通信点的度数.若一个通信点在循环中,会有 $f>1$ 成立.若 $f=0$,则表示该通信点不被调用或被有效的计算所覆盖.那么,由此我们可以给出通信点的定义(如下所示),其中的通信函数可以是最底层的通信原语,也可以是在通信原语基础上实现的复杂通信函数调用.但显而易见的是,通信函数的层次越高,在进行通信点分析时所需的信息越少,所建立的通信花费模型也越不精确;相反地,层次越低,用户所需分析的信息越多,但同时通信花费的模型会越精确.通信点的具体层次由用户可得到的信息和建立通信花费模型的难易程度来决定.

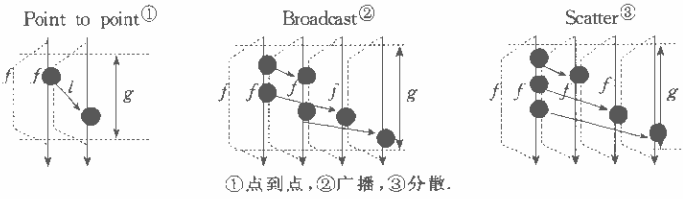

Fig. 1 Communication points example
图1 通信点图例

定义 1. 消息传递并行程序中的一次消息传递函数调用称作该程序中的一个通信点.

在传统的 Von Neumann 体系结构中,计算所用到的所有数据假定都可以在本地内存中取到.但是,随着现代大规模并行计算机的出现,这种假定不再成立了.当处理器需要访问不在本地内存的数据时,它需要借助程序员的帮助,显式地通过某种通信机制才能得到所需要的数据或进行同步.这种数据移动操作的不透明性,极大地增加了并行程序编制的困难.近些年来,关于分布式共享并行机和并行平台的研究取得了不小的进步[6],为并行程序的编制开辟了一条比较容易实现的道路,但这种方法的性能目前还不是很令人满意.有大量的用户还是希望获取更多的性能,尽管这需要投入大量的程序编制时间.

定义 2. 在程序的执行过程中,通信点被调用的次数称作该通信点的度数.

一般来讲,不同处理器网格中通信点的调用次数是不一样的,且每一通信点在不同处理器网格下的不同调用,其时间花费也是不一样的,我们把这作为每一通信点的权重,以区别不同的网格形状和调用.

定义 3. 通信点比重指是一个通信点从调用开始到完成通信任务返回之间所花费的时间,用 $g(p,l)$ 表示,其中 p 是参与该通信操作的进程数,l 是与所计算的问题规模有关的消息长度.

应当注意的是,一个并行算法的最适处理器网格不是一成不变的,它会随着算法所运行的并行机通信链路、通信软件设计、进程数和问题规模等的不同而不同.但在并行计算环境确定的情况下,一个并行算法的最适处理器网格的形状随处理器数和问题规模变化的规律是可以通过分析加以确定的,并可用于今后对该类并行程序的运行,以获取高性能.

1.2 参数定义

本文中用到的一些参数及其定义在表1中列出.表1中的$g_i(p,l)$是第i个通信点在消息长度为l,参与通信操作的进程数为p时的通信点比重.在下面的讨论中,我们假定$g_i(p,l)$可以用如下模型逼近:

$$g_i(p,l)=\begin{cases}\alpha+S(p)+(\beta+B(p))l, & \text{若 } i \text{ 为远地通信;}\\ T_{\text{localmem}}(l), & \text{若 } i \text{ 为本地内存操作.}\end{cases}$$

其中α是消息传递的起动时间,β是每一字节的花费[7],而$S(p)$和$B(p)$是和参与通信的处理器数有关的函数,是对文献[7]中点到点通信模型在多进程通信时的扩展.对于点到点通信,有$S(p)=B(p)=0$.T_{localmem}则是一个和本地处理器内存访问开销有关的函数,会随处理器和操作系统的不同而不同.在后面的实例分析中,我们将给出在SR2201上所用到的MPI通信函数和本地memcopy()函数的详细花费模型.

Table 1 Parameters used in this paper

表1 本文中用到的参数

$g_i(p,l)$	The weight of the ith communication point, when message longth is l and those are p processors involved in message possing①
CP_i	The ith communication point②
f_i	Degree of the CP_i③
$\theta_{P_0q_0}$	Total degree of communication point set on processor grid $p_0\times q_0$④

①当消息长度为l,参与通信处理器为p时,第i个通信点的比重,②第i个通信点,
③第i个通信点的度数,④处理器网格$p_0\times q_0$的总通信点集合度数.

表1中的f_i表示第i个通信点的度数.对于不在任何循环中的通信点,其通信点度数是1,而对于包含在循环中的通信点,其度数与循环的次数有关.

对于不同的处理器网格,它所包含的通信点数目和各点的度数是不一样的.我们把每一种处理器网格所包含的所有通信点组成一个集合,称作通信点集.并用参数$\theta_{P_0q_0}$表示网格$p_0\times q_0$的通信点集合中通信点的总度数.本文提出的最小度数通信点集合法(minimum degree of communication point set,简称MDCPS)正是通过选取总度数最少的通信点集合,从而确定相应的最适处理器网格形状的.$\theta_{P_0q_0}$可用下面的公式计算:

$$\theta_{P_0q_0}=\sum_{i=1}^{K}\sum_{j=1}^{f_i}g_i(p,l_j),$$

其中K是进程网格$p_0\times q_0$通信点集合的通信点个数.

在负载平衡与处理器网格无关的条件下,我们有如下定理成立.

定理1. 在负载分配均衡且与处理器网格选择无关的前提下,对于一个特定的机器,在进程数为P,问题规模为n时,能使通信点集合度θ最小的通信点集合所对应的处理器网格(p,q)是在该机器上该进程数和问题规模下的最适处理器网格.

证明:由并行程序的运行时间

$$t_p=t_{cp}+t_{cm},$$

其中t_p是并行运行时间,t_{cp}和t_{cm}是并行计算时间和并行通信时间.若一个处理器串行运行时的时间设为t_1,则P个处理器时的并行计算时间在负载平衡时为

$$t_{cp}=\frac{t_1}{P},$$

又由$t_{cm}=\theta$成立,则

$$t_p=\frac{t_1}{P}+\theta.$$

在算法一定的情况下,t_1不变,P的个数也是固定的,若使t_p最小,只有取θ最小,定理得证. □

1.3 方法描述

MDCPS方法的一个主要假设是,并行程序的计算量已经均衡地分配到各个处理器,且不会随进程网格形状的改变而出现不均衡的情况.另外,目前的方法是假定用户的总进程数与总物理处理器数一一对应,不会出现多

个进程在同一个处理器上分时的情况,否则,此时的处理器网格应叫做进程网格.

在以上假定下,对 MDCPS 方法的简要描述是:

(1) 首先,必须给出有通信函数调用的并行程序的算法描述.并对所用到的机器上的相应通信函数进行建模.对通信函数的建模可以一次完成,多次使用;

(2) 从算法描述中找出所有的通信点,并将它们列表.对于特定数目的处理器,算出其所有可能的处理器网格形状,并确定每一网格的通信点集合,其复杂性不超过 $O(K\sqrt{P})$,其中 K 是通信点的总个数,P 是处理器数.有些通信点在 $1\times P$ 和 $P\times 1$ 这些一维形状的网格中会变成本地内存操作或根本不被调用.因而可以去掉或需要特别考虑.但对于其他真正二维的网格,所有的通信点都要考虑;

(3) 从算法描述中确定每一通信点的度数.对于在循环内的通信点,确定循环次数是关键.通常不必给出问题的规模,用参数表示就可以了.待实际计算时再给出实际值.这样,只分析一次即可以多次调用;

(4) 利用前面测定的通信模型,分析每一通信点所用到的处理器数 P 和所发送的消息长度 L,从而确定其通信花费;

(5) 对于确定的处理器数目和问题规模,计算所有处理器网格的总通信点度数;

(6) 比较所有处理器网格的通信点集合总度数,其中总度数最小的通信点集所对应的处理器网格即是对该并行消息传递程序在该处理器数目和问题规模下的、该机器的最适处理器网格.

2 实例分析

本节介绍我们的实验平台 SR2201,并给出通过实验建立的部分通信函数的性能模型,然后分析 ScaLAPACK 软件包中的并行 Cholesky 分解程序,并给出分析结果的实验验证.

2.1 实验平台及 MPI 通信模型

本文的实验是在 SR2201 的 16 个计算结点上进行的,其结点采用 HP PA-RISC 的 150MHz HARP 1E 芯片,并进行了面向并行大规模数值计算的改装,提供了功能很强的伪向量功能,解决了数值计算中由于数据量大而经常出现的快速缓存不命中问题.各节点之间用三维交叉开关互联,由于链路的通信速度快,对用户的感觉就像是全互联一样.

为分析方便,我们对 SR2201 上的 MPI 通信函数进行了全面的测试,并且给出了堵塞点到点通信和基于树的广播算法的分段线性模型.MPI 的堵塞点到点通信模型如下所示(单位:μs):

$$T_{pp}(L)=\begin{cases}26.226277+0.013267L & 0\leqslant L\leqslant 128\text{B};\\ 47.504057+0.007188L & 128\text{B}\leqslant L\leqslant 16\text{KB};\\ 164.407858+0.004554L & 16\text{KB}\leqslant L\leqslant 1\text{MB};\\ 0.0+0.004686L & 1\text{MB}\leqslant L.\end{cases}$$

MPI 基于树的广播函数的模型如下(单位:μs):

$$T_{bc}(p,L)=\begin{cases}18.47+(\log_2(p)-1)*8.125+0.01L & 0\leqslant L\leqslant 128\text{B};\\ 70.29+\log_2(p)*0.006L & 128\text{B}\leqslant L\leqslant 16\text{KB};\\ 131.70+(\log_2(p)-1)*147.53+(0.004+(\log_2(p)-1)*0.0001)L & 16\text{KB}\leqslant L\leqslant 32\text{KB};\\ 205.45+(\log_2(p)-1)*232.82+(0.0027+(\log_2(p)-1)*0.0007)L & 32\text{KB}\leqslant L\leqslant 48\text{KB};\\ 228.74+(\log_2(p)-1)*293.5+0.004L & 48\text{KB}\leqslant L\leqslant 64\text{KB};\\ 251.994+(\log_2(p)-1)*339.27+0.004L & 64\text{KB}\leqslant L\leqslant 80\text{KB};\\ 292.30-(\log_2(p)-1)*418.0+0.0035L & 80\text{KB}\leqslant L\leqslant 96\text{KB};\\ 130.75-(\log_2(p)-1)*109.0+(0.0045+(\log_2(p)-1)*0.0044)L & 96\text{KB}\leqslant L.\end{cases}$$

由于某些通信点消失后变为本地内存 Copy,我们也对 Memcopy()函数和 BLAS 中的 DSWAP()(双精度行交换)函数进行了建模.Memcopy()的速率模型如下所示(单位:MB/s):

$$T_{\text{memccpy}}(L)=\begin{cases}1.137*L/8 & 0\leqslant L\leqslant 512\text{B};\\ 0.000135*e^{(\log_2(L)+2.20)}+5.0 & 512\text{B}\leqslant L\leqslant 3600\text{KB};\\ 150.0 & 3600\text{B}\leqslant L\leqslant 4089\text{B};\\ 36.0 & 4090\text{B}\leqslant L.\end{cases}$$

2.2 并行 Cholesky 分解

并行 Cholesky 分解是 ScaLAPACK 中的对称正定线性方程组求解测试程序,采用二维块循环的数据分布方式,假定处理器网格是 $p\times q$,分块大小为 $r\times r$ 的方块,(Myrow,Mycol)是进程在处理器网格上的逻辑坐标,(IAROW,IACOL)指示当前所处理的矩阵的分块全局坐标,矩阵的大小为 $n\times n$ 的方阵(并行 Cholseky 分解的算法描述参见文献[8],从并行 Cholesky 分解的算法描述,可以看出有 4 个通信点,它们是:

(1) Broadcast L_{kk} to all processes in same template column;

(2) Broadcast $A_{(k+1)k}$ to all processes in same template row;

(3) Transposing column block $A_{(k+1)k}$ to row block $A_{k(k+1)}$;

(4) Broadcast $A_{k(k+1)}$ segment to all processes along column Mycol.

从算法描述中可以得到每一通信点的度数和每一通信点的消息长度,见表 2. 其中,若用 lcm 表示 p 和 q 的最小公倍数,则 $lcmp=\frac{lcm}{p}$,$lcmq=\frac{lcm}{q}$. 其中第 3 个通信点比较复杂,涉及本地内存操作及点到点通信,其操作次数及每次的数据量与进程网格的形状关系很大,大体上需要 $lcmp$ 次串行本地内存操作和 $(lcmq-1)$ 次远程点到点通信才能完成. 而这些本地内存操作是点到点通信的数据准备,且处在通信的关键路径上. 这些因素在我们的分析中也有所考虑,但没有在表中列出.

Table 2 The degree of communication point and the kth message passing longth in cholesky factorization

表 2 Cholesky 分解中通信点的度数和第 k 次通信消息长度

CP	1	2	3	4
f	$\left\lceil\frac{n}{r}\right\rceil-1$	$\left\lceil\frac{n}{r}\right\rceil-1$	$(lcmq-1)*\left\lceil\frac{n}{r}\right\rceil-1$	$\left\lceil\frac{n}{r}\right\rceil-1$
l	$4r^2$	$8r*\left\lvert\frac{(n-k*r)}{p}\right\rvert$	$8r*\left\lvert\frac{(n-k*r)}{p*lcmp}\right\rvert$	$8r*\left\lvert\frac{(n-k*r)}{q}\right\rvert$

这样,通过 MDCPS 方法,利用前面给出的 MPI 通信模型,我们可以定量地计算出对于一定处理器数目,其相应处理器网格的通信点集合总度数,并通过选择具有最小度数的通信点集合,确定最适处理器网格形状. 当方矩阵规模 n 为 1000～10000,步长为 1000,分块大小 $r=100$ 时,对 4～16 个处理器的分析结果如图 2 的左半部分所示. 图中横坐标是矩阵的规模,纵坐标是不同处理器网格的总通信点集合度数,单位是 s. 从图中可看出,对这 4 个处理器的最适网格形状为 2×2,8 个为 4×2,16 个为 4×4.

我们用 ScaLAPACK 的 Cholesky 分解测试程序,对有 4、8、16 个处理器时不同处理器网格的性能进行了测试,其问题规模和块大小与前面的分析一致. 在 SR2201 上的实验结果如图 2 的右半部分所示. 图中的横坐标为问题规模,纵坐标为性能(MFlpos),它是由 Cholesky 分解的总浮点操作数 $O\left(\frac{1}{3}n^3\right)$ 除以所用时间得到的. 从图中可以看出,有 4、8、16 个处理器的实验结果与分析结果完全一致,而且对不同处理器网格通信点集合总度数的排序也与实验结果一致. 这说明 MDCPS 方法不但能发现最适处理器网格,而且对各处理器网格间的相互关系也是可以精确区别的. 在有 4 个处理器时,由于内存不足,问题规模只到 8000 为止.

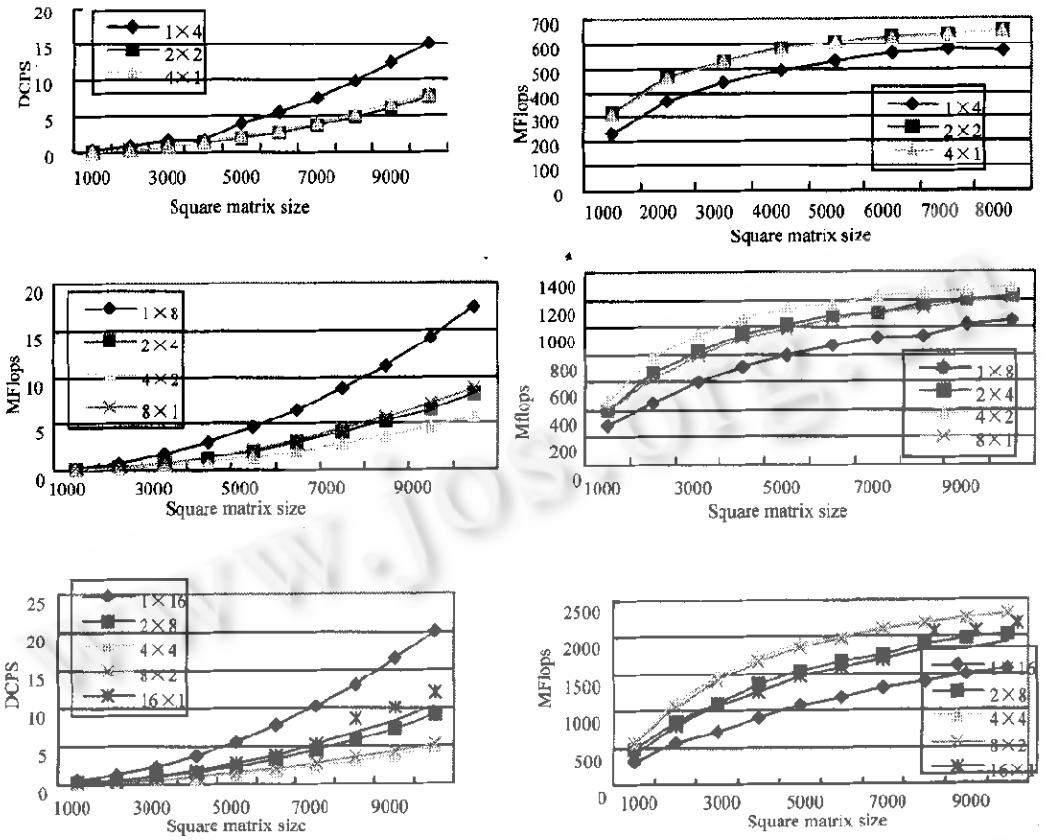

Fig. 2 The analytical result of MDCPS method for parallel Cholesky algorithm on SR2201 and its performance

图 2 MDCPS方法对并行 Cholesky 算法在 SR2201 上的分析结果及实际性能

3 结束语

本文在假定用户已经解决了并行消息传递程序的数据对齐和数据分布问题的基础上,提出了把确定并行算法的适用处理器网格作为数据分割第 3 步的观点,并提出了一个新的方法来选择最适网格.从性能分析的结果来看,MDCPS 方法能够比较准确地预测最适处理器网格.当然,由于现有的机器规模还不够大,MDCPS 方法的可扩展性还有待于进一步的研究.

References:

[1] High Performance Fortran Forum. High Performance Fortran Language Specification(version 2.0). 1997. http://www.netlib.org/hpf/hpf-v20-final.ps.gz.

[2] Blackford, L. S., Choi, J., Cleary, A., *et al*. ScaLAPACK: a portable linear algebra library for distributed memory computers-design issues and performance. In: Proceedings of the Supercomputing'96. IEEE Computer Society Press, 1996. http://www.supercomp.org/sc96/proceedings.

[3] MPI: a message passing interface standard international journal of supercomputer applications and high performance computing. MPI Forum, 1994,8(3~4):165~414.

[4] Kalns, E. T., Xu, H., Ni, L. M. Evaluation of data distribution patterns in distributed-memory machines. Technical Report, MSU-CPS-ACS-80, Michigan State University, 1992.

[5] Yan, Yong, Zhang, Xiao-dong, Song, Yong-sheng. An effective and practical performance prediction model for parallel computing on nondedicated heterogeneous NOW. Journal of Parallel and Distributed Computing, 1996,38(1):63～80.

[6] Shi, Wei-song, Hu, Wei-wu, Tang, Zhi-min, *et al*. A study of shared virtual memory. Journal of Computer Engineering and Science, 1998,20(A1):84～90 (in Chinese).

[7] Dongarra, J. J., Dunigan, T. Message-Passing performance of various computers. Technical Report, CS-95-299, University of Tennessee, 1996.

[8] Zhang, Yun-quan, Shi, Wei-song. Communication point based optimal process grid selection in MPP machines. In: Proceedings of the 5th National Graduate Workshop on Computer Science and Technology. Beijing: Institute of Computing Technology, The Chinese Academy of Sciences, 1998. 80～85 (in Chinese).

附中文参考文献:

[6] 施巍松,胡伟武,唐志敏,等.虚拟共享存储系统研究.计算机工程与科学,1998,20(A1):84～90.

[8] 张云泉,施巍松.基于通信点集合度的并行算法最适进程网格选择.见:第5届全国计算机科学与工程研究生学术讨论会论文集.北京:中国科学院计算技术研究所,1998.28～33.

Optimal Processor Grid Selection for Parallel Program Independent of Load Balance

ZHANG Yun-quan[1], SHI Wei-song[2]

[1](*Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*);

[2](*Institute of Computing Technology, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: zyq@mail.rdcps.ac.cn; wsshi@water.chpc.ict.ac.cn

http://www.rdcps.ac.cn

Received June 18, 1999; accepted September 28, 1999

Abstract: Physical processors are often viewed as a logical processor grid or process grid to ease the parallel algorithm implementation and to provide useful coordination information among parallel processes. However, the shape of processor grid has great impact on the final performance of user's parallel programs. How to select a suitable or even optimal processor grid for an parallel algorithm on certain parallel machines becomes an urgent problem. In this paper, a novel method named MDCPS (minimum degree of communication point set) is proposed, which tries to find out the optimal processor grid for parallel program independent the impaction of load balance through analysis on its communication pattern. The analysis results on ScaLAPACK parallel Cholesky factorization program match the experimental results well and show that the proposed method can select the optimal processor grid for parallel program successfully.

Key words: processor grid; parallel program; communication point; data distribution; load balance