文章编号:1001 - 9081(2009)01 - 0009 - 03

应用层并行 I/O 效率研究

张 娟,陆林生
(江南计算技术研究所,江苏 无锡 214083)
(zjtzcn@163.com)

摘 要:针对划分计算空间到多个计算区域(zone)这类问题,采用基于计算区域组织 I/O 时,首先需选择计算区域的主进程;其次当某一进程是两个计算区域的主进程时需指定计算区域数据访问先后顺序。设计了多级极大独立集算法实现上述过程。该算法利用图论中的连通图以及极大独立集概念规定了每个计算区域进行数据访问的进程以及优先级。经样例分析,采用此种方法达到 I/O 并行度最高,并实现在并行度最高情况下通信量最小。

关键词:并行 I/O;连通图;极大独立集;矩阵
中图分类号: TP301.6 **文献标志码**:A

Efficient parallel I/O scheduling on application level

ZHANG Juan, LU Lin-sheng
(*Jiangnan Institute of Computing Technology, Wuxi Jiangsu* 214083, *China*)

Abstract: Many samples suit to use collective I/O in one zone, which needs to find a main process to do I/O of this zone. When more than one zone has the same main process, priority level need be assigned to each zone. This paper presented a new algorithm, multilevel maximal independent sets, to solve the above problem. Through analysis, this algorithm can realize the highest parallel degree of I/O and communication load is the least at the highest degree.

Key words: parallel I/O; connected graph; Maximal Independent Sets (MIS); matrix

0 引言

并行 I/O 定义为由同一并行程序的多个进程(本文中如不另加说明,程序开始运行即建立进程与处理器间的对应关系,且进程数等于处理器数)产生对文件数据并发请求[1]。本文从应用层开展并行 I/O 研究,研究的目的是在系统软、硬件环境不改变的情况下[2],通过改善应用软件提高 I/O 并行度。

针对在分布式存储并行机上处理复杂数值模拟问题,应用层解决并行 I/O 问题主要有以下三类方法[3-4]。

第一类:并行程序的所有进程都参与数据访问,每个进程都拥有独立的数据文件用于记录本进程访问的数据。该方法并行度高,适合在计算资源严格保障的情况下,某些需频繁进行读写数据文件的算例。但这种方法存在局限,主要体现在:1)数据处理较困难,数据后处理时,无法直接采用各独立文件的数据,需进行拼合处理;而在大规模并行程序中,文件数目多,拼合数据环节易出错。2)采用该方法,中间重启时不能变更处理器数,对计算资源要求严格。

第二类:将待计算的数据划分为多个数据块,并行程序的所有进程在数据块间分配,从分配到某数据块的进程中,选出一进程负责访问该数据块并拥有独立的数据文件。该方法的优点:1)可直接利用记录的数据文件进行后处理;2)由中间步重启时,可任意选择处理器数。缺点是:负责数据访问的进程在写数据文件之前需以数据块为单元进行数据拼合,同时该进程还需负责读数据文件的工作,在读完数据文件后需将数据分发到相关进程,通信量大,并行效率较低。

第三类:选出一个或多个进程专门用于 I/O,此方法适合读写数据量较大的课题。

本文主要关注高端科学计算中采用结构化网格的数值模拟课题。此类问题,计算时间远远多于 I/O 时间,不适宜选用专门的进程用于 I/O;随着计算机技术发展和计算能力提高,解决的问题规模越来越大,待数值模拟的外形也越来越复杂,在生成网格过程中,往往将计算空间划分成多个计算区域(zone);考虑到负载平衡和通信最优等原因,需将计算区域拆分成更小的区域分配到进程;由于计算资源、环境等问题,计算过程中可能会发生被迫停止,甚至需改变并行规模。本课题决定在上述第二类方法的基础上,设计并行 I/O 算法。

基于计算区域组织 I/O,每个计算区域对应有各自的数据文件。选定某一进程用于该计算区域的数据访问,我们称该进程为此计算区域主进程。本文采用的任务分配方式支持进程可处理来自多个计算区域的数据。如图 1 所示,计算区域与进程之间是多对多的关系,任意指定计算区域的主进程极易发生两个或多个计算区域的主进程为同一进程的情况,导致这些使用同一进程作为主进程的计算区域的 I/O 只能顺序进行,降低 I/O 并行度,因此需设计合理的算法尽量使得不同计算区域拥有不同主进程。当无法避免两个或多个计算区域主进程为同一进程时,需指定此进程访问计算区域数据的先后顺序以避免数据访问混乱。本文中采用定义计算区域数据访问优先级的方式确定数据访问的先后顺序,并作如下规定:处于同一优先级的计算区域,它们的主进程可以并发地对这些计算区域进行数据访问;处于不同优先级时,优先级高的先进行数据访问。计算区域数多且并行规模大的情况下,指定计算区域的主进程和定义计算区域优先级是一个复杂的问题,本文利用图论知识设计多级极大独立集算法解决这一复

收稿日期:2008 - 07 - 28;**修回日期**:2008 - 10 - 12。 **基金项目**:国家 863 计划项目(2006AA1149)。
作者简介:张娟(1979 -),女,江苏通州人,博士研究生,主要研究方向:并行算法; 陆林生(1942 -),男,江苏常熟人,教授,博士生导师,主要研究方向:并行算法、并行识别。

杂问题。经实例验证,该方法提高了程序并行度并优化了通信。

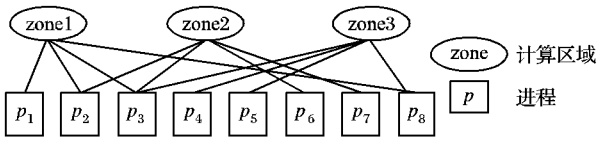

图1 任务分配示意图

1 预备知识

用平面图 $G=(V,E,\rho,\sigma)$ 的顶点集 $V=\{v_1,v_2,\cdots,v_Z\}$[5]14 表示 Z 个计算区域。经任务分配后,顶点 $v_i(1\leqslant i\leqslant Z)$ 代表的计算区域由 m_i 个更小的区块构成,可表示成小区块的集合 $v_i=\{d_{i1},\cdots,d_{i,m_i}\}$。进程数为 N,进程集合为 $P=\{p_j\mid \forall j,1\leqslant j\leqslant N\}$,顶点 v_i 拥有的进程的集合为 $S_i=\{p_j\mid \exists j,1\leqslant j\leqslant N\}$,进程 $p_j\in S_i$ 处理的数据为来自顶点 v_i 代表的计算区域的小区块为 $d_{ik}(1\leqslant k\leqslant m_i)$,小区块 d_{ik} 的数据量为 $|d_{ik}|$。由定义可知,$|S_i|=m_i$。顶点的权重 $\rho:V\rightarrow R$,$\rho(v_i)=|S_i|$ 表示顶点拥有的进程数。

定义1 定义 $mdc_i\in S_i$ 为处理顶点 v_i 中网格单元数最多的小区块的进程号。定义 $mc_i\in S_i$ 为顶点 v_i 的主进程,由 mc_i 执行顶点 v_i 代表的计算区域的I/O。

图 G 顶点之间的连线即边 $E=\{e_1,e_2,\cdots,e_K\}\subseteq V\times V$ 表示边两端顶点代表的计算区域拥有相同的进程。边的权重 σ:$E\rightarrow R$,表示边两端顶点拥有相同进程的数目,设图 G 中的某个边为 $e=\{v_i,v_j\}\in E$[6],$\sigma(e)=|S_i\cap S_j|$。该图的拓扑可由图的邻接矩阵 $\boldsymbol{A}$ 表示:

$$\boldsymbol{A}=(a_{ij})_{Z\times Z} \tag{1}$$

其中:

1) 当 $1\leqslant i,j\leqslant Z$ 且 $i\neq j$

$$a_{ij}=\begin{cases}1, & |S_i\cap S_j|\neq 0\\ 0, & |S_i\cap S_j|=0\end{cases} \tag{2}$$

2) 当 $i=j$ 时,$a_{ij}=0$。

根据图 G 的定义,可知图 G 在某些情况下,可能是非连通的,例如所有的计算区域间没有共享进程。平面图 G 是一般图,其顶点集 V 可以被唯一地划分为非空子集 $V_1,V_2,\cdots,V_k$,使得下面的条件成立[7]:a) 分别由 $V_1,V_2,\cdots,V_k$ 导出的一般子图 $G_1=(V_1,E_1),G_2=(V_2,E_2),\cdots,G_k=(V_k,E_k)$ 都是连通的;b) 对任意 $i\neq j,V_i$ 中任一顶点 x 与 V_j 中任一顶点 y 之间,不存在连接它们的途径。

定义2 我们称上述的图 $G_1,G_2,\cdots,G_k$ 为图 G 的连通分支。

定义3 定义 $level(p_i)(1\leqslant i\leqslant N)$ 为进程 p_i 的级别,表示进程 p_i 当前已被选作主进程的次数;定义 $level(v_i)(1\leqslant i\leqslant Z)$ 为顶点 v_i 的I/O优先级,$level(v_i)$ 越小优先级越高,同一优先级顶点可并发I/O,$level(v_i)$ 等于 v_i 主进程的当前级别。

设计并行I/O算法的目标之一是使得 $\max\limits_{1\leqslant i\leqslant Z}(level(v_i))$ 取得最小值。

定义4 若进程 p_i 为某一计算区域主进程,p_j 处理该计算区域中的某一小区块 d_{ik},由于同一计算区域中每个网格单元上的数据量相等,定义在I/O操作的拼合或发放数据中,两进程间的通信量 $\pi(p_i,p_j)$ 为 $|d_{ik}|$,则在整个I/O部分的通信量为 $flow=\sum\limits_{1\leqslant i,j\leqslant N}\pi(p_i,p_j)$。

设计并行I/O算法的目标之二是使得 $flow$ 取得最小,根据定义,选择网格单元数大的小区块所在进程为主进程可减少I/O部分的通信量。

将顶点集 $V=\{v_1,v_2,\cdots,v_Z\}$ 划分成若干不相关的子集,而且每一子集中的任意两点都不相邻,这样的顶点集合(指集合中任意两顶点都不相邻)叫做独立集[5]206。设图 G 的某一顶点集合是独立集,但是任意增加一顶点就破坏它的独立性,则称该独立集为极大独立集 $MIS(G)$。例如,图2中黑色顶点所组成的集合是相应图的极大独立子集[8]。若图中任意两顶点间都存在边,则称该图为完全图,如图3所示为一完全图。

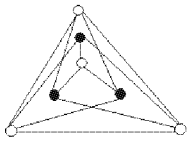

图2 极大独立集

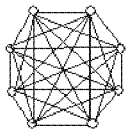

图3 完全图

2 多级极大独立集算法

本文采用多级极大独立集算法给定平面图 G 的顶点的主进程集合 $MC=\{mc_i\mid mc_i\in S_i,1\leqslant i\leqslant N\}$ 以及若 $mc_i=mc_j(i\neq j)$ 时,规定 v_i 和 v_j 进行数据访问的顺序。

根据上述连通分支的定义,任意图的任意两个连通分支间不会拥有相同的进程。根据极大独立集的定义,极大独立集中的各元素之间不会拥有相同的进程。多级极大独立集算法是基于图论的算法。设计基于图论的多级极大独立集算法的最终目的是为了解决基于计算区域的I/O问题,若计算区域之间共享主进程则不可能并发进行I/O操作,此时需要合理规定计算区域进行数据访问的顺序。映射到图 G 中,多级极大独立集算法实现I/O优先级划分并寻找具有同一优先级的顶点。多级极大独立集算法步骤如下:

1) 初始化 $level\leftarrow 0$。

2) $level\leftarrow level+1$;采用广度优先算法找出图 G 的连通分支。

3) 判断连通分支是否是完全的,若是直接转至第5步。

4) 找出各连通分支中的极大独立集,所有极大独立集中顶点的级别赋值为 $level$,并从图中去除极大独立集中顶点。

5) 利用2.2节描述的算法补充可处于同一优先级顶点,顶点的级别赋值为 $level$,从图中去除已赋值的顶点。

6) 判断是否还有剩余顶点,若有则构造下一级子图 G',$G\leftarrow G'$ 并跳转至第2步。

具体内容将在下述四个小节中给予详细介绍。

2.1 极大独立集的查找

查找某一连通分支的极大独立集算法经平移变换算法[9]演变而来。算法如下:

输入:连通图 $G=(V,E)$,$V=\{v_1,v_2,\cdots,v_Z\}$,顶点 v_i 拥有的进程的集合为 $S_i=\{p_j\mid \exists j,1\leqslant j\leqslant N\}$,$E=\{e_1,e_2,\cdots,e_K\}\subseteq V\times V$,图 G 的邻接矩阵 $\boldsymbol{A}=(a_{ij})_{Z\times Z}$。

输出:图 G 的元素个数最多的极大独立集 $MIS(G)$。

1) 根据 $|S_i|(1\leqslant i\leqslant Z)$ 由小到大对顶点集 $V=\{v_1,v_2,\cdots,v_Z\}$ 中顶点排序,得到 $V'=\{v'_1,v'_2,\cdots,v'_Z\}$,利用数组 h 建立新顶点集与原顶点集中的顶点对应关系;

2) 根据排序后的顶点以及图 G 的邻接矩阵得到排序后新顶点集的邻接矩阵 $\boldsymbol{A'}=(a'_{ij})_{Z\times Z}$;

3) do $i=1,Z$

4) $ch=0$

5) do $j=i+1,Z$ // 查找前 i 个元素为0的行

6) if $(a'_{jk}=0(1\leqslant k\leqslant i))$ then

7)　　$ch = j$;exit
8)　　endif
9)　　enddo
10)　　if ($ch \neq 0$) then
11)　　将 ch 与 $i+1$ 进行一个平移变换,包括行和列,得到新的

$$A' = \begin{bmatrix} 0 & A_{21}^{T} \\ A_{21} & A_{22} \end{bmatrix}$$

其中:

$$A_{22} \in R^{(n-i)\times(n-i)}$$

$$A_{21} = \begin{bmatrix} a'_{i+1,1} & a'_{i+1,2} & \cdots & a'_{i+1,i} \\ a'_{i+2,1} & a'_{i+2,2} & \cdots & a'_{i+2,i} \\ \vdots & \vdots & & \vdots \\ a'_{n,1} & a'_{n,2} & \cdots & a'_{n,i} \end{bmatrix}$$

12)　　$r = h(ch)$;$h(ch) = h(i+1)$;$h(i+1) = r$
13)　　else
14)　　$index = i$
15)　　exit
16)　　endif
17)　　enddo

得到极大独立集 $MIS(G) = \{v'_{h(1)}, v'_{h(2)}, \cdots, v'_{h(index)}\}$。顶点 $v'_i(i = h(1), \cdots, h(index))$ 对应的进程集为 S'_i,由于这是第一级的极大独立集,$mc'_i = mdc'_i$,$level(mc'_i) = 1(i = h(1), \cdots, h(index))$,得到级别为 1 的顶点级 $V_{(1)} = MIS(G)$。

2.2　补充同一级中顶点

构造新的平面图 $G^{(1)} = (V^{(1)}, E^{(1)})$, $V^{(1)} = V - MIS(G) = \{v_1^{(1)}, v_2^{(1)}, \cdots, v_{n'}^{(1)}\}$,$n' < Z$,$E^{(1)} = E - \{e_i \mid 1 \leqslant i \leqslant k$ 且 e_i 的任一端点属于 $MIS(G)\}$,顶点 $v_i^{(1)} \in V^{(1)}$ 拥有的进程的集合为 $S_i^{(1)}$,最大块网格单元数的小区块所在进程号为 $mdc_i^{(1)}$。由 $V^{(1)}$ 中根据顶点包含的进程数由小到大逐个选择,若 $level(mdc_i^{(1)}) = 0$,为减少通信量,主进程 $mc_i^{(1)} = mdc_i^{(1)}$,则 $V_{(1)}$ 赋值为 $V_{(1)} \cup \{v_i^{(1)}\}$ 且平面图顶点集 $V^{(1)}$ 赋值为 $V^{(1)} - \{v_i^{(1)}\}$,$E^{(1)}$ 赋值为 $E^{(1)} - \{e_i \mid 1 \leqslant i \leqslant k$ 且 e_i 的任一端点为 $v_i^{(1)}\}$;若 $level(mdc'_i) \neq 0$,为减少通信量,根据处理的小区块的数据量由大到小从 $S_i^{(1)}$ 中选择进程为主进程,若存在 $p_j \in S_i^{(1)}$ 满足 $level(p_j) = 0$,则仍将 $v_i^{(1)}$ 加入到 $V_{(1)}$ 并更改平面图 $G^{(1)}$,并退出,继续选择后一顶点试探着加入 $V_{(1)}$,若不存在 $p_j \in S_i^{(1)}$ 满足 $leve(p_j) = 0$,则仍退出,继续选择后一顶点试探着加入 $V_{(1)}$。直至试探完所有的顶点,得到经修改后平面图 $G^{(1)} = (V^{(1)}, E^{(1)})$,当前拥有主进程的顶点级别都定为 1。

2.3　构造下一级平面图

经 2.1 ~ 2.2 节,多级极大连通图中的第一级顶点主进程查找完毕,按照此步骤继续由图 $G^{(1)} = (V^{(1)}, E^{(1)})$ 中查找属于第二级的顶点以及第二级顶点的主进程,此时,判断顶点是否可以加入 $V_{(2)}$ 的依据为:该顶点分配到的进程中,是否存在进程 p_j 满足 $leve(p_j) = 1$。得到图 $G^{(2)} = (V^{(2)}, E^{(2)})$。直至得到的 $G^{(n)} = (V^{(n)}, E^{(n)})$ 为完全图或所有顶点赋有非零级别值,可省略极大独立集的查找过程,根据顶点拥有的进程数由小到大直接选择顶点试探加入。

目前经计算空间剖分成的计算区域数较少,所以,上述试探查找的方法可行。

2.4　工程应用问题

为了很好地实现负载平衡,任务分配实现了计算区域跨进程分配和进程跨计算区域分配,即计算区域的数据可以分配到不同的进程处理和进程可处理来自不同计算区域的数据,超越了计算区域数和进程数限制。有可能导致在基于计算区域组织的并行 I/O 中出现以下情况:

两顶点 v_i 和 v_j,对应的进程集合为 $S_i = \{\cdots, p_i, \cdots, p_j, \cdots\}$ 和 $S_j = \{\cdots, p_i, \cdots, p_j, \cdots\}$,$p_i$ 为 v_i 的主进程,p_j 为 v_j 的主进程,且两个主进程处于同一级,出现 p_i 和 p_j 相互对发,引发通信堵塞。在工程中,采用分两次发送和接收的方法,首先进程号大的进程发送给进程号小的,进程号小的进程先接收;然后进程号小的进程发送给进程号大的,进程号大的进程先接收。此举可有效避免发生通信堵塞。

3　实例分析

为了更好说明问题,选用一个样例,采用上述的多级极大独立集算法构建其并行 I/O。样例是模拟飞行器前翼周围流场情况,分为 12 个计算区域,运行 32 个进程。对应的平面图 G 如图 4 所示,椭圆所代表的顶点表示计算区域,任务分配后,每个计算区域分到的进程在椭圆内部标识,其中斜体加下划线的进程将处理其所在计算区域中具有最大网格单元数的子区块,例如顶点 v_1 中的 p_{20} 进程。

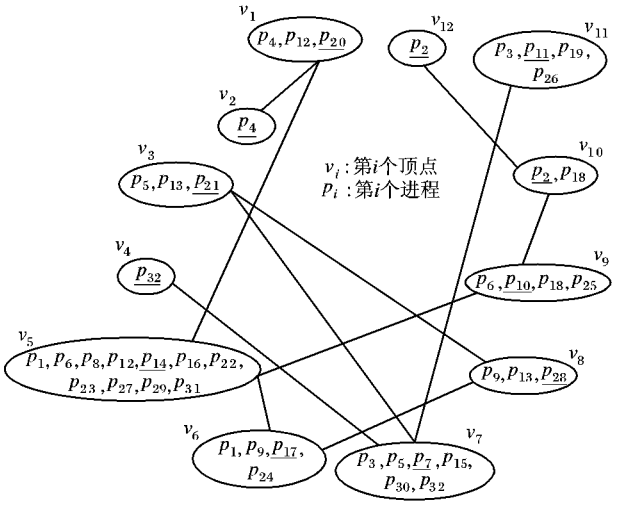

图 4　样例进程分配示意图 G

表 1　进程分配表

顶点	进程
v_1	$p_4, p_{12}, \underline{p_{20}}$
v_2	$\underline{p_4}$
v_3	$p_5, p_{13}, \underline{p_{21}}$
v_4	$\underline{p_{32}}$
v_5	$p_1, p_6, p_8, p_{12}, \underline{p_{14}}, p_{16}, p_{22}, p_{23}, p_{27}, p_{29}, p_{31}$
v_6	$p_1, p_9, \underline{p_{17}}, p_{24}$
v_7	$p_3, p_5, \underline{p_7}, p_{15}, p_{30}, p_{32}$
v_8	$p_9, p_{13}, \underline{p_{28}}$
v_9	$p_6, \underline{p_{10}}, p_{18}, p_{25}$
v_{10}	$p_2, \underline{p_{18}}$
v_{11}	$p_3, \underline{p_{11}}, p_{19}, p_{26}$
v_{12}	$\underline{p_2}$

经多级极大独立集算法求得各顶点对应的主进程在表 1 中加下划线标识,无顶点共享主进程,因此,所有计算区域可并发 I/O。相比较那些选择计算区域中具有最大网格单元数的子区块所在进程作为该计算区域主进程的算法来看,在前

(下转第 15 页)

拥塞丢包,传输速率的平滑性将得到提高。为了更直观比较发送速率的变化情况,定义相邻时间间隔吞吐量比例 $R_i = \frac{T_i}{T_{i+1}}$,$T_i$ 和 T_{i+1} 为相邻时间间隔的吞吐量(在无线自组网中,往返时延 RTT 通常需要持续几秒钟,因此,选择 10 s 作为短期吞吐量的计量单位,可以保证每个时间间隔都包含几个 RTT,减少突发性对短期吞吐量的影响),$R_i = 1$ 意味相邻时间间隔发送端的发送速率相同,$R_i < 1$ 意味发送速率增加,反之意味发送速率减小。图 7 显示了随机选取的 6 对不同协议的吞吐量比例的概率分布。

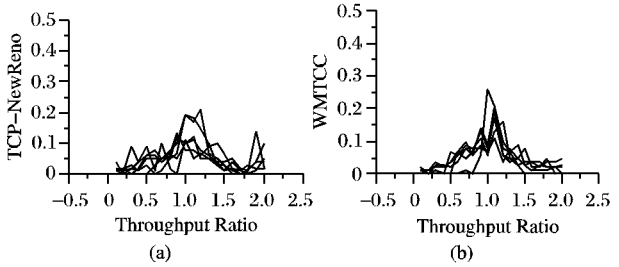

图 7　短期吞吐量变化

可以看出 TCP-NewReno 的吞吐量比例分布范围更广,发送速率表现出更强的抖动性。而采用 WMTCC 协议的数据流吞吐量比例基本分布于 0.5 ~ 1.5,并且集中于 1.0 附近,说明发送速率的变化更加平滑。

5　结语

本文讨论了无线网络中流媒体的传输机制,提出了一种适用于无线网络的流媒体传输机制,它通过发送探测报文区分拥塞丢包和链路丢包,动态调节发送速率,从而获得较高的网络利用率。仿真实验表明算法在无线高误码率网络中有效可行。该方法的有效实施,在于网络中的路由机制采用优先级丢弃策略,但目前网络中很多的路由仍采用 drop-tail 策略,因此在大范围部署上受到一定限制。但是由于无线自组网络通常节点数量有限,因此该协议在无线自组网络中部署方便,有较高应用价值。

参考文献:

[1] SRIPANIDKULCHAI K, MAGGS B, ZHANG H. An analysis of live streaming workloads on the Internet[C]// Proceeding of ACM Internet Measurement Conference (IMC). Taormina, Italy: ACM Press, 2004: 41 - 54.

[2] BALAKRISHNAN H, SECHAN S, AMIR E. Improving TCP/IP performance over wireless networks[C]// ACM MOBICOM '95. Berkeley, CA: ACM Press, 1995: 2 - 11.

[3] FLOYD S, HANDLEY M, PANDHYE J, *et al*. Equation-based congestion control for unicast applications[EB/OL]. [2008 - 05 - 05]. http://www.icir.org/tfrc/tcp-friendly.pdf.

[4] STOCKHAMMER T. Streaming video over variable bit-rate wireless channels[J]. IEEE Transactions on Multimedia, 2004, 6(2): 268 - 277.

[5] 陈志刚, 邓晓衡, 张连明, 等. 无线网络中 TCP 友好流媒体传输改进机制[J]. 通信学报, 2007, 28(5): 22 - 28.

[6] KUN TAN, FENG JIANG, QIAN ZHANG, *et al*. Congestion control in multihop wireless networks. IEEE Transactions on Vehicular Technology, 2007, 56(2).

[7] SHARMA G, MAZUMDAR R, SHROFF N B. Delay and capacity trade-offs in mobile Ad Hoc[J]. IEEE/ACM Transactions on Networking, 2007, 15(5): 981 - 992.

[8] MENG XIAO - QIAO, NANDAGOPAL T, WONG S H Y, *et al*. Scheduling delay-constrained data in wireless data networks[C]// WCNC 2007: IEEE Wireless Communications and Networking Conference. Hongkong: IEEE Press, 2007.

[9] CHEN K, NAHRSTEDT K. Limitations of equation-based congestion control in mobile Ad Hoc networks[C]// ICDCSW '04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC. Washington, DC: IEEE Computer Society, 2004: 756 - 761.

[10] YANG Y R, LAM S S. General AIMD congestion control[C]// Proceedings of the 8th International Conference on Network Protocols. Osaka, Japan: [s.n.], 2000.

[11] BANSAL D, BALAKRISHNAN H. Binomial congestion control algorithms[C]// Proceeding IEEE INFOCOM. Washington, DC: IEEE Press, 2001: 631 - 640.

[12] BIAZ S, VAIDYA N H. De-randomizing congestion losses to improve TCP performance over wired-wireless networks[J]. IEEE/ACM Transactions on Networking, 2005, 13(3): 596 - 608.

(上接第 11 页)

期任务分配相同的情况下,图4所示顶点 v_{10} 和 v_{12} 的主进程将都是 p_2,因此 v_{10} 和 v_{12} 代表的计算区域无法实现并行 I/O,必然增加 I/O 时间。

4　结语

本文由应用层出发研究提高 I/O 并行度,针对具有计算空间划分成多个计算区域,任务分配算法实现进程跨区域分配以及计算区域数据跨进程分配,以区域为单位组织并行I/O等特点的算例,提出了多级极大独立集算法。实例分析表明,采用多级极大独立集算法,很好地解决了这一类算例中的问题,在 I/O 并行度达到最高的情况下,通信量取得最小值。

参考文献:

[1] DONGARRA J, FOSTER I, FOX G. 并行计算综论[M]. 莫则尧, 陈军, 曹小林, 等译. 北京: 电子工业出版社, 2005.

[2] OLDFIELD R A, WOMBLE D E, OBER C C. Efficient parallel I/O in Seismic imaging[J]. The International Journal of High Performance Computing Applications, 1998, 12(3): 333 - 344.

[3] THAKUR R, GROPP W, LUSK E. Data sieving and collective I/O in ROMIO[C]// Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation. Washington, DC: IEEE Computer Society, 1999: 182 - 189.

[4] KOTZ D. Disk-directed I/O for MIMD multiprocessors[J]. ACM Transactions on Computer Systems, 1997, 15(1): 41 - 74.

[5] 卢开澄, 卢华明. 图论及其应用[M]. 2 版. 北京: 清华大学出版社, 1995.

[6] 刘鑫, 陆林生. 数据不规则问题并行计算的负载平衡策略的研究[J], 计算机应用, 2004, 24(10): 108 - 121.

[7] BRUALDI R A. 组合数学[M]. 冯舜玺, 罗平, 裴伟东, 译. 北京: 机械工业出版社, 2005.

[8] 李有梅, 徐宗本, 孙建永. 一类求解最大独立集问题的混合神经演化算法[J]. 计算机学报, 2003, 26(11).

[9] 张大方. 基于矩阵的极大独立点集生成算法[J]. 电子学报, 1998, 26(5): 222.