类的成员函数是对类的属性进行操作的方法，或者说是对框架中的槽值进行操作的函数。例如前面所提到的 set-slot-value()，show-slot-value()和 get-slot-value()等。类的成员函数可根据需要定义，由于此问题较为简单，故本文从略。

2.6 框架和实体的建立

框架和实体的建立过程是在已经定义的框架和实体的数据结构上进行的。

2.6.1 建立框架和槽链。建立一个框架的主要任务是根据该框架所包含的诸槽的槽名，建立各个槽结点和由这些槽结点所形成的槽链。例如，对图1中的病人简况框架，它所包含的诸槽的槽名可用如下字符数组说明：

```
char * patient-slot [ ] = {"name", "sex", "age"};
```

根据此数组，建立一个框架的函数为：

```
frame * set-up-class-slot (char * frame-name, char *slot[]){
  frame * temp=new frame;
  int i=0;
  temp->frame-name=frame-name;
  temp->slot-list=NULL;
  while(strcmp(slot[i],SLOT-END)){
    slot-node * temp1=new slot-node;
    temp1->slot-name=slot[i];
    temp1->slot-next=temp->slot-list;
    temp->slot-list=temp1;
    i++;
  }
  return temp;
}
```

用相应的实参调用此函数，就可建立起一个框架结构，并返回指向该框架的指针。此指针实际上是指向该框架的槽链的指针。

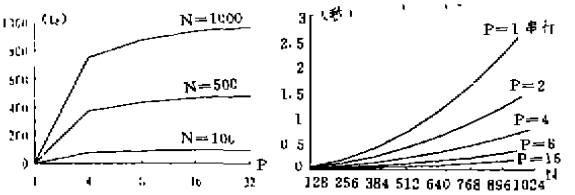
对上述框架，建立其槽链的函数为：

```
add-slot-list(frame * f,patient-frame * p){
  p->frame-name=f->frame-name;
  p->slot-list=f->slot-list;
}
```

2.6.2 建立实体和实体链。建立一个实体的主要任务是对该实体的框架的各个槽赋值，这是由相应类的成员函数来完成的。一实体被建立后，还应将其加入到相应的实体链中。下面是将一实体加入相应实体链的函数：

```
void add-entity-list (char * entity-name, patient * s, patient-frame * sl){
  patient-entity * temp=new * patient-entity;
  temp->entity-name=entity-name;
  temp->frame-of-entity=s;
  temp->next-entity=sl->entity-list;
  sl->entity-list=temp;
};
```

（上接第79页）



图二 通信开销与 P 的关系　　图三 同代过程串行与并行执行的响应时间

从(4)，(5)二式可得加速比：

$$S_p=\frac{T_1}{T_{p0}+T_{com}}$$

$$=\frac{P(1+1/N)}{1+(p\delta-1-\delta)/N+(1-1/P)t_s/N}$$

当 N 足够大，且 P 具有一定规模时 $S_p\approx P/(1+t_s/N)$，t_s 基本上为一常数，单位是与执行一次乘除法时间的比。在我们的实验中，构成虚拟机节点的主机都是 SUN SparcStaion 1，除首次起动时的同步接收等待较长，以后 t_s 的开消是一常数。我们对以下假定的上三角阵进行了求解。

$a_{ij}=1, b_i=(n+i-1)(n-i)/2, i,j=0,1,\cdots,n$ $\delta=32$ 并对不同的 N，P 及串行算法进行了比较，结果见图三。从图三可以看出，随 N 的增大，加速也更明显，这与加速比模型分析是一致的。

参 考 文 献

[1]Al Geist, et al., PVM3 User's Guide And Reference Manual, May, 1993

[2]V. S. Sunderam, PVM: A framework for parallel distributed computing, J. Concurrency: Practice and Experience, 2(4)1990

[3]陈景良，《并行算法引论》

计算机科学1996 Vol. 23 №. 3

基于 PVM 的线性方程组的并行解法

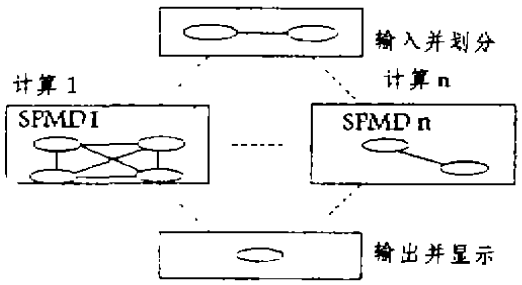
崔振乾 叶澄清
(浙江大学计算机系 杭州310027)

摘 要 In this paper, we present a parallel solution of linear equations based on PVM, which is achieved through parallelization of traditional Gauss resolution and back-substitution schemes seperately. Through theoretic analysis and experiment, this algorithm is shown to provide better performance.

关键词 Parallel Virtual Machine, Single-program Multi-data, Granularity, Wrap-around distribution.

PVM(Parallel Virtual Machine)是一种基于现成局域网(典型的是 Ethernet)的异构并行计算环境,一个纯软件系统,由美国 Oak Ridge 国家实验室、田纳西大学等联合开发而成,异构是指构成虚拟机环境的节点机可以是一般的工程工作站,也可以是 MPP、SMP 等计算机。这种基于消息传递的并行计算环境为利用现有的工作站环境资源进行并行程序的开发与研究提供了一种有效的解决方案。由于其简洁、高效及广泛的适应性、易用性,已成为最盛行的并行开发环境。

在 PVM 环境中,应用程序通常被划分成具有相互依赖关系的子任务,然后被分配到虚拟机的各结点上以进程的形式运行,相互之间以消息传递方式来进行同步、通信,完成整个应用的并行计算,从这一点上看,PVM 提供了最一般意义上的 MIMD 形式的并行计算。但在 PVM 环境的实际应用中,典型的二种计算模型是单程序多数据模型即 SPMD 与主从模型即 Master/Slave。前者各子任务仅是数据不同,而后者则一组子任务(Slaves)在 Master 的控制下协同运行,这二种模型在实际应用中通常是结合在一起的(图一)。



——任务间的通信与同步;……节点间的通信与同步

图一 PVM 的计算模型

PVM 提供给用户的是一个接口库(pvmlib)及在各节点上后台运行的守护进程 pvmd。用户可以用 C 或 FORTRAN 调用其库来完成并行程序的编写。

一、算法描述

1. 消去过程

设线性方程组:$AX=B$。其中 $A=(a_{ij})_{N\cdot N}$, $B=(b_0,b_1,\cdots,b_{N-1})$, $X=(x_0,x_1,\cdots,x_{N-1})^T$, $i,j=0,1,\cdots N-1$。高斯消去的过程即是得到 $LY=B$ 与 $UX=Y$ 的过程,L 为下三角矩阵,U 为上三角矩阵。串行解法是:

```
for k=0 to N-2;
  for i=k+1 to N-1;
        l_ik = a_ik / a_kk
        b_i = b_i - l_ik * b_k
        for j=k+1 to N-1
              a_ij = a_ij - l_ik * a_kj
```

(1)

我们标记上述最后四行的计算为(1)。

假定有 N 个处理器 $P_i(i=0,1,\cdots,N-1)$,求解 LU 的一个直观并行过程可描述如下:

step1 把 a_0送 $P_i(i=1,2,\cdots,N-1)$, a_i 表示 A 的第 i 行向量,下同。由 P_i 计算 $l_{i0}=a_{i0}/a_{00}$及$(a_i)^1=(a_i)^1-l_{i1}(a_1)^1$; $i=1,2,\cdots,N-1$,其中,$(a_i)^k=(a_{ik},\cdots,a_{i,n-1})$下同。

step2 $(a_1)^1$送 $P_i(i=2,3,\cdots,N-1)$。由 P_i 计算 $l_{i1}=a_{i1}/a_{11}$及$(a_i)^2=(a_i)^2-l_{i1}(a_1)^2$, $i=2,3,\cdots N-1$。

如此重复,直至完成。

这一方法存在三个缺点:①每个处理器的计算与通信比值很低,两者不能重叠进行。②各处理器的负载相当不均衡,每一步都使得能进行并行工作的

崔振乾 硕士生,叶澄清 教授,主要研究领域:网络与并行计算、智能计算、图形图像处理。

处理机个数递减。[3]在实际应用中，$N \gg P$，假定 $N=P$ 是不现实的，一个改进的方法是假定 $N=mP$，对任一 P_i 所要处理的行数是区间$[mi, m(i+1)-1]$中的行。可推知，每完成 m 行处理也闲置一处理器。这一方法之负载不平衡得到一定程度的缓和，计算与通信的(数据传送)的比值提高。

我们的策略是依然按连续行把 A 分块、块大小为 δ，为方便表示，假定 $N=S*\delta$ 及 $S=M*P$，也即把 N 行矩阵分成块大小为 δ，总块数为 S 的段，每个节点平均分到 M 块。当 $\delta=1$ 时，即为按行分配，块编号为 $s_i=(i=0,1,\cdots,S-1)$；交叉分配的方法是对任意 P_i，属于它的块集合为 $R_{pi}=\{s_j \mid j \bmod p=i, j=0,1,\cdots,S\}$，属于 s_i 的行区间为$[\delta_i, \delta(i+1)-1]$. LU 分解过程的并行算法可描述为：

对 P_0：则发送 a_0 至其他处理器。对 R_{p0} 中的集合 $s_0=[a,b]$进行如下处理：

```
for k=a to b-1
  for i=k+1 to b
应用(1)，并发送结果给其它处理器.
标识 s0集合完成计算，对 Rp0中其它未经处理的块集合
  si=[a,b]，进行如下更新.
    for k=a to b-1
      for i=c to d
        应用(1)
```

对 $P_i(i\neq 0)$

```
Repeat
    等待接收一行，记收到的行编号为 k；
    对Rpi中的任一集合 si=[a,b]，进行如下更新.
    for i=a to b 应用(1)
    若a==k+1则该集合中的行继续进行如下运算，并标
      识该集合完成计算。
      for k=c to d-1
        for i=k+1 to d
          应用(1)，并发送结果给其它处理器.
    若b==N-1，则完成整个应用的并行计算.
    否则，对 Rpi中的任一未完成计算的集合 si=[c,d]，进
      行如下更新.
      for k=a to b
        for i=c to d
          应用(1)
      Until Rpi中的集合全部标识完成计算
```

上述算法求解的结果是得到上三角阵 L，实际依然放在$(a_{ij})_{N\times N}$中。

2. 回代过程的并行算法

回代过程串行算法可简单描述为：

```
for i=N-1 downto 0
```

$$x_i=x_i-\sum_{j=i+1}^{N-1} a_{ij} * x_j \qquad (2)$$

$$x_i=x_i/a_{ii} \qquad (3)$$

并行的思想即是对上面的求和过程也分段进行。每当一处理节点求解出部分解，即把该解发送给其它处理器，这些处理节点得到部分解后即按(2)求部分和。对系数矩阵进行分块的方法与 LU 分解相同，假定向量 B 的值已存在于 X 中，算法可形式描述如下：

```
若为 PP-1号处理器，则对 RpP-1中的行块集合 sS-1=[a,b]
进行如下处理。
  for i=b downto a
    运用(2)、(3)
  发送 X[a]至 X[b]到其它处理器。标识集合 sS-1完成求
    解。
  对 RpP-1中的其它未完成处理的集合 si=[c,d]，利用
    上面部分解，求部分和：
    for i=d downto c
      对 j∈[a,b]运用(2)
  否则 Pi(i≠P-1)
    Repeat
      接收已得到的解 X[a]到 X[b]
    对Rpi中任一未完成求解的集合 si=[c,d]，求部分和
      for i=d downto c
        对 j∈[a,b]运用(2)
    若(d+1)==a，则下面计算可得到 xk，k∈[c,d]的解
      for i=d downto c
        对 j∈[i+1,d]运用(2)
        运用(3)
      发送 X[c]至 X[d]到其它处理器。
      若 c==0则完成整个求解过程。
      标识集合 si=[c,d]完成求解。
      利用得到的解 X[c]至 X[d]更新 Rpi中任一未完成求
        解的集合。
Until Rpi中的集合全部完成求解
```

二、性能分析

考虑到线性方程组 LU 并行分解过程与并行回代过程的类似，下面仅对回代过程进行计算复杂度及加速比的详细分析。对串行算法其乘除法的执行次数为：$T_s=N(N+1)/2$，并行处理时对任一处理节点 P_i 集合 $s_j\in R_{pi}$，$s_j=[\delta j,(j+1)\delta-1]$区间内的执行次数为：

$$Ts_j=\sum_{k=\delta j}^{\delta(j+1)-1}(N-k)+\delta=(2N-\delta+1)\delta/2-\delta^2 j$$

由此可得对 P_i，其执行的总次数为：

$$T_{pi}=\sum_{j\in\{k \mid k \bmod P=i, k=0,\cdots,S-1\}} Ts_j$$

$$=\frac{N}{2P}(N+\delta p+1-\delta-2\delta i) \qquad (4)$$

当 $i=0$，即 P_0 完成处理时，为最大的执行时间。

每个节点机的通信开销可分成两部分，一是接收来自其它节点的解，这是一个同步接收的过程，另一部分是发送本节点部分解的过程，PVM 中发送采用异步过程，因此基本上可以与计算重叠进行，因此通信开销主要体现在接收，可表示为：

$$T_{com}=(N-N/P)t_\delta=N(1-1/P)t_\delta \qquad (5)$$

其中 t_δ 为接收 δ 个解的开销。

δ 对 t_δ 的影响并不大，上面的并行算法任何时刻至多只有一个节点求出的解在网上广播，因此数据通信量对通信延迟的影响不大。t_δ 主要体现在同步接收时的等待、接收建立等。N 与通信开销的关系可见图二。从图二可以看出对给定的 N 随 P 的增大，通信开销也增大，当 P 增大到一定值时，通信开销基本上维持不变。图中 T_{com}的是相对于 t_δ 的时间单位。

(下转第77页)