一个可预测并行程序效率的评价模型*

陈昌生[1]，孙永强[1]，何积丰[2]

[1](上海交通大学 计算机科学与工程系，上海 200030)

[2](华东师范大学 计算机系，上海 200063)

E-mail: Sun-YQ@cs.SJTU.edu.cn

http://www.sjtu.edu.cn

摘要：并行程序的性能分析，特别是效率分析往往需要程序在实际运行后，根据实验结果再对并行算法进行优化，或改变数据的分配策略，甚至重新选择并行算法.结合通用并行计算模型BSP(bulk-synchronous parallel)，提出一种有效的并行程序效率评测模型，使得程序员在设计、分析阶段即可对程序效率进行分析和评估，并据此进一步优化程序.实验结果表明，该模型的预测是精确的.

关键词：并行程序设计；BSP(bulk-synchronous parallel)模型；效率评价准则

中图法分类号：TP302 **文献标识码**：A

在并行处理领域中，并行体系结构的类型很多，并且相互间往往差别很大，使得不同的体系结构成为不同特殊领域内的专用体系结构.这种依赖于体系结构的并行程序的可移植性、可扩展性都很差，编程较复杂，在很大程度上阻碍了并行处理的普及推广和发展.串行处理领域之所以发展得如此迅猛，主要是因为有统一的体系结构模型，即"冯诺依曼计算机"(Von Neumann computer).程序员可在详细程序设计之前，分析程序所用资源(时间、空间等)的复杂性.在并行处理领域中，在使用目前较为流行的PVM(parallel virtual machine)和MPI(message-passing interface)时，并行程序的性能分析往往依赖于程序运行后的实验数据，效率评价也需要运行后的实验数据.BSP(bulk-synchronous parallel)模型将并行程序员带入了一个新天地.这种模型[1~3]提供了既可扩展，又独立于体系结构的并行程序设计的途径.BSP计算机很简单，只使用有限的4个参数即可刻画，任何并行计算结构都可由BSP模型来表示.BSP模型不仅可以作为体系结构模型，也可以作为并行程序设计模型，它的编程与串行非常接近.本文提出的基于BSP的可预测并行效率模型不但可以有效地在程序运行前对程序效率进行预测，而且还可以帮助程序员找到影响效率的主要因素，从而进一步优化程序.实验结果显示，该模型的预测是精确的.

1 BSP模型简介

BSP计算机[1~3]是由带存储器的处理器集合、全局通信网络和全局障碍同步(global barrier synchronisation)机制组成.设一个本地操作(比如加法或乘法)所需时间为一个时间步，则BSP计算机的性能可由4个参数来描述：处理器数目p；全局同步的时间步数(同步周期)l；处理器速度s，比如每秒时间步数；1秒内所有处理器完成的本地操作数与1秒内网络所发送的字(word)数之比g.任何可扩展并行系统都可看做是一个BSP计算机，通过确定参数l和g来描述.

BSP并行程序复杂性包括4个基本部分：

* **收稿日期**：1999-06-28；**修改日期**：1999-09-10

基金项目：国家自然科学基金资助项目(69783001)

作者简介：陈昌生(1969－)，男，江苏人，博士生，主要研究领域为分布式计算，并行数据库；孙永强(1931－)，男，浙江人，教授，博士生导师，主要研究领域为计算机理论，程序语言；何积丰(1944－)，男，上海人，教授，博士生导师，主要研究领域为计算机软件.

$$T_{para}=T_{comp}+T_{comm}+T_{disk}+T_{sync}.$$

复杂性分析中的磁盘I/O操作,即T_{disk},可以作为一种特殊的通信操作,因此,BSP模型中的超步可归结为局部计算、全局通信和同步3个阶段.

超步$s(i)$所用时间可如下确定:

假设w_j^i——$s(i)$中第j个处理器完成本地计算所用的时间步.

$h_{out_j}^i, h_{in_j}^i$——分别为$s(i)$中第j个处理器发送或接收的消息数($1\leqslant j\leqslant p$).

令 $$w^i=\max\{w_j^i\},\ h_{out}^i=\max\{h_{out_j}^i\},\ h_{in}^i=\max\{h_{in_j}^i\},$$

则超步$s(i)$所用的时间步 $T_i=w^i+\max\{h_{out}^i, h_{in}^i\}*g+l$.

整个BSP程序运行所用时间步 $T_{para}=\sum_{i=1}^{S}T_i=W+H*g+S*l$.

显然,W,H,S都是问题规模n(比如矩阵相乘中的矩阵阶数)和处理器数p的函数.可见,BSP程序设计的优化目标将是$W(n,p),H(n,p),S(n,p)$尽可能地小,p尽可能地大.在大多数情况下,要求我们注意负载均衡和减少通信量.

定义1. h-关系是指每个处理器至多发送h条消息和至多接收h条消息的寻径问题.

定义2. 完全h-关系是指每个处理器确切地发送和接收了h条消息.

假定BSP程序运行过程中满足h-关系且通信和计算过程不可重叠,则有

$$\text{BSP程序运行时间步}=\sum_{i=1}^{S}(T_{comp_i}+T_{comm_i}+l)=\sum_{i=1}^{S}(T_{comp_i}+g*h+l).$$

若假定通信和计算过程可以重叠,则每个超步的时间开销为$\max\{T_{comp_i}, g*h\}+l$.

2 BSP并行程序效率评价模型

常用的并行效率评价准则[4]有加速比Sp、并行效率Eff以及负载平衡E_{load}等,但是它们只是大体上反映了一个并行程序在某一并行环境下的总体性能[5,6],对于在其他并行环境下该算法是否有效以及影响并行算法效率的关键因素是什么,如何去修改,则无能为力.现有的BSP性能分析模型[5]都没有考虑数据的分布和平衡问题,如前所述,在BSP中,数据分布策略、负载平衡是很重要的,因此,新模型将充分考虑算法的数据分布策略,揭示由此产生的负载平衡度.根据BSP模型的特点,增加了多个评价准则,以便程序员较全面、客观地分析影响效率的主要原因.这些评价准则和后面介绍的伪代码语言规范一起构成了评价模型的基础.

BSP并行程序效率评价准则有以下几个:

$$Sp=\frac{T_{seq}}{T_{para}},\tag{1}$$

$$Eff=\frac{Sp}{P}=\frac{T_{seq}}{P*T_{para}},\tag{2}$$

$$E_{load}=\frac{\sum_{i=1}^{P}\sum_{j=1}^{S}(t_{ij}^{comp}+t_{ij}^{comm}+l)}{\max_{i=1}^{P}\left\{\sum_{j=1}^{S}(t_{ij}^{comp}+t_{ij}^{comm}+l)\right\}*P},\tag{3}$$

$$E_{comp}=\frac{\sum_{i=1}^{P}\sum_{j=1}^{S}(t_{ij}^{comp}+l)}{\sum_{i=1}^{P}\sum_{j=1}^{S}(t_{ij}^{comp}+t_{ij}^{comm}+l)},\tag{4}$$

$$E_{idem}=\frac{\sum_{i=1}^{P}\sum_{j=1}^{S}(t_{ij}^{comm}+l)}{\max_{i=1}^{P}\left\{\sum_{j=1}^{S}(t_{ij}^{comm}+l)\right\}*P},\tag{5}$$

$$E_{lscm}=\frac{\sum_{j=1}^{S}\left(\max_{i=1}^{P}\{t_{ij}^{comm}+l\}-\min_{i=1}^{P}\{t_{ij}^{comm}+l\}\right)}{\left(\sum_{j=1}^{S}\sum_{i=1}^{P}(t_{ij}^{comm}+l)\right)/p}. \tag{6}$$

其中 T_{seq} 为计算问题的串行程序开销;T_{para} 为计算问题的 BSP 并行程序开销,$T_{para}=T_{comp}+T_{comm}+T_{sync}$,$T_{comp}$,$T_{comm}$,$T_{sync}$ 分别为并行程序计算、通信、同步所用时间步数;t_{ij}^{comp},t_{ij}^{comm} 分别表示第 i 个处理器在第 j 个超步的 CPU 时间步和通信时间步.由于 BSP 模型只需 4 个参数(p,g,l,s)即可描述,类似于前面程序性能的分析,E_{load},E_{cmmp},E_{ldcm},E_{lscm}将是以上 4 个参数及超步数 S 的函数,这几个参数在程序运行之前即可确定.

E_{load}反映了 BSP 程序计算和通信负载平衡能力;E_{cmmp}反映了 BSP 程序中通信处理所占的比重,一般情况下越小越好;E_{ldcm}反映了 BSP 程序通信的总体负载平衡度,即各个处理器间通信负载的平衡程度,一般情况下越高越好.由 BSP 的性能分析模型可知,计算和通信负载平衡对性能的影响很大,如果计算和通信负载总体上都很不错,但效率仍不理想,这就要看 E_{lscm},它反映了同一超步内的数据通信的平衡度.E_{load},E_{ldcm}反映的都是 BSP 程序的总体平衡情况,BSP 程序是由许多超步构成的,因此,对于超步间锯齿型的不平衡情况,它们无法刻画,但 E_{lscm}能揭示这一点.

总之,影响并行程序效率的因素有很多,也很复杂,我们只能考虑那些主要因素.

为了在分析设计时能有效地反映出数据分布策略,根据 BSP 编程特点,本文定义了一种简单的伪代码规范系统.它的主要规范为

para

〈programs〉　　表示封装一个并行程序.

endpar

P_i:Send(dest,Numofdata)表示 i 进程向 dest 进程发送 Numofdata 大小的数据;

P_j:Get(source,Numofdata)表示 j 进程从 source 进程接受了 Numofdata 大小的数据.

Send 和 Get 是成对出现的.

Sync 表示 BSP 同步

该规范结构简单,特别适合 BSP 程序员进行并行程序的分析和设计.以上评价模型所需数据也容易分析出来.

3 实例研究

矩阵相乘是科学计算中的典型计算,许多应用问题都涉及到大规模的矩阵相乘,如大规模线性方程组求解、大规模偏微分方程求解、反问题等,有一定的普遍性.矩阵 $C=A_{n\times n}*B_{n\times n}$的串行计算的复杂性为 $T_{seq}=O(n^3)$,而并行程序的计算复杂性则依赖于任务映射策略和数据通信量.下面讨论矩阵相乘的两种 BSP 并行算法.

两个算法的主要区别表现在数据分布的策略上,设 P 个处理器的编号为 $P(i,j)(i,j=0,\ldots,\sqrt{p}-1)$.

算法 1.

(1) 将 A 矩阵划分成 $\sqrt{p}$ 个 $\frac{n}{\sqrt{p}}\times n$ 子矩阵,将第 i 个子矩阵发送至 $P(i,j)(i,j=0,\ldots,\sqrt{p}-1)$;将 B 矩阵划分成 $\sqrt{p}$ 个 $n\times\frac{n}{\sqrt{p}}$ 子矩阵,将第 i 个子矩阵发送至 $P(j,i)(i,j=0,\ldots,\sqrt{p}-1)$;

(2) 每个 $P(i,j)$同时进行本地矩阵相乘.

算法 2.

(1) 将 A,B 均匀划分为 P 个 $\frac{n}{\sqrt{p}}\times\frac{n}{\sqrt{p}}$ 子矩阵,将 $A(i,j)$仅发送至对应的 $P(i,j)$;将 $B(i,j)$仅发送至对应的 $P(j,i)$;

(2) 每个 $P(i,j)$同时进行本地矩阵相乘;

(3) 每个 $P(i,j)$同时将本地 A 的子矩阵在 $P(i,j)$阵列中往右传,B 的子矩阵往下传;$P(i,j)$完成本地矩阵

相乘;

(4) 重复步骤(3) $\sqrt{p}-1$ 次.

为了便于模型的分析,两个算法用前面提出的伪代码规范描述如下:

算法 1.

```
main
  para
    call GetInput
    call Seqmatmult
    call finalOut
  endpara
subroutine GetInput
for row=0 to n-1
{
    i=row/(n/√p),j=0,...,√p-1
  P0:Send (i*√p+j,n of A)
     Send (j*√p+i,n of B)
  Pk:Get (0,n of A)
     Get (0,n of B)
     Sync
}

subroutine Seqmatmult (A,B,C)
for i=1 to n/√p
  for j=1 to n/√p
    for k=1 to n/√p
      C[i,j]=C[i,j]+A[i,k]*B[k,j]

Subroutine finalOut
for row=0 to n-1
{
    i=row/(n/√p),j=0,...,√p-1
P(i*√p+j):Send (0,n/√p of C)
P0: Get (i*√p+j,n/√p of C)
    Sync

}
```

算法 2.

```
main
  para
    call GetInput
    call Seqmatmult
    for i=1 to √p-1
    {
  Pk: Send ((k+1) mod √p,n²/p of A)
      Send ((k+√p) mod p,n²/p of B)
  Pj: Get ((j+1) mod √p,n²/p of A)
      Get ((j+√p) mod p,n²/p of B)
      Sync
      Call Seqmatmult
    }
    Call finalOut
  endpara

subroutine GetInput
for row=0 to n-1
{
   i=row/(n+√p),j=0,...,√p-1
P0:Send (i*√p+(j-i) mod √p,n/√p of A)
   Send (((i-j) mod √p)*√p+j,n/√p of B)
Pk:Get (0,n/√p of A)
   Get (0,n/√p of B)
   Sync
}

subroutine Seqmatmult (A,B,C)
for I=1 to n/√p
  for j=1 to n/√p
    for k=1 to n/√p
      C[i,j]=C[i,j]+A[i,k]*B[k,j]

Subroutine finalOut
for row=0 to n-1
{
    I=row/(n/√p),j=0,...,√p-1
P(i*√p+j): Send (0,n/√p of C)
P0: Get (i*√p+j,n/√p of C)
    Sync
}
```

下面先用效率预测模型预测以上两种算法的性能及效率,最后再与实验结果进行比较.

假设 P_i^{comm} 表示第 i 个处理器在 BSP 程序中通信的时间;P_i^{all} 表示第 i 个处理器在 BSP 程序中运行的时间,则分析程序后可得到(其中 ω 表示所传输数据类型与 word 之比,如 Linux 下的 double 型长是 4 个 word,$\omega=4$):

算法 1 的预测结果如下:

$P_0^{comm}=(2*\sqrt{p}+1)*n^2*g*\omega+2*n*l$,

$P_0^{all}=P_0^{comm}+n^3/p$,

$P_k^{comm}=(1+1/\sqrt{p})*n^2*g*\omega+2*n*l$,

$P_k^{all}=P_k^{comm}+n^3/p$,

$T_{para}=(2*\sqrt{p}-1)*n^2*g*\omega+2*n*l+n^3/p$,

代入评价公式,得到:

$Sp=T_{seq}/T_{para}$,

$Eff=Sp/p$,

$E_{load}=(P_0^{all}+(p-1)*P_k^{all})/(P_0^{all}*p)$,

$E_{cmmp}=(P_0^{comm}+(p-1)*P_k^{comm})/(P_0^{all}+(p-1)*P_k^{all})$,

$E_{ldcm}=(P_0^{comm}+(p-1)*P_k^{comm})/(P_0^{comm}*p)$.

算法 2 的预测结果如下:

$P_0^{comm}=(3+2*(\sqrt{p}-1)/p)*n^2*g*\omega+(2*n+\sqrt{p}-1)*l$,

$P_0^{all}=P_0^{comm}+n^3/p$,

$P_k^{comm}=(3/\sqrt{p}+2*(\sqrt{p}-1)/p)*n^2*g*\omega+(2*n+\sqrt{p}-1)*l$,

$P_k^{all}=P_k^{comm}+n^3/p$,

$T_{para}=(3+2*(\sqrt{p}-1)/p)*n^2*g*\omega+(2*n+\sqrt{p}-1)+n^3/p$,

代入评价公式,类似于算法 1,得到相应的性能预测表达式.

比较两个算法的 T_{para} 的差,得到:

$$(2*\sqrt{p}+1)*n^2*g*\omega-(2*n*l+n^3/p)-((3+2*(\sqrt{p}-1)/p)*n^2*g*\omega+(2*n+\sqrt{p}-1)+n^3/p)=2*(p-1)*(\sqrt{p}-1)*n^2*g*\omega/p-(\sqrt{p}-1)*l.$$

4 实验结果分析

我们在高性能微机群集环境下实现了以上两种算法.环境为 4 台 586 微机,OS 为 RedHat Linux,计算机间通过 100Mbps 高速以太网连接.将 BSP 的 4 个参数值* 代入以上两个算法的性能预测表达式.此时两个算法 T_{para} 的差 $2*(p-1)*(\sqrt{p}-1)*n^2*g*\omega/p-(\sqrt{p}-1)*l>0(n>20)$,故预测算法 2 应该较优,得到的预测性能结果见表 1,可以看出算法 2 的效率以及负载平衡较算法 1 好.两个算法实际运行后的性能结果见表 2.表 2 的结果同样显示出算法 2 优于算法 1.图 1 是两个算法的预测和实际的加速比图.这些图表都显示了预测的与实际结果非常接近,说明以上并行程序性能和效率预测模型是精确的,同时也验证了 BSP 并行计算模型有着很好的可预测性.理论分析和实验结果都表明,这两种算法中对应的 E_{cmmp} 都差不多,但负载平衡相差较明显,因而造成这两种并行算法效率差异的主要因素是负载平衡度的不同.

* 这些值可从 BSP 系统的参数测试值表中查到,或用 BSP 提供的测试软件去测试.

Table 1 Predictable results for algorithm 1 and algorithm 2

表 1 算法 1 和算法 2 的预测结果

N	Predictable results for algorithm 1①					Predictable results for algorithm 2②				
	Sp	Eff	E_{load}	E_{cmmp}	E_{ldcm}	Sp	Eff	E_{load}	E_{cmmp}	E_{ldcm}
100	0.32	0.08	0.63	0.87	0.60	0.41	0.10	0.80	0.87	0.73
280	0.90	0.23	0.63	0.64	0.53	1.14	0.29	0.80	0.64	0.72
400	1.20	0.30	0.66	0.54	0.51	1.49	0.37	0.82	0.54	0.71
640	1.66	0.41	0.71	0.41	0.50	1.99	0.50	0.85	0.41	0.70
1000	2.12	0.53	0.76	0.30	0.49	2.45	0.61	0.88	0.30	0.69

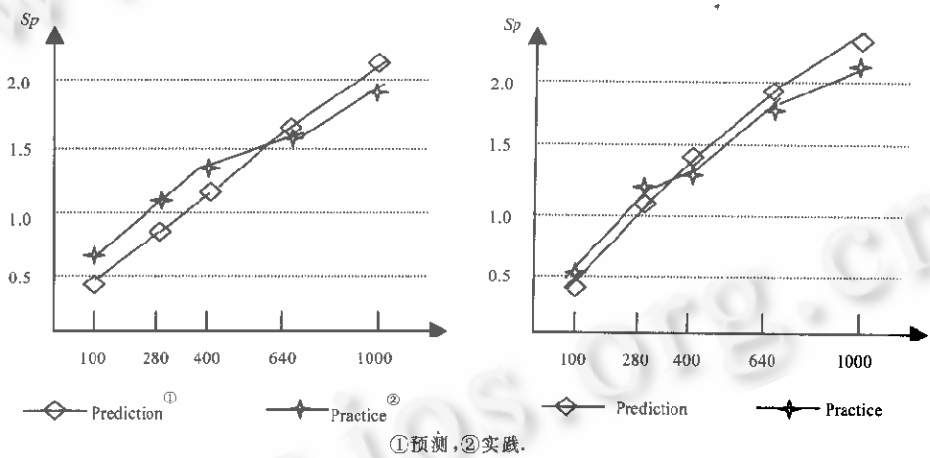
①算法 1 的预测结果,②算法 2 的预测结果.

Table 2 Practical results for algorithm 1 and algorithm 2

表 2 算法 1 和算法 2 的实际运行结果

N	Practical results for algorithm 1①					Practical results for algorithm 2②				
	Sp	Eff	E_{load}	E_{cmmp}	E_{ldcm}	Sp	Eff	E_{load}	E_{cmmp}	E_{ldcm}
100	0.57	0.14	0.86	0.94	0.85	0.49	0.12	0.90	0.94	0.90
280	1.10	0.28	0.79	0.71	0.73	1.24	0.31	0.85	0.75	0.81
400	1.30	0.33	0.79	0.75	0.68	1.36	0.34	0.85	0.63	0.77
640	1.52	0.38	0.82	0.39	0.63	1.93	0.48	0.86	0.46	0.73
1000	1.8	0.46	0.86	0.24	0.70	2.32	0.58	0.88	0.32	0.70

①算法 1 的实际运行结果,②算法 2 的实际运行结果.



①预测,②实践.

(a) Algorithm 1
(a) 算法 1

(b) Algorithm 2
(b) 算法 2

Fig. 1
图 1

5 结束语

在某一并行环境下的 BSP 模型的 g,l,s 通过实验得到以后,程序员即可根据以上介绍的性能分析效率评价模型,对算法和程序进行分析、评价乃至优化.所有这些都可在算法分析设计、编程调试阶段完成,无需运行程序.BSP 程序可移植性强,且同一程序仅只要替换不同并行环境的 g,l,s 即可得到该环境下的性能分析和并行效率.本文提出的 BSP 程序的可预测并行效率的评价模型,使得在设计、分析或编译、调试阶段即可根据当前并行环境从多个并行算法中选择较优的算法,在保持并行程序代码的可移植性下,始终可以选取具有较优性能的程序,即实现了并行程序效率的可移植性.另外,本文的伪代码结构可进一步规范化,这样可以通过开发一个工具来自动评估 BSP 程序的性能和效率.

References:

[1] McColl, W.F. Scalable computing. LNCS 1000, Berlin: Springer-Verlag, 1995. 46~61. http://www.bsp-worldwide.org.

[2] Valiant, L.G. A bridge model for parallel computation. Communications of the ACM, 1990,33(8):39~47.

[3] Jonathan, M., Hill, D., Skillicorn, D.B. Lessons learned from implementing BSP. Technical Report, PRG-TR-21-96, Oxford University Computing Laboratory, 1996. http://www.bsp-worldwide.org.

[4] Hwang, Kai. Advanced Computer Architecture: Parallelism, Scalability, Programmability. New York: McGraw-Hill, Inc., 1993.

[5] Ding, Y.Z., Stefanescu, D.C. Efficient scheduling of loop nests for BSP programs. Technical Report, TR-04-95, Oxford University Computing Laboratory, 1995. http://www.bsp-worldwide.org.

[6] Yuan, Wei, Sun, Yong-qiang. Parallel programming with predictable performance. Journal of Software, 1997,6(supplement):465~472 (in Chinese).

附中文参考文献:

[6] 袁伟,孙永强.可预测并行性能的并行程序设计.软件学报,1997,6(增刊):465~472.

An Evaluation Model for Predicting the Efficiency of Parallel Programs

CHEN Chang-sheng[1], SUN Yong-qiang[1], HE Ji-feng[2]

([1]*Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai* 200030, *China*);

([2]*Department of Computer Science, Normal University of East China, Shanghai* 200063, *China*)

E-mail: Sun-YQ@cs.SJTU.edu.cn

http://www.sjtu.edu.cn

Received July 28, 1999; accepted September 10, 1999

Abstract: The performance of parallel programs, especially the parallel efficiency evaluation, usually needs to get data after running the program. Then through checking the experimental results, the programmer optimizes the parallel algorithms, modifies data placement policies, or even selects other algorithms. In this paper, based on the BSP parallel programming model, an evaluation model for predicting the efficiency of parallel programs is presented, which allows programmer to evaluate the efficiency at the stages of algorithm design and programming, and then take some measures to optimize the programs further. The experimental results show that the evaluation predication model is accurate.

Key words: parallel programming; BSP (bulk-synchronous parallel) model; efficiency evaluation rule