文章编号:1671 - 6833(2009)02 - 0080 - 04

关于并行程序设计方法的分析与研究

王文义[1], 赵建建[1], 王若雨[2]

(1.中原工学院 并行处理技术研究所,河南 郑州 450007; 2.河南电力职工大学 网络信息中心,河南 郑州 450051)

摘 要:并行程序设计与并行计算机的体系结构密切相关,因此其复杂性要远远大于串行程序设计.介绍了数据分解和循环体依赖等概念,提出了一个 cache 利用率和并行计算机有效速度的近似关系模型.通过该模型和一个实例,阐述了在并行程序设计中降低和拆解计算目标中数据依赖的方法过程,从而达到尽可能多地发掘指令级并行性,提高 cache 利用率即提高并行系统有效速度的目的.

关键词:循环体依赖;Cache 利用率;循环展开;指令级并行

中图分类号:TP 311 文献标识码:A

0 引言

高性能计算(HPC)技术[1]已日益显示出其重要性,因为有太多与国家发展息息相关的领域和课题都离不开它.并行程序设计的方法研究是 HPC 的重要组成部分之一.出于对并行计算机系统的成本和并行程序设计复杂性(严重依赖硬件特性)的考虑,尽管随着时间的推移,该领域已逐渐向普通民众露出了庐山面目,但是要想达到像串行程序设计那样普及还需要走很长的路.笔者在长期从事并行处理技术研究的实践活动中,总结了一些并行程序设计需要注意的事项和并行程序在某些高性能计算机上获得的运行数据,在此整理出来,以飨读者.

1 数据依赖性

1.1 循环体依赖分类

在并行程序设计中,为了发掘潜在的并发任务执行的可能性,我们必然要从目标数据的藕合度即数据依赖性[2]入手.为了实现并行执行,经常要对目标数据进行必要的拆解或变换.与串行计算机一样,并行计算机的性能也同样是大部分消耗在了对循环体的迭代执行上,因此应该十分关注循环依赖(Loop - Carried Dependence)问题.通常,循环依赖存在下述几种情况:

(1) 回溯式循环依赖 B - LCD (Backward Loop - Carried Dependence)

```
DO j=2,N
  A(j) = A(j-1) *2.0
ENDDO
```

这里,由于 A(j)依赖于由先前迭代中所产生的 A(j-1),所以我们无法把循环中的每次迭代置为并发作业执行.

(2) 前向式循环依赖 F - LCD (Forward Loop - Carried Dependence)

```
DO j=1,N
  A(j) = A(j+1) *2.0
ENDDO
```

由于 A(j)依赖于在循环执行前被赋值的 A(j+1)并将在下一次迭代中被计算,因此也无法把循环中的每次迭代设置为并发任务执行.

(3) 输出循环依赖 O - LCD(Output Loop - Carried Dependence)

```
DO j=1,N
  A(L(j)) = B(j) + C(j)
ENDDO
```

由于我们不知道下标 L(j)是否含有重复值,如存在 L(3) = L(12) = 7 的情况,那么两次不同迭代就可能把值赋给一个相同的地址.若这时把迭代设置成并发任务,就必须要十分小心.

收稿日期:2008 - 10 - 20;修订日期:2009 - 01 - 11

基金项目:河南省基础与前沿技术研究项目(082300410300)

作者简介:王文义(1947 -),男,河南洛阳人,中原工学院教授,研究方向为并行处理技术和高性能科学计算,E-mail:wwy@zzu.edu.cn.

万方数据

(4)外观循环依赖 A - LCD(Apparent Loop - Carried Dependence)

```
DO j = 1,N
  A(j) = A(L(j)) + C(j)
ENDDO
```

对于这种情况,由于无法确定下标 L(j)是属于上述情况的哪一种,故称为外观循环体依赖.除非程序员对程序非常了解并且确信不存在数据依赖,否则就不能冒然对它进行并行化.

1.2 降低数据依赖与实现并行计算

通过下面实例,说明降低模型中的数据依赖继而实施并行计算的优点.

在量子色动力学 QCD[3](Quantum Chromo-Dynamic)研究中,费米矩阵 $M_{i,j}$ 是一个大规模非厄密复数矩阵.即使采用最基本的 $24^3 \times 32$ 规模,其非零元素也需要 4GB 的存储空间.我们可以对其进行分解处理以降低数据依赖程度,算法过程如下.

$M_{i,j}$ 的下标结构为:

$$M_{i,j} = M_{(x,y,z,t,\mu,\alpha),(x',y',z',t',\mu',\alpha')},$$

首先让

$$n = x + (y-1) * nx + (z-1) * nx * ny + (t-1) * nx * ny * nz,$$

然后把 n 分解为 $L = n/(\mathrm{NCPU}/2)$,以得到 $M^k_{i,j}$,其中 $i = 1,2,\cdots,L; j = 1,2,\cdots,n; k = 1,2,\cdots(\mathrm{NCP}/2)$.于是我们可以把 $V = M \cdot g$ 并行化为

$$V'_1 = M^1_{i,j} \cdot g$$
$$V'_2 = M^2_{i,j} \cdot g$$
$$\cdots\cdots$$
$$V'_{\mathrm{NCPU}/2} = M^{\mathrm{NCPU}/2}_{i,j} \cdot g$$

由于矩阵 M 的物理定义为[3]:

$$M_{aax,b\beta y} = \delta_{a,b}\delta_{\alpha,\beta}\delta_{x,y} - k\sum_{\mu=1}^{4}[(1-\gamma_\mu)_{\alpha\beta}U_\mu(x)_{ab}\delta_{x,y-\hat{\mu}} + (1+\gamma_\mu)_{\alpha\beta}U_\mu(x)_{ab}\delta_{x,y+\hat{\mu}}]$$
$$= I + M_{for} + M_{bac}$$

于是,$M \cdot V = V + M_{for} \cdot V + M_{bac} \cdot V$,这种形式适合于并行化处理.根据该方法,对 QCD 大规模矩阵求逆处理,可以获得很好的加速比.测试结果如表 2 所示.

表 1　QCD 大规模矩阵求逆的测试(SPP2200,8 个处理器)

Tab.1　The test for cosmically matrix reverse of the QCD(SPP2200,8processor)

迭代次数	CPU 时间/s	Wall Clock 时间/s	加速比
109	5 322.8	835.7	6.37

由此我们不难看出进行数据分解后采用并行算法处理大规模问题时所具有的优势.

2　局部缓存的利用率与系统有效速度

一般高性能并行机中的各个处理器自身都带有局部高速缓存[4](Cache),而处理器对自身局部 Cache 的访问速度明显地要比访问其它存储器快得多.因此要想设计出具有较高计算性能的并行应用程序,就必须要考虑如何合理地使用 Cache.

在影响并行机应用性能的各种延迟中,Cache 延迟差不多要占到整个延迟的 70% 左右,因此如何保证在运算时让数据到达 Cache 而无须等待是提高 Cache 的利用率(CUR Cache Using Ratio)的关键.

在 Cache 缺失延迟情况下,并行计算机的有效速度可以近似地表示为:

$$V_E \approx V_P * (1 - \mathrm{CMR}) = V_P * \mathrm{CUR} \quad (1)$$

式中:V_E,V_P 分别代表并行计算机的有效速度和理论峰值速度;CMR 代表 Cache 缺失率(Cache Missing Ratio),并且

$$\mathrm{CMR} = (\text{Cache 缺失时间})/(\text{CPU 时间}) * 100\% \quad (2)$$

$$\mathrm{CUR} = \text{指令执行总次数}/\text{访问数据总数} * 100\%\ [5] \quad (3)$$

对 CUR,我们可以这样理解,即基于当计算机的运算部件开始工作时,参与运算的数值应该已存在于 Cache 之中,因此可以把它视为是 Cache 中平均每个数据参加运算的次数,那么若平均每个数据参加运算的次数越多,则表示 Cache 的利用率也就越高,因此计算机的有效速度也就越高.

3　指令级并行与测试

3.1　指令级并行

并行程序设计中的指令级并行(细粒度)技术主要考虑推测式执行、指令的动态调度、软件流水和循环展开 4 种[6].考虑到高性能计算的核心问题往往都集中在对若干海量矩阵的处理(乘、转置、旋转、特征值等)上,同时实现该处理过程又离不开消耗大量计算机性能的循环,因此本文将侧重讨论循环展开技巧.

3.2　循环展开

对于多重嵌套循环体的并行设计,应该尽可能多地考虑如何把最大的计算工作量改造为适合于并行处理以获得理想的计算速度上.在并行程

万方数据

序设计中,循环展开是一个很好的降低数据依赖的方法,循环展开的目的是让指令实现并行化,以能够在无数据依赖的情况下得到顺序执行.

实例程序Ⅰ(F90):

```
DO I=1,N
  A(I)=A(I)+B(I)*C
ENDDO
```

我们可以将其改造为:

```
DO I=1,N,4
  A(I)=A(I)+B(I)*C ;A(I+1)=A(I+1)
  +B(I+1)*C;
  A(I+2)=A(I+2)+B(I+2)*C ;A(I+3)
  =A(I+3)+B(I+3)*C
ENDDO
```

在该程序中,由于循环体内各语句之间不存在数据依赖,因此它适合于并行处理.以理论物理的高性能计算课题——QCD中的Monto Carlo模型(F90程序[7-8])为例:

程序Ⅰ:

```
INTEGER,PARAMETER::N=1000000
COMPLEX(16),dimension(3,3,N)::w,u,v
w=0.0_high
DO i=1,N
  k=j(i)
DO j1=1,3
DO j2=1,3
DO j3=1,3
  w(j2,j1,i)=w(j2,j1,i)+u(j2,j3,i)*v(j3,
  j1,k)
ENDDO
ENDDO
ENDDO
ENDDO
```

程序存在四重循环,并且矩阵 w,u,v 都是复数类型.由于在循环体中对 w 矩阵各分量的运算过程中存在循环依赖,因此我们无法把循环中的每次迭代置为并发作业执行.为此,我们在保留外层循环控制变量 i 的前提下,通过对第2~4层循环进行展开,从而把上述程序改造成如下形式:

```
    w(1,1,i)=u(1,1,i)*v(1,1,k)+u(1,2,
i)*v(2,1,k)+u(1,3,i)*v(3,1,k)
    ……
    w(2,2,i)=u(2,1,i)*v(1,2,k)+u(2,2,
i)*v(2,2,k)+u(2,3,i)*v(3,2,k)
    ……
    w(3,3,i)=u(3,1,i)*v(1,3,k)+u(3,2,
i)*v(2,3,k)+u(3,3,i)*v(3,3,k)
```

做此处理后,由于 w,u,v 是复数矩阵,仍然不便于并发执行.鉴于上述9项中每一项的前两个下标都是常量并都是复数类型,所以为了实现指令级并行,我们可以把每一句的实部与虚部拆解为独立的两部分,并对拆解后 w 矩阵的各个下标进行重新命名:

```
    w(1,i)= ……      /* 原w(1,1,i)的实部
    */
    w(2,i)= ……      /* 原w(1,1,i)的虚部
    */……
    w(18,i)= ……     /*原w(3,3,i)的虚部
    */
```

然后我们得到下面与程序Ⅰ等价的程序程序Ⅱ:

```
INTEGER,PARAMETER::N=1000000
real(8),dimension(18,N)::u,v,w /* 把原来的
复数定义改为相应的实数定义并且数组需重新说
明 */
w=0.0_high
DO i=1,N
k=j(i)
w(1,i)=u(1,i)*v(1,k)-u(2,i)*v(2,k)+
u(3,i)*v(7,k)-u(4,i)*v(8,k)+u(5,i)*
v(13,k)-u(6,i)*v(14,k)
……
w(6,i)=u(1,i)*v(6,k)+u(2,i)*v(5,k)+
u(3,i)*v(12,k)+u(4,i)*v(11,k)+u(5,i)
*
v(18,k)+u(6,i)*v(17,k)
……
w(18,i)=u(13,i)*v(6,k)+u(14,i)*v(5,k)
        +u(15,i)*v(12,k)+u(16,i)*
        v(11,k)+u(17,i)*v(18,k)+u(18,
        i)*v(17,k)
```

至此,我们把一个四重循环变成了一个单循环顺序执行的形式,其中各个语句彼此独立,耦合度(依赖度)均等于零.它具有如下特征:

(1)循环中所有18个句子彼此都不存在依赖关系;

(2)循环中的每个句子都有12次运算,因此是负载平衡的;

(3)每次循环的总运算次数为18*12=216次.

3.3 Cache利用率与有效速度测试

上述Monto Carlo模型程序在实现指令级并

万方数据

行前 CUR = (27 * 3)/(27 * 4) = 3/4 = 0.75,实现指令级并行后 CUR = 216/234 = 0.923.

该程序在几种不同体系结构的高性能并行计算机上的测试结果列在表 2 之中.实验结果(均使用 4 个处理器)表明,实现指令级并行可以大幅提高 Cache 利用率(均达到了 45% 左右),考虑到当前绝大多数并行计算机的实际效率仅有峰值速度的 25% 左右的现象,因此说实现指令级并行对于提高并行应用程序的有效性能的效果是明显的.

表 2　QCD Monto Carlo 代码的并行机测试

Tab. 2　The Monto Carlo code test for QCD On parallel computers

机型	体系结构	浮点指令数/(条·周期$^{-1}$)	V_P/Mflops	指令并行前		指令并行后	
				V_E/Mflops	CUR/%	V_E/Mflops	CUR/%
Oringin2000	分布式共享	2	390	39.8	10.1	170.9	43.8
IBM SP2	分布式	4	1500	214.7	14.3	736.5	49.1
InfiniBand Ⅲ集群	分布式	2	4400	403.2	9.2	1820.4	41.4

4　结束语

对于大多数从事 HPC 课题的技术人员来说,都很清楚并行计算机的运行性能大部分都消耗在对大容量矩阵的运算处理上,而该处理的实现又离不开具有数据依赖的多重循环.笔者侧重从软件(也少量涉及到硬件,如不同处理器的浮点部件个数等)出发,提出如何运用循环展开技术以及 Cache 利用率的概念,通过实例分析和在不同体系结构的并行计算机上的实验验证,来说明如何降低循环层次和数据依赖程度以提高指令级并行的过程,从而向读者展示了一条可行的提高并行应用程序执行效率的途径.

参考文献:

[1] 张林波,池学斌.并行计算导论[M].北京:清华大学出版社,2006.

[2] HWANG K. Advanced computer architecture: parallelism scalability programmability [M]. New York: McGraw - Hill Inc,1993:51 - 57.

[3] DONG S,WANG W. A parallelization test on SPP1200 of a large scalar matrix computation - lattice Quantium Chromodynamic(QCD) and the hierarchical problem [C]//Proceeding of Convex/HP user group worldwide conference, 1996:437 - 454.

[4] HAO J, ROSE K A. Making B + - trees cache conscious in main memory[C]//Proc of 2000 ACM SIGMOD International Conference on Management of Data,Dallas,2000:475 - 486.

[5] 王文义,董绍静. 高性能科学计算的并行程序设计方法研究[J]. 计算机工程,2002,28(12):83 - 85.

[6] HENNESSY J L, PATTERSON D A. Computer architecture:A quantitative approach[M]. Third Edition. 北京:机械工业出版社,2004:113 - 180.

[7] GROUP W, LUCK E, SKJELLUM A. Using MPI: portable parallel programming with the message passing interface [M]. 2nd Edition. Cambridge, MA: MIT Press, 1999.

[8] 汪同庆. Fortran90 程序设计[M]. 武汉:武汉大学出版社,2004.

Analysis and Study of Methods on Parallel Programming

WANG Wen - yi[1], ZHAO Jian - jian[1], WANG Ruo - yu[2]

1. Institute of Parallel Processing Technology,Zhongyuan Institute of Technology, Zhengzhou 450007, China; 2. Network and Information Center, Henan University of Electric Power Workers,Zhengzhou 450051, China)

Abstract: Parallel programming has close relation with the parallel computer architecture, so its complexity is far larger than serial programming. The paper introduced the concept of data decomposition and loop - carried dependence etc., and presented an approximately relational model between cache using ratio and the parallel computer effective speed. In order to excavate the instruction - level parallelism as much as possible in the program and to enhance cache using ratio, namely effective speed of parallel system, the paper also presented a method of how to split and reduce the degree of the data dependence in whole computing objects, with the above - mentioned model and an instance.

Key words: loop - carried dependence;cache using ratio; loop unrolling;instruction - level parallelism

万方数据