ISSN1000-9825 Journal of Software 软件学报 2000,11(4):507~514

CC-NUMA 并行系统通信优化的变换技术*

陈渝 李春江 杨学军 陈福接

(国防科学技术大学计算机学院 长沙 410073)

E-mail: fqian@nudt.edu.cn

摘要 减少通信开销是并行编译优化的主要目标之一.该文针对具有 cache 一致性的非一致存储访问并行系统(CC-NUMA)的特点,提出通过结合计算变换和数据变换,在统一的代数框架下对并行程序进行通信优化的策略和方法.通过实验测试,验证了此策略和方法是行之有效的.

关键词 并行编译,通信优化,计算变换,数据变换,具有 Cache 一致性的非一致存储访问.

中图法分类号 TP311

具有 cache 一致性的非一致存储访问 CC-NUMA(cache coherent-non uniform memory access)并行系统,比对称多处理 SMP(symmetric multi-process)共享存储系统有着更好的可扩展性,而且它在硬件上提供了 cache 的一致性维护和全局地址空间的访问,简化了并行编程.这些使得它在技术角度和商业角度上都有很好的发展前景,正逐渐成为超级计算的主流之一.因此,如何通过并行编译优化提高并行程序在 CC-NUMA 并行系统中的执行效率,有着重要的理论和实际意义.

在 SMP 系统中,内存是全局统一编址,具有一致的内存存取时间,不存在远地内存数据访问的问题,但由于自身结构的限制,使得它的可扩展性差.CC-NUMA 系统在硬件上作了很大的革新,在结构上突破了在 SMP 中可扩展性差的缺点.它在硬件上实现了统一的全局编址,所有的内存空间可以被任意节点访问.远地节点内存的数据可以被本地节点的 cache 缓存,由硬件保证 cache 的一致性.

目前,有不少学者试图通过计算变换开发数据的局部性.例如,Wolf 和 Lam[1]在定义和分析了数据局部性的基础上,通过单模(unimodular)变换来开发程序的并行性和减小通信开销,但却没有考虑计算变换的局限性和数据变换的特点.Wei Li 和 Pingali[2]使用线性变换技术来进行数据局部性优化,他们统一考虑了计算变换和数据变换,但所考虑的仅是简单的线性变换,没有对计算扭曲变换和数据扭曲变换进行分析.Bugnion 等人[3]提出用 page color 技术来提高 SMP 系统中的 cache 利用率,但由于内存页的粒度比 cache 行的粒度大,所以不能精确地分析出 cache 的使用情况并加以改进.

综上所述,大多数学者在计算变换方面进行了深入的研究,但对数据变换的研究还不够.而且,目前关于数据局部性的研究主要是针对 SMP 共享存储并行系统(不考虑数据的分布)和非共享存储并行系统(对远地数据不能进行本地 cache),关于 CC-NUMA 并行系统的数据局部性的研究还很少.因此,有必要对 CC-NUMA 以及数据变换进行深入的研究.

本文主要考虑在 CC-NUMA 系统中,如何通过变换技术来开发数据的局部性,减少访存冲突通信开销,特别是,由于 cache 和内存页访问冲突引起的通信开销.由于在 CC-NUMA 系统中,cache 和内存页存在着对同一内存单元的多个拷贝,同时也存在着远地内存访问.因此,影响并行程序性能的主要访存通信开销包括:cache 真共享冲突开销、cache 假共享冲突开销、内存页的假共享冲突、cache 不命中开销、页不命中开销和远地访存

---

* 本文研究得到国家杰出青年科学基金(No.69825104)资助.作者陈渝,1972 年生,博士生,主要研究领域为并行编译优化.李春江,1975 年生,硕士生,主要研究领域为并行编译优化.杨学军,1963 年生,教授,博士生导师,主要研究领域为并行计算机系统结构,并行操作系统,并行编译.陈福接,1942 年生,教授,博士生导师,主要研究领域为并行计算机系统结构.

本文通讯联系人:陈渝,长沙 410073,国防科学技术大学计算机学院博士生队

本文 1998-12-15 收到原稿,1999-06-21 收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

开销[4].

本文减少访存开销的出发点是通过分析并行程序的访存模式,在统一的代数和集合表示框架下把计算变换和数据变换结合起来,解决CC-NUMA中存在的cache冲突.至于减少远地访存开销的技术,许多学者对此都进行了较为深入的研究,而且在CC-NUMA系统中,如果远地内存的数据能被本地cache存储,并且有良好的重用性和局部性,则数据通信开销就可以较好地得到控制.

1 计算变换、数据变换的代数和集合表示框架

为了表示嵌套循环,本文定义n层嵌套循环的迭代空间$I\subset Z^n$和一个m维数组的数据空间$A\subset Z^m$.设n层嵌套循环中第k层是并行循环(假设并行系统实际只并行执行1层并行循环,其他并行循环串行执行),它的取值范围是$1\sim N_k$,则可定义并行迭代子空间集合I_p(为了简化定义,本文假设嵌套循环中只存取1个数组空间)如下:

定义 1.1. 并行迭代子空间集合:

$$Ip=\{Ip_{i_k}\mid Ip_{i_k}=\{(*,\ldots,i_k,\ldots,*)\},1\leqslant i_k\leqslant N_k\}.$$

其中i_k是第k个循环迭代向量元素的值,$*$表示相应迭代向量元素的所有可能取值.具有相同的$i_1,i_2,\ldots,i_{k-1}$值的Ip_{i_k}迭代子空间可并行执行,但在并行执行后可能要进行同步,I_p内迭代是串行执行的.

定义 1.2. 迭代空间对数据空间的访问模式映射$h(I)\to A$,即

$$h(I)=\{a=i*T+c\mid i\in I,a\in A\}.$$

其中a是m维数据向量,i是n维迭代向量,T是$n*m$维存取矩阵,c是m维偏移向量.

定义 1.3. 并行迭代子空间集合Ip的访问集合:

$$Gp=\{Gp_{i_k}=\mid Gp_{i_k}=\{h(Ip_{i_k})\mid Ip_{i_k}\in Ip\},1\leqslant i_k\leqslant N_k\}.$$

所有Gp_{i_k}的并是A的一个子集.

定义 1.4. 迭代空间变换的表示$I'=I*R+r$,R是变换矩阵,r是偏移向量.

定义 1.5. 数据空间变换的表示$A'=A*S+s$,S是变换矩阵,s是偏移向量.

定理 1.1. 如果Ip中的$Ip_1,\ldots,Ip_{N_k}$相应的访问数据集$Gp_1,\ldots,Gp_{N_k}$两两相交为空,那么就没有cache真共享冲突(不考虑由于cache容量有限所带来的影响).

证明:Cache真共享冲突是由于并行执行的Ip_i和Ip_j对同一个内存单元进行了写操作而造成的.由题设可知,对Ip中任意Ip_i和Ip_j的相应访问数据集Gp_i,Gp_j有$Gp_i\cap Gp_j=\varnothing$,则不存在$Ip_i$和$Ip_j$对同一个内存单元进行写操作,即不存在cache真共享冲突. □

假设存在一个迭代变换矩阵R和相应的迭代偏移向量r,通过它们对迭代空间进行变换可消除cache真共享冲突.同时,假设存在一个数据变换矩阵S和相应的数据偏移向量s,通过它们对数据空间进行变换可减少或消除cache假共享冲突.这样就可以把它们结合起来,同时进行变换,以减少或消除cache冲突.

首先,通过计算变换改变程序的循环索引的表示:$I'=I*R+r$.这时,新的存取模式为$h(I')=((I'-r)*R-1)*T+c$,再计算出通过数据变换产生的新的存取模式:$h'(I')=(((I'-r)*R-1)*T+c)*S+s$.最后,根据新的存取模式修改数组的索引下标.这样就可以统一地用代数方法消除cache冲突(下面的变换都是以Fortran语言作为参考的).

2 用计算变换消除cache真共享冲突

2.1 并行计算机系统模型

为了简化讨论,本文考虑全局统一编址的CC-NUMA的多层次存储结构,它的内存层次可分成3层:远地内存空间、本地内存空间和本地cache.处理机对远地内存空间的访问开销比本地内存要大.同时,cache可以保存本地或远地内存空间的数据,并通过写回策略来保证cache的一致性.由于只考虑真共享引起的冲突,本文假设本地cache具有无限大的容量来排除由于cache容量有限所带来的其他开销.同时,假设每个cache行只含一

© 中国科学院软件研究所 http://www.jos.org.cn

个字,以忽略假共享引起的冲突.

2.2 程序模型

本文仅考虑最外层循环为串行循环的嵌套循环(下面的分析也适用于有串行关系的多个并行循环体).为了简化讨论,本文假设忽略并行循环内语句间的相关关系,循环的下限和增量步长都是1,所有数组下标表达式为线性函数.嵌套并行结构中仅含一层并行循环,它可能包含一个串行循环,并被包含在一个串行循环中,如果存在多层并行循环,则仅并行执行其中一层[5].

基于上面的假设,本文重点考虑三层嵌套循环结构:一个并行循环,它包含一个串行循环,同行它也被一个串行循环包含.如下所示:

```
DO I1=1,N1
  ...
    DOALL I2=1,N2
      ...
        DO I3=1,N3
          ...
          A1(h1(I1,I2,I3))=...
          ...=A1(h2(I1,I2,I3))
          ...
        ENDDO
      ...
    ENDDOALL
  ...
ENDDO
```

2.3 优化策略

为了更好地说明问题,本文首先讨论简单的单个二维数组 A.它的内存布局是列优先的,其下标表达式为单个线性函数.对于多个数组或多线性函数的情况可以此为基础进行推广.下面是一个具体的例子.

例1:

```
REAL A(M,M)
  DO I1=1,N
    DOALL I2=1,N
      DO I3=1,N
        A(I2+I3-1,I1+I3-1)=...
        ...=A(I2+I3-1,I1+I3-1)
      ENDDO
    ENDDOALL
  ENDDO
```

例1中的迭代空间 $I=(i_1,i_2,i_3)$,$1\leqslant i_1,i_2,i_3\leqslant N$,数据空间 $A=(a_1,a_2)$,$1\leqslant a_1,a_2\leqslant M$,迭代空间到数据空间的映射为 $h(I)$,通过分析我们可以得到存取矩阵

$$T=\begin{pmatrix}0 & 1\\ 1 & 0\\ 1 & 1\end{pmatrix},\quad c=(-1,-1).$$

根据 Z. Shen[6]对12个大型科学应用程序的统计结果,存取矩阵 T 中的元素值往往很小,其绝对值大于1的只有2.05%,最大系数绝对值为2.因此,本文的例子具有一定的普遍性.

根据定义1.1,可以得到并行迭代子空间 $Ip_k=\{(*,k,*)\}$,$1\leqslant k\leqslant N$,根据定义1.3,Ip_k 的访问数据集 $Gp_k=\{A(h(i))\mid i\in Ip_k\}$.通过对 Gp_k 的分析,我们可以发现它们的交集不为空,而且交集中元素个数很大.例如

$$Gp_1=\{(I_3,I_1+I_3-1)\mid 1\leqslant I_1,I_3\leqslant N\},\quad Gp_2=\{(I_3+1,I_1+I_3-1)\mid 1\leqslant I_1,I_3\leqslant N\},$$

则

$$Gp_1\cap Gp_2=\{(I_3,I_1+I_3-2)\mid 2\leqslant I_1,I_3\leqslant N\}.$$

为了使 Gp_k 之间的交集为空,需要对循环进行重组,使得重组后 Ip'_k 之间的访问数据集不相交.

© 中国科学院软件研究所 http://www.jos.org.cn

定理2.1. 对于 Ip_k 中的迭代子空间 $\{(k_1,k,*)\}$,把它移入另一个并行迭代子空间 Ip_j 中,只要保证在同步点上的同步和迭代顺序,则重构后的 Ip' 子空间并行执行不影响程序执行的准确性.证明略.

在例1中可以明显地看到,每执行 I_1 的一次循环,就要完成一次同步.这样,对于一个确定的 I_1,I_2 的 N 个循环可以任意分布在 P 个处理机上执行(假设不考虑负载平衡).如果通过对 Ip 中的迭代子空间进行重新分布,使 Ip 子空间之间满足定理1.1,那么就可以消除 cache 真共享冲突了.为此,本文给出如下定义和定理.

定义2.1. 对于 I 中迭代子空间 $I_1\{(e_1,k_1,*)\}$ 和迭代子空间 $I_2\{(e_2,k_2,*)\}$,$(e_1,k_1)\neq(e_2,k_2)$,令它们的数据访问集分别为 G_1 和 G_2.如果 $G_1\cap G_2\neq\varnothing$,则记为 $I_1\delta I_2$.

定理2.2. 对于 I 中迭代子空间 $I_1\{(e_1,k_1,*)\}$ 和 I 中迭代子空间 $I_2\{(e_2,k_2,*)\}$,$e_1\neq e_2$,则 $I_1\delta I_2$ 当且仅当存在 j_1 和 $j_2(1\leqslant j_1,j_2\leqslant N)$,满足:$(e_1,k_1,j_1)*T=(e_2,k_2,j_2)*T$.

证明:如果 $I_1\delta I_2$,则根据定义2.1,有相应的 $G_1\cap G_2\neq\varnothing$,根据定义1.2,存在 $i_1=(e_1,k_1,j_1)$,$i_2=(e_2,k_2,j_2)$,使 $a=i_1*T=i_2*T$;反之亦然. □

定义2.2. 对于 I 中迭代子空间 $I_1\{(e_1,k_1,*)\}$ 和迭代子空间 $I_2\{(e_2,k_2,*)\}$,$e_1\neq e_2$,如果 $I_1\delta I_2$,则 $I_1,I_2\in Ip'_k$.

下面介绍如何构造满足定义2.2的 Ip' 并行迭代子空间.以例1为例,通过对定理2.2的进一步分析计算,发现满足 $I_1\delta I_2$ 的情况可简化为下面的形式:

令　　$i_1=(e_1,k_1,j_1)$,　$i_2=(e_2,k_2,j_2)$,　$\Delta i=i_1-i_2=(e_1-e_2,k_1-k_2,j_1-j_2)$,

则定义2.2的条件可简化为

$$\Delta i*T=0. \tag{1}$$

令 $\Delta e=e_1-e_2,\Delta k=k_1-k_2,\Delta j=j_1-j_2$,解式(1)可以得到

$$\Delta e=\Delta j,\quad \Delta k=\Delta j,\quad \Delta e=\Delta k,\quad 0\leqslant\Delta j\leqslant N-1.$$

即对于迭代子空间 $I_1=\{(e,k,*)\}$ 和任意迭代子空间 $I_2=\{(e_2,k_2,*)\}$,只要满足

$$r=e_2-e=k_2-k,\quad \text{且}0\leqslant r\leqslant N-1,$$

它们就同属于一个 Ip'_i.本文定义它包含在并行迭代子空间 Ip'_i 中,表示为

$$Ip'_i=\{(e+r,k+r,*)\mid r\in Z,1\leqslant e+r\leqslant N,1\leqslant k+r\leqslant N\},$$

其中 Z 为整数集合.

如果用迭代变换矩阵和迭代偏移向量来表示变换后的 $Ip'_k=\{(N-k+r,r,*),1\leqslant r\leqslant N$ 且 $1+k-N\leqslant r\leqslant k\},1\leqslant k\leqslant 2N-1$,则

$$R=\begin{bmatrix}1&-1&0\\0&1&0\\0&0&1\end{bmatrix},\qquad r=(0,N,0).$$

显然,所有的 Ip' 可并行执行,而且相互之间不存在真共享冲突.但在 CC-NUMA 系统中,Ip' 间可能产生大量的远地数据访问和伪共享冲突.因此,本文下一步将采用数据变换技术,以消除 cache 假共享冲突和减少远地数据访问.

3　用数据变换消除 cache 假共享冲突

本节取消了第2节中对系统模型的部分假设,认为 cache 行包含多个字,即存在 cache 假共享冲突,这样更符合实际的 CC-NUMA 系统.在 CC-NUMA 系统中,数组的布局应尽量提高并行程序的局部性,将并行程序所需的数据尽量放置在本地内存.但是在例1中,Ip' 中的迭代子空间 I'_{ij} 的存取模式是沿着对角线方向访问的,因此采用标准的 Block,Cyclic 或 Block(n)方式来放置数组,都不能消除 cache 伪共享冲突.但是,通过数据变换却可以解决这个问题.我们首先介绍多维数组线性地址的代数表示和数据变换的代数表示.

3.1　数组的线性地址代数表示

为了反映出多维数组在内存的实际放置,需要对数组进行线性表示[7].对一个 n 维数组,建立一个 $(n+1)$ 维

© 中国科学院软件研究所 http://www.jos.org.cn

定位向量 P,一个$(n+1)$维索引向量 m,以二维数组 $A(n,n)$为例,它的元素 $A(i,j)$的索引向量 m 为$(i,j,1)$,它的定位向量 P 为$(1,n,-(n+1))$.设起始地址为0,则 $A(i,j)$的线性地址可表示为

$$Addr(A(i,j))=P*m^t=(1,n,-(n+1))*\begin{bmatrix}i\\j\\1\end{bmatrix}=i+n*j-n-1. \tag{2}$$

这里,m 和 P 的最后一个元素是用来表示地址偏移量的,所以 m 的最后一个元素值为1.为了更清晰地表示迭代子空间对数组的访问模式,本文定义了迭代的线性地址访问模式:

$$Addr(I)=P*(h(I))^t. \tag{3}$$

由于间隔访问距离与数组线性地址的偏移量无关,所以,可把定位向量 P 的最后一个表示偏移量的元素去掉,用 P'表示.P'实际上表示了数组索引下标的相邻变化在数组线性地址中的间隔.同时,为了反映出迭代子空间相邻元素对数组线性地址的访问距离,本文定义了间隔向量 iv(interval vector):

$$iv^T=I'*P'^T. \tag{4}$$

间隔向量 iv 实际上反映了程序对数据访问的间隔距离.对于 n 层嵌套循环,可以得到一个 n 维向量,其中第 i 个元素的绝对值表示了相应的第 i 层循环对数据访问的间隔距离.特别是最后一个元素的绝对值表示了最内层循环对数据访问的间隔距离.

定理3.1. 设一个 cache line 的长度为 m,有一个 n 维间隔向量 $iv(v_1,v_2,\ldots,v_n)$,其中第 k 个向量元素 $|v_k|$是可并行执行的循环对数据访问的步长,称 iv 的空间局部性是优良的,如果:

(1) $|v_k|>m$;

(2) $|v_{(i+1)}|\leqslant|v_j|$,$1\leqslant j\leqslant n-1$且 $j\neq k$;

(3) $|v_n|<m$.

定义3.1. 设一个 cache line 的长度为 m,设有两个 n 维间隔向量 $iv_1(v_{11},v_{12},\ldots,v_{1n})$和 $iv_2(v_{21},v_{22},\ldots,v_{2n})$,其中第 k 个向量元素 v_{1k},v_{2k}是可并行执行的循环对数据访问的步长,称 iv_1的空间局部性优于 iv_2,如果:

(1) $|v_{1k}|\gg m>|v_{2k}|$;

(2) 令 $niv'_1=(|v_{1n}|,\ldots,|v_{1(k+1)}|)$,$niv'_2=(|v_{2n}|,\ldots,|v_{2(k+1)}|)$,则 $niv'_1\leqslant niv'_2$"$\leqslant$"表示按字典序小于).

3.2 数据变换的代数表示

在 Fortran 或 C 语言中,对多维数组在内存中的线性地址存放有直接的规定.对于某些程序而言,由于其嵌套循环中的数据访问模式的特点,可能使其局部性受到很大的损失,因此需要通过数据变换来提高数据访问的局部性.目前,数据变换的类型[1]主要有3类:转置(permutation)变换、扩维(extension dimension)变换和扭曲(skew)变换.当然,还存在其他类型的数据变换,但这3种数据变换更具有通用性和实用性.

3.2.1 转置变换

一个 n 维转置变换矩阵 S 可以改变一个 n 维数组的线性地址空间,它用来改变 n 维数组中各维的优先次序.以二维数组为例,它的转置变换矩阵只有一种:

$$S=\begin{pmatrix}0&1\\1&0\end{pmatrix},\qquad \text{偏移向量 } s=(0,0).$$

它实际上是把以列优先的二维数组变成以行优先的二维数组.通过分析可以发现,对于一个 n 维数组,它的转置方式有 $n!-1$种,即它有 $n!-1$种转置矩阵,而且这些矩阵都是 unimodula 矩阵.因为通常的应用软件很少有超过5维的数组,大部分都是2维和3维数组,所以转置变换矩阵的数目不会很大.

3.2.2 扩维变换

一般而言,扩维变换只针对 n 维数组的某一维.$n*(n+1)$或 $n*(n+2)$维扩维变换矩阵可改变一个 n 维数组,使之变成一个$(n+1)$或$(n+2)$维数组,它本身不改变数组的空间布局,但再通过数据分布,就可以改变数组的物理线性地址空间.由于在 CC-NUMA 系统中,数据的分配是以内存页为单位的,如果对数据的分布间隔小于一页,那么在支持页复制的 CC-NUMA 系统中就会产生在页这一级的伪共享或远程存取.同理,如果对数据的分布间隔小于一个 cache 行,就会产生 cache 一级的伪共享.因此,可以通过扩维变换,使数据的分布间隔大于

© 中国科学院软件研究所 http://www.jos.org.cn

一页或一个 cache 行.以一维数组 $A(N)$为例,设对于不同的分布策略,有不同的扩维变换矩阵[8]:

Block 分布:$S=(1/b,1\%b)$,

Cyclic 分布:$S=(1\%p,1/p)$,

Block(k)分布:$S=((1/k)\%p,1/(p*k),1\%k)$,

其中 p 是处理机个数,b 等于 N/p,k 是 block(k)分布的块大小.虽然在变换矩阵中存在整数除和整数取模的计算,但在实际的变换中可以通过各种优化技术来消除或尽量减少这两种计算.

3.2.3　扭曲变换

一个 n 维扭曲变换矩阵实际上是用来偏移一个 n 维数组某几维的数据.以二维数组为例,如果要把一个二维数组的列向量左偏移,它的扭曲变换矩阵则是

$$S=\begin{pmatrix}1 & -1\\ 0 & 1\end{pmatrix}.$$

虽然这样产生的矩阵会有很大的空间被浪费掉,但在实际问题中要用到数组扭曲变换的,一般是处理带状数组(banded matrix)的科学计算.而带状数组经过扭曲变换后,增加的数据量只占整个数组的一小部分.

对于稠密数组,本文采取了另一种扭曲变换矩阵.它利用循环偏移保证了变换后的矩阵不会增加内存开销,本文称之为循环扭曲变换矩阵.以二维数组 $A(N,M)$为例,如果要把一个二维数组的列向量向左循环偏移,它的代数表示为 $A'(i',j')=A(i,(j-i) \bmod (M)+1)$,它的扭曲变换矩阵可表示为

$$S=\begin{pmatrix}1 & -1\\ 0 & 1\end{pmatrix}_{(2)},\qquad s=(0,1).$$

其中,下标向量元素值表示相应维要做的模操作.

在实际应用中,通常只需对多维数组的某一维进行扭曲变换.每一维的扭曲变换类型又可分为4种,以二维数组为例,如果对列向量进行扭曲,则分别是左偏移、右偏移、左循环偏移、右循环偏移.对于 n 维数组,则一般有 $4n$ 种扭曲变换形式.下面将结合例1来具体说明如何通过数据变换来改善程序的空间局部性,并消除 cache 伪共享冲突.

3.3　优化策略

这里我们只对扭曲变换进行分析.虽然例1通过计算变换使重构后的 Ip' 之间存取的数据集的交为空,但通过对 Ip' 中的迭代子空间的存取模式进行分析就会发现问题.通过分析可以得到经过计算重构后的访问模式为

$$T'=R^{-1}*T=\begin{pmatrix}1 & 1\\ 1 & 0\\ 1 & 1\end{pmatrix},\quad c'=-r*R^{-1}*T+c=(-N-1,-1),\quad P=(1,M,-(M+1)),\quad P'=(1,M).$$

根据式(4),可得到间隔向量:

$$iv^T=T'*P'^T=(M+1,1,M+1)^T.$$

它的间隔向量 iv 不满足定义3.1,这样可导致大量的伪共享冲突和远地内存访问.通过分析我们发现,采用数据变换可以很好地解决这个问题.为了不增加额外的数据空间,本文采用左循环转移扭曲变换,令扭曲变换矩阵为

$$S=\begin{pmatrix}1 & -1\\ 0 & 1\end{pmatrix}_{(2)},\qquad \text{偏移向量 } s=(0,1),$$

则重构后的数组下标 $(i',j')=(i,j)*S+s$.直接用 S 来对 iv 进行代数表示比较困难,但通过分析重构后的代数下标,把取模操作用代数表达式和条件语句可以重新表示如下:

令 $S'=\begin{pmatrix}1 & -1\\ 0 & 1\end{pmatrix}$,

(1) $(i',j')=(i,j)*S'+(0,1),j\geqslant i$;

(2) $(I',j')=(i,j)*S'+(0,M+1),j<i$.

可以发现,实际上,循环转移变换的变换矩阵在各种条件下都可以认为是相同的,只是在不同的约束条件下,偏

© 中国科学院软件研究所 http://www.jos.org.cn

移向量是不一样的.这样就可以用 S' 表示变换后的定位向量 $\boldsymbol{P}''$:

$$\boldsymbol{P}''=S'*\boldsymbol{P}'^{T}=\begin{pmatrix}1 & -1\\ 0 & 1\end{pmatrix}*\begin{pmatrix}1\\ M\end{pmatrix}=\begin{pmatrix}1-M\\ M\end{pmatrix},$$

而间隔向量 $\boldsymbol{iv}'$ 可表示为

$$\boldsymbol{iv}'^{T}=T'*\boldsymbol{P}''=T'*S'*\boldsymbol{P}'=\begin{pmatrix}1 & 1\\ 1 & 0\\ 1 & 1\end{pmatrix}*\begin{pmatrix}1-M\\ M\end{pmatrix}=(1,1-M,1)^{T}.$$

可以看出,变换后的间隔向量满足定义3.1,所以它的空间局部性是优良的,而且不会增加内存开销.在实际的数据变换中,为了节省数据变换的时间,通过优化,我们取消了模运算,使数组变换的开销更小.

通过上述分析,我们可以发现,如果单独使用计算变换或数据变换,对并行程序的优化都有一定的局限性.通过统一的代数框架,我们可以将两种变换技术很好地结合起来,简化并行程序的变换步骤,降低编译器的开销,进一步提高并行程序在CC-NUMA 系统中的执行效率.

4 试验结果

由于条件限制,无法在 CC-NUMA 并行计算机系统下进行测试.我们的测试是在4个 CPU 的 SGI Power Challenge 上进行的,它的主存是128MB,每个 CPU 有1MB 的二级 cache.由于它是具有一致共享主存的紧耦合并行计算机系统,与 CC-NUMA 系统一样都具有统一的共享地址空间,并且所有的内存单元都能被 cache.因此,它在一定程度上可以体现 CC-NUMA 内存子系统的特性.我们使用了本机提供的并行编译器,进行显式并行编译.同时,为了更好地对各种变换进行比较,没有给编译器加优化开关.我们对4个程序进行了测试,它们分别是本文中的例1,Banded SYR2K,MatrixTranspose,Vpenta.BandedSYR2K 是求解带状对称矩阵的 Rank-2K 更新(update)的计算程序;MatrixTranspose 是求解二维数组转置的计算程序;Vpenta 是测试程序包 NASA-kernel中的一个测试程序.测试结果见表1.

Table 1 Speedup of transformation

表1 不同变换的加速比

Program①	Number of processors②	Base speedup③	Base+Comp transform speedup④	Base+Comp+Data transform speedup⑤
Example 1⑥	2	1.53	1.73	2.37
	4	2.56	2.75	4.71
Banded SYR2K	2	1.59	2.51	6.41
	4	2.80	4.81	13.83
Matrix Transpose	2	1.82	1.89	1.88
	4	3.10	3.10	3.03
Vpenta	2	1.55	1.79	1.82
	4	2.67	2.97	3.25

①程序,②处理机数,③一般并行加速比,④一般并行+计算变换加速比,
⑤一般并行+计算变换+数据变换加速比,⑥例1.

例1的数据存取模式是从二维数组的左上角到右下角沿对角线方向进行的,通过计算变换,可以消除程序的真共享冲突;通过数据变换可以消除假共享冲突,并提高数据重用性.因为假共享冲突在例1中占据通信开销的主要部分,因此得到了超线性加速比.BandedSYR2K 的读数据存取模式是从二维数组的右上角到左下角沿对角线方向进行的,它虽然没有 cache 伪共享冲突,但却造成大量的 TLB 不命中,大大影响了程序的性能.通过数据变换提高了数据访问的局部性,减少了在页这一层的开销,因此也得到超线性加速比.通过计算变换和数据转置变换不能解决 MatrixTranspose 中两个数组相反的数据线性地址,但通过数据扩维变换,可以减少在页这一级的伪共享冲突,同时改善了 cache 的利用率.但由于我们进行试验的计算机是 SMP 系统,故不存在页这一级的伪共享冲突,所以通过数据扩维变换只能改善 cache 的利用率.通过计算变换或数据转置变换可以使 Vpenta 程

© 中国科学院软件研究所 http://www.jos.org.cn

序中的二维数组和三维数组的访问模式沿数组的列方向进行,提高了数据的局部性.由于它具有和 Matrix-transpose一样的特性,因此,通过数据的扩维变换,可以改善 cache 的利用率.

5 结 论

由于 CC-NUMA 具有物理分布的全局共享主存,并且能 cache 所有的共享内存单元.所以,减少在内存页一级和在 cache 一级的真假共享冲突并提高它们的局部性,能够有效地提高并行程序的性能.本文论述了如何结合计算变换和数据变换来减少程序的通信开销的问题.实践表明,二者结合起来能得到更好的效果.下一步,我们还将对计算变换和数据变换的理论基础、计算变换和数据变换的其他类型作更深入的研究.同时,我们准备在即将完成的 DSM 模拟器上进行更精确的实验和评测.

参考文献

1 Wolf M E, Lam M. A data locality optimizing algorithm. In: Proceedings of the SIGPLAN'91 Conference on Programming Language Design and Implementation. Toronto, Canada, ACM SIGPLAN Notice, 1991,26(6):30～44

2 Li Wei. Compiler cache optimizations for banded matrix problems. Technical Report, Department of Computer Science, University of Rochester, 1995

3 Bugnion E, Anderson J M, Mowry T C *et al*. Compiler-Directed page coloring for multi-processors. In: Proceedings of the 7:h International Conference on Architecture Support for Programming Languages and Operating Systems. Boston, MA: ACM SIGPLAN Notice, 1996,31(9):244～255

4 Chen Yu, Yang Xue-jun, Chen Fu-jie. Communication optimization for NUMA parallel system. In: Li Xiao-ming ed. Workshop on the Frontier of Supercompiler and Parallel Computing Environment. Beijing: Beijing University Press, 1998. 18～29
(陈渝,杨学军,陈福接.基于 NUMA 结构的通讯优化技术.见:李晓明编.并行编译和并行计算环境前沿技术研讨会.北京:北京大学出版社,1998.18～29)

5 Jin Guo-hua. Program parallelizing techniques and tools [Ph.D. Thesis]. Changsha: National University of Defense Technology, 1993
(金国华.程序并行化技术及其工具的研究[博士学位论文].长沙:国防科学技术大学,1993)

6 Shen Z, Li Z, Yew P. An empirical study of Fortran programs for paralleling compilers. IEEE Transactions on Parallel and Distributed Systems, 1990,1(3):45～53

7 Cierniak M, Li Wei. Unifying data and control transformations for distributed shared-memory machines. In: Proceedings of SIGPLAN'95 Conference on Programming Language Design and Implementation. ACM SIGPLAN Notice, 1995,30(6):205～217

8 Chandra R, Chen D K, Cox R *et al*. Data distribution support on distributed shared memory multiprocessors. In: Proceedings of the SIGPLAN'97 Conference on Programming Language Design and Implementation. ACM SIGPLAN Notice,1997,32(5):334～345

Transformation Methods for Communication Optimization on CC-NUMA Parallel System

CHEN Yu LI Chun-jiang YANG Xue-jun CHEN Fu-jie

(*School of Computer National University of Defense Technology Changsha* 410073)

Abstract Reducing communication overhead is one of the goals of parallel compiling optimization. According to the characteristics of CC-NUMA parallel architecture, the authors propose some strategies and methods to optimize the communication of parallel programs by combining computing transformation and data transformation based on unified algebraic representation in this paper. Experimental results of optimization are presented, and the efficiency of the optimization policies and the methods is proved.

Key words Parallel compiling, communication optimization, computing transformation, data transformation, CC-NUMA (cache coherent-non uniform memory access).

© 中国科学院软件研究所 http://www.jos.org.cn