ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, Supplement, December 2010, pp.270−283

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一类基于迭代空间条块的并行有限差分 Stencil 算法*

张纪林[1+], 狄 鹏[2], 蒋从锋[1], 张 伟[1], 徐向华[1], 万 健[1], 任永坚[1]

[1](杭州电子科技大学 计算机学院,浙江 杭州 310018)

[2](北京科技大学 信息工程学院,北京 100083)

A Parallel Finite Difference Stencil Algorithm Based on Iterative Space Alternate Tiling

ZHANG Ji-Lin[1+], DI Peng[2], JIANG Cong-Feng[1], ZHANG Wei[1], XU Xiang-Hua[1], WAN Jian[1], REN Yong-Jian[1]

[1](School of Computer Science and Engineering, Hangzhou Dianzi Unviersity, Hangzhou 310018, China)

[2](School of Information and Engineering, University of Science and Technology Beijing, Beijing 100083, China)

+ Corresponding author: jilin.zhang@hdu.edu.cn

Zhang JL, Di P, Jiang CF, Zhang W, Xu XH, Wan J, Ren YJ . A parallel finite difference stencil algorithm based on iterative space alternate tiling. ***Journal of Software***, **2010,21(Suppl.):270−283.** http://www.jos.org.cn/1000-9825/10028.htm

Abstract: Difference stencils are fundamental computations throughout a broad range of scientific and engineering computer programs. In order to optimize data locality and communication overhead, this paper proposes a novel alternate tiling stencil algorithm on distributed memory machines by exploiting the property of the iterative algorithm. The serial execution process of this iterative method is given, which introduces the sequence of iterative space tile as the sequence of execution, and uses time skewing technique to divide iteration space. In this process, nodes of the tile can be traversed many times to improve data locality. The parallel algorithm based on iteration space tile technique is presented, which uses an improved polyhedral model to implement the iteration space tiling algorithm and reorders the tiles of iteration space to reduce cache misses, and the cost of communication and synchronization. The theoretical comparison is given between alternate tiling and other parallelization techniques. Finally numerical results are presented to confirm the effectiveness of serial and parallel execution models of alternate tiling finite difference stencil algorithm, specifically compared with domain-decomposition and red-black iterative methods, and show that the new parallel iterative method has a good data locality, parallel efficiency and scalability.

Key words: finite difference stencil; iterative method; alternate tiling; polyhedral model; data locality; communication optimization

摘 要: 高效的并行有限差分 Stencil 算法对于求解大型线性方程组是十分重要的.针对并行有限差分 Stencil 算法中数据局部性差、同步和通信开销大的问题.首先改进传统有限差分 Stencil 算法,提出了多层对称遍历有限差分

* Supported by the National Natural Science Foundation of China under Grant Nos.60873023, 60973029, 61003077 (国家自然科学基金); the National Basic Research Program of China under Grant No.2007CB310906 (国家重点基础研究发展计划(973)); the Zhejiang Provincial Natural Science Foundation of China under Grant Nos.Y1101104, Y1101092, Y1090940 (浙江省自然科学基金); the Zhejiang Provincial Education Department Research Foundation of China under Grant No.2010000711 (浙江省教育厅科研项目)

Received 2010-06-15; Accepted 2010-12-10

Stencil 算法.然后给出了以迭代空间条块序作为执行序的串行算法,通过沿时间轴对迭代空间进行时滞划分,在不改变迭代算法性质的同时,对迭代空间条块内部多次迭代计算,提高算法的数据局部性.最后提出一种基于迭代空间条块的并行算法,该算法利用改进的多面体模型对迭代空间网格划分,并通过网格条块重排序减少了 Cache 缺失率、通信启动和同步次数.理论分析和实验结果表明,该并行模型比传统的区域分解方法和红黑排序并行算法具有更好的数据局部性,并行效率和可扩展性.

关键词: 有限差分 Stencil;迭代算法;交错网格条块;多面体模型;数据局部性;通信优化

随着科学计算需要解决的问题渐趋复杂和并行计算机的迅速发展,并行计算已成为解决大规模并行科学计算问题必不可少的手段.在计算数学和计算物理等科学与工程计算领域,很多问题最终都归结为求解稀疏线性代数方程组.因此,在并行计算机上高效求解大规模稀疏线性代数方程组已成为当前科学计算领域的一项重要任务[1].

由于受舍入误差、计算机内存和计算复杂度的限制,对大规模问题,直接求解该类方程组几乎是不可能的,通常采用有限差分 Stencil 计算方法[2,21].迭代法的主要思想是通过构造有效的迭代格式,在有限步数内收敛于方程的精确解.

有限差分 Stencil(finite difference stencil)并行算法的实现一直是数值计算的重要研究对象.关于有差分 Stencil 算法的并行化设计与实现,前人已经做了很多工作,Zhang[3]通过使用基于区域分解的多色排序方法实现了面向集群的并行 GS 算法,但是当数据量增大时,数据局部性成下降趋势,并且在每次迭代计算过程,都需要通信和同步.Xie[1],Rohallah[4],Wallin[5]等人分别对 GS 并行算法进行了优化.循环分块(loop tiling)技术是重要的循环转换技术之一,主要用于提高循环并行度和数据局部性优化.研究人员对迭代空间分块已经做了很多研究工作.这些研究主要集中在两个方面:(a) 数据局部性优化研究[6,7,9,11,18,19,22,23];(b) 提高并行效率的分块尺寸/形状研究[5,8,10,12,16,24,25].通过数据依赖向量实现数据分块使其能够最大化复用,但是分块技术会引入数据块的空间依赖性,导致其基于空间条块的流水线执行方式不利于并行执行.

我们认为 Stencil 迭代算法并行化还有 4 个问题需要进一步的优化.(1) 数据局部性问题.大多数方法能够有效的提高迭代内数据局部性,但对迭代间的数据局部性优化效果有限.(2) 可扩展性问题.传统的并行化迭代算法在迭代内和迭代间都需要同步操作以维护数据依赖关系.处理机进行全局的同步,会增加开销时间,当处理机的台数增多时,全局同步的代价变得更加重要,并且影响算法的可扩展性.(3) 通信和同步开销问题.由于传统的并行化算法需要在每次迭代过程中通过通信操作得到边界数据,通信开销制约了并行算法的效率.并且当问题给定时, 随着处理机台数的增大,并行纯计算时间在减少,而通信时间在不断增加,这必将影响并行算法的可扩展性.(4) 依赖关系问题.迭代空间划分所引入的条块依赖关系严重阻碍了循环的并行化.因此需要研究减少通信时间的新方法.

鉴于此,本文首先提出多层对称有限差分 Stencil 迭代算法,并且通过将网格条块序引入串行算法的执行序,提高了串行算法的数据局部性.通过对迭代空间进行区域分解和条块重排,实现迭代的并行化.算法通过循环交错分块技术[15],沿时间轴对迭代空间进行划分,有效地降低了迭代算法的通信和同步开销.本文第 1 节给出多层对称迭代算法数学描述、串行执行过程和性能模型.第 2 节给出基于多面体模型的算法并行化过程.第 3 节是算法比较、实际测试以及性能分析.最后总结全文.

1 多层对称有限差分 Stencil 迭代及串行执行算法

1.1 多层对称差分Stencil迭代数学描述

许多实际问题进行数值模拟时,常利用偏微分方程作为是数学模型.考虑二维 Possion 方程经过 5 点差分之后形成的离散方程:

$$4u_{i,j}-u_{i-1,j}-u_{i+1,j}-u_{i,j-1}-u_{i,j+1}=h^2 f_{i,j},\ u_{i,j}\in\Omega=[0,1]\times[0,1] \tag{1}$$

其中,Dirichlet 边界条件为 $u(i,j)=0,[i,j]\in\partial\Omega$.

对求解区域[0,1]×[0,1]使用网格划分,在 x 轴和 y 轴分别用空间间隔 $h=1/(n+1)$的长度把求解区域网格化,网格点为(ih,jh),其中,$i,j=0,\ldots,n+1$.我们用 $u_{i,j}(i,j=0,\ldots,n)$表示 $u(ih, jh)$ 的有限差分近似值.

边界值离散问题通常会引入稀疏线性方程组 $Au=f$.其中 u 定义为差分方程所求解的未知数组,A 定义为稀疏矩阵,其矩阵结构与数值和差分方式有关.

以五点差分迭代为例说明多层对称有限差分 Stencil 迭代算法.偏微分方程经过差分产生一组线性迭代方程,初始值 $u_{i,j}^{(0)}$ 通过行优先排序.方程如式(2)所示,其中 k 代表迭代次数.

$$u_{i,j}^{(k+1)}=1/4\left(h^2 f_{i,j}+u_{i-1,j}^{(k+1)}+u_{i,j-1}^{(k+1)}+u_{i+1,j}^{(k)}+u_{i,j+1}^{(k)}\right),\ i,j=1,2,3,\ldots,n \tag{2}$$

通过改变更新未知数 $u_{i,j}$ 的顺序,线性迭代方程也可以表示为

$$u_{i,j}^{(k+1)}=1/4\left(h^2 f_{i,j}+u_{i+1,j}^{(k+1)}+u_{i,j+1}^{(k+1)}+u_{i-1,j}^{(k)}+u_{i,j-1}^{(k)}\right),\ i,j=n,n-1,n-2,\ldots,1 \tag{3}$$

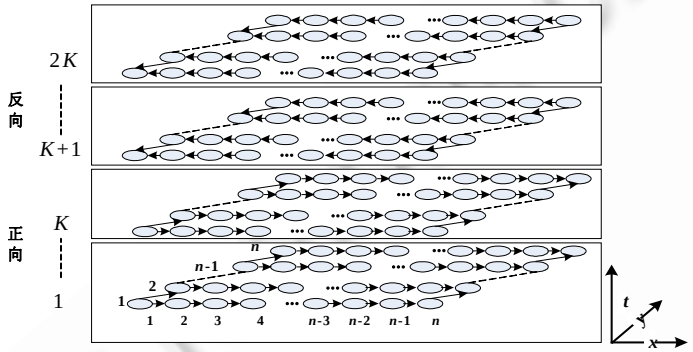


Fig.1 Multi-Layer symmetric five-point Stencil algorithm

图 1 多层对称 Stencil 算法

多层对称差分 Stencil 迭代是通过交替使用不同方向迭代求解方法,对线性迭代方程求数值解.对网格点 $u_{i,j}=\left\{u_{i,j}^{k}\mid k\in[(p-1)K+1,p\times K]\right\}$, $p=1,3,5,...,2P+1$(P 为自然数),迭代采用“向前”(左下至右上)的迭代方法,由式(2)计算,其迭代过程称为奇数 K 次迭代;对网格点 $u_{i,j}=\left\{u_{i,j}^{k}\mid k\in[(p-1)K+1,p\times K]\right\}$, $p=2,4,6,...,2P$ (P 为自然数),迭代采用“向后”(右上至左下)的迭代方法,由式(3)计算,其迭代过程称为偶数 K 次迭代.

利用 Saul’yev 的 SSOR 理论证明[26]可简单推导出多层对称差分 Stencil 迭代收敛性与 SSOR 算法相同.

1.2 多层对称差分Stencil迭代局部性优化

传统的 Stencil 迭代方法执行序是,在一次迭代内部依据网格点的顺序,依次对所有网格点进行迭代更新操作,文献[32]对 Stencil 迭代的 Cache 缺失因素作了详细的分析.

如图 2 所示,其原因在于当数组大于 Cache 容量时,本次更新的数据在下一次更新之前已经被写回内存.而且当数据量大时,传统迭代算法中多次迭代会导致数据 Cache 的容量缺失.此外,多次迭代使数据的地址转换信息会周期性地在 TLB 中进行存取,也是影响其性能的一个重要因素.当矩阵规模增加时,大量的 TLB 容量缺失会严重影响程序的性能.

```
for (t=1; t<=T; t++)
    for (i=1; i<N–1; i++)
        for (j=1; j<N–1; j++)
            A[t+1][i][j]=(A[t+1][i–1][j]+A[t][i+1][j]+A[t][i][j]
                          +A[t+1][i][j–1]+A[t][i][j+1])/5
```

Fig.2 Traditional 5-point difference iterative algorithm

图 2 传统 5 点差分迭代算法

因此,本文提出迭代空间条块串行迭代执行序,其核心思想是:改变以往传统的以迭代次序为执行序的特

点,将迭代空间分块引入执行序中,如图 3 所示.

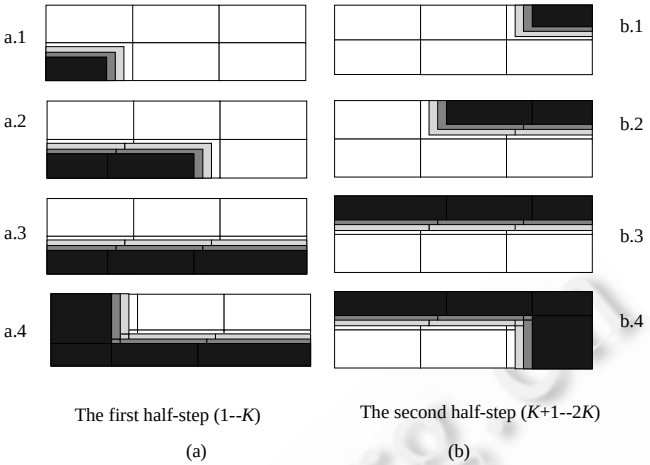


Fig.3 Time skewing schedule for Multi-layer symmetric five-point Stencil algorithm

图 3 迭代空间交错条块串行 Stencil 算法

1.2.1 条块式多层对称差分 Stencil 迭代算法

在传统的 Stencil 迭代算法中,处理单元必须通过遍历并更新全部网格点完成一次迭代过程,当数据量增大时,数据局部性较差.为此,我们采用迭代空间交错条块式串行 Stencil 迭代,通过对迭代空间进行时间轴方向划分成网格条块,实现对同一网格块进行递归式多次迭代步更新,从而在不改变串行 Stencil 迭代算法性质的同时,提高条块内数据局部性.为描述执行过程,首先给出数据空间和迭代空间的定义.

数据空间:在迭代计算中,m 维网格点 $x(i_1,\ldots,i_m)$组成 m 维数据空间 $data\_space(m)$.

迭代空间:$n-1$ 维数据空间 $data_space(n-1)$和迭代维 T 的组合可以被看作 n 维的迭代空间 $iter\_space(I_1,\ldots,I_{n-1},T)$,其中的每个点都可由一个 $n\times1$ 维的列向量来表示,即 $\overline{I}=(i_1,...,i_{n-1},t)^T$,其中 $i_1,...,i_{n-1}$ 从左至右分别代表网格点在数据空间中的 $I_1,\ldots,I_{n-1}$ 维的维度坐标.数据空间中网格点 $x(i_1,\ldots,i_m)$在 k 次迭代的值,在迭代空间中可表示为 $u^k(i_1,\ldots,i_{n-1})$.例如,三维迭代空间 $iter_space(I,J,T)$由二维网格点 $x(i,j)$和时间维 T 组成.$u^k(i,j)$表示网格点 $x(i,j)$在 k 次迭代的值.

以二维 Possion 方程为例,迭代空间交错条块串行迭代算法执行过程如下:

步骤 1.网格块划分.

用区域分解方法将迭代空间 $iter_space(I,J,T)$在 T=0 处进行数据划分.如图 3 所示,定义 l_1 为子空间 $sub_iter_space(p,q,0)$的行数,定义 l_2 为子空间 $sub_iter_space(p,q,0)$的列数,划分后使得每个子空间 $sub_iter_space(p,q,0)$中的网格点数为 $R=l_1\times l_2$,l_1 与 l_2 满足式(4):

$$l_1>K\cap l_2>K \tag{4}$$

其中,K 为单向迭代次数.

步骤 2.在网格块的基础上,沿时间轴对迭代空间进行划分.

划分方法采用时滞技术[10,11],对每层迭代的子空间修正边界,图 3 中灰线边界表示修正后的边界.

定义有向图 $G(V,E)$存放相邻条块的关系.若条块 v_i 与条块 v_j 边界相连且 $v_i<v_j$,则$<v_i,v_j>\in E$.定义 $P_{node}(v_i,v_j,k)$为在第 k 次迭代中属于条块 v_i 但与条块 v_j 相邻的边界数据.修正算法描述如图 4 所示.

步骤 3.按空间网格条块顺序执行 Stencil 迭代算法.

网格条块生成后,按条块顺序执行迭代算法.以条块内部网格点层作为内部执行序,条块内部迭代次数作为中间执行序,条块序作为外部执行序,更新每个网格点的值.执行过程如图 3 所示,a.1~a.4 为条块正向执行顺序,b.1~b.4 为条块反向执行顺序,其中数据块中 3 种由浅入深的灰色分别代表连续的 3 次迭代计算.由于该串行算法只是改变了网格的排列顺序,并没有改变 Stencil 迭代算法的性质,因此该算法并没有改变传统迭代算法的算

法复杂度.

```
/*条块边界修正算法*/
for (n=0; n<N; n+=2)
        for (k=n×K+1; k<=(n+1)×K; k++){//奇数 K 次迭代修正边界 Foreach⟨vi,vj⟩E
        sub_iter_space(vi,k+1)=sub_iter_space(vi,k)−Pnode(vi,vj,k)
        sub_iter_space(vj,k+1)=sub_iter_space(vj,k)+Pnode(vi,vj,k)
          }
for (k=(n+1)×K+1; k<=(n+1)×K; k++){//偶数 K 次迭代修正边界 Foreach⟨vi,vj⟩E
        sub_iter_space(vj,k+1)=sub_iter_space(vi,k)−Pnode(vj,vi,k)
        sub_iter_space(vi,k+1)=sub_iter_space(vi,k)+Pnode(vj,vi,k)
}
```

Fig.4 Boundary revise algorithm of alternate tiling iteration execution model

图 4 条块迭代算法执行模型中的边界修正算法

2 并行交错条块有限差分 Stencil 算法

我们通过网格条块重排序,实现交错条块有限差分 Stencil 算法的并行化.该算法有效的降低传统差分 Stencil 算法的通信和同步开销.由于数据空间中网格点的排序是任意的,因此在保证迭代过程偏序关系的情况下,改变迭代空间中网格点的排列顺序时,算法的敛散理论仍可以适用[7],据此迭代空间交错条块并行有限差分 Stencil 迭代方法可有效的实现迭代的并行化.

2.1 多面体模型描述

传统的多面体模型[27−30]通过迭代空间多面体矩阵 B,依赖矩阵 D,超平面矩阵 H,条块依赖关系矩阵 S 描述迭代空间的数据划分及依赖关系.但传统多面体模型中引入超平面执行顺序会降低迭代空间并行化的效率,因此需要改变超平面方向消除引入的数据依赖关系,提高迭代计算的并行性.在标准条块迭代算法中超平面由一组法向量组成.法向量定义了条块在同一超平面方向的两面,其中 $h_i(l)$定义为 I 方向第 l 条块区域 $tile(l,I)$在 I 方向超平面的后平面,$h_i(l+1)$定义为第 l 条块区域的 I 方向超平面的前平面.在条块中前平面不会引入 I 方向其他条块的依赖,而后平面中的数据在计算时需要 I 方向相邻条块的数据,因此引入了 I 方向数据依赖.例如:$\forall title(p,I)\in subdomain(n_1)$,$\forall title(q,I)\in subdomain(n_2)$,且 $title(p,I)$与 $title(q,I)$相邻.在计算 $title(p,I)$时,会使用 $title(q,I)$的边界数据.在边界条块中通过建立 I 方向“负超平面”可以减少条块之间在 I 方向的依赖关系.将后平面 $h_i(l)$替换为负超平面 h'_i 满足:

$$\forall d_k\in D,\ h'_i\cdot d_k\leqslant 0 \tag{5}$$

举例说明如图 5 所示,该条块区域为区域分解后在各处理器上的第一个条块(如图 6(b)中标注 4.1 的数据块).当引入 J 向负超平面和 I 向负超平面,由超平面组成的迭代条块可以独立的执行迭代计算,而不需要引入其他依赖关系.从而在各处理器执行过程中,此类条块可以被同时执行更新迭代.

在维度 j 中可以独立执行的条块区域用 $\neg j$ 表示,其余非独立执行部分用 j 表示.在每一维度 j 中,条块区域均可被分为两类:独立执行的条块区域和非独立执行的条块区域.因此在 n 维空间,迭代空间中共有 2^n 类条块区域.由于我们仅考虑在迭代空间的并行执行过程,因此不考虑时间维.例如在图 3 中 2 维迭代空间(I,J)中条块区域可分为以下 4 类:$\neg j\cap\neg i$, $\neg j\cap i$, $j\cap\neg i$, $j\cap i$.基于条块顺序的迭代执行的基本准则是:在每一维中先执行独立条块,然后将相应维度的边界依赖数据转递给非独立条块;当非独立条块得到所需要的依赖数据后,再执行非独立条块数据的迭代更新.例如在图 6(b)中,在对称遍历的正向(白色)迭代过程中,属于 $\neg j\cap\neg i$ 类别的条块为 1,2,3,4;属于 $\neg j\cap i$ 类别的条块是 5,6,7,8;属于 $j\cap\neg i$ 类别的条块是 9,10,11,12;属于 $j\cap i$ 类别的条块是 13,14,15,16.而在对称遍历的反向(灰色)迭代过程中,属于 $\neg j\cap\neg i$ 类别的条块为 16,15,14,13;属于 $\neg j\cap i$ 类别

的条块是 12,11,10,9;属于 $j \cap \neg i$ 类别的条块是 8,7,6,5;属于 $j \cap i$ 类别的条块是 4,3,2,1.其执行顺序和通信顺序如图 7(b)所示.

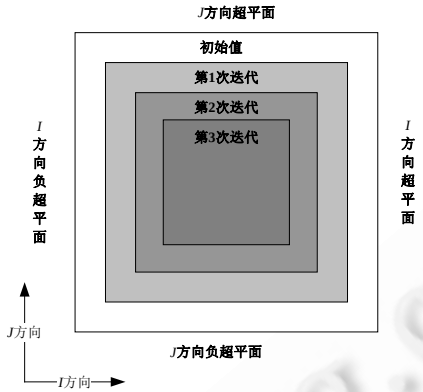


Fig.5 Tiling hyperplane partition: top view

图 5 条块超平面划分图:俯视

2.2 并行执行算法

为实现网格条块的并行化,需要建立网格条块之间的依赖关系.迭代空间交错条块并行 Stencil 迭代方法的并行度可通过依赖关系表示.图 6 给出了三维(空间维+时间维)迭代空间的两种交错条块划分,对应的网格条块依赖关系如图 7 所示.图 7(a)显示网格条块按条块顺序串行执行迭代更新,从正向第 1 块数据块 *b_tile*(1)(白色数据块 1)到反向第 1 块数据块 *t_tile*(1)(灰色数据块 1)长度为 32,因此没有并行度.图 7(b)显示通过增加边界条块的负超平面进而改变网格条块的执行顺序,可以实现 4 个进程并行执行迭代更新,因此并行度为 4.为了描述方便,本文在边界条块增加负超平面之后将图 7(b)的条块顺序标号改为如图 7(b)的顺序标号,进程 1 依次执行 *b_tile*(1.1),*b_tile*(1.2),*b_tile*(1.3),*b_tile*(1.4),*t_tile*(1.4),*t_tile*(1.3)*t_tile*(1.2)*t_tile*(1.1)条块,进程 2 依次执行 *b_tile*(2.1),*b_tile*(2.2),*b_tile*(2.3),*b_tile*(2.4),*t_tile*(2.4),*t_tile*(2.3),*t_tile*(2.2),*t_tile*(2.1)条块.进程 3 依次执行 *b_tile*(3.1),*b_tile*(3.2),*b_tile*(3.3),*b_tile*(3.4),*t_tile*(3.4),*t_tile*(3.3),*t_tile*(3.2),*t_tile*(3.1)条块,进程 4 依次执行 *b_tile*(4.1),*b_tile*(4.2),*b_tile*(4.3),*b_tile*(4.4),*t_tile*(4.4),*t_tile*(4.3),*t_tile*(4.2),*t_tile*(4.1)条块.其中,4 个进程在正向执行条块迭代更新过程中,分别在执行所属第 1 个条块及第 3 个条块后执行同步操作以维护条块间的数据依赖关系.同样 4 个进行在反向执行条块更新过程中,也分别在执行所属第 1 个条块及第 3 个条块后执行同步操作以维护条块间的数据依赖关系.显然,第 2 种划分方式的执行速度是第 1 种划分方式的 4 倍.这两种方式最大的不同在于初始化时的排序方式不同.因为排序方式的不同代表了相邻条块间的执行顺序的差异,因此条块排序影响了条块间的数据依赖关系.通过改变网格条块间数据依赖关系,提高条块执行的并行度.如图 7(b)所示.

区域分解方法是实现分布式内存并行化的主要方法,但是传统的区域分解方法仅在空间维度上实现求解空间的分解,并没有考虑时间维度.为了实现并行类似于 STENCIL 迭代本身有串行性质的迭代方法,交错条块并行算法改进了区域分解方法,使其沿时间维度对求解空间进行划分,形成不同的网格条块.为了方便说明,将迭代空间 *iter_spact*(I_1,I_2,T)划分为 4 个子空间 sub_domain1,sub_domain2,sub_domain3,sub_domain4.如图 8(a)所示.

空间条块的边界网格点必须发送给其他相邻子空间以维护数据依赖关系.子空间中的条块按通信类别分为 4 类:“发送条块”、“接收条块”、“混合条块”、“非通信条块”.“发送条块”在执行更新后将自身边界网格数据发送给“接收条块”.“接收条块”必须在接受其他子空间条块发送的边界网格数据后才执行更新.“混合条块”在执行前后需要接收数据和发送数据.其余的条块为“非通信条块”,其自身计算不需要其他处理器中网格条块的边界值.在交错条块算法中,按条块的执行序可以减少处理器之间的通信开销.当迭代执行 2K 次时,通信执行两次,通信数据量为 2V,通信时间满足式(6):

$$T_{comm} = 2 \times T_s + 2 \times \frac{V(\text{number of points to transmit on a half of tile})}{r(\text{transfer rate})} \tag{6}$$

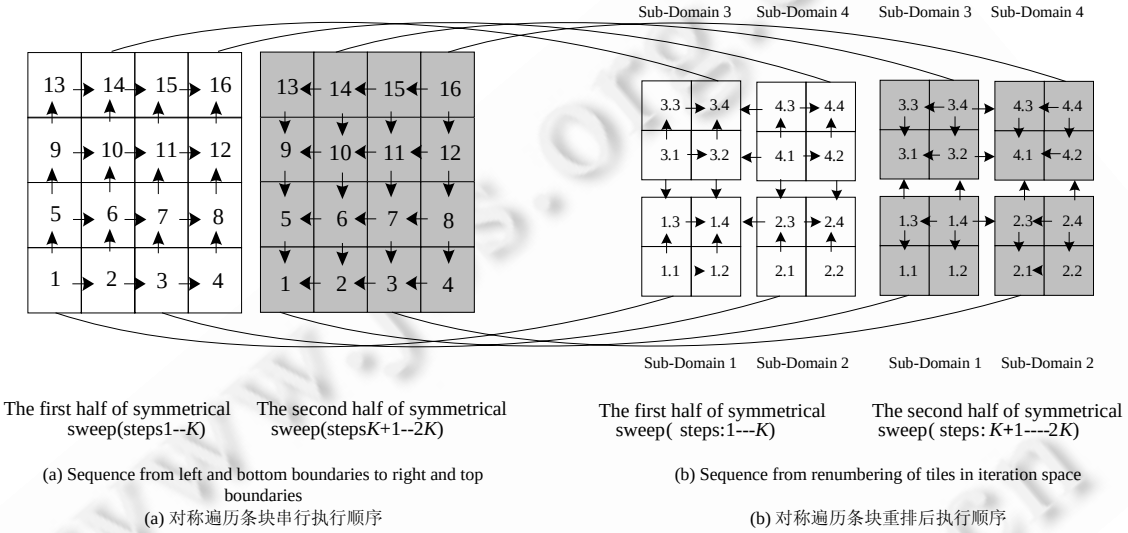
$$T'_{comm} = 2 \times k \times T_s + 2 \times \frac{V(\text{number of points to transmit on a half of tile})}{r(\text{transfer rate})} \tag{7}$$

交错条块并行 Stencil 算法中需要正反方向各一次通信,每一次通信需要 K 个边界值,T_s 是通信的启动时间.如式(6)、式(7)所示,T_{comm} 是交错条块算法的通信时间,T'_{comm} 是传统的区域分解并行 Stencil 算法中的通信时间.显然,在交错条块算法的通信开销中,启动时间比区域分解方法减少了 $2\times(k-1)T_s$.

并行执行算法描述如下,如图 8(b)所示:

步骤 1.根据处理器拓扑进行数据划分.

依据处理器数目 P,及处理器的拓扑结构 $P=P1\times P2$.将空间计算区域划分为 $P1\times P2$ 个子空间 *sub_domain*.如图 6 所示,迭代区域被划分成 P 个子空间,且 $P1=P2=Q$.

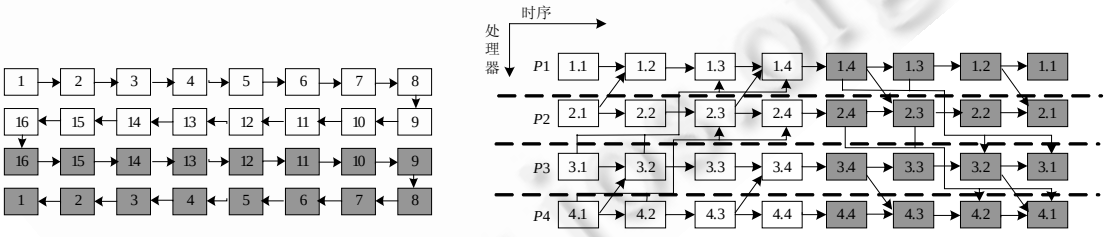


(a) Sequence from left and bottom boundaries to right and top boundaries
(a) 对称遍历条块串行执行顺序

(b) Sequence from renumbering of tiles in iteration space
(b) 对称遍历条块重排后执行顺序

Fig.6 Two applications of alternate tiling to a three-dimensional iteration space

图 6 三维迭代空间的两类条块划分方法



(a) Tile dependence graph from Figure 6(a)
(a) 图6(a)划分方法的条块依赖图

(b) Tile dependence graph from Figure 6(b)
(b) 图6(b)划分方法的条块依赖图

Fig.7 Tile dependence graphs for alternate tilings shown in Figure 6

图 7 图 6 所示两类条块划分方法的条块依赖图

步骤 2.指定迭代方向.

交错条块迭代算法分为奇数和偶数 k 次迭代,并且奇数 k 次迭代与偶数 k 次迭代执行方向相反.例如,使用 LB-RT 顺序(左下到右上)执行奇数 K 次迭代,使用 RT-LB 顺序(右上到左下)执行偶数 K 次迭代.

步骤 3.对子空间条块划分.

各子区域通过时间轴进行划分算法,同串行迭代算法,并且奇数 K 次和偶数 K 次划分方向相反.

步骤 4.对网格条块重新排序.

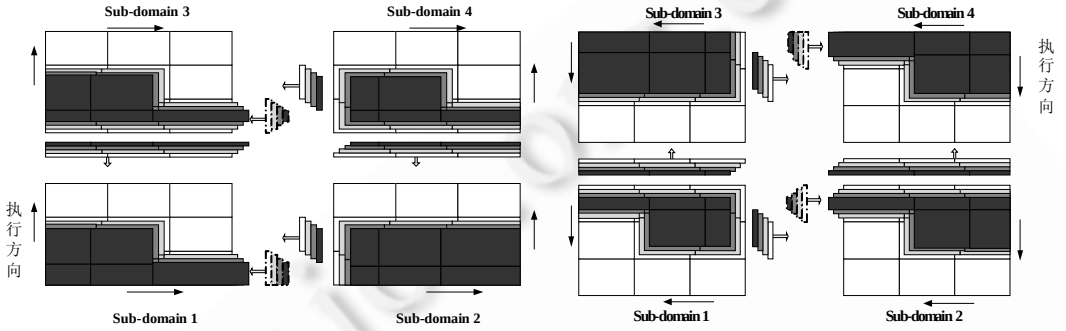
根据处理器个数 P 按照 *cyclic*(P)排列对所有的数据块进行重排序.

步骤 5.以网格条块为单位执行奇数 K 次迭代数据更新.

执行奇数 K 次迭代更新,更新顺序参照重排序之后每个子空间内的条块序.当更新奇数 K 次中的“发送条块”和“混合条块”后,需要将边界数据发送给相应的“接收条块”和“混合条块”,而接收数据的条块必须在接收完数据后再进行迭代更新.

步骤 6.以网格条块为单位执行偶数 K 次迭代数据更新.

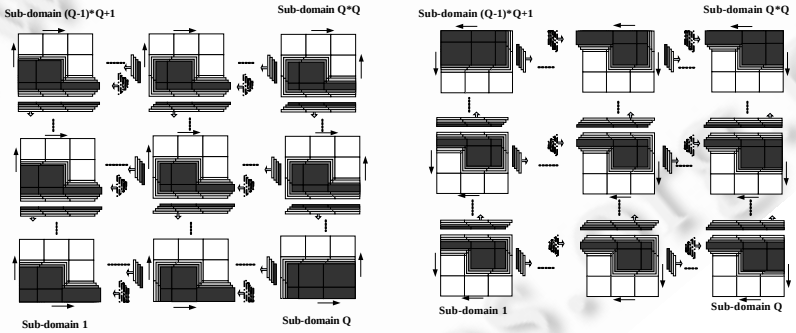
执行偶数 K 次迭代更新,更新顺序按照每个子空间内的条块序反序执行.当更新偶数 K 次中的“发送条块”和“混合条块”后,需要将边界数据发送给相应的“接收条块”和“混合条块”,而接收数据的条块必须在接收完数据后再进行迭代更新.



(a.1) The first half of *symmetrical sweep*
(a.1) 对称遍历过程的正向部分

(a.2) The second half of *symmetrical sweep*
(a.1) 对称遍历过程的反向部分

(a) Four sub-domains parallelization
(a) 4个子区域的并行化



(b.1) The first half of *symmetrical sweep*
(b.1) 对称遍历过程的正向部分

(b.2) The second half of *symmetrical sweep*
(b.2) 对称遍历过程的反向部分

(b) Multi sub-domains parallelization
(b) 多个子区域的并行化

Fig.8 Parallelization of alternate tiling algorithm by domain decomposition

图 8 基于区域分解的交错条块有限差分 Stencil 并行化

3 算法比较与实际测试

3.1 算法比较

交错条块并行化差分 Stencil 迭代方法不同于其他差分 Stencil 迭代并行化方法,如多色排序法[13,17]、区域

分解法.后两种方法的并行度分别与颜色数和划分的块数有关,并且在每次迭代过程中,都需要进行相邻网格块的通信,总的通信时间如式(6)所示.此外,对于多色排序法和区域分解法来说,在迭代内部和迭代之间都存在同步操作,而同步操作能够严重影响并行效率.但是,交错条块法以条块作为执行序,改变了上述方法单纯的以网格点作为执行序的特点,其优势不仅在于提高了数据局部性,而且仅在条块之间以及每个 K 次迭代之间需要通信和同步操作.因此,在高性能网络环境下,理论上交错条块法可以减少数据通信的启动次数,通过最小化 Cache 缺失率、通信以及同步开销有效提高差分 Stencil 迭代算法的并行化效率.此外,交错条块方法不仅适用于共享式内存体系结构,而且适用于分布式内存集群.

在测试过程中,我们发现并行交错条块法由于边界点破坏了原有的收敛率,达到收敛时的执行的迭代次数比串行交错条块法的执行迭代次数高出约 5%左右,但是由于并行算法本身提高了并行度,因此整体执行时间明显小于串行算法执行时间,以收敛率为代价换取整体并行时间的大幅减少.

3.2 性能测试

为了测试与比较多种平台下的差分 Stencil 迭代的交错条块算法、多色排序法[13]、区域分解法在串行执行模式的数据局部性.实验平台采用 3 种主流处理器:Intel Xeon,AMD Optern,MIPS R16K.程序采用 C 语言和 PAPI 库[21]测试数据局部性.为了测试上述 3 种算法的并行性能,实验平台采用 16 节点的 Xeon Cluster.其中,每个节点配置 Intel Xeon 3.0GHz/1024Kbyte L2 缓存,2Gbyte 内存,节点通过千兆以太网互联.安装的操作系统 Fedora Core 3,MPI[14]运行环境为 Argonne 实验室开发的 mpich-1.2.7,使用 ch_p4 设备.

3.2.1 执行时间测试

实验针对式(1)进行求解,网格规模分别为 1024×1024,2048×2048,4096×4096.这 3 种算法的执行时间见表 1,红黑 Stencil 迭代执行时间普遍比传统 Stencil 迭代时间长.特别是在网格规模为 1024×1024 和 4096×4096 时,红黑 Stencil 算法比传统 Stencil 算法分别慢 60%和 24%.由于红黑 Stencil 执行模型中数据局部性远比传统 Stencil 差,因此造成了大量的 Cache 缺失,执行时间也相应减慢.此外,交错条块 Stencil 算法的执行时间普遍比 Stencil 时间短,其原因也是由于前者的数据局部性比后者高.图 9 给出了在不同规模的分块情况下,交错条块算法在 3 种平台的执行时间曲线.从图 9 中可以看出,在不同的分块情况下,交错条块算法的执行时间要优于其余两种算法.在 Xeon 平台下,当数据块为 128×256(32K 字节)时执行时间最短,其原因在于 Cache 缺失和 TLB 缺失都很低(如图 10 和图 11 所示).

Table 1 Execution time of GS, RBGS and ATGS for all problem sizes

表 1 不同规模的 GS,RBGS,ATGS 算法执行时间

Total nodes	GS (s)			RBGS (s)			ATGS (s)		
	Xeon	Opteron	MIPS	Xeon	Opteron	MIPS	Xeon	Opteron	MIPS
1024×1024	0.24	0.27	1.56	0.39	0.34	2.61	0.13	0.23	1.06
2048×2048	0.98	1.10	6.3	1.30	1.36	11.23	0.53	0.93	4.95
4096×4096	3.95	5.72	27.45	4.88	8.15	47.22	2.16	4.35	20.22

3.2.2 存储性能测试

本文对 3 种规模网格下的交错条块算法进行 Cache 缺失和 TLB 缺失统计,统计曲线如图 10 和图 11 所示.从图 10 中可以清楚的看出数据块的大小存在最优选择.在 3 种体系结构下,当数据块分别小于 128×256,128×128,256×256 时,Cache 缺失变化不大;相反,则 Cache 缺失会随数据块的增加而迅速增加.其原因在于 L2Cache 的容量缺失会随着数据块的增加而迅速增加.从图 11 中可以看出在 3 种体系结构下,TLB 缺失与数据尺寸均保持先增后减的趋势.其原因在于本文的程序使用行优先数组,数据块尺寸小时,造成大量的 TLB 预取缺失,数据块尺寸大时,TLB 预取缺失会大量减少.

3.2.3 并行效率和可扩展性测试

为了验证交错条块差分 Stencil 迭代方法的加速比和可扩展性,针对式(1)求解,对比区域分解法、红黑排序法以及交错条块法的并行效果,网格规模为 8192×8192.我们给出了在处理器个数不同的情况下,3 种方法各自的加速比和效率.从图 12 中可以看出,随着处理器个数的增长,交错条块划分方法的并行效率始终大于红黑排

序法和区域分解法.此外,区域分解法和红黑排序法的并行效率下降,说明随着处理器个数的增长,通信和同步开销严重影响了程序的执行时间.由于交错条块方法通过沿时间轴对迭代空间进行划分,有效减少了通信和同步开销,因此加速比和可扩展性高于区域分解方法和红黑排序法(如图 12 所示),具有明显优势.

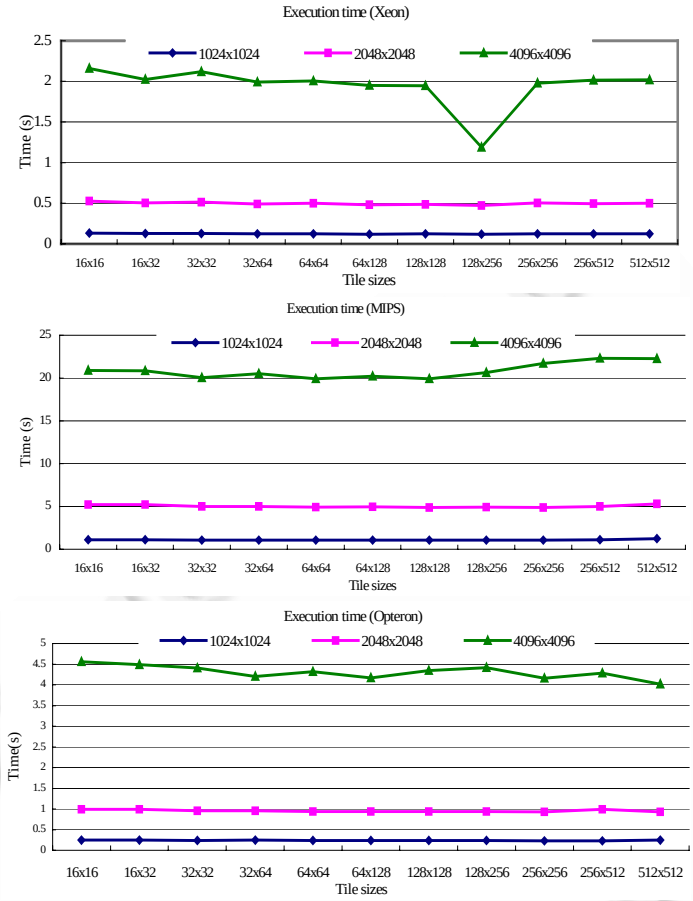


Fig.9 Execution time of ATGS at different tile sizes for all problem sizes on three microprocessors: Xeon; MIPS; Opteron

图 9 3 类处理器(Xeon,MIPS,Opteron)中,交错条块有限差分 Stencil 计算在不同计算规模下条块尺寸与执行时间关系

Table 2 L2 cache misses of GS, RBGS and ATGS for all problem sizes on all platforms

表 2 不同平台不同计算规模下的算法 2 级 Cache 缺失

Total nodes	GS			RBGS			ATGS		
	Xeon	Opteron	MIPS	Xeon	Opteron	MIPS	Xeon	Opteron	MIPS
1024×1024	35 174	68 113	1 049 822	71 888	136 052	2 100 111	18 506	16 880	105 770
2048×2048	138 338	271 039	4 197 825	278 532	541 882	8 396 005	84 610	60 429	499 938
4096×4096	543 755	1 049 849	16 786 190	1 104 673	2 099 974	33 570 952	527 682	199 830	2 351 519

Table 3 TLB misses of GS, RBGS and ATGS for all problem sizes on all platforms

表 3 不同平台不同计算规模下的算法 TLB 缺失

Total nodes	GS			RBGS			ATGS		
	Xeon	Opteron	MIPS	Xeon	Opteron	MIPS	Xeon	Opteron	MIPS
1024×1024	54 955	32 468	4 320	85 363	64 435	8 573	8 821	5 651	333
2048×2048	240 232	131 839	17 715	374 673	262 536	34 656	34 710	100 118	1 584
4096×4096	857 911	528 915	73 607	1 370 420	1 058 415	144 334	134 249	2 038 769	12 059

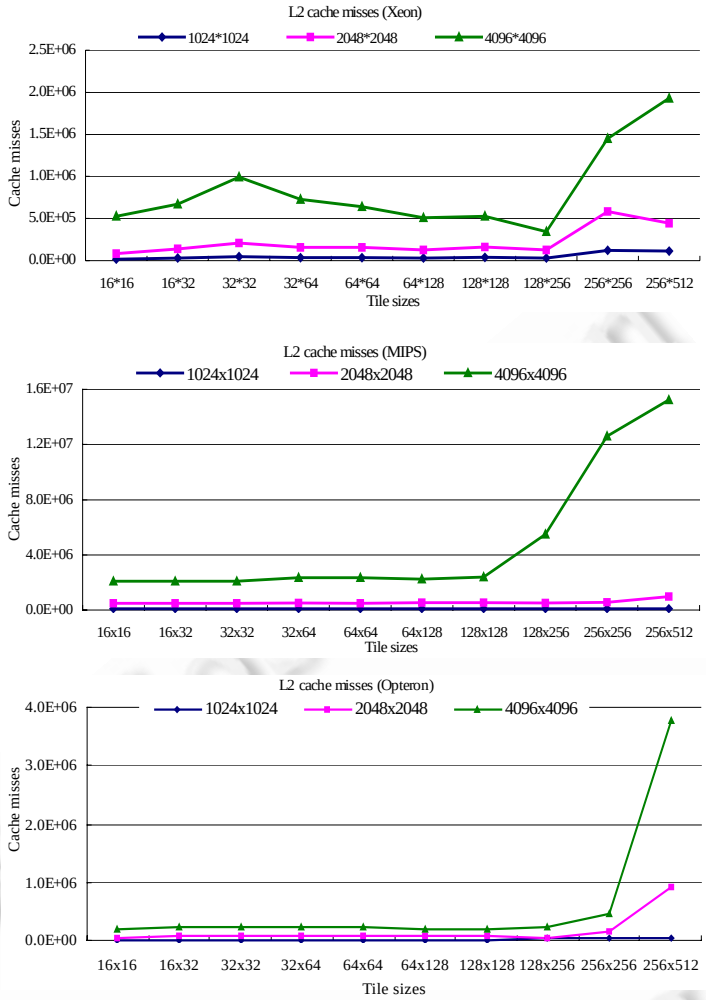


Fig.10 L2 cache misses of ATGS at different tile sizes for all problem sizes on three microprocessors: Xeon; MIPS; Opteron

图 10 3 类处理器(Xeon,MIPS,Opteron)中, 交错条块有限差分 Stencil 计算在不同计算规模下条块尺寸与 2 级 cache 缺失关系

4 结 论

本文通过分析迭代方法的并行化效率不高的原因,提出了一种基于空间条块的并行有限差分 Stencil 迭代算法.在不增加数据通信量的情况下,通过负超平面对迭代空间进行划分,减少了通信启动开销和同步时间,提高了数据局部性.并且通过对空间迭代块的重新排序,改变了块间数据依赖性,可有效地实现迭代算法的并行化.从理论上对比并分析了交错条块法与区域分解法和红黑排序法的并行计算性能.分析表明交错条块法还可有效地减少通信开销和同步时间.理论分析和测试结果都表明,交错条块迭代算法是一种具有较高加速比和较好可扩展性的并行迭代算法.下一步工作需要在充分研究集群存储结构的基础上,设计和实现用于有限元分析的非规则迭代算法并行化.并且实现多级空间条块划分模型及分析框架以实现迭代算法的自动调优[31].

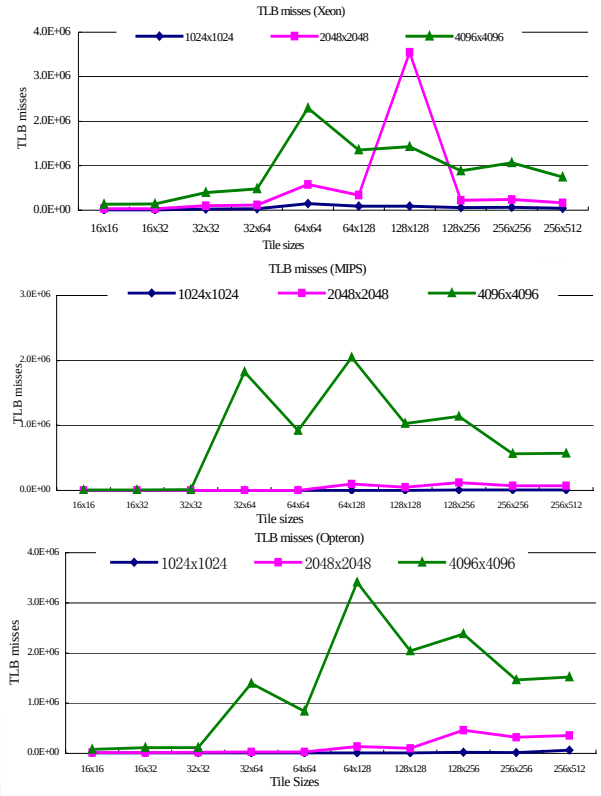


Fig.11 TLB misses of ATGS at different tile sizes for all problem sizes on three microprocessors: Xeon; MIPS; Opteron

图 11 在 3 类处理器(Xeon,MIPS,Opteron)中,交错条块有限差分 Stencil 计算在不同计算规模下条块尺寸与 TLB 缺失关系

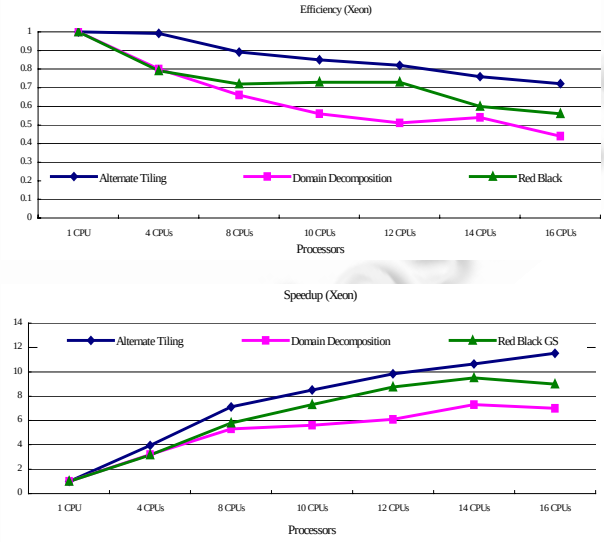


Fig.12 A comparison of the efficiency and speedup of different parallel GS algorithms on different number of processors, total nodes are 8192×8192

图 12 计算规模为 8192×8192 时并行算法的加速比和并行效率比较

References:

[1] Xie D. A new block parallel SOR method and its analysis. SIAM Journal on Scientific Computing, 2006,27:1513−1533.

[2] Saad Y. Iterative Methods for Sparse Linear Systems. 2nd ed., SIAM, 2003.

[3] Zhang C, Lan H, Ye Y, Estrade BD. Parallel SOR iterative algorithms and performance evaluation on a Linux cluster. In: Proc. of the Int'l Conf. on Parallel and Distributed Processing Techniques and Applications. Las Vegas: CSREA Press, 2005. 1042−1048.

[4] Tavakoli PDR. New stable group explicit finite difference method for solution of diffusion equation. Appl. Math. Comput., 2006,181:1379−1836.

[5] Wallin D, Lof H, Hagersten E, Holmgren S. Multigrid and Gauss-Seidel smoothers revisited: Parallelization on chip multiprocessors. In: Proc. of the 20th Annual Int'l Conf. on Supercomputing. 2006. 145−155.

[6] Bondhugula U, Hartono A, Ramanujan J, Sadayappan P. A practical automatic polyhedral parallelizer and locality optimizer. In: ACM SIGPLAN Programming Languages Design and Implementation (PLDI). 2008.

[7] Strout MM, Carter L, Ferrante J, Kreaseck B. Sparse tiling for stationary iterative methods. Int'l Journal of High Performance Computing Applications, 2004,18(1):95−114.

[8] Xue J. Loop Tiling for Parallelism. Boston: Kluwer Academic Publishers, 2000.

[9] Huang QG, Xue JL. Xavier Vera: Code tiling for improving the cache performance of PDE solvers. In: Proc. of the ICPP 2003. IEEE, 2003. 615−626.

[10] Hogstedt K, Carter L, Ferrante J. Selecting tile shape for minimal execution time. In: Proc. of the SPAA'99. 1999. 201−211.

[11] Douglas CC, Hu J, Kowarschik M, Rude U, Weiss C. Cache optimization for structured and unstructured grid multigrid. Electronic Trans. on Numerical Analysis, 2000,10:21−40.

[12] McCalpin J, Wonnacott D. Time skewing: A value-based approach to optimizing for memory locality. Technical Report, DCS-TR-379, Department of Computer Science, Rugers University, 1999.

[13] Melhem RG, Ramarao KVS. Multicolor ordering of sparse matrices resulting from irregular grid. ACM Trans. on Math. Software, 1988,11:117−138.

[14] MPI-2: Extensions to the message-passing interface. Technical Report, http://www.mpi-forum.org

[15] Hu CC, Zhang JL, Wang J, Li JJ. A new parallel Gauss-Seidel method by iteration space alternate tiling. In: Proc. of the 16th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2007). 2007. 410.

[16] Goumas GI, Drosinos N, Karakasis V, Koziris N. Coarse-Grain parallel execution for 2-dimensional PDE problems. In: Proc. of the 21st Int'l Parallel and Distributed Processing Symposium (IPDPS 2007). 2007. 1−8.

[17] Adams LM, Ortega JM. A multi-color SOR method for parallel computation. In Proc. of the Int'l Conf. on Parallel Processing. 1982. 53−58.

[18] Weiss C, Karl W, Kowarschik M, Rude U. Memory characteristics of iterative methods. In: Proc. of the Conf. on Supercomputing. 1999. 31.

[19] Athanasaki M, Sotiropoulos A, Tsoukalas G, Koziris N. Pipelined scheduling of tiled nested loops onto clusters of SMPs using memory mapped network interfaces. In: SC. 2002. 1−13.

[20] Hackbusch W. Iterative solution of large sparse systems of equations. Applied Mathematical Sciences, Vol.95. 1993.

[21] Moore S, Cronk D, Wolf F, Purkayastha A, Teller P, Araiza R, Aguilera M, Nava J. Performance profiling and analysis of DoD applications using PAPI and TAU. In: Proc. of the DoD HPCMP UGC 2005. Nashville: IEEE, 2005.

[22] Song Y, Li Z. New tiling techniques to improve cache temporal locality. In: Proc. of the PLDI 1999. 1999. 215−228.

[23] Renganarayana L, Rajopadhye S. A geometric programming framework for optimal multi-level tiling. In: Proc. of the SC 2004. 2004. 18.

[24] Andonov R, Balev S, Rajopadhye S, Yanev N. Optimal semi-oblique tiling. IEEE Trans. on Par. &Dist. Sys., 2003,14(9):944−960.

[25] Athanasaki E, Anastopoulos N, Kourtis K, Koziris N. Exploring the performance limits of simultaneous multithreading for memory intensive applications. The Journal of Supercomputing, 2008,44(1):64−97.

[26] Saul'yev VK. Integration of Equations of Parabolic Type Equation by the Method of Net. Pergamon Press, 1964.

[27] Pouchet L-N, Bastoul C, Cohen A, Vasilache N. Iterative optimization in the polyhedral model: Part I, one-dimensional time. ACM Int'l Conf. on Code Generation and Optimization (CGO 2007). San Jose, 2007.

[28] Bastoul C. Improving data locality in static control programs [Ph.D. Thesis]. University Paris 6, Pierre et Marie Curie, 2004.

[29] Classen M, Griebl M. Automatic code generation for distributed memory architectures in the polytope model. In: Proc. of the Parallel and Distributed Processing Symp. 2006.

[30] Ancourt C, Irigoin F. Scanning Polyhedra with DO Loops. In: PPOPP. 1991. 39−50.

[31] Asanovic K, Bodik R, *et al*. The landscape of parallel computing research: A view from Berkeley. Technical Report, No. UCB/EECS-2006-183, EECS Department University of California, 2006.

[32] Leopold C. Cache miss analysis of 2D Stencil codes with tiled time loop. Int'l Journal of Foundations of Computer Science, 2003, 14(1):39−58.

张纪林(1980－),男,山东济南人,博士,讲师,主要研究领域为并行计算与并行编译技术,性能调优.

狄鹏(1981－),男,博士生,主要研究领域为并行计算,并行编译技术.

蒋从锋(1980－),男,讲师,主要研究领域为服务计算,虚拟机.

张伟(1977－),男,讲师,主要研究领域为无线网络,演化计算.

徐向华(1965－),男,副教授,主要研究领域为并行计算,并行编译.

万健(1969－),男,博士,教授,主要研究领域为高性能计算,虚拟机性能评测.

任永坚(1963－),男,博士,教授,主要研究领域为云计算,网络存储,高性能计算.