ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.6, June 2008, pp.1274−1282
DOI: 10.3724/SP.J.1001.2008.01274

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

迭代空间交错条块并行 Gauss-Seidel 算法*

胡长军, 张纪林+, 王 珏, 李建江

(北京科技大学 信息工程学院,北京 100083)

Iterative Space Alternate Tiling Parallel Gauss-Seidel Algorithm

HU Chang-Jun, ZHANG Ji-Lin+, WANG Jue, LI Jian-Jiang

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

+ Corresponding author: E-mail: zhangjilin@acm.org

Hu CJ, Zhang JL, Wang J, Li JJ. Iterative space alternate tiling parallel Gauss-Seidel algorithm. *Journal of Software*, 2008,19(6):1274−1282. http://www.jos.org.cn/1000-9825/19/1274.htm

Abstract: In order to optimize data locality, communication and synchronization overhead, this paper proposes a multi-layers symmetric Gauss-Seidel method. Then the serial execution model of this iterative method is given, which introduces the sequence of iterative space tile as the sequence of execution, and divides iteration space by time skewing. In this model, nodes of the tile can be updated many times to improve data locality. The parallel GS execution model based on iteration space tiling is presented, which uses an improved iteration space partition algorithm and reorders the tiles of iteration space to reduce cache misses, communication and synchronization cost. Finally the numerical results are presented to confirm the effectiveness of Gauss-Seidel parallelized with alternate tiling method, specifically compared with owner-computing and red-black Gauss-Seidel methods, and show that the new parallel iterative method has better parallel efficiency as well as scalability.

Key words: Gauss-Seidel algorithm; alternate tiling; data locality; communication optimization

摘 要: 针对并行 GS(Gauss-Seidel)迭代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 GS 迭代,提出了多层对称 GS 迭代算法.然后给出了以迭代空间条块序作为执行序的串行执行模型.该模型通过对迭代空间进行“时滞”划分,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执行模型.该模型改进了迭代空间网格划分,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.实验结果表明,迭代空间交错条块并行算法比传统的区域分解方法和红黑排序并行算法具有更好的并行效率和可扩展性.

关键词: Gauss-Seidel 算法;交错网格条块;数据局部性;通信优化

中图法分类号: O241 **文献标识码**: A

许多物理应用问题的求解都归结为求微分方程数值解,其核心是高效地求解线性方程组,GS(Gauss-Seidel)

* Supported by the National Natural Science Foundation of China under Grant No.60373008 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z105 (国家高技术研究发展计划(863)); the Key Project of the Ministry of Education of China under Grant No.106019 (国家教育部科学技术研究重点项目)

Received 2007-09-10; Accepted 2007-11-06

和 SOR 迭代算法以其高效性和实现简单的特性成为大型线性方程组求解的重要算法,并且常用作非定常迭代算法的高效预条件子,是许多应用软件的核心算法之一[1].

在并行处理系统中,通信和同步开销以及数据局部性是许多并行算法程序设计必须优先考虑的问题[2].通信和同步开销越大,程序的并行效率就越低.数据局部性越高,程序的并行效率就越高.

GS 算法由于本质上的串行特性,其并行算法的实现一直是数值计算的重要研究对象.关于 GS 迭代算法的并行化设计与实现,已有很多工作,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,数据局部性呈下降趋势,并且在每次迭代计算过程,都需要通信和同步.Xie[4,5],Rohallah[6,7],Wallin[8]等人分别对 GS 并行算法进行了优化.但我们认为还有 3 个问题没有解决.(1) 数据局部性问题.大多数能够有效地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) 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,全局同步的代价会变得更加重要,并且影响算法的可扩展性.(3) 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,因此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,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 GS 算法,并且通过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,通过对网格条块进行重排,实现 GS 的并行化.算法通过时滞(time-skewing)技术[9,10],沿时间轴对网格空间进行划分,有效地降低了 GS 算法的通信和同步开销.本文第 2 节给出多层对称 GS 算法的数学描述.第 3 节描述多层对称 GS 算法的串行执行模型.第 4 节给出串行算法的并行化过程.第 5 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 GS 算法及串行执行模型

1.1 多层对称GS算法

设一维椭圆偏微分方程为

$$u_{xx}=0,\ x\in\Omega=[0,1] \tag{1}$$

其中,Dirichlet 边界条件为

$$u(x)=x,\ x\in\partial\Omega \tag{2}$$

对求解区域[0,1]使用网格划分,用空间间隔为 $h=1/(n+1)$的长度把求解区域网格化,网格点为 $x_i\{x_i=ih,\ i=0,\ldots,n+1\}$.我们用 $u_i(i=0,\ldots,n)$表示 $u(ih)$ 的有限差分近似值.对式(1)进行 3 点有限差分近似得:

$$u_{i-1}-2u_i+u_{i+1}=0,\ i=1,\ldots,n \tag{3}$$

考虑边界值条件 $x_0=0$ 和 $x_{n+1}=1$,将差分格式(3)使用矩阵形式表示为

$$AU=B \tag{4}$$

其中,

$$A=\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}_{n\times n}.$$

$U=[u_1,\ldots,u_n]^T$ 并且 $B=[b_1,\ldots,b_n]^T$.GS 算法使用初始值 $u_i^{(0)}$ 通过行优先排序,产生一组线性迭代方程,如式(5)所示,其中 k 代表迭代次数.u

$$u_i^{(k+1)}=\frac{1}{2}\left(u_{i-1}^{(k+1)}+u_{i+1}^{(k)}\right),\ i=1,\ldots,n \tag{5}$$

通过改变更新未知数 u_i 的顺序,线性迭代方程也可以表示为式(6)

$$u_i^{(k+1)}=\frac{1}{2}\left(u_{i-1}^{(k+1)}+u_{i+1}^{(k)}\right),\ i=1,\ldots,n \tag{6}$$

多层对称 GS 算法是通过交替使用不同方向的 GS 迭代求解方法,对线性迭代方程求数值解.对网格点 $u_i=\left\{u_i^k \mid k\in[(p-1)K+1,p\times K]\right\}$,$p$=1,3,5,…,2$P$+1($P$ 为自然数),迭代采用"向前"的 GS 迭代方法,由式(5)计算,其迭代过程称为奇数 K 次迭代;对网格点 $u_i=\left\{u_i^k \mid k\in[(p-1)K+1,p\times K]\right\}$,$p$=2,4,6,…,2$P$ (P 为自然数),迭代采用"向后"的 GS 迭代方法,由式(6)计算,其迭代过程称为偶数 K 次迭代.

1.2 串行执行模型

如图 1 所示,当迭代次数 $k\in[1,K]$ 时,对所有网格点使用式(5)计算,网格点更新顺序如箭头指向.类似地,当迭代次数 $k\in[K+1,2K]$ 时,对所有网格点使用式(6)计算,网格点更新顺序为由右向左.

传统的 GS 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.但这种"串行"执行序不仅其数据依赖关系阻碍了 GS 算法并行化执行,而且当数据量大时,其数据局部性较差.因此,本文提出迭代空间条块串行 GS 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,将迭代空间条块[11−14]引入执行序中,如图 2 所示.

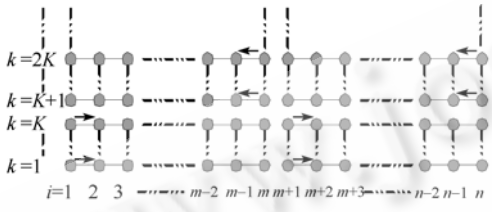


Fig.1 Multi-Layers symmetric Gauss-Seidel algorithm

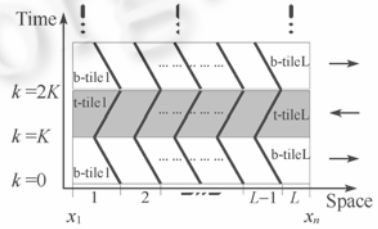


Fig.2 Serial execution model of iterative space alternate tiling GS

图 1 多层对称 Gauss-Seidel 算法

图 2 迭代空间交错条块串行 GS 执行模型

为方便算法描述,对数据空间和迭代空间的定义如下:

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,二维迭代空间 *iter_space*(I_1,T)由一维网格点 $x(i_1)$和时间维 T 组成.$u^k(i_1)$表示网格点 $x(i_1)$在 k 次迭代的值.

在传统的 GS 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串行 GS 执行模型,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 GS 迭代算法性质的同时,提高条块内数据局部性.

以一维椭圆微分方程为例,迭代空间交错条块串行 GS 执行模型如下:

步骤 1. 网格块划分.

用区域分解方法将迭代空间 *iter_space*(I_1,T)在 T=0 处进行数据划分.如图 2 所示,划分后使得每个子数据空间 *sub_iter_space*(l,0)(其中,$1\le l\le$ L)中的网格点数为 R,R 满足式(7):

$$R>2K \tag{7}$$

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[8,9],对每层迭代的数据子空间修正其边界,图 2 中的黑线表示修正后的边界.修正算法描述如下:

步骤 2.1. 在奇数 K 次迭代中,修正第 1 块网格条块 $sub_iter_space(1,k)$的右边界,使 $sub\_iter\_space(1,k)=sub\_iter\_space(1,k-1)-P^{y}_{node}(1,k-1)$.其中, $P^{y}_{node}(1,k-1)$为第 1 个网格块在第 k–1 次迭代步的右边界点.

步骤 2.2. 在奇数 K 次迭代中,修正第 L 块网格条块 $sub_iter_space(L,k)$的左边界,使 $sub\_iter\_space(L,k)=sub\_iter\_space(L,k-1)+P^{y}_{node}(L-1,k-1)$.其中, $P^{y}_{node}(L-1,k-1)$为第 L–1 个网格块在第 k–1 次迭代步的右边界点.

步骤 2.3. 在奇数 K 次迭代中,修正其余网格条块 $sub_iter_space(L,k)$的左右边界,其中 $1<l<L$.使 $sub\_iter\_space(l,k)=sub\_iter\_space(l,k-1)+P^{y}_{node}(l-1,k-1)-P^{y}_{node}(l,k-1)$.

步骤 2.4. 在偶数 K 次迭代中,修正第 L 块网格条块 $sub_iter_space(L,k)$的左边界,使 $sub\_iter\_space(L,k)=sub\_iter\_space(L,k-1)-P^{y}_{node}(L,k-1)$.

步骤 2.5. 在偶数 K 次迭代中,修正第 1 块网格条块 $sub_iter_space(1,k)$的右边界,使 $sub\_iter\_space(1,k)=sub\_iter\_space(1,k-1)-P^{y}_{node}(2,k-1)$.

步骤 2.6. 在偶数 K 次迭代中,修正其余网格条块 $sub_iter_space(l,k)$的左右边界,其中 $1<l<L$.使 $sub\_iter\_space(l,k)=sub\_iter\_space(l,k-1)+P^{y}_{node}(l+1,k-1)-P^{y}_{node}(l,k-1)$.

网格条块的边界修正后,在奇数 K 次迭代中,网格块和相应的边界点组成了下层空间网格条块,如图 2 中的 b_tile1~b_tileL.在偶数 K 次迭代中,网格块和相应的边界点组成了上层网格条块,如图 2 中的 t_tile1~t_tileL.

步骤 3. 按空间网格条块顺序执行 GS 算法.

网格条块生成后,按条块顺序执行 GS 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.由于该串行算法只是改变了网格的排列顺序,并没有改变 GS 算法的性质,因此,该算法并没有改变传统 GS 算法的算法复杂度.

2 并行交错条块 GS 算法

我们通过网格条块重排序,实现 GS 的并行化.该算法有效地降低了 GS 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 GS 迭代过程偏序关系情况下,改变迭代空间中网格点的排列顺序,GS 的敛散理论仍可以适用[2],据此迭代空间交错条块并行 GS 迭代方法可以有效地实现 GS 迭代的并行化.

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 GS 迭代方法的并行度可通过依赖关系表示.图 3 给出了二维(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 4 所示.图 4(a)显示网格条块按条块顺序($b\_tile(1)$~$b\_tile(4)$~$t\_tile(4)$~$t\_tile(1)$)串行执行迭代更新,因此没有并行度.图 4(b)显示通过改变网格条块的执行顺序,可以实现两个进程并行执行的迭代更新,因此并行度为 2.如图 4(b)所示,进程 1 依次执行 $b\_tile(1)$、$b\_tile(3)$、$b\_tile(5)$、$t\_tile(5)$、$t\_tile(3)$、$t_tile(1)$条块,进程 2 依次执行 $b\_tile(2)$、$b\_tile(4)$、$b\_tile(6)$、$t\_tile(6)$、$t\_tile(4)$、$t_tile(2)$条块.其中,进程 1 和进程 2 需要在 $b_tile(1)$和 $b_tile(2)$完成迭代更新后同步操作,以维护 $b_tile(5)$和 $b_tile(2)$的数据依赖关系.同样,在 $t_tile(5)$和 $t_tile(6)$完成迭代更新后同步操作,以维护 $t_tile(5)$和 $t_tile(2)$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 2 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来提高条块执行的并行度.如图 3(b)所示.

并行算法执行模型描述如下,如图 5(b)所示.

步骤 1. 依据处理器数目 m,将空间计算区域划分为 m 个子空间 sub_domain.第 1 块子空间的大小和最后一块子区域的大小通过添加和减少 $K/2$ 个数据进行修正(式(8),式(9)),其他子区域的大小为 n/m.

步骤 2. 交错条块 GS 算法分奇数和偶数 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 LR 顺序(左到右)执行奇数 K 次迭代,使用 RL 顺序(右到左)执行偶数 K 次迭代.

步骤 3. 各子区域通过时间轴进行划分算法,同串行 GS 算法的步骤 2.

步骤 4. 对网格条块重新排序.

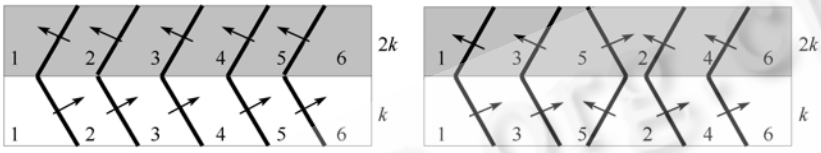
根据多色排序方法将所有的网格块进行重新排序,相邻的网格块使用不同颜色,并对相同颜色使用连续的数字标示.

步骤 5. 以网格条块为单位执行奇数 K 次迭代数据更新.

首先,执行奇数 K 次迭代更新,更新所有 sub_domain 中的“发送条块”以及第 1 个条块 *b_tile*(1);然后,将边界数据发送给相邻 sub_domain 的“接收条块”;最后,更新所有 sub_domain 中的“接收条块”以及最后一个条块 *b_tile*(2*m*).

步骤 6. 以网格条块为单位执行偶数 K 次迭代数据更新.

首先,执行偶数 K 次迭代更新,更新所有 sub_domain 中的“发送条块”以及最后条块 *t_tile*(2*m*);然后,将边界数据发送给相邻 sub_domain 的“接收条块”;最后,更新所有 sub_domain 中的“接收条块”以及第 1 个条块 *t_tile*(1).



(a) Initial execution sequence of tiles (b) Execution sequence of tiles from reordering

(a) 网格条块原始执行顺序 (b) 网格条块重排后的执行顺序

Fig.3 Execution sequence of tiles

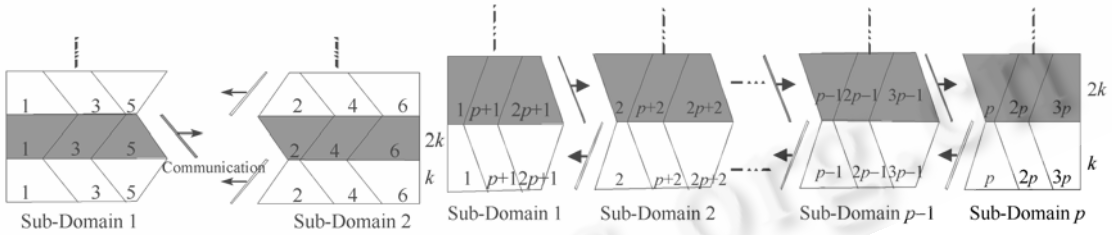
图 3 网格条块执行顺序

(a) Tile dependence graph from Fig.3(a) (b) Tile dependence graph from Fig.3(b)

(a) 图 3(a)的条块关系依赖图 (b) 图 3(b)的条块关系依赖图

Fig.4 Tile dependence graphs

图 4 网格条块间关系依赖图



(a) Parallel execution model of two sub-domains (b) Parallel execution model of multi sub-domains

(a) 两个子空间的并行执行模型 (b) 多个子空间的并行执行模型

Fig.5 Parallel execution model of iterative space alternate tiling GS

图 5 并行交错条块 GS 算法执行模型

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 GS 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,T)划分为两个子空间 sub_domain1 和 sub_domain2,如图 5(a)所示.为了保证 CPU 之间的负载均衡,sub_domain1 的网格点数 n_1 和 sub_domain2 的网格点数 n_2 必须满足式(8)和式(9).

$$n_1=n/2-K/2 \tag{8}$$

$$n_2=n/2+K/2 \tag{9}$$

其中,n 代表迭代空间网格点数,K 代表单向迭代次数.

网格条块的计算量可认为是条块内部网格点的函数,因此,$b_tile(1)$的计算量使用函数 $f\left(Ks-\frac{(1+K)\times K}{2}\right)$表示,$b_tile(2)$的计算量使用函数 $f(Kt-(1+K)\times K)$ 表示.s 和 t 分别代表 $b_tile(1)$和 $b_tile(2)$的网格点数.从负载均衡的角度考虑,第 1 块与第 2 块计算量必须相等,所以,s 和 t 必须满足以下关系:

$$s=t-(K+1)/2 \tag{10}$$

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.子空间中的条块分为 3 类: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.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,条块的执行序为先执行“发送条块”,再执行“非通信条块”和“接收条块”.“发送条块”完成迭代更新后才发送边界网格数据,这样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 2K,通信时间满足式(11):

$$T_{comm}=2\times T_s+2\times\frac{K(\text{number of points on a half of tile})}{r(\text{transfer rate})} \tag{11}$$

$$T'_{comm}=2\times k\times T_s+2\times\frac{K(\text{number of points on a half of tile})}{r(\text{transfer rate})} \tag{12}$$

交错条块并行 GS 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(11)、式(12)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 GS 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 $2\times(k-1)T_s$.

3 算法比较与实际测试

3.1 算法比较

交错条块并行化 GS 方法不同于其他 GS 并行化方法,如多色排序法、区域分解法(DDGS).多色排序法是传统的 GS 并行化方法之一,通过对矩阵图相邻节点赋予不同的颜色,且相同颜色的节点可以并行执行以提高执行速度.颜色数与并行化方法中的并行度相同.区域分解方法通过对所有网格点进行数据划分,从而实现并行化执行 GS.区域分解方法的并行度与所划分的数据块个数有关.多色排序法和区域分解法在每次迭代过程中,都需要进行相邻网格块中边界网格点的通信,通信所需要的数据为边界数据,总的通信时间见式(11).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,而同步操作将严重影响并行效率.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 GS 算法的并行化效率.此外,交错条块方法不仅适用于共享中的内存,而且适用于分布式内存机器.

3.2 性能测试

实验平台是一个 16 节点的 Cluster,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 Redhat Linux 9.0,MPI[15]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 数据局部性测试

交错条块划分 GS 算法既能优化时间局部性,又能优化空间局部性.假设 s 为数据空间均分后每个子空间内的网格数,k 为条块内迭代次数的一半.在上例的 GS 迭代算法中,对于每一个网格点有 2 个双字型变量存取数组,3 个双字型变量用于迭代计算,1 个实型变量用于敛散性判断,1 个整型变量用于存取数组下标.因此,对于子空间内的网格迭代,用于计算和存储的空间为 m,m 满足式(13),其中 i 为整型变量字节数.

$$m=(5\times2\times i+1\times i+1\times i)\times(k+s) \tag{13}$$

为了验证交错条块 GS 方法的数据局部性,使用不同的 L2 缓存块大小作为 m 的给定值.针对一维椭圆偏微分方程求解,网格规模为 10 万网格点.表 1 给出的结果是在单个节点上 GS 算法在不同划分大小的情况,当 m 值为 L2cache 大小时,L2cache 缺失率最低,相应的执行时间是传统 GS 算法的 75%.由图 6 中可以看出,在实验环境中(Intel Xeon 3.0GHz/1024Kbyte L2 缓存),随着 cache 分块大小的缩减,执行时间递增.

3.2.2 并行效率和可扩展性测试

为了验证交错条块 GS 方法的加速比和可扩展性,针对一维椭圆偏微分方程求解,对比区域分解法、红黑排序法(RBGS)以及交错条块法(ATGS)的并行效果.求解的一维椭圆偏微分方程使用中心差分格式进行离散,网格规模为 50 万,离散后形成大型规则带状稀疏线性方程组.我们给出了在不同处理器个数的情况下,3 种方法各自的加速比和效率.从表 2 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 0.9,而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地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,有效地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 7、图 8 所示),具有明显优势.

Table 1 Comparing the L2 cache misses retired and execution time of alternate tiling GS with several partitions and original GS (OGS)

表 1 网格条块及传统 GS 方法的执行时间和 L2 cache 缺失率测试数据

Partition size	L2 cache misses retired (%)	Execution time (s)
L2	4.63	453.33
1/2 L2	7.02	483.27
1/4 L2	48.34	488.01
1/8 L2	53.80	487.32
OGS	99.40	601.85

Table 2 A comparison of speedup (S) and efficiency (E) in different algorithms

表 2 不同算法的加速比和效率测试数据

CPU	ATGS (S/E)		RBGS (S/E)		DDGS (S/E)	
1	1	1	1	1	1	1
2	1.918	0.96	1.6	0.8	1.87	0.935
4	3.863	0.966	2.9	0.73	3.6	0.9
6	5.751	0.958	3.72	0.62	3.9	0.65
8	7.673	0.959	4.16	0.52	4.2	0.525
10	9.185	0.919	4	0.4	4.3	0.43
12	11.081	0.923	3.72	0.31	3.96	0.33
14	12.843	0.917	3.5	0.25	3.78	0.27
16	14.667	0.917	3.2	0.2	3.36	0.21

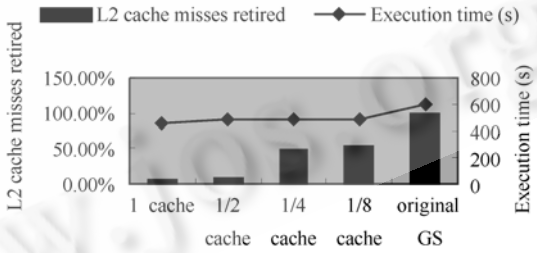


Fig.6 A comparison of L2 cache misses retired and execution time from different partition sizes and original GS

图 6 网格条块及传统 GS 方法的执行时间和 L2 cache 缺失率效果比较

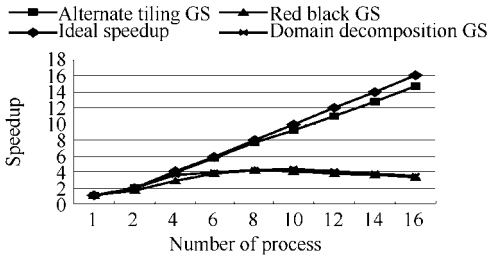


Fig.7 A comparison of the speedup in different parallel GS algorithm on different number of processors

图 7 不同并行 GS 算法的加速比比较

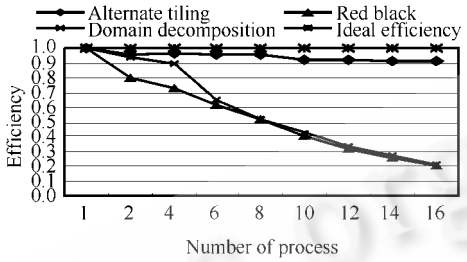


Fig.8 A comparison of the efficiency in different parallel GS algorithm on different number of processors

图 8 不同并行 GS 算法的效率及可扩展性比较

4 结 论

本文通过分析传统的 Gauss-Seidel、SOR 等迭代方法的并行化效率不高的原因,提出了一种交错条块并行化方法.通过对迭代空间进行划分,在不增加数据通信量的情况下,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 GS 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 GS 算法是一种具有较高加速比且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优化局部性和并行性的多级空间条块划分模型及分析框架,以提高传统迭代算法在集群应用中的局部性和执行效率.

References:

[1] Saad Y. Iterative methods for sparse linear systems. 2nd ed., Philadelphia: SIAM, 2003.

[2] 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.

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

[5] Xie D. New block parallel SOR methods by multi-type partitions. In: Proc. of Int'l Conf. on Parallel Processing Workshop. Montreal: IEEE Computer Society Press, 2004. 165−172.

[6] Tavakoli R, Davami P. New stable group explicit finite difference method for solution of diffusion equation. Applied Mathematics and Computation, 2006,181(2):1379−1836.

[7] Tavakoli R, Davami P. A new parallel Gauss-Seidel method based on alternating group explicit method and domain decomposition method. Applied Mathematics and Computation, 2007,188(1):713−719.

[8] 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. Cairns: ACM Press, 2006. 145−155.

[9] McCalpin S, 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.

[10] Wonnacott D. Time skewing for parallel computers. In: Proc. of the 12th Int'l Workshop on Languages and Compilers for Parallel Computing. London: Springer-Verlag, 1999. 477−480.

[11] Strout MM. Performance transformations for irregular applications [Ph.D. Thesis]. San Dieco: University of California, 2003.

[12] Huang QG, Xue JL, Vera X. Code tiling for improving the cache performance of PDE Solvers. In: Proc. of the Int'l Conf. on Parallel Processing. Kaohsiung: IEEE Press, 2003. 615−626.

[13] Rivera G, Tseng CW. Tiling optimizations for 3D scientific computations. In: Proc. of the 2000 ACM/IEEE Conf. on Supercomputing. Washington: IEEE Press, 2000.

[14] Lakshminarayanan R, Manjukumar H, Rinku D, Sanjay R. towards optimal multi-level tiling for stencil computations. In: Proc. of the Parallel and Distributed Processing Symp. IEEE Press, 2007. 1−10.

[15] Message passing interface forum. MPI-2: Extensions to the message-passing interface. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html

胡长军(1963－),男,河北沧州人,博士,教授,CCF 高级会员,主要研究领域为并行计算,并行编译技术,并行软件工程,网络存储体系结构,数据工程,软件工程.

王珏(1981－),男,主要研究领域为并行计算,并行编译技术.

张纪林(1980－),男,硕士,主要研究领域为并行计算,并行编译技术,并行软件工程.

李建江(1971－),男,博士,副教授,CCF 会员,主要研究领域为并行计算,并行编译,多线程技术.