片级三维寄生电容的并行提取算法

郑蓝舟[1] 喻文健[1] 尹 航[2] 王泽毅[1]
[1](清华大学计算机科学与技术系 北京 100084)
[2](百度公司 北京 100080)
(zhenglz05@mails.thu.edu.cn)

摘 要 随着多核 CPU 和分布式机群的日益普及,并行计算被日益广泛地应用于科学与工程实践中,以解决复杂的数值模拟问题.提出片级三维寄生电容的并行提取算法,它基于三维层次式块边界元素法,应用双向重叠组合思想将芯片划分为 4 类大小不同的"窗口";采用可变长的动态混合队列进行静态/动态结合的任务调度方法将全部"窗口"分配到不同进程,并在稀疏矩阵求和及进程间的规约求和运算中采用了提高并行效率的技术,达到了较好的负载平衡和较高的加速比.在分布式机群上采用消息传递接口编程的实验,验证了文中算法的有效性.

关键词 并行计算;重叠组合法;三维寄生电容;全芯片提取;消息传递接口
中图法分类号 TN47

A Parallel Algorithm for Chip-Level 3D Parasitic Capacitance Extraction

Zheng Lanzhou[1] Yu Wenjian[1] Yin Hang[2] Wang Zeyi[1]
[1](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[2](*Baidu Company, Beijing* 100080)

Abstract With prevalence of multi-core CPU and distributed clusters, parallel computation is widely applied to scientific research as well as engineering practices, for solving complex numerical simulation problem. A parallel computational approach to the capacitance extraction is proposed, which is based on 3D hierarchical block boundary element method (HBBEM). It applies the two-way overlap and combination idea by dividing the chip into four kinds of the "windows" with different sizes, and then uses the variable-length mixed dynamic queue to schedule the windows to different processes via a combination of static and dynamic task-scheduling methods. It also adopts techniques to increase the efficiency of parallel computation in summation of sparse matrices and reduce communication cost between processes, and to gain efficient workload balance and high speedup. Experimental results on distributed clusters implemented by message passing interface (MPI) have validated the proposed method.

Key words parallel computation; overlap-combination method; 3D parasitic capacitance; full chip extraction; message passing interface

随着半导体技术的飞速发展,集成电路芯片的特征尺寸不断减小,工作频率不断升高,使互连寄生效应成为影响集成电路性能的关键因素.在 GHz 以上纳米工艺数字电路和数模混合电路中,为了得到

收稿日期:2008-01-22;修回日期:2008-05-23.基金项目:国家自然科学基金(90407004);清华信息科学与技术国家实验室基础研究基金.郑蓝舟,男,1983 年生,硕士研究生,主要研究方向为互连寄生参数提取算法.喻文健,男,1977 年生,博士,助理研究员,主要研究方向为 VLSI 互连寄生参数提取、三维边界元快速算法与应用、互连线建模与分析等.尹 航,女,1983 年生,硕士,主要研究方向为 VLSI 互连寄生电容提取.王泽毅,男,1940 年生,教授,博士生导师,主要研究方向为 VLSI CAD 中的器件模拟、互连寄生参数提取与分析及其串并行数值方法.

高精度的时延和串扰分析,需要快速、准确地计算金属连线间的耦合寄生电容,这使得提取所有导体间全耦合电容矩阵变得越发重要.

提取三维寄生电容的方法主要有 2 类:一类是通过数值求解三维静电场方程提取电容,也称场求解器;另一类是基于模式匹配方法获取三维互连结构的寄生电容.虽然三维场求解器的计算精度高,但其运算速度较低,因此工业界的芯片级电容提取多采用后一类方法.

目前,任意拐角及大量通孔在多层保形介质结构中已广泛应用,极大地增加了互连结构的复杂性.为了得到较精确的结果,必须采用场求解器.场求解器方法可分为 2 类:1)局部方法.为导体设置一组偏压,再通过方程求解得到完整耦合电容矩阵中的一列.当前广泛应用的场求解器几乎都属于局部方法,它在提取关键路径的耦合电容时效率较高,但在全路径提取时需要对所有线网设置偏压,速度较慢.2)全局方法.不预设偏压,通过对电路网络的矩阵运算一次性地得到电路的全耦合电容矩阵,该方法更适宜于解决全芯片、全路径的提取.文献[1]提出的三维层次式块边界元素法(hierarchical block boundary element method, HBBEM)是一种提取三维耦合电容的全局方法,它将芯片窗口划分成层次小块后计算各小块的边界电容矩阵,并对边界电容矩阵作层次式合并,从而得到完整的全耦合电容矩阵.与局部方法相比,HBBEM 算法在完整耦合电容的提取速度上有明显提高.

为了适应一般版图结构的全芯片、全路径电容提取,文献[2]中将文献[3]的方法加以改进,提出了双向区域重叠组合方法进行电容提取,即先将芯片(或待求解区域)沿 2 个方向划分为一系列重叠的窗口,然后采用 HBBEM 算法计算每个窗口以及窗口重叠区域的电容并,最后通过这些电容矩阵的加减运算得到完整耦合电容矩阵.文献[2]方法具有较高的精度和计算效率.

由于全芯片提取问题规模很大,提取速度成为制约算法性能的最重要因素.随着并行计算技术的发展,如何采用并行技术来提高提取算法的效率成为新的研究热点.文献[4]提出一种用于寄生参数提取的并行化的多级矩阵压缩算法,通过对空间进行层次式划分,并用空间八叉树对划分后的节点进行组织来对远场和近场分别求解.文献[5]在文献[4]的基础上对其算法做了一系列改进,包括使用广播机制、调整任务分配、采用 *MPI_Gather* 函数代替 *MPI_Reduce* 函数提高数据收集效率等,并且还实现了并行的 LU 求解器来进行全波模拟.文献[2]基于重叠组合法对芯片级电容提取作了并行计算的初步尝试,但其仅在多 CPU 的单机上进行实验.本文在文献[2]的基础上,实现了真正的并行化芯片级电容提取算法,并在大规模机群上完成了并行计算实验.

1　HBBEM 算法

由于在重叠组合方法中需要反复使用场求解器求解窗口的全耦合电容矩阵,因此场求解器的运算效率直接影响算法的整体运算效率.HBBEM 算法全局化的特点使其能够快速地计算出窗口的全耦合电容矩阵,以保证整个并行算法的效率.

对于如图 1 所示的多介质区域,设每一个介质区域都是均匀介质,则区域 i 的 Laplace 方程可转化为直接边界积分方程

$$c_s u_s^{(i)} + \int_{\partial\Omega_i} q_s^* u^{(i)} \mathrm{d}\Gamma = \int_{\partial\Omega_i} u_s^* q^{(i)} \mathrm{d}\Gamma \qquad (1)$$

其中,$u_s^{(i)}$ 是介质 i 中源点 s 的电势,$q=\partial u/\partial \boldsymbol{n}$ 是边界上一点电势的法向导数,c_s 是与源点附近边界几何形状有关的常数.$u_s^* = 1/4\pi r$ 是 Laplace 方程基本解,其沿单位外法向 $\boldsymbol{n}$ 的方向导数为 $q_s^* = \partial u_s^*/\partial n = -(\boldsymbol{r},\boldsymbol{n})/4\pi r^3$,$\boldsymbol{r}$ 为源点到被积分点的向量(r 为相应的欧氏距离).$\partial\Omega_i$ 是包围介质 i 的边界,变量的上标(i)为其所属介质区域 i 的标识.

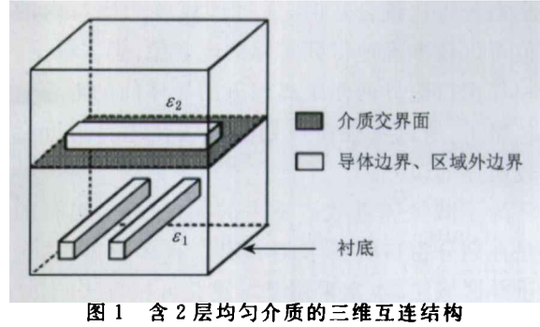

图 1　含 2 层均匀介质的三维互连结构

将区域 i 的边界离散为一系列边界元并采用常数元配置法,可将式(1)离散化为

$$\boldsymbol{Hu} = \boldsymbol{Gq} \qquad (2)$$

其中 $\boldsymbol{H}$ 和 $\boldsymbol{G}$ 均为方阵,维度为区域 i 的边界元数目.在式(2)中,$\boldsymbol{G}$ 一般为非奇异矩阵,则可以令 $\boldsymbol{A}' = \boldsymbol{G}^{-1}\boldsymbol{H}$,并得到

$$\boldsymbol{A}'\boldsymbol{u} = \boldsymbol{q} \qquad (3)$$

对于式(3),利用介质交界面上电势和电位移的连续性条件,可以消去介质交界面的变量,再通过

导体表面电荷变量凝聚技术得到方程 $Au=q$. 其中，向量 u 和 q 只包含所求解区域外边界上的变量以及每个导体的电势和总电荷，矩阵 A 被称为边界电容矩阵(boundary capacitance matrix, BCM). BCM 已经包含了全耦合电容矩阵的信息.

求 BCM 的主要开销为求 G^{-1} 的开销. 将芯片层次式划分成较小的区域，则每个区域的 G 矩阵规模较小，求逆速度较快. 在求得 2 个相邻区域的 BCM 后，可以根据交界面的连续性条件将 2 个 BCM 合并，进而得到较大区域的 BCM. 通过自底向上层次式的合并，可得到整个区域的完整 BCM，进而得到整个区域全耦合电容. 有关 HBBEM 算法的详细内容见文献[1].

2 并行重叠组合算法

2.1 重叠组合算法基本思想

对芯片级电容提取而言，由于问题规模较大，通常采用将其划分为较小窗口后再进行计算的方法. 如果要提取的是关键路径上的电容，则可沿关键路径划分窗口；如果要提取全芯片的耦合电容矩阵，则需要对全芯片作窗口划分.

根据静电场的电势叠加原理，在不同窗口中求得的电容值具有可叠加性. 此外，根据电容的局部特性，距离较远或者被其他导体屏蔽的导体间的电容很小，可在一定条件下忽略. 所以在理论上，当窗口大小划分得比较合适时，通过计算各个窗口内的电容值可以较准确地得到实际的电容值. 但实际上，由于相邻窗口边界两侧距离很近的导体间的电容效应被忽略了，所以常规窗口划分方法在边界附近会有比较明显的误差.

为了减少邻近效应带来的误差，一个可行的方法是在划分窗口时要求相邻窗口有部分重叠，这就是重叠区域法. 本文采用二维重叠区域方法.

在文献[2]工作的基础上，本文简化了窗口划分类别，这将有利于并行任务的调配. 下面简要介绍所采用的二维重叠区域划分方法：首先对 x 方向和 y 方向分别设置宽度为 x_o 和 y_o 的重叠区域，然后在整个芯片边界保留 1/2 重叠宽度的一圈. 将去除外围一圈的芯片沿 x 方向和 y 方向划分为 $m\times n$ 个相同大小的基本区域，每个基本区域加上周围 1/2 重叠的外延就是一个完整的主窗口(A 类窗口)，设其尺寸为 $x_l\times y_l$，主窗口之间互相重叠就得到了一次重叠窗口(B 类窗口和 C 类窗口)和二次重叠窗口(D 类窗口). 按照这种划分模式，每一类窗口都有相同的几何尺寸. 图 2 所示为一个 4×3 的二维重叠区域划分的示意图，图中的实线是基本区域的边界，虚线为重叠区域的边界.

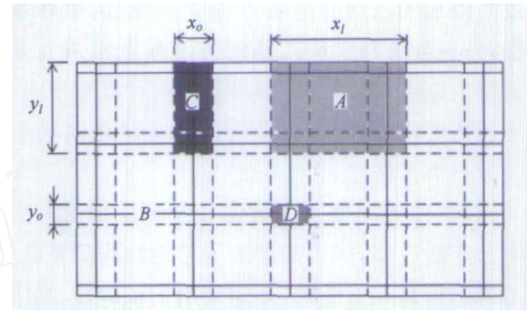

图 2 二维重叠组合法窗口划分示意图

得到各类窗口的电容矩阵后，完整的芯片级电容矩阵可通过它们的代数和叠加得到. A 类窗口为主窗口，电容矩阵取正号；B，C 类窗口为一次重叠窗口，电容矩阵取负号；D 类窗口为二次重叠窗口，电容矩阵应取正号. 全芯片电容矩阵

$$C=\sum C_A-\sum C_B-\sum C_C+\sum C_D \quad (4)$$

其中，C_A，C_B，C_C 和 C_D 分别表示 4 类窗口所对应的电容矩阵，C 为全芯片电容矩阵.

2.2 MPI 简介

消息传递接口(message passing interface, MPI)[6-7]是一种基于消息传递编程模型的并行语言实现. 目前几乎所有的并行计算机都支持 MPI 的标准通信库，MPI 已经成为消息传递编程模型的一个事实标准.

严格来说，MPI 只是一个并行库，而不是一种语言，其实现必须依赖于特定的串行语言. 在 MPI-1 的标准中，规定了 MPI 和 FORTRAN77 语言与 C 语言的绑定，而在更新的 MPI-2 标准中又增加了和 FORTRAN90 语言与 C++的绑定，使得 MPI 可以应用于面向对象的编程. 本文的并行算法程序采用 MPI+C/C++的形式实现.

2.3 并行算法设计

重叠组合算法处理芯片级提取可分 2 步：1)将芯片划分为多个计算相对独立的窗口，并对每个窗口进行求解；2)对计算得到的各窗口电容矩阵进行代数求和，即通过窗口电容矩阵的叠加得到完整耦合电容矩阵. 并行计算时，芯片划分得到的窗口数在一般情况下总是远大于并行计算进程数的，所以每个进程需要完成多个窗口任务的计算，并得到多个局部的窗口电容矩阵. 矩阵求和的并行处理也可分

2 步实现:1)各个进程将本地运算得到的各个窗口矩阵进行求和;2)将各个进程运算得到的矩阵进行总的叠加得到最终结果.这样,进程间的通信和数据传输只发生在第二步中,显著地减少了通信开销.第二步操作需要所有进程同时参与,可采用 *MPI_Reduce* 函数实现.该并行算法的设计流程如图 3 所示.

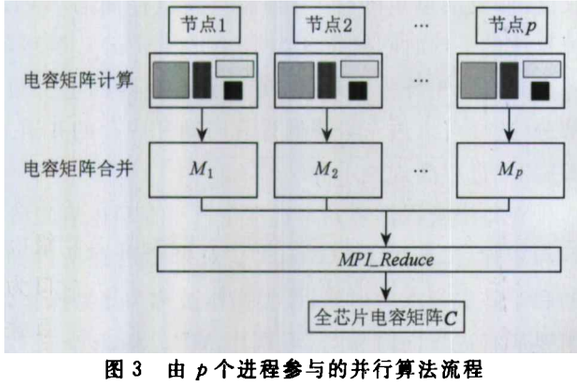

图 3　由 p 个进程参与的并行算法流程

2.3.1　窗口提取任务调度

并行算法的第一步是将每个窗口的计算任务分配到各个进程以提取窗口电容矩阵,它占据了整个算法绝大部分的时间开销.理想状况下,该步骤的时间开销应为

$$N_{avg}=\frac{N}{p}=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}a_{i,j}+\sum_{i=1}^{m}\sum_{j=1}^{n-1}b_{i,j}+\sum_{i=1}^{m-1}\sum_{j=1}^{n}c_{i,j}+\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}d_{i,j}}{p}.$$

其中,N 为总计算量,N_{avg} 为每个进程的平均计算量,p 为进程数,m 和 n 为 $x-y$ 这 2 个方向上的窗口划分数;$a_{i,j}$,$b_{i,j}$,$c_{i,j}$ 和 $d_{i,j}$ 分别为 A,B,C 和 D 这 4 类窗口的计算量.如果仅以窗口面积大小作为衡量其计算量的标准,一般来说,A 类窗口的计算量远大于 D 类窗口,而 B 和 C 2 类窗口的计算量介于这二者之间.

在理想情况下,同一类型中不同窗口的计算量差别不大,可将同类型窗口平均分配到 p 个进程上,以获得比较平衡的负载.但是在实际芯片中,不同区域的布线密度往往差别很大,同样大小的窗口落在布线稠密区域时其计算量会明显大于落在布线稀疏区域的窗口;同时,HBBEM 是一个层次式算法,其时间开销不仅仅与该窗口内边界元数目相关,还与区域内的导体分布及具体的层次划分有明显关系,因此难以得到文献[5]中那样简单的任务量估算公式.

在窗口计算量很难预估的前提下,本文采用动静态混合的任务分配策略来平衡进程间负载.由于每个类型窗口的计算时间大致在一个数量级上,所以大部分任务可以静态分配,少量任务进入动态任务队列做动态分配,以降低动态分配较大的额外网络开销.假设静态分配到第 i 个进程的任务量为 F_i,则达到负载平衡所需动态队列的任务量 F 满足

$$F\geqslant\sum_{i=1}^{p}(\max(F_i)-F_i) \tag{5}$$

如果式(5)的条件满足,在忽略网络延迟差异的前提下,最早和最晚结束计算的进程间的结束时间差距,即最长空闲时间,不会超过任务池中任务量最大的一个任务对应的计算时间.为了降低最长空闲时间,要求任务池中每个任务的计算量较小.但是在任务池中总任务量满足式(5)的情况下,如果每个任务的计算量都较小,则需要的任务数就较多,而每分配一个任务池中的任务都需要在进程间进行通信,较多的任务分配会带来较多的额外通信开销.实际上,任务池中的任务是作为一个任务队列存在的,每当空闲进程来请求任务时,就将队列头上的任务分配给空闲进程.在大部分情况下,进程快结束时分配到的都是任务队列尾端的任务,所以最长空闲时间主要与任务队列尾端的任务相关.为了兼顾负载均衡和通信开销,本文采用了混合任务队列.任务队列主要由 A 类窗口对应的任务组成,这类窗口计算量大,可以减少队列中的任务数;队列尾端则是 D 类窗口对应的任务,这类窗口计算量最小,可以降低进程的最长空闲时间.

此外,本文通过实验还发现,随着进程数 p 的增加,保持负载平衡所需任务池的任务量也会随之增加.所以我们采用的动态分配任务队列是变长任务队列,即队列的长度由进程数确定,在进程数多时适当增加队列长度.

还要注意:为了减少数据的传输,在动态分配阶段应只把任务编号传给需要对其计算的进程,由进程根据收到的编号和输入文件来直接得到需要的数据.

算法 1. 并行算法.

Step1. 根据输入参数进行窗口划分.

Step2. 根据进程数设置动态队列长度(包括 A 任务个数和 D 任务个数).

Step3. 将 B,C 2 类任务和 A 与 D 2 类任务中参与静态分配的任务按如下规则分配到各个进程:进程 i 计算每类窗口任务中标号为 i,$i+p$,$i+2p$,…的任务.

Step4. 将动态队列中 A 类任务分配给空闲进程.

Step5. 将动态队列中 D 类任务分配给空闲进程.

算法 1 中，Step1，Step2 为串行处理，Step3～Step5 为并行处理. 其中，Step3 采用的静态分配方法可有效地避免将导体比较集中或者比较稀疏的某些区域的窗口被分配给同一进程计算.

一个并行程序在 p 个处理器上的运行时间满足

$$T(N,p)=\sigma(N)+\varphi(N)/p+\kappa(N,P) \quad (6)$$

其中，T 为运行时间，σ 为串行部分用时，φ 为并行部分用时，κ 为处理器通信和同步用时，N 为问题规模. 对算法 1，串行部分 σ 只有窗口划分等简单运算(Step1 与 Step2)，它与并行部分 φ 相比时间开销很低，基本可以忽略. 并行部分在采用变长混合动态队列后可基本达到负载平衡，使式(6)中的 $\varphi(N)/p$ 接近理想情况. $\kappa(N,p)$ 通常为 p 的增函数，尤其在算法 1 中，动态队列长度随 p 增加而增加，则通信开销也随之增加. 但是动态队列本身任务数量不多，且并行部分计算量较大，使通信开销在 T 中所占比例不大. 因此从整体来看，算法 1 的运行时间和进程数接近反比关系，即接近并行计算的理想情况. 后面的实验将验证这个结论.

2.3.2 稀疏矩阵求和

在得到各个任务窗口的电容矩阵后，各个进程需完成本进程计算所得电容矩阵的合并计算. 合并计算如式(4)，是稀疏矩阵的求和运算.

为了提高算法效率，本文采用平衡二叉排序(AVL)树进行矩阵的存储和计算. AVL 树是一棵具有以下特征的二叉树：对于该树的任意一个节点，其左子树中节点的键值均小于该节点，右子树中节点的键值均大于该节点，且左右子树的高度差不超过 1. 对参与求和的矩阵中的每一个元素，以它在矩阵中的位置下标作为键值插入 AVL 树，插入过程中搜索到键值相同的节点时将电容值相加. 这样，在插入所有待求和稀疏矩阵的元素后，便得到由 AVL 树表示的耦合电容矩阵.

AVL 树的主要优点是运算速度较快，且可以直接得到有序的结果. 假设结果矩阵的元素数为 M，则 AVL 树的高度为 $\log M$，所以每插入一个元素的平均时间复杂度是 $O(\log M)$. 设待插入的矩阵元素个数为 N，则总时间复杂度为 $O(N\log M)$.

下面将 AVL 树算法与理论最优的 Hash 表方法进行对比. 如采用 Hash 表进行稀疏矩阵求和，插入一个元素的平均时间复杂度为 $O(1)$，插入 N 个元素的时间复杂度即为 $O(N)$. 但是，Hash 表中的结果是无序的，而第 2.3.3 节中进程间矩阵规约求和需要有序的向量，因此必须对结果进行排序. 采用平均性能最优的快速排序法，其时间复杂度为 $O(M\log M)$. Hash 表方法的总时间复杂度为 $O(N+M\log M)$，相对于 AVL 树方法并无明显优势.

对整个矩阵建立一棵 AVL 树，由于结果矩阵的元素较多，其搜索长度也比较长. 如果对于矩阵的每一行建立一棵 AVL 树，则可以大大地缩短搜索长度. 假设结果矩阵有 n 行，那么按行存储的 AVL 树算法的平均时间复杂度为 $O(N\log(M/n))$. 最后，考虑到电容矩阵为对称阵，可以只存矩阵的上三角部分，这样可以进一步降低计算时间和内存的开销.

2.3.3 进程间规约求和

在每个进程都得到本进程中所有窗口电容矩阵的和矩阵后，算法 1 的 Step3 是对每个进程所得到的和矩阵进行求和运算，该类操作被称为规约操作，需要所有进程同时参与，可调用 *MPI_Reduce* 函数完成.

由于进程间通信只能传输连续的数据，因此，需将 AVL 树表示的矩阵行向量转换成数组存储的压缩向量形式，然后调用 *MPI_Reduce* 函数来完成规约操作.

如果对矩阵整体进行规约，则每个进程只有一个待规约元素，每步运算依次只有 $p/2$，$p/4$，$p/8$，… 个处理器在进行运算. 如果采用行向量表示，则每个进程的待规约元素为 k 个，由 *MPI_Reduce* 函数的实现可知，当规约元素个数大于进程数时，可以让几乎所有处理器都处在计算状态，从而可以提高并行效率[8].

3 实 验

本文的实验环境为 HP Server rx2600，机群共 128 个节点，每个节点有 2 个 1.3 GHz 主频的 Intel Itanium 2 处理器和 4 GB 的内存；机群的网络带宽为 10 Gb. 本文实验最多采用了 30 个 CPU 进行并行计算，并行软件环境为 MPICH.

实验的输入数据为一个小规模 CPU 芯片的布线分布，芯片尺寸为 716.88 μm×724.24 μm，该芯片布线包含约 37 000 个矩形，共 3 036 个线网.

对芯片进行 40×40 的二维划分，2 个方向上的重叠宽度均为 10 μm. 表 1 所示为该划分模式下、在不同进程数下算法各个主要处理步骤的时间开销.

表 1　不同进程下时间开销　　s

CPU 数目	窗口计算时间	进程内矩阵求和时间	规约求和时间
5	52 314	1.68	30
10	26 130	2.09	35
15	17 456	2.79	44
20	13 240	3.29	50
25	10 638	2.58	50
30	8 984	2.82	48

表 1 中的窗口计算时间和进程内的矩阵求和时间都是最慢进程的时间开销.可以看出,算法主要的时间开销为计算窗口内的局部电容矩阵的时间.由于采用了变长的混合动态任务队列,窗口计算部分的并行程度非常好,时间开销基本与进程数呈反比.因为结果矩阵的规模不会随进程数增加而明显变化,所以进程内矩阵求和时间变化较小.由于矩阵规约求和为全局操作,所以时间开销会随进程数增加而略有增加.

图 4 所示为不同 CPU 数目下的加速比曲线,并行算法加速比的定义为串行计算时间和并行计算时间的比值.

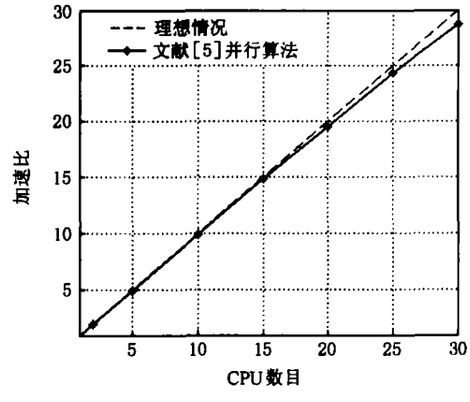

图 4　不同 CPU 数的加速比曲线

图 4 中,上方曲线为理想状况,可以看出,本文的并行算法有很高的并行性,加速比接近理想状况.这是因为该算法的主要时间开销部分(即窗口计算部分)能基本达到负载平衡,而通信开销又相对不大,从而保证了较高的并行效率.

图 5 所示为文献[5]中的并行电容提取算法的加速比曲线.

注意,图 5 中所取的加速比不是并行计算中标准的串行计算时间和并行计算时间的比值,而是 2 个 CPU 的并行计算时间和多个 CPU 的并行计算时间的比值.由于基准值已经是并行计算时间,包含一部分并行开销,所以加速比曲线会优于标准的加速比曲线.对比图 4,5 可以发现,本文算法的加速比曲线更接近理想曲线,而且在并行进程数更多的情况下,保持了良好的加速比特性.

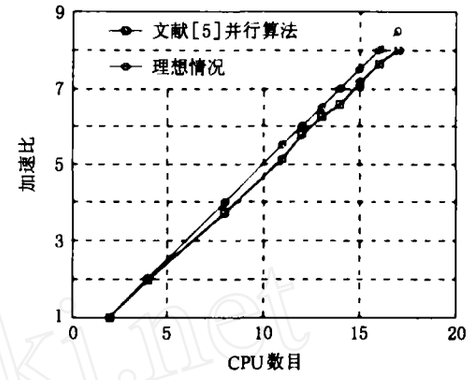

图 5　文献[5]并行算法加速比曲线

图 6 所示为计算窗口电容矩阵时不同处理器数目下最大空闲时间的曲线.最长空闲时间可以在一定程度上反映负载均衡状况.从图 6 中可以看出,本文算法的最大空闲时间一直维持在较低的水平上,说明该算法的负载平衡状况良好.

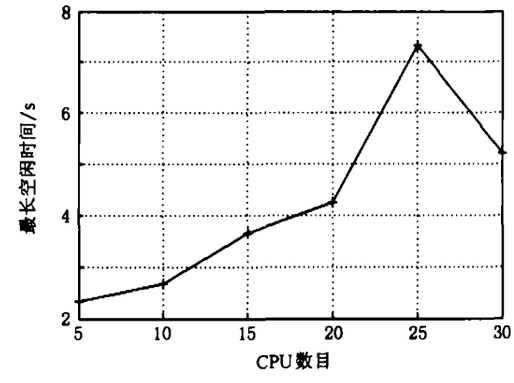

图 6　最长空闲时间曲线

最后对芯片采用新的划分参数,观察其对并行算法的性能影响.新划分参数为 50×50 划分,2 个方向的重叠宽度均为 8 μm.设原划分方式为划分 1,新划分方式为划分 2,图 7 所示为 2 种划分方式下加速比的对比曲线.

从图 7 中可以看出,新的划分方式的加速比要略低于原有划分方式,这是由于采用新的划分方式后,整体的计算量下降了.原划分方式的串行计算时间是 258 164 s,而新划分方式的串行计算时间是 134 390 s,约为原划分方式的一半.但是采用新划分方式的通信开销则和原划分方式基本相当,使通信

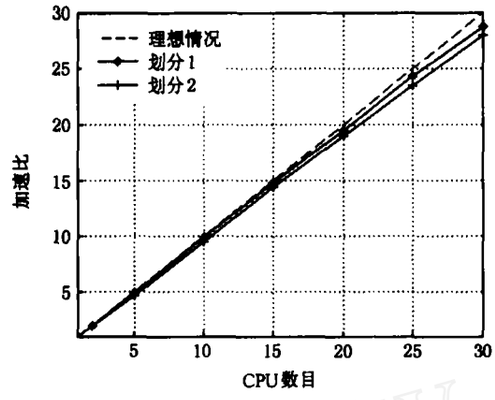

图 7 2 种不同划分方式下的加速比曲线

开销在总开销中的比重增大，并最终导致了加速比的下降. 通过这个实验说明，较小的窗口和重叠宽度可以减少重叠组合算法的串行计算时间. 但是，随着 CPU 数目的增加，串行速度较快的算法并行效率下降也会相对稍快. 此外，速度也不是窗口选择的唯一标准，较小的窗口和重叠宽度也会增加重叠组合方法的误差.

4 小 结

本文在文献[2]的基础上，利用 MPI 语言，实现了能够应用于大规模并行机群的芯片级并行提取算法，并对该算法的负载均衡和通信开销等影响并行性能的关键因素进行了较为详细的分析.

重叠组合算法的内核为 HBBEM 算法，与传统方法相比，该算法在提取速度上有明显的优势，但是其运算量受到较多因素的影响，较难估计. 本文提出了适应该算法特点的变长混合动态任务队列，较好地实现了负载平衡，也尽量降低了网络通信的开销，保证了算法有很高的并行效率.

在稀疏矩阵求和过程中，采用了先进程内合并，再进程间规约的两步求和方法. 应用 AVL 树算法及矩阵分行存储的技术，进一步提高了并行算法的效率. 实验结果表明，本文算法在大规模的分布式机群上也有很高的并行效率，有很强的并行扩展性.

参 考 文 献

[1] Lu Taotao, Wang Zeyi, Yu Wenjian. Hierarchical block boundary-element method (HBBEM): a fast field solver for 3-D capacitance extraction [J]. IEEE Transactions on Microwave Theory and Techniques, 2004, 52(1): 10-19

[2] Yin Hang, Yu Wenjian, Lu Taotao, *et al*. The overlap-combination approach to 3-D chip-level capacitance extraction and its parallel implementation [J]. Journal of Computer-Aided Design & Computer Graphics, 2006, 18(2): 238-244 (in Chinese)
(尹 航，喻文健，陆涛涛，等. 重叠组合法的芯片级三维寄生电容提取及其并行实现[J]. 计算机辅助设计与图形学学报，2006，18(2)：238-244)

[3] van Genderen A J, van der Meijs N P. Hierarchical extraction of 3D interconnect capacitances in large regular VLSI structures [C] //Proceedings of International Conference on Computer-Aided Design, San Francisco, 1993: 764-769

[4] Chuanyi Y, Swagato C, Dipanjan G, *et al*. A parallel low-rank multilevel matrix compression algorithm for parasitic extraction of electrically large structures [C] // Proceedings of the 43rd ACM/IEEE Design Automation Conference, San Francisco, 2006: 1053-1056

[5] Wang X R, Jandhyala V. Parallel algorithms for fast integral equation based solvers [C] //Proceedings of the 16th IEEE Conference on Electrical Performance of Electronic Packaging, Atlanta, 2007: 249-252

[6] Michael J Q. Parallel programming in C with MPI and OpenMP [M]. Beijing: Tsinghua University Press, 2004: 74-76, 134-135 (in Chinese)
(Michael J Q. MPI 与 OpenMP 并行程序设计[M]. 陈文光，武永为，译. 北京：清华大学出版社，2004：74-76，134-135)

[7] Du Zhihui. MPI parallel programming [M]. Beijing: Tsinghua University Press, 2001: 15-18 (in Chinese)
(都志辉. 高性能计算并行编程技术——MPI 并行程序设计[M]. 北京：清华大学出版社，2001：15-18)

[8] Rolf R. New optimized MPI reduce algorithm [OL]. [2008-01-22]. http://www.hlrs.de/organization/par/services/models/mpi/myreduce.html