应用驱动的并行程序性能优化研究

邸瑞华 蒋海华 吕 海
(北京工业大学计算机学院 北京 100124)

摘 要 从应用角度出发,分析、归纳各种应用中的核心计算过程,利用符合多核处理器芯片架构的并行计算模型对这些核心计算过程进行优化,得出可以被重复利用的高性能可扩展的软件库,它既可以支持新应用的高效开发,也可以保证程序性能的可扩展性。以分层并行计算模型思想为指导,从应用驱动的并行程序性能优化的角度出发,首先提出了面向多核处理器芯片体系结构的并行算法设计模型,在此基础上对并行扫描算法进行分析优化,得出新的具有良好扩展性、高性能的g-scan算法。之后深入研究13种核心计算实体之一的稀疏线性代数计算实体,应用g-scan算法设计实现了新的稀疏矩阵-向量运算算法,并将其应用于结构工程领域中广泛使用的有限元分析,大大提升了其执行效率。

关键词 多核处理器,计算实体,扫描算法,有限元分析
中图法分类号 TP393 **文献标识码** A

Research on Application-driven Parallel Program Performance Tuning

DI Rui-hua JIANG Hai-hua LV Hai
(College of Computer Science, Beijing University of Technology, Beijing 100124, China)

Abstract Multi-core processor provides multiple threads parallel execution capability, and makes applications to have huge potential for performance improvement, but makes it enormously challenge to efficiently develop high-performance program. Meanwhile, through the old process of performance optimization, the scalability is difficult to be guarantee. From application point of view, attributing core calculation to patterns and motifs, and optimizing these motifs can produce reusable library that can support efficient develop new application, and also can guarantee the scalability of the application performance. A layered parallel computing model was used for guidance in this article. From the perspective of application-driven parallel program performance optimization, this article designed a new parallel multi-core processor computing model, which can be used in the architecture of multi-core processor chip. Based on this model, g-scan algorithm was designed which has good extendibility together with high performance after analyzing and optimizing some fundamental parallel algorithms. At the last, newly designed parallel algorithm was applied to OpenSeesSP which is a finite element software widely used in structure engineering.

Keywords Multi-core processor, Computing motif, Scan algorithm, Finite element method

1 引言

一直以来,处理器芯片厂商通过不断提高主频和指令级并行执行能力来提升处理器的性能。如今这种方法受到内存带宽、指令级并行度、单线程性能、功耗等诸多因素的限制,已无法延续下去。从应用需求来看,日益复杂的科学计算、多媒体、虚拟化等多个应用领域都在呼唤更为强大的计算能力。在技术困境和应对计算能力需求的双重驱动下,多核处理器芯片作为处理器芯片厂商的应对策略,成为提升处理器性能的事实上的解决方案,这使得提升处理器性能的思路转变成为不断增加单个处理器芯片中处理核心的数目。这样,在当前和未来所有的计算机都成为了SMP即共享内存的并行计算机。已有研究结果[1-3]预言,处理器芯片中的处理核心数目会每两年翻一番。

计算机核心体系结构的巨大变化,使得软件性能随硬件性能提升而提升的"免费午餐"结束[4]。目前如何利用多核处理器芯片提供的计算能力是一个难题。这是由于要充分发挥多核处理器芯片的计算能力,必须使在其上运行的程序并行化,而且要利用好片上的计算和存储资源,这就要求并行执行的应用任务划分适中,同步和通信粒度适中,但是只有少数的并行计算专家才具备这样的能力。

另一方面,对于已开发的应用程序,多核处理器芯片并不能有效地发挥其全部的性能,这与过去的单核处理器芯片性能提升促使应用性能提升的情形是不同的。多核处理器芯片

到稿日期:2012-06-30 返修日期:2012-08-02 本文受中国教育科研网格二期建设项目(ChinaGrid 2)资助。
邸瑞华(1947-),女,博士生导师,主要研究方向为网络应用及网络分布计算环境、分布式系统的体系结构、软件工程等;蒋海华(1984-),男,讲师,主要研究方向为高性能计算、分布式计算等;吕 海(1982-),男,博士生,主要研究方向为高性能计算、并行计算、分布式计算,E-mail:lvhai@emails.bjut.edu.cn。

要求开发出的应用程序的性能能够随着处理核心数目的增加而提高。虽然应用程序在运行时可以调用操作系统原语，查到当前运行平台的处理核心数目，但对应用开发者来说，开发自适应计算平台体系结构的并行应用程序无疑是巨大的挑战。

本文以分层并行计算模型思想为指导，从应用驱动的并行程序性能优化的角度出发，首先提出了面向多核处理器芯片体系结构的并行算法设计模型，在此基础上对并行扫描算法进行分析优化，得出新的具有良好扩展性、高性能的 g-scan 算法。之后深入研究 13 种核心计算实体之一的稀疏线性代数计算实体，应用 g-scan 算法设计实现了新的稀疏矩阵-向量运算算法，并将其应用于结构工程领域中广泛使用的有限元分析，大大提升了其执行效率。

2 背景

2.1 并行程序性能优化的一般流程

软件性能优化[5]一般需要经历以下几个阶段：收集数据、分析数据、寻找解决方法、修改代码实施改进和测试评估性能，如图 1 所示。首先，利用性能分析工具收集程序执行过程中的性能数据，找到程序中执行最耗时的部分，通常称其为热点；然后分析热点存在的原因，确定瓶颈所在，如内存访问效率低、循环不恰当、没有采用最优算法、编译器和硬件特性没有考虑等，通常每次集中解决一个性能问题；接下来寻找解决这些问题的方法，修改程序实施优化；最后重新编译执行修改后的程序，通过修改前后执行时间的对比来验证所做的优化是否发挥作用，能否带来性能的提升，评估性能是否可以接受，以此来决定是否重复整个过程，进一步优化程序性能。最后一步评估是很重要的，因为对于程序来说，没有最快，只有更快，如果想要提升性能总会找到地方，这就需要用户权衡性能提升量和工作量或者花费的代价，来决定什么时候终止优化过程。知道何时停止优化是非常重要的，不要花费大量的精力去优化一个仅能提高少许速度的函数。

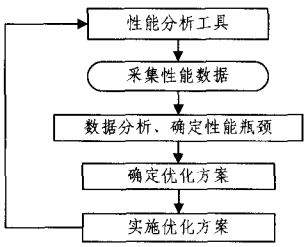


图 1 并行程序性能优化的一般流程

从并行算法层面讲，对并行程序性能优化的关键是对应用程序中关键算法的分析优化，而在科学计算领域中，各种各样的应用层出不穷，且每种应用的算法大多都存在着不少差异。这样，对于某种应用的优化分析成果不易被其他的应用所利用。从并行程序实现角度看，由于并行编程模型从上到下的各个方面：程序整体任务划分（如流水线、障同步、Map Reduce 等）、任务并行执行机制（如分治、任务并行、数据并行）、数据结构（如分布式数组、共享数组、共享哈希表等）、运行时同步通信机制（如锁机制、消息传递、集合通信等）都不同，且对于每个应用根据自身特点采用的是不同的实现方案，因此对一个应用的优化成果，也不易为其他应用所采用。从并行程序执行环境层面来看，对于一个具体的应用的优化都是基于一个具体计算平台的体系结构，当计算平台的体系结构发生改变时，之前的优化成果便无法再重新利用；另一方面，这个层面的优化往往是既与应用的计算特性（计算模型、通信模型、访存模式）紧密相关，也跟计算平台的体系结构紧密联系的，因此一种应用的优化成果往往不能被另外的应用所采用。

2.2 计算实体

有研究人员推断[6]，在科学计算领域中按应用计算模式的不同，共有 7 种计算实体，称之为 7 个“矮人”。这其中包括：稠密线性代数计算、稀疏线性代数计算、结构化网格计算、非结构化网格计算、空间方法计算、偏微分方法、蒙特卡洛模拟。美国伯克利大学的研究者根据这个想法，从机器学习、数据库视频游戏的计算应用中进一步发掘计算实体，扩充了 6 种计算实体，分别是：组合逻辑计算、图遍历算法、动态编程、回溯算法、分支界定算法、图像处理算法。未来如果有新的应用出现，将会有更多的计算实体加入到这里。发掘这些计算实体是非常有用的，因为在各种各样的领域应用中，最核心的计算可以由这些计算实体来表现。这样对于大多数的编程人员来说，只需弄清楚其开发的应用中的计算实体，然后直接利用由领域专家设计的最优的计算方法，就能解决应用里最核心的计算过程。这种方法将并行性能优化的巨大挑战交给了各个领域的专家而不是普通的应用开发者，极大提高了应用开发效率，同时保证了对这些计算实体的优化成果可以十分容易应用于各种相关的实际应用中。这种以应用驱动的并行程序性能优化成为了一种趋势。

2.3 分层并行计算模型

并行计算的一般过程是，对给定计算问题，根据某个并行计算模型分析并设计解决该问题的并行算法，然后用特定编程语言或编程模型实现该算法，最后在目标计算平台编译并行程序，运行求解给定的计算问题。整个并行计算过程表现出来的性能取决于一系列因素。

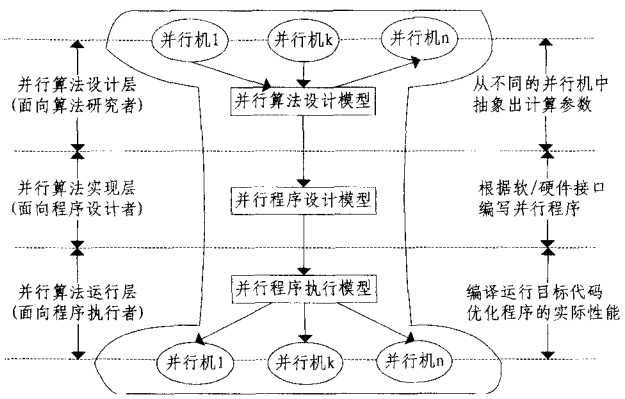


图 2 分层并行计算模型

我国中科院陈国良院士提出了分层并行计算模型[7]，如图 2 所示，他将并行计算的各个阶段限制和约束到各自模型下来进行，其分别是并行算法设计模型、并行程序设计模型、并行程序执行模型。并行算法设计模型面向算法研究人员，是算法设计者和体系结构专家之间的桥梁，从算法角度将不同并行计算机抽象为一种通用的并行机，算法设计者在其上设计和分析算法；并行程序设计模型面向并行应用开发人员，按照程序的执行流程，选用某种语言正确地实现并行算法。并行程序执行模型面向程序运行者，从性能效率的角度指导

程序运行者，将不同编程模型实现的并行程序在具体的计算平台上编译、优化和运行。分层后，目标单一，各负其责，易于设计和实现。目前并行算法设计模型和并行程序设计模型的研究相对成熟，并行程序执行模型的研究尚处于起步阶段。

3 应用驱动的并行程序性能优化

以应用驱动的并行程序性能优化过程如图 3 所示，即从顶层并行算法设计模型着手，对多核处理器芯片的并行算法设计模型进行深入的研究，设计新的面向多核处理器体系结构的算法设计模型，在此基础上对当前的 13 种计算实体进行研究。本文重点分析、优化其中的稀疏线性代数计算实体，对于稀疏线性代数实体的优化过程是首先基于新设计的并行算法设计模型对相关的基础并行算法进行分析优化，最后应用已有成果对稀疏线性代数计算实体进行优化。

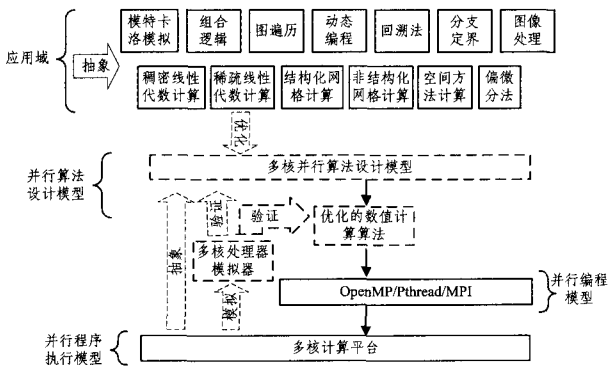


图 3 应用驱动的并行程序性能优化示意图

4 UPMM 多核并行计算模型

多核处理器芯片的架构主要有 3 种类型：同构超标量多核芯片、同构简单多核芯片、异构多核芯片。同构超标量芯片是指把多个已经通用的复杂的超标量处理核心集成到一个芯片，这样的架构的创新之处在于其片上缓存的组织结构。由于其中的每个处理核心为通用的复杂超标量处理核心，使得这样架构的多核处理器仍然能够具有出色的单线程处理能力。同构简单多核芯片是把多个设计相对简单的标量处理核心或中介核心集成到一个芯片上。由于每个处理核心较为简单，因此在一个芯片上可以集成大量的这种芯片。这样的架构设计完全从只注重单个线程的性能转移到了提高多线程吞吐率上了。异构多核处理器在一个芯片上将一个或多个复杂超标量处理核心和多个简单标量处理核心集成在一起，这样处理器既能保证一定的单线程性能，也能保证众多标量处理器所带来的多线程的高吞吐率。本文所讨论的多核处理器芯片的架构主要集中在同构多核处理器，把异构多核处理器作为未来的研究工作。

假设多核处理器芯片中有 P 个处理核心，每个处理核心可以访问 Q 级缓存，L_0 指的是片上每个处理核心所私有的寄存器，L_{Q-1} 指的是主存；在每个级别的缓存中，缓存容量是相等的，并且其被共享的处理核心数目也是相等的；级别 q 有 N_q 个缓存，缓存容量大小为 C_q，被 M_q 个处理核心所共享；所有缓存按块划分，块大小为 B；私有的缓存不能被其他处理核心直接访问，处理核心要做一个数据操作，数据必须先读到其私有缓存中；数据在不同级别的缓存和主存之间进行移动。模型的逻辑结构如图 4 所示。

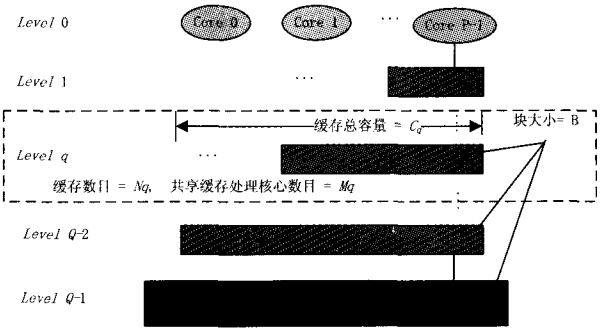


图 4 UPMM 多核并行计算模型

该模型中所用到的主要参数定义如下：

P 为片上处理内核数。

Q 为缓存分级数，其中在第 $q(0 \leqslant q \leqslant Q-1)$ 级缓存，缓存数为 N_q，每个缓存容量为 C_q，共享一个缓存的处理内核数为 M_q。

B 为缓存分块大小。

在本模型中，以 I/O 复杂度来度量整个模型复杂度，即在主存和缓存之间数据块并行移动的次数。例如，一个算法 F 在该模型上做计算时，同时从主存读入一次 P 个大小为 B 的数据块，则其 I/O 复杂度为 O(1)，而不是 O(P)。模型中多个处理内核是通过读写在共享缓存或主存中的数据来通信的，各个处理内核可以并发读写不同的缓存或主存中的数据，当多个处理内核同时读写同一个缓存或主存中的数据块时，采用 PRAM 模型中的 3 种变形来对待，具体如下：

1)同时读，同时写(CRCW)，多个处理内核可以同时读写同一个缓存块或主存块。

2)同时读，独占写(CREW)，多核个处理内核可以同时读同一个缓存块或主存块；但是在同一时刻，只有一个处理内核可以写一个缓存块或主存块。

3)独占读，独占写(EREW)，在任意一个时刻，只有一个处理内核可以读或写一个缓存块或主存块。

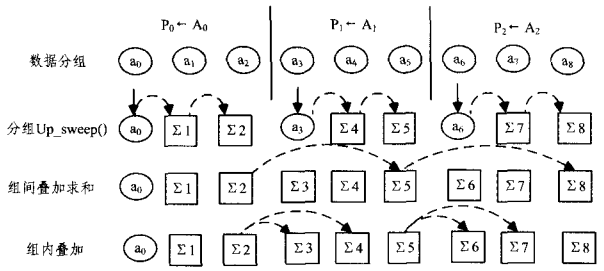
在 EREW 和 CREW 的情形下，处理器多个处理内核同时写一个缓存块或主存块可以用很多方法来解决，最简单的方法是串行排队写，这种方法中一次写的 I/O 复杂度为 O(P)。也可以用类似于二叉树的方式来解决，即利用辅助的缓存，在一次写中先两两归并写的结果，在下一次写中，同样地两两归并上一次写的结果，这样一次多个处理内核同时写同一个缓存块或内存块的 I/O 复杂度为 O($\log P$)。在 CRCW 情形下，多个处理内核同时写不同的缓存块或主存块是不会产生矛盾的，同时写一个缓存块或主存块，可以借鉴 CRCW PRAM 模型中的冲突解决机制。在本文中，着重考虑 CREW 情况。

5 稀疏线性代数实体的优化

13 种计算实体之一的稀疏线性代数计算中，使用最广泛的是稀疏矩阵-向量运算，在著名的数学库 BLAS、PETSc 中都有其具体的实现，大多数科学计算应用在计算的最后阶段都大量地进行这个运算。稀疏矩阵-向量运算的一般形式是：$Y=Y+A\times X$，其中 Y，X 是稠密向量，即向量中零元的个数极少，A 是稀疏矩阵，即矩阵中非零元的个数极少。计算的具体表达式为：$y_i=y_i+a_{ij}\times x_j$。从该计算过程可以看出，对于

任何一个 y_i，其计算不依赖于 Y 中任何其他的元素，可以独立地以任何顺序进行，这体现了这个运算内在的并行性。

为加速计算过程，在具体实现时通常采用基于扫描算法的向量运算。扫描算法[8-10]对一个数组 $X=\{x_0, x_1, \cdots, x_{n-1}\}$ 中的元素做二元关联运算，可以是任何二元数学运算。基于 UPMM 模型，本文设计了适合于多核处理器芯片架构的并行扫描算法：g-scan 算法，如图 5 所示。




```
B[P][m]
Group_prefix_sum(){
 Partition(A,B[])
 for all B[i], i→[0,P-1]
  in parallel do
   for j←0 to (m(i)-1),
    b[i][j]← b[i][j]+b[j]
 for i←0 to P-2 do
  b[i+1][m(i)-1]←b[i][m(i)-1]+b[i+1][m(i)-1]
 for i←1 to p-1 do
  in parallel do
   for j←0 to m(i)-2
    b[i][j]←b[i-1][m(i)-1]+b[i][j]
}
```

图 5　g-scan 扫描算法

在稀疏矩阵-向量运算过程中应用 g-scan 算法，可以极大地提高其在多核芯片架构的计算平台上的性能。新的计算过程如下所述：

1）首先计算一个 w 维的向量 V，计算方法是：$v_i=a_i\times x_{e_i}$，其中 a_i 是 A 中第 i 个非零元，e_i 是向量 E 中第 i 个元素，x_{e_i} 是向量 X 中第 e_i 个元素。

2）然后计算一个 w 维的标志位向量 F，计算方法是如果 E 中元素的索引在 T 向量中出现，则 $f=1$，否则 $f=0$，这是用来标记向量 V 中的分块的开始位置。

3）有向量 T，计算得出一个 w 维的向量 D，其中 $d_i=t_{i+1}-1$，这样可以做依据索引，在向量 V、T 上做前缀和扫描。

4）最后，对向量 Y 中的每一个元素 y_i，向量 V 中的每一个元素 v_{d_i} 求和。

以上为主要的计算步骤，算法的详细流程在后面的图 6 中给出。为了做前缀和，还必须要自定义一个⊕计算操作。本文中使用参考文献[11,12]给出的基于扫描算法做稀疏矩阵-向量运算的⊕操作，这个定义如图 6 所示。

$$v_j=\begin{cases}v_i+v_j, & f_j=0\\ v_j, & f_j=1\end{cases},\quad f_j=\begin{cases}f_i, & f_j=0\\ f_j, & f_j=1\end{cases}$$

$$A=\begin{bmatrix}a_0 & 0 & 0 & a_1 & 0\\ 0 & a_2 & 0 & 0 & 0\\ 0 & 0 & a_3 & 0 & 0\\ 0 & a_4 & 0 & a_5 & a_6\\ 0 & 0 & 0 & 0 & a_7\end{bmatrix}$$

$B=(a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7)$

$E=(0,3,1,2,1,3,4,4)$

$T=(0,2,3,4,7,8)$

SpMV

$$Y=\begin{bmatrix}y_0\\ y_1\\ y_2\\ y_3\\ y_4\end{bmatrix}+\begin{bmatrix}a_0 & 0 & 0 & a_1 & 0\\ 0 & a_2 & 0 & 0 & 0\\ 0 & 0 & a_3 & 0 & 0\\ 0 & a_4 & 0 & a_5 & a_6\\ 0 & 0 & 0 & 0 & a_7\end{bmatrix}\begin{bmatrix}x_0\\ x_1\\ x_2\\ x_3\\ x_4\end{bmatrix}$$

Computer V: $V=(a_0x_0, a_1x_3, a_2x_1, a_3x_2, a_4x_1, a_5x_3, a_6x_4, a_7x_4)$

Computer F: $F=(1,0,1,1,1,0,0,1)$

Computer D: $D=(1,2,3,6,7)$

Segmented V on V and F: $(a_0x_0, a_0x_0+a_1x_3, a_2x_1, a_3x_2, a_4x_1, a_4x_1+a_5x_3, a_6x_4, a_7x_4)$

scan on: 1　2　3　6　7

Computer Y: $Y=(y_0+a_0x_0+a_1x_3, y_1+a_2x_1, y_2+a_3x_2, y_4+a_4x_1+a_5x_3+a_6x_4, y_5+a_7x_4)$

图 6　基于 g-scan 算法的稀疏矩阵-向量运算

6　实验及优化后的程序性能分析

实验首先给出 g-scan 算法及采用 g-scan 算法的稀疏矩阵-向量运算算法的性能分析，实验采用多核芯片处理器模拟器进行。最后，使用实际的多核集群计算平台，分析优化了稀疏矩阵-向量运算算法在结构工程领域中广泛使用的开源有限元分析软件 OpenSees 的并行版本 OpenSeesSP 的性能。

1）g-scan 算法性能分析

采用模拟器的实验分别用 2 核、4 核、8 核、16 核进行实验分析；实际多核处理器的计算机为一片 IBM HS21 刀片服务器，其架构为 8 个处理核心，内存为 16G，操作系统为 Redhat 5.5，Gcc 为 4.1.2。用于对比组相连并行前缀和算法的是基于 EREW-PRAM 的前缀和算法 PRAM-scan[13] 以及串行前缀和算法。测试的性能指标为 wall clock，实验测试输入数据为单精度浮点数，测试结果如图 7 所示。

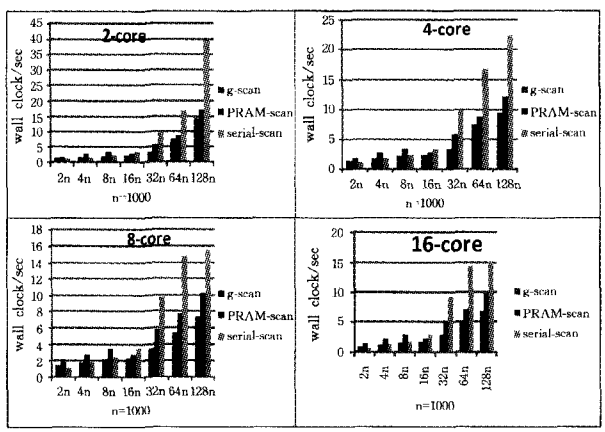


图 7　g-scan 算法的性能测试

由图 7 可以看出，基于组相连的并行前缀和算法具有很好的性能，而且根据处理器核的数目，保持了良好的可扩展性。在 4 个实验的开始阶段，由于输入数据较小，并行带来的开销使得两种并行算法都无法发挥其性能优势。

2）优化稀疏矩阵-向量运算算法性能分析

本实验对新设计的并行稀疏矩阵-向量运算算法进行性能测试，测试用的输入数据依据矩阵 A 的规模而定，初始规模为 1000×1000，矩阵中非零元的个数随矩阵规模增大而增大，逐次增加 10000，初始为 15000。实验选取 wall clock 作为评测指标，实验结果如图 8 所示。采用 Intel 的 MKL 来对比本文设计的稀疏矩阵-向量运算算法的实现。

由图 8 可以看出，新设计的稀疏矩阵-向量运算算法在性能上比 Intel MKL 的实现好一些，随着矩阵规模及非零元的个数增加，该算法的性能优势也越来越明显，并且性能一直很

稳定；而 MKl 库则随着矩阵规模和非零元数目的变化，表现出很不稳定的性能。

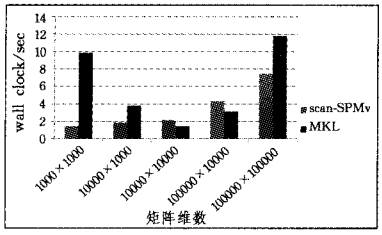


图 8　稀疏矩阵-向量运算算法性能评测

3)实际应用性能分析

本文实验结果分析所使用的硬件环境是一个有 8 个节点 IBM 高性能 HS21 刀片服务器的集群计算平台，集群中所有节点的处理器都是多核结构的，具体硬件系统的参数如下：

8 个 HS21 刀片服务器组成高性能计算集群；每个刀片服务器有两个 Intel(R) Xeon(R) CPU E5440 Quad core，即 8 个计算核心；每个刀片服务器配置 16G 内存；每个刀片服务器配置 146G 本地硬盘；每个刀片服务器配有两块网卡，分别是 1000M 和 10000M 以太网网卡；8 个刀片服务器通过 1000M/10000M 网络互联。该集群的软件环境为 RedHat Linux 5.4 32 位，openmpi 1.1.3，gcc 4.1.2，支持 OpenMP，图 9 给出了性能测试结果。

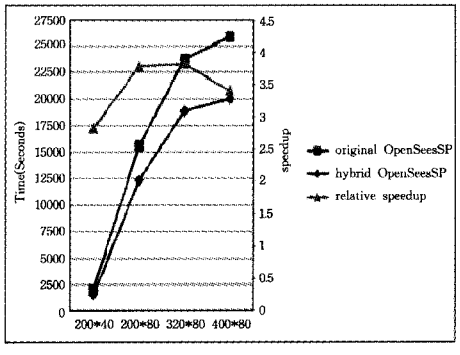


图 9　实际应用中并行稀疏矩阵-向量运算算法

从图 9 中可以看出，采用优化设计并行稀疏矩阵-向量运算算法的 OpenSeesSP 性能明显提升，这表明动力有限元分析时求解线性/非线性方程组是 OpenSees 的主要性能瓶颈，优化的并行稀疏矩阵-向量运算求解算法有效地提高了主进程方程组求解的效率。

结束语　本文从应用驱动的并行程序性能优化的角度出发，以分层并行计算模型思想为指导，首先提出了面向多核处理器芯片体系结构的并行算法设计模型，在此基础上对并行扫描算法进行分析优化，得出新的具有良好扩展性、高性能的 g-scan 算法。之后深入研究 13 种核心计算实体之一的稀疏线性代数计算实体，应用 g-scan 算法设计实现了新的稀疏矩阵-向量运算算法，并将其应用于结构工程领域中广泛使用的有限元分析，大大提升了其执行效率。

在以后的研究工作中可以考虑对其他计算实体进行性能分析及优化，以及在更多领域的应用中设计新的稀疏矩阵-向量运算算法。

参 考 文 献

[1] Agarwal A. The Why, Where and How of Multicore [C]// Workshop on EDGE Computing Using New Commodity Architectures (EDGE). 2006:96-100

[2] Asanovic K, Bodik R, Catanzaro B C, et al. The Landscape of Parallel Computing Research: A View from Berkeley[R]. UCB/EECS-2006-183. EECS Department, University of California, Berkeley, Dec. 2006:133-138

[3] Dubey P. A platform 2015 workload model: Recognition, mining and synthesis moves computers to the era of tera[R]. Technical report. Intel Corporation, 2005:99-102

[4] Sutter H. A fundamental turn toward concurrency in software [J]. Dr. Dobb's Journal, 2005, 30(3):16-22

[5] Gerber R, Bik A J C, Smith K B, et al. The Software Optimization Cookbook: High-performance Recipes for IA-32 Platforms [M]. Intel Press, 2006

[6] Colella P. Defining Software Requirements for Scientific Computing (presentation)[M]. 2004:28-49

[7] 陈国良，苗乾坤，孙广中，等. 分层并行计算模型[J]. 中国科学技术大学学报，2005，38(7)：54-57

[8] Hillis W D, Steele G L Jr. Data Parallel Algorithms[J]. Comm. ACM, 1986, 29(12):1170-1183

[9] Blelloch G E. Scans as Primitive Parallel Operations [J]. IEEE Trans. Computers, 1989, 38(11):1526-1538

[10] Blelloch G E. NESL: A Nested Data-Parallel Language (Version 2.6)[R]. CMU-CS-93-129. School of Computer Science, Carnegie Mellon Univ., 1993:338-349

[11] Sengupta S, Harris M, Garland M. Efficient Parallel Scan Algorithms for GPUs[R]. Technical report. NVIDIA Corporation, Dec. 2008:99-102

[12] Blelloch G E, Heroux M A, Zagha M. Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors [R]. CMU-CS-93-173. School of Computer Science, Carnegie Mellon Univ. and Cray Research, Inc., Aug. 1993:107-121

[13] Blelloch G E. Prefix Sums and Their Applications[R]. CMU-CS-90-190. School of Computer Science, Carnegie Mellon Univ, http://www.cs.cmu.edu/~guyb/papers/Ble93.pdf, Nov. 1990:23-30

(上接第 28 页)

[10] Christiaens F, Beyne E, Vandevelde B, et al, Compact transient thermal models for the polymer stud grid array[C]// Thermal Management of Electronic Systems. 1997

[11] Szekely V, Van Bien T. Fine structure of heat flow path in semiconductor devices: a measurement and identification method [J]. Solid-State Electronics, 1988, 31(9):1363-1368

[12] Yang Y, Master R, Tosaya E, et al. Transient Frequency-Domain Thermal Measurements with Applications to Electronic Packaging [J]. IEEE Trans: Components and Packaging Technologies, 2012, 2(3):448-456

[13] Gerstenmair Y C, Wachutka G. Calculation of the temperature development in electronic systems by convolution integrals [C]// Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 16th Annual IEEE. 2000

[14] Schweitzer D. A fast algorithm for thermal transient multisource simulation using interpolated Zth functions [J]. IEEE Trans: Components and Packaging Technologies, 2009, 32(2):478-483