面向高性能数值计算的并行计算模型 DRAM(h)

张云泉

(中国科学院软件研究所并行计算实验室 北京 100080)
(中国科学院计算机科学重点实验室 北京 100080)

摘 要 提出了一个基于存储层次的新并行计算模型 DRAM(h),并在该模型下对两个经典并行数值计算算法的不同实现形式:四种形式并行下三角方程求解(PTRS)和六种形式无列选主元并行 LU 分解(PLU),进行了分析.模型分析表明,具有近乎相同时间和空间复杂性的同一算法不同实现形式,在该模型下会有完全不同的存储复杂度.作者在日立公司 SR2201 MPP 并行机、曙光 3000 超级服务器和中国科学院科学与工程计算国家重点实验室(LSEC)的 128 节点 Linux Cluster 等三种并行计算平台上对模型分析结果进行了实验验证.结果表明,该模型分析在绝大多数情况下都能较好地与不同实验平台上的实验结果吻合.个别出现偏差的分析结果,在根据计算平台的存储层次特点修改模型分析的假定后,也能够进行解释.这说明了该模型对不同形式的算法实现进行存储访问模式区分的有效性.对在计算模型中加入指令/线程级并行的可行性和方法的研究是下一步的工作.

关键词 高性能数值计算;存储复杂性;并行计算模型.
中图法分类号 TP311

DRAM(h): A Parallel Computation Model for High Performance Numerical Computing

ZHANG Yun-Quan

(*Laboratory of Parallel Computing, Institute of Software, Chinese Academy of Sciences, Beijing* 100080)
(*Key Laboratory of Computer Science, Chinese Academy of Sciences, Beijing* 100080)

Abstract In this paper, a new parallel computation model DRAM(h), which has h-level memory hierarchies, was proposed. With this new model, we performed memory complexity analysis on different implementation forms of two classical parallel numerical linear algebra algorithms, i.e., four forms of parallel lower triangular solver(PTRS) and six forms of parallel LU factorization without column pivoting(PLU). Under DRAM(h) model, we find out that the different implementation forms of the same algorithm can have different memory complexity though they have almost the same time and space complexity under traditional RAM model. Finally, we validate our analytical results with experimental results on three parallel computing platforms, i.e., HITACHI SR2201, DAWNING3000 and 128 node LSEC Linux Cluster. In most cases, our model's analytical results match well with experimental results, which indicates the effectiveness of our new model on clarifying the different memory access pattern of various forms of the same algorithm. Some mismatches can be well explained through slightly modification on model analysis assumptions according to platform memory hierarchy features.

Keywords high performance numerical computing; memory complexity; parallel computation model

收稿日期:2002-03-12;修改稿收到日期:2003-01-06.本课题得到国家"九七三"重点基础研究发展规划项目《大规模科学计算研究》(G1999032805)、国家自然科学基金资助项目(60303020)、中国科学院计算机科学重点实验室青年科学基金(SYSQN0101)、中国科学院计算技术研究所知识创新课题和中国科学院软件研究所培育项目(CXK25628)的部分资助.**张云泉**,男,1973 年生,博士,副研究员,主要研究方向为大型并行数值软件、并行程序设计和性能评价、并行计算模型. E-mail: zyq@mail.rdcps.ac.cn.

1 引　言

过去,由于处理器的浮点处理单元速度往往较慢,不受访存带宽的限制,一个应用程序的性能经常用每秒百万浮点操作(Mflops)来表示.而现在,具有超标量流水、指令/线程级并行且主频几个 GHz 这样性能的浮点处理部件的速度已远远超出了存储访问的速度,这使得目前所声称的 MFLOPS 性能数据只代表数据全部在一级高速缓存中命中甚至寄存器中的情形,而这种速度在程序进行大量存储访问时是不可能达到的[5].

这种从浮点操作比访存开销大或近乎等价到小几个数量级的转变[1],使得传统的以计算步数为核心的 RAM 计算模型失去了精确分析算法优劣的部分能力.现在,如果只谈论一个算法的浮点操作次数,很难知道它在一个机器上会有多快.我们更不能用 RAM 模型对具有同样的浮点操作数却有不同访存模式的算法进行性能优劣的评价.它们最后的性能往往依赖于具体的实现技术.一个经过特别优化的算法实现的性能比普通未优化的版本往往高出不止一个数量级.

造成这种现象的原因主要是现代处理器为了平衡越来越严重的处理速度与存储访问速度之间的失配而采用的多级高速缓存设计[1].若算法的实现没有考虑到这些多级高速缓存,当然不能很好地利用,从而使得程序的性能由速度最慢的存储层决定.

由此,单位时间存储访问的 RAM 模型假设[8]变得不适用了.一度被忽视的存储访问复杂性[13,15]在算法设计和实现中被重新认识.提出一个在存储层次方面有所考虑的计算模型,成为一个迫切需要解决的问题.

然而,绝大多数现有的并行计算模型[4,6,7]忽略了本地的存储层次及其数据重用,或做了太理想化的假定,如本地模型是 RAM 模型[8].这些模型过于强调简单性和广泛可用性,从而导致分析的精度过低[12,16].但是,在实际算法设计和实现中分析精度很重要,即使常数倍的性能改进也是很有用的.这些简化的和广泛可用的并行模型可以在算法设计的最初阶段采用,但却不适用于算法实现和性能调试阶段.对本地存储层次的忽视导致这些模型在分析涉及大数据量操作,尤其是数值线性代数计算的算法时,精度明显不能满足要求[16].在这些领域最有用的模型应是那些能够较容易和精确地对本地存储层次中的数据移动进行建模的模型.虽然通用性是许多并行模型的目标,但过于强调通用性,反而处处不适用.我们要强调的是:传统的 RAM 模型需要重新考虑其对访存开销的假定,一个对高性能并行数值计算有用的并行计算模型必须建立在改进的,能够利用存储层次的新 RAM 模型之上.

在本文中我们将提出一个新的对存储层次有充分考虑的并行计算模型,可用于在高性能数值计算算法设计和实现时的精确分析.该模型主要基于我们以前提出的存储复杂性[13]概念,我们称该模型为有 h-层存储层次的分布式 RAM 模型——DRAM(h).

本文提出的 DRAM(h)模型是一个分布式存储的并行计算模型,它把传统的基于单位访存开销假定的 RAM 模型转换成访存开销随数据在存储层次中的位置,所访问的时间和访问模式而变化的 RAM 模型.远地内存访问在 DRAM(h)被看作存储层次的另一层,但其开销用 LogP 模型的方法进行建模并根据消息的长短看作连续或不连续访问.本文的最初研究结果发表在文献[17]中.

2 串行计算模型 RAM(h)

在本节,我们先介绍 RAM(h) (RAM with h-level Memory Hierarchies)模型的基本定义和概念.

在过去的十几年里,由于越来越大的浮点运算速度和访存速度失配,导致单位时间存储访问开销假定有些过时,人们提出了几个能够弥补这一缺陷的计算模型. Hierarchy Memory Model 模型[10]对访问地址为 a 的存储单元,假定其开销为 $f(a)$,其 $f(a)$是地址 a 的单调上升函数;而 Block Transfer (BT)[11]模型又增加了对块传递的假定:对长度为 b,起始地址为 a 的块的访问,其开销变为 $f(a)+b-1$ 而不是$\sum_{i=a}^{a+b-1}f(i)$.

以上两个模型都根据所访问的数据地址来决定相应的开销,这在诸如磁带之类的顺序访问存储介质上是适用的,但对于随机访问的高速缓存、DRAM 和硬盘是不适用的.在这些介质中,访问开销主要由其所在的存储层次数决定,而具体的访存地址不占主导地位.每一层都有其相对固定的访问开销. UMH 模型[9]就根据这一现象假定其存储访问开销为存储层次数的函数 $f(k)$,其中 k 是存储层次数.但该模型忽略了存储访问中的时间和空间局部性,因为作者认为这些目标太模糊且会使模型更复杂.在 UMH 模型中对这些特点的忽视导致在 UMH 模型下的算法总是重复地从存储层次的高层向底层(靠近处理器)传递本来可以在底层重复利用的数据,从而花费更多的数据传递时间,影响实际

计算的性能;同时对数据预取和计算与数据取重叠提出了更高的要求;而且,对存储访问开销函数统一和简单的假定使得该模型更不现实,现实中的存储层次访问开销不会像假定的那样理想;使得 UMH 模型更加不现实的另一个特点是它过于理想的存储访问与计算重叠假定.在这种理想的存储访问调度中,几乎所有的存储访问开销可以通过本地计算与存储访问的重叠和所有总线上的并发传递来隐藏,在现代处理器设计中这是不容易达到的要求[14].能够满足所需要的处理速度和统一数据传递开销函数,且具有强大的各级总线并发传输功能的处理器设计尚不成熟.一个可能的处理器是 SR2201 的计算节点.在这些节点上通过改装增加了预取和延后存的功能.有些经过精心调试(循环交换、循环展开、Tilling 等)的程序,几乎可以隐藏所有的数据访问开销.但该处理器也只能实现从主存直接到寄存器的预取和延后存的功能(Cache Bypassing),而不是所有的存储层次.然而 UMH 模型的设计者表达了在所有的存储层次都提供对预取和延后存功能支持的愿望.

基于以上观察,我们在本文提出一个现实的计算模型 RAM(h).很明显的,RAM(h)模型将建立在 RAM 模型之上,但其在存储开销方面作了扩展.我们把 RAM 模型中只有一层单位时间开销的存储扩展为 h 层存储,离处理器越远其容量越大,存储访问速度越慢;离处理器越近其容量越小,存储访问速度越快.算法所需要的数据假定先存在离处理器最远的存储层次,只在计算需要的时候才逐层取到离处理器最近的一层.在此过程中,每一层将会有该数据的一个拷贝.这些拷贝会保持在各存储层次中,直到计算结束或由于映射冲突和容量不足被别的数据替换出去为止.随后若一个曾被访问过的数据再次被访问,该数据将被从其所驻留的离处理器最近的层次中取到,而不是像 UMH 模型中的总是从最远的一层取到.每次存储访问的开销由数据所驻留的离处理器最近的层次数和数据重用的能力决定.数据所在的层次越低,开销越小;重用越好,开销也越小.数据在该模型中将以块传递,对某一分块中任一数据单元的访问将导致该数据所在的整个数据块被从最远端取到离处理器最近的层次.但是每次所传递的分块大小会逐层减少或不变.对存储层次为 h、长度为 l 的数据块的第一次访问的开销为 $\left\lceil \frac{l}{b(h-1)} \right\rceil c(h)+\left\lceil \frac{l}{b(h-2)} \right\rceil c(h-1)+\cdots+\left\lceil \frac{l}{b(0)} \right\rceil c(1)+l\times c(0)$ 而不是 $lf(h)$,其中的 l 是块的长度,$b(i)$ 是第 i 层分块的大小,$c(i)$ 是第 $(i-1)$ 层从第 i 层读/写单位数据块 $b(i-1)$ 的开销函数;$c(0)$ 是进行计算时数据读/写的时间,一般不计算在模型访存开销中,我们设为零,且 $f(i)=\sum_{j=1}^{i}c(j)$.每一层的存储访问开销函数是从该层读/写一个大小为 b_0 的第 0 层块所需的开销.因而我们总有对 $i>1$,开销函数满足 $f(i)>f(i-1)$ 成立,这由 $f(i)=f(i-1)+c(i)$,$c(i)>0$ 决定.一个简单的情形是 $l=b(1)$,由于我们有 $b(i-1)\leq b(i)$ 成立,那么从第 h 层对该块大小的数据连续取的开销为 $f(h)+\left(\left\lceil \frac{b(1)}{b(0)} \right\rceil -1\right)\times c(1)$,而这正是现有处理器中高速缓存行存取的开销.

对于数据存取和计算之间的重叠,RAM(h)模型并不进行特别的要求.这是因为确定二者的确切重叠情况比较复杂和困难.本文提出该模型的目的之一是鼓励用户对数据在不同存储层次间的流动进行优化,并尽量减少数据移动开销在整个计算中所占的比重.如果能够通过重叠把一部分开销隐藏起来,是有意义的.而且,在 RAM(h)模型中,对存储访问的读和写操作并不加以区分,虽然这是以牺牲模型分析的精度为代价,但却可以简化分析复杂性.

一旦数据被取入第零层,我们即认为数据访问的阶段结束,开始进入计算阶段.在第零层对数据进行的操作其存储访问开销假定为零,其实际的开销将记入浮点运算的时间.

为了对不同的算法或算法的不同实现进行评价,我们在文献[13]提出了存储复杂性的概念并具体给出了存储复杂性的评价尺度-存储复杂度的定义和实际验证.我们提出算法的复杂性包括计算复杂性和存储复杂性,其中的计算复杂性包含传统的时间复杂性和空间复杂性,是一个算法的静态属性,不会随着算法的不同实现形式有太大变化;而存储复杂性是一个算法的动态属性,会随着算法实现形式和实现平台的不同而出现较大变化.我们定义算法的存储复杂性是并行或串行算法的数据分布和数据访问模式与程序所运行的计算平台存储层次之间交互所形成的、算法所访问的数据的分布状态以及由此决定的算法访存开销.

存储复杂性分析是对一数值程序中占主导地位的存储访问模式特征的提取.数据在程序运行过程中,会由于访问时间、与其它数据先后顺序和访问模式等的不同,在存储层次中处于不同的状态(k)和在该状态下不同的访问次数(f),从而导致不同的数据存储复杂度 $d(k)$(存储访问开销).设 $d_i(k)$ 为第 i

个数据在状态 k 的存储复杂度,$f_i(k)$为数据 i 在状态 k 下被访问的次数,则算法的存储复杂度 $m(n)$定义为

$$m(n)=\frac{\sum_{i=1}^{p(n)}\sum_{k=1}^{h}f_i(k)d_i(k)}{\sum_{i=1}^{p(n)}\sum_{k=1}^{h}f_i(k)},$$

其中,$p(n)$是问题规模 n 的函数,表示在算法中用到的所有不同数据的数目,h 是存储层次的深度. 为了对算法的不同实现形式进行统一的比较,我们定义数据在不同存储层次之间的总移动次数与总浮点操作次数的比 $q(n)$为

$$q(n)=\frac{\sum_{i=1}^{p(n)}\sum_{k=1}^{h}f_i(k)}{C(n)},$$

其中 $C(n)$为算法的时间复杂性,一般用浮点操作次数代替,那么 $m(n)$和 $q(n)$的乘积 T_{fp}表示算法的每次浮点操作中花在数据移动上的时间,即 $T_{fp}=m(n)q(n)$(也叫做规范化存储复杂度). 我们在该文献中给出了存储复杂度的具体分析方法,这里限于篇幅,不再详细介绍.

对该文中的分析方法略作调整,就可用于 RAM(h)模型的分析之中. 利用存储复杂度进行分析,原来的算法分析除了计算复杂度(含时间和空间复杂性)之外,会另有一个 T_{fp}给出. 通过给出的存储复杂度,我们可以大致知道该算法所进行的数据移动所集中的存储层次和访存模式,并对其可能达到的性能有一个量的判断.

存储复杂度可以有效区分同一算法的不同实现形式或同一算法实现形式在不同计算平台上的区别. 一般的,这些实现形式会有同样或接近的时间复杂度和空间复杂度,在传统的分析方式下,我们是无法分辨出各形式之间的优劣的. 而在 RAM(h)模型下进行了存储复杂度分析后,我们可以进一步对这些不同的形式进行区分,从而为选取性能更高的实现形式提供指导.

3　并行计算模型 DRAM(h)

在 RAM(h)的基础上,我们可以建立并行版本的模型 Distributed RAM(h),简称 DRAM(h). 该模型以 RAM(h)为本地模型,并通过互连网把 p 个这样的处理单元连接起来,而互连网的具体拓扑结构不在该模型关心的范围内[6]. 每一个处理单元都有自己的本地存储层次,各处理器间通过点到点的消息传递交换信息. 在我们的并行模型中,把消息传递统一地看成另一层存储层次访问. 我们需要的只是下层通信的几个参数如短消息开销函数、长消息开销函数等. 发送操作看作存储访问中的写操作,接收操作看作存储访问的读操作. 一般的由于发送和接收总是匹配的,且其时间开销在阻塞通信时是一样的,因而在发送和接收时双方都将此看作一次存储访问操作且开销相同. 在我们的模型中即假定消息传递的原语是阻塞的发送和接收. 其开销将用 LogP 模型[6]的分析方法提供. 其它的复杂集合(Collective)通信函数如广播、归约和同步等可以通过点到点发送和接收实现,因而其开销函数也可以通过基本原语的分析得到. 在并行模型中,我们不准备把非阻塞发送和接收原语包含在模型中,因为一方面这会破坏我们的非重叠存储访问模型假定;另一方面,决定什么时候在并行算法中使用非阻塞消息传递也是比较困难的且不是所有的平台都支持非阻塞消息传递. 阻塞原语给出的是完成消息传递任务所需时间的上限. 我们模型的主要目的是分析并行程序的访存开销,并鼓励尽量通过连续的存储访问和重用降低存储访问的开销,而不是去隐藏这些开销. 若用户的程序在我们的模型下有好的存储访问行为,那么很自然的一些延迟隐藏技术能够很好地使用,并进一步降低存储访问开销在总开销中所占的比例.

我们把集合消息传递看作并发的存储访问. 为了对并发访问时的拥塞和等待开销进行建模,我们在单次传递开销函数之前乘以一个以参与通信的并发进程数为参数的函数. 而该函数由集合消息传递的具体实现方式决定. 实际上,在分析每次点到点消息传递开销的时间时,我们可以利用 LogP 模型来进行. 但是,对于数值线性代数计算来说,LogP 模型中的所有参数并不是都有用. 其中的参数 g 对性能的影响不大,这是因为在数值线性代数计算中的通信频率相对较低. 如果一个数值线性代数的并行程序有极高频率的通信,那该程序的性能肯定不会很好,也不会是一个好的实现形式. 因而 L 和 o 参数可以如下包含进我们的模型:

$$t_s=\frac{\alpha+\beta l}{l},$$

其中 $\alpha=o$,$\beta=\frac{L}{l}$,l 是消息长度.

若 l 非常小,则 o 的部分会占主导地位,此时 t_s 表示为 T_s,可以用 $T_s=\frac{o}{l}$近似. 当消息长度很长时,L 部分为主,那么 t_s可以用 $t_s=\frac{L}{l}=\beta$近似.

因而在这样的公式下,我们的模型鼓励长的消息传递并给长消息传递低的开销 β;对短的消息传递,软件开销部分会主导每一字节的消息传递. 这与

通常的广泛接受的并行程序实现经验是一致的.

为了区分并行算法的不同实现形式,我们的并行模型的存储复杂度分析以串行模型分析方法为基础,并把它扩展到包含远地存储的访问开销[13,15].

4 模型分析

在本节的模型分析中,我们用并行 DRAM(h)模型来分析 4 种不同形式的并行下三角方程求解(PTRS)和 6 种不同形式的并行无列选主元 LU 分解(PLU).由于篇幅的限制,我们没有给出全部算法的分析过程和算法描述,我们以 PTRS 的 $ij-c$ 形式和 PLU 的 $kij-r$ 形式为例子演示应用本文提出的模型分析过程.关于其它形式的具体算法描述和分析过程可参见文献[3,15].本文中用到的各参数定义见表 1.

PTRS 的 $ij-c$ 算法形式和 PLU 的 $kij-r$ 算法形式分别如图 1(a)和(b)所示.

```
for i=1 to n
  t=0;
  for j=1 to i-1
    if (j in mycols) then
      t=t+x(j)l(i,j);
    endif
  end
  MPI_Reduce(s,t);
  if (i in mycols) then
    x(i)=(b(i)-s)/l(i,i);
  endif
end
```

(a) PTRS $ij-c$ 算法形式

```
left=(MyId-1+p)%p;
right=(MyId+1)%p;
for k=1 to n-1
  if (k in myrows) then
    MPI_Send(a(k,1), n, right);
  else
    MPI_Recv(a(k,1), n, left);
    if (right!=k%P)
      MPI_Send(a(k,1), n, right);
    endif
  for i=k+1 to n
    if (i in myrows){
      l(i,k)=a(i,k)/a(k,k);
      for j=k+1 to n
        a(i,j)=a(i,j)-l(i,k)a(k,j);
      end}
    end
end
```

(b) PLU $kij-r$ 算法形式

图 1 PTRS 的 $ij-c$ 算法形式和 PLU 的 $kij-r$ 算法形式

上面算法形式中采用的都是一维(按行或按列)的数据分配方式,因此,$myrows$ 或者 $mycols$ 表示一个进程所分配到的矩阵行或列的集合.$MyId$ 是一个进程在整个一行或一列参加计算的进程中的排列序号.在下面的分析中我们假定数组以行为主存储,进程个数为 p,问题规模 n 比较大,且 $p \ll n$.

PTRS $ij-c$ 形式采用的是列卷帘数据分布方式,也叫标量积形式.在该算法形式中,我们看到有一个消息传递语句,$MPI\_Reduce(s, t)$;两个计算语句,$t=t+x(j)l(i,j)$和 $x(i)=(b(i)-s)/l(i,i)$.针对每个计算进程,应用 $DRAM(h)$模型针对上述三个语句的分析如下:

消息传递语句 MPI_Reduce 是集合(Collective)通信语句,目前比较流行的实现算法为二叉树算法.对 p 个进程来说,相当于 $\log_2 p$ 次 MPI_Send/MPI_Recv 调用;由于每次只传递一个双精度浮点数,因此是短消息,在模型中的数据复杂度为 T_s,每个进程对该语句的调用次数为 n,因此总的远程通信数据复杂度约为 $n(\log_2 p)T_s$;

计算语句 $t=t+x(j)l(i,j)$ 的总调用次数约为 $\frac{n(n+1)}{2p}$,t 可以认为是寄存器变量,其数据复杂度为 0;$x(j)$由于在外层循环中被重复以步长 1 连续访问,且规模为 n,可以假定全部在 $L1$ 高速缓存中命中,其数据复杂度为 t_{L1};$l(i,j)$虽然是步长为 1 的连续访问,但由于没有重用,假定其全部在主存中,其数据复杂度为 t_{1m};因此该语句调用的总数据复杂度为 $\frac{n(n+1)}{2p}(t_{L1}+t_{1m})$;

计算语句 $x(i)=(b(i)-s)/l(i,i)$ 的总调用次数约为 $\frac{n}{p}$,$x(i)$和 $b(i)$由于在循环中被以步长 1 连续调用,但没有重用,且规模为 n,可以假定全部在内存中命中,其数据复杂度为 t_{1m};s 可以认为是 $L1$ 高速缓存命中,其数据复杂度为 t_{L1};$l(i,i)$是步长为 n 的无重用非连续存储访问,假定其全部在主存中命中,其数据复杂度为 t_m;因此,该语句调用的总数

据复杂度为$\frac{n}{p}(t_{L1}+2t_{1m}+t_m)$;

根据存储复杂度的定义,PTRS $ij-c$ 每个进程的存储复杂度分析结果应为

$$m(n,p)=\frac{\frac{n}{p}(t_{L1}+2t_{1m}+t_m)+\frac{n(n+1)}{2p}(t_{L1}+t_{1m})+n(\log_2 p)T_s}{\frac{4n}{p}+\frac{n(n+1)}{p}+n(\log_2 p)}$$
$$=\frac{(n+3)t_{L1}+(n+5)t_{1m}+2t_m+2p(\log_2 p)T_s}{2n+10+2p(\log_2 p)}.$$

由于该算法的计算复杂度为 n^2+3n,总的浮点数存储访问次数为 $n^2+5n+np(\log_2 p)$,因此在 n 很大的情况下,$q=\frac{n^2+5n+np(\log_2 p)}{n^2+3n}\approx 1+\frac{p(\log_2 p)}{n}\approx 1$,那么

$$T_{fp}=m(n,p)\times q$$
$$\approx\frac{(n+3)t_{L1}+(n+5)t_{1m}+2t_m+2p(\log_2 p)T_s}{2n+10+2p(\log_2 p)}.$$

PLU $kij-r$ 形式采用的是行卷帘数据分布方式,很明显的,本地计算的形式将采用串行 LU 的 kij 实现形式,而在广播时的通信拓扑为一个流水环.在该算法形式中,我们看到有两个成对的消息传递语句,$MPI_Send()$和 $MPI\_Recv()$;有两个主要的计算语句,$l(i,k)=a(i,k)/a(k,k)$和 $a(i,j)=a(i,j)-l(i,k)a(k,j)$.应用 $DRAM(h)$模型,针对上述四个语句的分析如下:

消息传递语句 MPI_Send 和 MPI_Recv 在该算法中是成对,且我们的模型中假定读和写的开销一样,由于其消息长度为 n,因此是长消息传递,其数据复杂度为 t_s,调用次数为 $2n(n-1)$,该语句调用的总数据复杂度为 $2n(n-1)t_s$;

计算语句 $l(i,k)=a(i,k)/a(k,k)$ 的总调用次数约为$\frac{n(n+1)}{2p}$,$l(i,k)$和 $a(i,k)$的访问模式相同,都是无重用非连续存储访问,因此其数据复杂度为 t_m.$a(k,k)$由于下标在最内层循环不变,可以看作寄存器变量,数据复杂度为零.该语句调用的总数据复杂度为$\frac{n(n+1)}{p}t_m$;

计算语句 $a(i,j)=a(i,j)-l(i,k)a(k,j)$ 的总调用次数约为$\frac{(n-1)n(2n-1)}{6p}$,其中的 $l(i,k)$由于其下标在最内层循环中不变,可以作为寄存器变量,其数据复杂度为零;$a(i,j)$是步长为 1 的连续存储访问,在最外层循环 k 中虽然有重用,但由于其数据量比较大,不能保证在下次访问时的高速缓存命中,因此其第一次读访问的数据复杂度为 t_{1m};由于第二次写访问很近,因此数据复杂度为 t_{L1};$a(k,j)$是连续存储访问,且可以在 i 层循环重用,虽然数据量不大,但由于和数据量很大的 $a(i,j)$在一起,不能保证在 $L1$ 高速缓存命中,因此数据复杂度为 t_{1L2}.由此,该语句调用的总数据复杂度为$\frac{(n-1)n(2n-1)}{6p}(t_{L1}+t_{1L2}+t_{1m})$;

根据存储复杂度的定义,$PLU\ kij-r$ 每个进程的存储复杂度分析结果为

$$m(n,p)=\frac{\frac{(n-1)n(2n-1)}{6p}(t_{L1}+t_{1L2}+t_{1m})+\frac{n(n+1)}{p}t_m+2n(n-1)t_s}{\frac{(n-1)n(2n-1)}{2p}+\frac{n(n+1)}{p}+2n(n-1)}$$
$$\approx\frac{2t_s+\frac{1}{p}t_m+\frac{n}{3p}(t_{L1}+t_{1L2}+t_{1m})}{2+\frac{n+1}{p}}\approx\frac{2t_s+\frac{n}{3p}(t_{L1}+t_{1L2}+t_{1m})}{2+\frac{n}{p}}.$$
$$=\frac{n(t_{L1}+t_{1L2}+t_{1m})+6pt_s}{4p+2n}.$$

由于该算法的计算复杂度约为$\frac{2}{3}n^3$,总的浮点数存储访问次数为$\frac{(n-1)n(2n-1)}{2}+n(n+1)+2pn(n-1)\approx n^3$,因此在 n 很大的情况下,$q\approx\frac{3}{2}$.则

$$T_{fp}=m(n,p)\times q\approx\frac{3t_s+\frac{n}{2p}(t_{L1}+t_{1L2}+t_{1m})}{2+\frac{n}{p}}$$

通过分析,我们得到如表 2 和表 3 所示的结果.表中的算法形式列中,X-Y-Z 中的 X 表示算法形式中的循环下标顺序的字符串,最左边的是最外层循环;Y 是表示数据分布方式的字符串(行(r)或列(c)分布);如果有 Z 则表示循环数据分配方式(cyclic).

表 1 本文模型分析中用到的参数及其定义

参数	定义	参数	定义
p	并行进程数	t_{L2}	非连续(步长大于缓存行长)访问二级高速缓存单位数据开销(ns)
n	计算问题的规模	t_{1m}	单位步长连续访问主存的单位数据开销(ns)
q	算法实现形式中总浮点数据存储访问次数与总浮点操作数的比值	t_m	非连续访问主存的单位数据开销(ns)
T_{fp}	每次浮点操作存储访问开销[13]	t_s	长消息传递单位数据开销(ns)
t_{L1}	访问一级高速缓存单位数据开销(ns)	T_s	短消息传递单位数据开销(ns)
t_{1L2}	单位步长连续访问二级高速缓存单位数据开销(ns)		

表 2 不同形式的并行下三角方程求解算法的 T_{fp} 分析结果(数组存储行为主)

算法形式	q	T_{fp}
$ij-c$	$1+\frac{p\log_2 p}{n}$	$\frac{(n+3)t_{L1}+(n+5)t_{1m}+2t_m+2p\log_2(p)T_s}{2n+10+2p\log_2(p)}$
$ji-r$	$\frac{3}{2}+\frac{p\log_2 p}{n}$	$\frac{3(n+1)t_{L1}+3(n+5)t_{1m}+3(n+3)t_m+6p\log_2(p)T_s}{6(n+3)+4p\log_2(p)}$
$ij-r-cyc$	$1+\frac{4p}{n}$	$\frac{(n+3)t_{L1}+(4p+n-1)t_{1m}+2t_m+4(p-1)T_s}{2n+8p+8}$
$ji-c-cyc$	$\frac{3}{2}+\frac{2p}{n}$	$\frac{3(n+7)t_{1m}+3(n+1)(t_{L1}+t_m)+12(p-1)T_s}{6n+8p+10}$

表 3 6 种不同形式的并行无列选主元 LU 分解算法的 T_{fp} 分析结果(数组存储行为主)

算法形式	q	T_{fp}
$kij-r$	$\frac{3}{2}$	$\frac{n(t_{L1}+t_{1L2}+t_{1m})+6pt_s}{4p+2n}$
$kij-c$	$\frac{3}{2}$	$\frac{n(t_{L1}+t_{1L2}+t_{1m})+6pt_s}{4p+2n}$
$kji-r$	$\frac{3}{2}$	$\frac{n(t_{L1}+t_{L2}+t_m)+6pt_s}{4p+2n}$
$ikj-c$	$\frac{3}{2}$	$\frac{n(2t_{L1}+t_{1m})+3pT_s}{2p+2n}$
$jki-r$	$\frac{3}{2}$	$\frac{n(t_{L1}+t_{L2}+t_m)+3pT_s}{2p+2n}$
$jik-r$	1	$\frac{n(t_{1m}+t_m)+3pT_s}{3p+2n}$

5 实验验证

本节我们用 4 种形式的并行下三角方程求解(PTRS)算法和 6 种形式的并行无列选主元 LU 分解(PLU)算法实现在日立 SR2201 MPP 并行机、曙光 3000 超级服务器和中国科学院 LSEC 的 128 节点 LinuxCluster等三种平台上对本文提出的并行计算模型 DRAM(h)的分析结果进行实验验证. 本地计算部分,由于在 SR2201 上主要的计算部分没有伪向量化,因而相应的存储层次参数用文献[13]中 PA-RISC 150(O(2))列的数据. 为了分析的方便,我们在表 4 中把 DRAM(h)模型所用到的所有参数的实际测量值给出. 如无特别说明,消息传递采用的是 MPI. 其中 L 是短消息的长度(双精度浮点数个数).

SR2201 采用经过改装的 HP PA-RISC HARP 1E 150Mhz 处理器,16KB L1 高速缓存,512KB 片外高速缓存,每个节点内存 256MB,节点间采用三维交叉开关互连,通信带宽单向 300MB/s. 曙光 3000 采用 IBM Power3-II 375MHz 处理器,64KB L1 非阻塞数据缓存(128B 行长,128 路组相联),4MB(最大 8MB) L2 缓存,每一计算节点是 4 路 SMP, 2GB 内存. 采用 Myrinet 进行节点间互连. LSEC 128 节点 Linux Cluster 采用 PIII 550E 处理器,32KB L1 缓存,256KB 片上全速 L2 缓存. 每个节点 512MB 内存,节点间用 100Mb 快速以太网连接.

表 4 各实验平台上 DRAM(h)模型所需的存储层次参数 (单位:ns)

参数	t_{L1}	t_{1L2}	t_{L2}	t_{1m}	t_m	t_s	T_s
SR2201(-O2)	8.0	18.0	64.0	30.0	531.0	36.8	26000/L
曙光 3000(-O3)	4.1	16.1	68.1	64.1	336.0	73.3	22990/L
LSEC Linux 机群(-O3)	3.67	6.95	10.05	41.0	62.0	720.0	102410/L

5.1 并行下三角方程求解

根据表 2 中的分析结果和表 4 中的模型参数,我们在图 2,图 3 和图 4 中分别给出了并行下三角方程求解在上述三个实验平台上的 T_{fp} 值随问题规模(1000～25000)的变化情况(图(a))和相应的实验结果(图(b)). 由于在问题规模较小时的分析结果太大且我们的分析是为了问题规模较大的情况,为了分析的方便,分析结果没有从 1000 的问题规模开始给出.

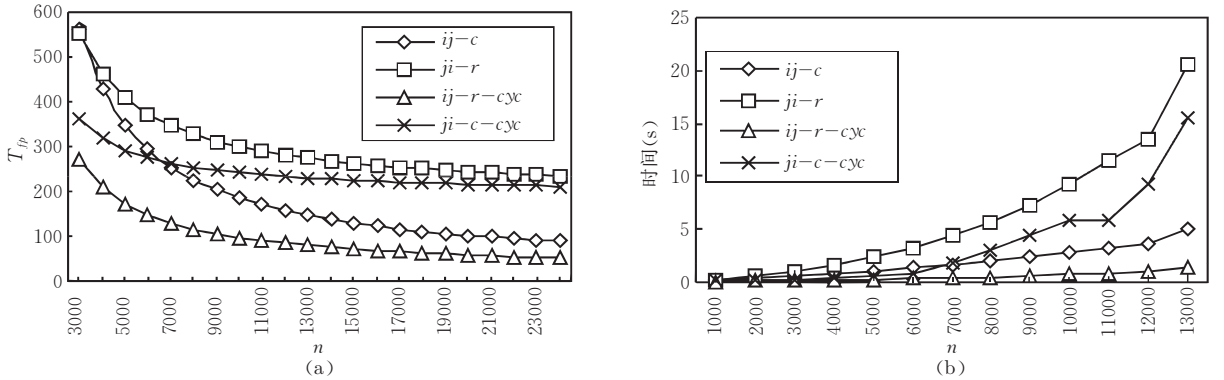

图 2　SR2201 上不同形式的 PTRS 在 $p=16$ 时的 T_{fp} 值和相应的实验结果

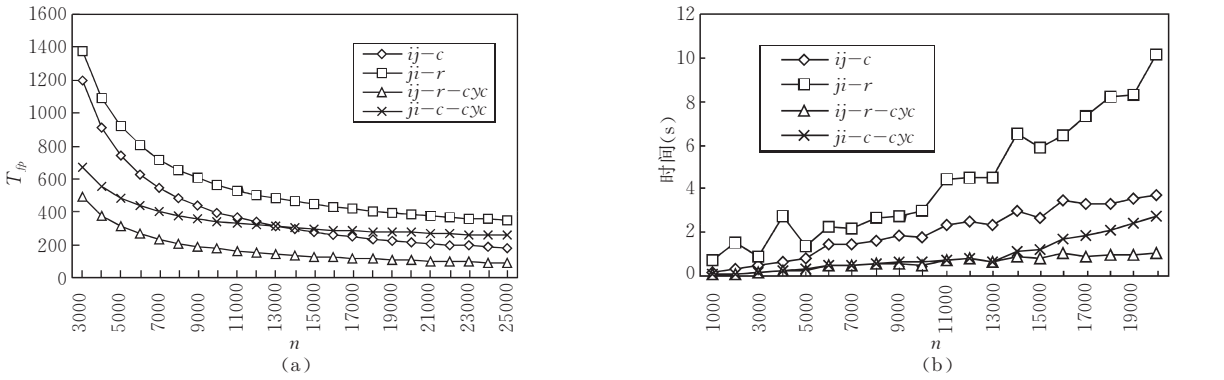

图 3　曙光 3000 上不同形式的 PTRS 在 $p=32$ 时的 T_{fp} 值和相应的实验结果

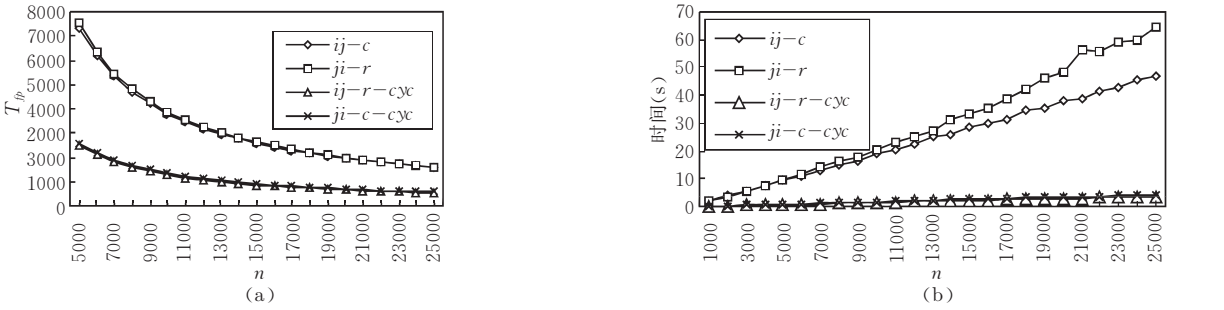

图 4　LSEC Linux Cluster 上不同形式的 PTRS 在 $p=64$ 时的 T_{fp} 值和相应的实验结果

从图 2～图 4 中的图(a)可以看出:

对于 SR2201,16 处理器时的分析结果是:

(1) 在 $n<7000$ 时:$T_{fp}^{ji-r}>T_{fp}^{ij-c}>T_{fp}^{ji-c-cyc}>T_{fp}^{ij-r-cyc}$;

(2) 在 $n>7000$ 时:$T_{fp}^{ji-r}>T_{fp}^{ji-c-cyc}>T_{fp}^{ij-c}>T_{fp}^{ij-r-cyc}$.

对于曙光 3000,32 处理器时的分析结果是:

(1) 在 $n<14000$ 时:$T_{fp}^{ji-r}>T_{fp}^{ij-c}>T_{fp}^{ji-c-cyc}>T_{fp}^{ij-r-cyc}$;

(2) 在 $n>14000$ 时:$T_{fp}^{ji-r}>T_{fp}^{ji-c-cyc}>T_{fp}^{ij-c}>T_{fp}^{ij-r-cyc}$.

对于 LSEC Linux 机群,64 处理器时的分析结果是:

$T_{fp}^{ji-r}>T_{fp}^{ij-c}\gg T_{fp}^{ji-c-cyc}>T_{fp}^{ij-r-cyc}$.

我们用 C 语言实现了这四种形式的并行下三角方程求解程序.为了防止过多的优化对最后性能的影响,我们对任何一种形式都避免特别的改进.每一个实验平台上测试的最大问题规模是在该平台上内存容量所容许的最大规模.在 SR2201 上 $p=16$ 时的问题规模从 1000～15000,步长 1000;在曙光 3000 上 $p=64$ 时的问题规模从 1000～20000,步长 1000;在 LSEC Linux 机群上 $p=64$ 时的问题规模从 1000～25000,步长 1000.测试结果分别如图 2～图 4 的图(b)所示.

从图 2～图 4 的图(b)可以看出,这些形式的运行时间从大到小的排列如下:

在 SR2201 上 16 个处理器时：

(1) 若 $n<7000$：$ji-r>ij-c>ji-c-cyc>ij-r-cyc$；

(2) 若 $n>7000$：$ji-r>ji-c-cyc>ij-c>ij-r-cyc$.

在曙光 3000 上 32 个处理器时，问题规模一直测到 20000 为止，一直有

$ji-r>ij-c>ji-c-cyc>ij-r-cyc$.

在 LSEC Linux 机群上 64 个处理器时，问题规模一直测到 25000 为止，一直有

$ji-r>ij-c>ji-c-cyc>ij-r-cyc$.

综合分析和实验结果，可以看出：

在 SR2201 上 $p=16$ 时，运行时间的排列顺序与分析排列顺序完全吻合. 从图 2 中可以看出，在 $p=16$ 时，$ji-c-cyc$ 形式在 $n<7000$ 时对 $ji-r$ 和 $ij-c$ 形式有性能优势；在 $n>7000$ 时，其性能会比 $ij-c$ 差一些. 这和我们对交叉点的预测完全一致.

在曙光 3000 上 $p=32$ 时，分析所给出的性能排列关系在 $n\leq 14000$ 前与实验完全吻合；但是所预测的交叉点并不在 14000，而从总体的趋势上看，会比 20000 大. 由于机时的限制，我们没能完全测到交叉点能够出现的问题规模. 由于曙光 3000 的二级高速缓存很大，但其内存小问题规模不能测到太大，我们在进行分析时对非连续存储访问的假定不是总成立，即非连续的存储访问在问题不是很大时，也会有在二级高速缓存命中的情况出现，这会引起在表 2 公式中的所有 t_m 变成 t_{L2}. 如果这样，那么我们在曙光 3000 的分析结果会变成如图 5 中所示. 在这种假定下的分析结果与实验结果完全吻合.

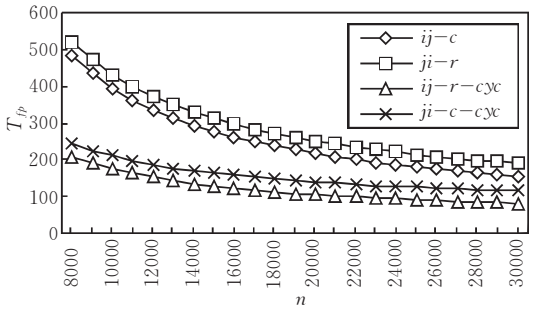

图 5 曙光 3000(32 处理器)在假定二级高速缓存很大时的分析结果

在 LSEC Linux 机群上 $p=64$ 时的分析结果与实验结果完全一致. 由图 2～图 4 的(a)中分析得出，$ij-c$ 和 $ji-r$ 形式看上去没有太大的差别实际上从绝对值来说，二者的差别一直在 40ns 左右.

5.2 并行无列选主元 LU 分解

根据表 3 中的分析结果和表 4 中的模型参数，我们在图 6～图 8 中分别给出了并行无列选主元 LU 分解(PLU)在 SR2201、曙光 3000 和 LSEC Linux 机群上 T_{fp} 值随问题规模(1000～15000)的变化情况(图(a))和相应的实验结果(图(b)). 曙光 3000 上的实验由于机时和机器规模有限，只做了 16 个处理器的实验.

从图 6～图 8 中的图(a)可以看出：

在 SR2201 上当处理器数为 16，根据模型分析结果有：

(1) $n<3000$ 时，$T_{fp}^{jki-r}\approx T_{fp}^{jik-r}>T_{fp}^{ikj-c}>T_{fp}^{kji-r}>T_{fp}^{kij-r}\approx T_{fp}^{kij-c}$；

(2) $n>3000$ 时，$T_{fp}^{jki-r}\approx T_{fp}^{jik-r}>T_{fp}^{kji-r}>T_{fp}^{ikj-c}>T_{fp}^{kij-r}\approx T_{fp}^{kij-c}$.

在曙光 3000 上当处理器数为 16 时，根据模型分析结果有：

(1) $n<4000$ 时，$T_{fp}^{jki-r}\approx T_{fp}^{jik-r}>T_{fp}^{ikj-c}>T_{fp}^{kji-r}>T_{fp}^{kij-r}\approx T_{fp}^{kij-c}$；

(2) $n>4000$ 时，$T_{fp}^{jki-r}\approx T_{fp}^{jik-r}>T_{fp}^{kji-r}>T_{fp}^{ikj-c}>T_{fp}^{kij-r}\approx T_{fp}^{kij-c}$.

在 LSEC Linux 机群上处理器数为 32 时，根据模型分析结果有

$$T_{fp}^{jik-r}\approx T_{fp}^{jki-r}\approx T_{fp}^{ikj-c}>T_{fp}^{kji-r}\approx T_{fp}^{kij-r}\approx T_{fp}^{kij-c}.$$

我们用 C 语言实现了这 6 种形式的 PLU. 在 SR2201 上 $p=16$ 时的问题规模从 1000～10000，步长 1000；在曙光 3000 上 $p=16$ 时的问题规模从 1000～15000，步长 1000；在 LSEC Linux 机群上 $p=64$ 时的问题规模从 1000～7000，步长 1000. 测试结果分别如图 6～图 8 的(b)所示. 可以看出，这些形式的运行时间从大到小的排列如下：

在 SR2201 上有 16 个处理器时：

(1) $n<5000$ 时，$jki-r>jik-r>ikj-c>kji-r>kij-r\approx kij-c$；

(2) $n>5000$ 时，$jki-r>jik-r>kji-r>ikj-c>kij-r\approx kij-c$.

在曙光 3000 上有 16 个处理器时：

(1) $n\leq 8000$ 时，$jki-r>jik-r>ikj-c>kji-r>kij-r\approx kij-c$；

(2) $n>8000$ 时，$jki-r>kji-r>jik-r>ikj-c>kij-r\approx kij-c$.

在 LSEC Linux 机群上有 32 个处理器时：

$jik-r\approx jki-r\approx ikj-c>kji-r\approx kij-r\approx kij-c$.

由综合分析和实验结果，可以看出：

在 SR2201 上 $p=16$ 时,运行时间的排列顺序与计算模型分析的排列顺序完全吻合,我们的模型成功分析出了 $ikj-c$ 和 $kji-r$ 两种形式随着问题规模的变大,会出现性能曲线交叉点的情况;只是模型对问题规模变大时的性能交叉点位置分析的不是很准确,这可能是由模型的简化引起的.

在曙光 3000 上 $p=16$ 时,也存在和 SR2201 上类似的问题.另外,对 $jik-r$ 形式的分析结果在问题规模不超过 5000 时是正确的,但在超过 5000 之后与分析结果出现偏差.变成比 $kji-r$ 形式还要小.目前不能作出对曙光 3000 上的这一现象很好的解释.

而在 LSEC Linux 机群上,我们的实验结果与计算模型分析的排列顺序完全吻合.由于通信的软件开销过大,6 种形式的算法实现明显地被分成了两类,而两类内部的差别不是很大.

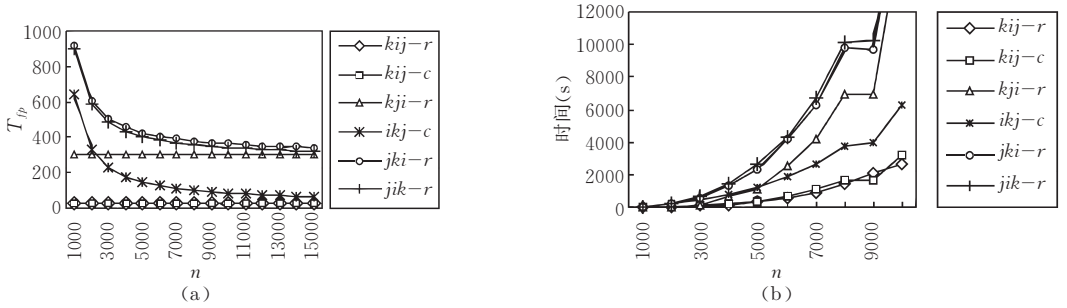

图 6　在 SR2201 上不同形式的 PLU 在 $p=16$ 时的 T_{fp} 值和相应的实验结果

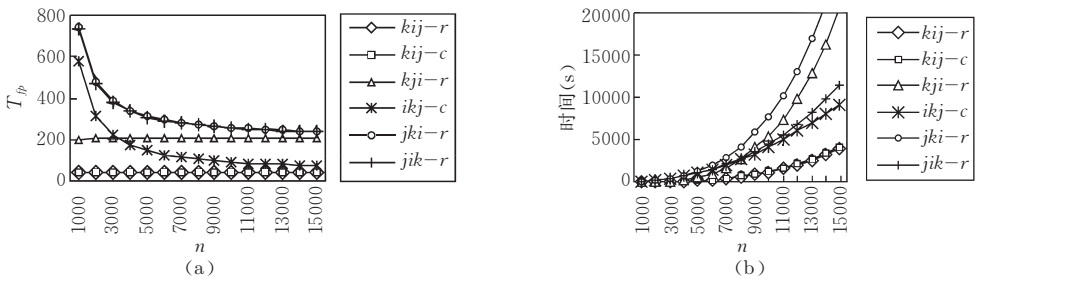

图 7　在曙光 3000 上不同形式的 PLU 在 $p=16$ 时的 T_{fp} 值和相应的实验结果

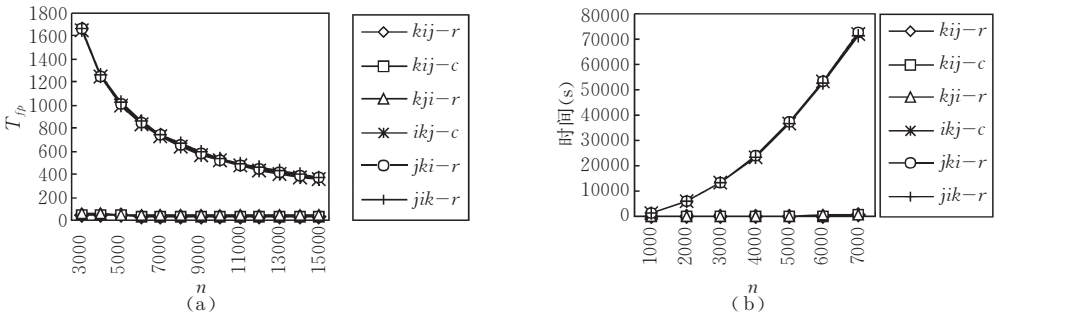

图 8　在 LSEC Linux Cluster 上不同形式的 PLU 在 $p=32$ 时的 T_{fp} 值和相应的实验结果

6　结束语

本文通过把存储复杂性的概念嵌入传统串行计算模型 RAM,在存储层次方面对该经典模型的分析能力进行了加强,提出了新的串行计算模型 RAM(h);在新串行模型的基础上,通过结合 LogP[6] 模型的特点,进一步将该模型扩展为面向分布式存储的并行计算模型 DRAM(h).通过在三种计算平台上对 PTRS 和 PLU 两个典型并行数值计算算法不同实现形式的模型分析和实验验证,我们的模型分析结果在绝大多数情况下都能很好地与在不同实验平台上的实验结果吻合.模型在有些情况下甚至能精确地分析出不同形式的算法实现出现性能交叉点的情况.对一些出现偏差的分析结果,在根据计算平台的存储层次特点修改模型分析的假定后,也能够进行解释.这说明了我们的新模型对不同形式的算法实现形式进行存储访问模式区分的有效性.我们提出的新计

算模型能够很好地对同一算法的不同实现形式在不同的计算平台上进行有效的性能优劣分析,为用户选取最适合某一计算平台的算法实现形式提供理论指导.

目前本文提出的计算模型的分析方法还比较复杂,需要在保持分析精度的前提下,进一步研究更加方便实用的分析方法,降低模型分析的难度;在更多计算平台上对本文提出的并行计算模型进行更多的验证,并根据实验结果对模型进行改进,以进一步提高分析精度,降低模型分析的难度以及在指令/线程级并行方面的研究是我们将来的工作.

参 考 文 献

1 Goodman J R. Using cache memory to reduce processor-memory traffic. In: Proceedings of the 10th Annual Symposium on Computer Architecture(ISCA-10), Stockholm, Sweden, 1983. 124～131

2 Burger D, Goodman J R. Guest editors' introduction: Billion-transistor architectures. IEEE Computers, 1997, 30(9): 46～48

3 Ortega James M. Introduction to parallel and vector solution of linear systems. Frontiers of Computer Science. New York: Plenum Press, 1988

4 Valiant L. A bridging model for parallel computation. Communication of ACM, 1990, 33(8): 103～111

5 McVoy Larry, Staelin Carl. lmbench: Portable tools for performance analysis. In: Proceedings of Winter 1996 USENIX, San Diego, CA, USA, 1996. 279～284

6 Culler D, Karp R, Patterson D *et al*. LogP: Towards a realistic model of parallel computation. In: Proceedings of PPoPP IV, San Diego, CA, USA, 1993. 1～12

7 Gibbons Phillip B, Matias Yossi, Ramachandran V. Can a shared-memory model serve as a bridging model for parallel computation? In: Proceedings of SPAA'97, Newport, Rhode Island, USA, 1997. 72～83

8 Cook S A, Reckhow R A. Time bounded random access machines. Journal of Computer and Systems Sciences, 1973, 7(4): 354～375

9 Alpern B, Carter L, Feig E, Selker T. The uniform memory hierarchy model of computation. Algorithmica, 1994, 12(2/3): 72～109

10 Aggarwal A, Alpern B, Chandra A, Snir M. A model for hierarchical memory. In: Proceedings of the 19th Annual ACM Conference on Theory of Computing, New York, USA, 1987. 305～314

11 Aggarwal A, Alpern B *et al*. Hierarchical memory with block transfer. In: Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angels, California, USA, 1987. 204～216

12 Amato N M *et al*. Predicting performance on SMPs: A case study——the SGI Power Challenge. In: Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, 2000. 729～737

13 Zhang Yun-Quan, Sun Jia-Chang, Tang Zhi-Min, Chi Xue-Bin. Memory complexity on numerical programs. Chinese Journal of Computers, 2000, 23(4): 363～373(in Chinese)
(张云泉,孙家昶,唐志敏,迟学斌. 数值计算程序的存储复杂性分析. 计算机学报, 2000, 23(4): 363～373)

14 Badawy A H, Aggarwal A, Yeung D, Tseng C W. Evaluating the impact of memory system performance on software prefetching and locality optimizations. In: Proceedings of 2001 International Conference on Supercomputing (ICS'01), Sorrento, Italy, 2001. 486～500

15 Zhang Yun-Quan. Performance optimizations on parallel numerical software package and study on memory complexity[Ph. D. dissertation]. Beijing: Institute of Software, Chinese Academy of Sciences, 2000(in Chinese)
(张云泉. 高性能并行数值软件性能优化及存储复杂性研究[博士学位论文]. 北京:中国科学院软件研究所, 2000)

16 Juurlink Ben, Wijshoff Harry A G. A quantitative comparison of parallel computation models. ACM Transactions on Computer Systems, 1998, 16(3): 271～318

17 Zhang Yun-Quan. DRAM(h,k): A parallel computation model for numerical computing. In: Proceedings of 2001 National Conference on Intelligent Computer of "863" National Plan, Beijing, 2001. 218～225(in Chinese)
(张云泉. 面向数值计算的并行计算模型 DRAM(h,k). 见: "八六三"计划智能计算机主题学术会议论文集《智能计算机研究进展》, 北京, 2001. 218～225)

ZHANG Yun-Quan, born in 1973, Ph. D., associate professor. His major research interests include high performance parallel computing, with particular emphasis on the design of large scale parallel numerical software library, parallel programming, performance modeling and evaluation, and parallel computational model.