摘 要

长期以来，大规模计算的应用需求推动并行计算技术不断发展，并行计算机的峰值计算能力稳步提高。当前，基于共享存储的片上多核处理器搭建集群系统，成为并行体系结构的发展趋势，并行计算已经进入了千万亿次并行计算机的时代。但是，并行计算机的实际应用水平不高，应用程序实测性能远低于系统的峰值性能。因此，充分发挥并行计算机的计算能力，加速应用程序的执行速度，逐渐成为并行计算领域中亟需解决的一个关键问题。

未来，共享存储系统是搭建大规模并行处理系统的基本单元。围绕提高并行计算机的实际应用效率，缩小实际应用性能与机器峰值性能之间的差异，本文以共享存储系统为目标平台，研究并行计算模型以及程序性能优化关键技术，主要研究工作分为两部分：一是研究分层的并行计算模型，为并行算法设计和并行程序执行提供理论基础和分析方法，其中重点研究片上多核系统的程序执行模型；二是研究共享存储系统上的程序性能优化技术，以提高并行应用程序的实际性能，同时为程序执行模型提供思路和借鉴。本文针对计算模型和优化技术的研究，可以有效地提高并行应用的性能，充分发挥并行计算机的计算能力，具有重要的学术价值和广泛的应用前景。具体而言，本文的主要研究成果、贡献和创新点可概括为以下几点：

(1) **提出分层的并行计算模型** 随着并行机体系结构的快速发展变化，传统单一的并行计算模型变得越来越复杂，难以使用。本文对并行计算模型分层研究，把并行计算模型分为并行算法设计模型、并行程序设计模型和并行程序执行模型三个层次，分别给出了各层模型的特点及研究内容。

(2) **优化共享存储系统上消息传递的通信性能** MPI是一种流行的并行编程接口，同时支持分布存储并行机和共享存储并行机。针对MPI在共享存储上的驱动程序通信性能不高，本文提出一种共享存储系统上MPI消息传递优化方法，利用共享内存系统上进程间通讯机制和自旋等待同步策略，实现了进程间直接数据复制，减少了消息传递延迟，提高了共享存储系统上点对点和集合通信性能，优化了实际应用程序的通信性能。

(3) **优化共享存储系统上典型应用程序的性能** 本文研究两个典型的并行应用，分别在两种共享存储系统上的优化方法。一个是在对称多处理机上，基于MPI的生物信息领域的应用Mfold的并行优化；另一个是在片上多核系统上，基于OpenMP的信息检索领域的应用CBIR的并行优化。针对应用和系统

特点，设计了高效的并行算法，挖掘共享存储系统的多级并行度，有效优化了应用的指令级并行性、数据级并行性和线程级并行性，加速了应用程序在共享存储系统上的速度，为这一类平台上开发高效应用程序提供了借鉴。

(4) **提出面向片上多核系统的定量程序执行模型** 结合对共享存储系统上应用程序性能优化的研究，本文提出面向片上多核系统的定量程序执行模型CRAM(h)。CRAM(h)模型考虑了指令执行行为、层次存储访问行为及并行处理行为，抽取关键性能参数，对程序执行时间进行建模。实验表明，模型评估的程序执行时间与实际程序运行时间基本一致。

关键词： 并行计算 共享存储系统 分层并行计算模型 对称多处理机 多核处理器 性能优化 性能评测 程序执行模型

ABSTRACT

In the past few decades, parallel computing keep developing continuously promoted by the requirements of large computation problems. At the same time, the peak performance of parallel computer increases steadily. Nowadays, large scale cluster based on Chip Multi-Processor (CMP) has become the mainstream of parallel architecture. Parallel computing has entered a petaflops era. However, the application level of parallel computer is very low. The performance of real application is much lower than the peak performance provided by parallel computer. Consequently, fully utilizing the computational power of parallel computer and accelerating the performance of real application have become the critical issues in parallel computing.

In the future, shared memory system will be the base unit to build large scale parallel computer. This dissertation focuses on improving the efficiency of a parallel computer and bridging the gap between the real performance of applications and the peak performance of parallel computers. The main contents include parallel computation model and program performance optimization techniques on shared memory architecture. First, a layered parallel computation model is proposed to provide theoretical fundamental and analysis approaches for parallel algorithm design and parallel program execution. Specially, parallel execution model is emphatically studied. Second, in order to improve real application's performance on shared memory system, program performance optimization techniques are studied in depth. The study on computation model and performance optimization would effectively increase application's performance on parallel computer, which are valuable in theory and practice. Specifically, the main contents and contributions of this dissertation are as follows.

(1) **Layered parallel computation model** With the rapid development of parallel architecture, the conventional unified parallel computation model becomes more and more complex, which is difficult to use. A layered parallel computation model is proposed in this dissertation, which consists of parallel algorithm design model, parallel programming model and parallel execution model. The properties of each model are presented, as well as the research spots.

(2) **Optimization of MPI communication on SMP** MPI is widely used in parallel programming, which supports both distributed and shared memory system. But cur-

rent MPI communication device on SMP has low performance. In this dissertation, an optimized MPI communication method is proposed for SMP. In the optimized communication, IPC (Inter-process communication) is employed for data transfer, and a spin-waiting strategy is used for synchronization. The new communication method reduces message passing delay, increases the performance of point-to-point communication and collective communication, and delivers high communication performance in real application.

(3) **Optimization of typical applications on shared memory system** In this dissertation two typical applications on different shared memory systems are studied. One is parallel optimization of Mfold on SMP using MPI, which is an application in bioinformatics. The other is parallel optimization of content-based image retrieval (CBIR) on CMP using OpenMP, which is an application in information retrieval. Based on the characteristics of the applications and the architectures, high performance parallel algorithms are designed to exploit multi-level parallelism of the shared memory systems. The proposed optimization techniques significantly improve the ILP (Instruction Level Parallelism), DLP (Data Level Parallelism) and TLP (Thread Level Parallelism), finally accelerate the two applications on shared memory systems, and provide insights into designing high performance applications on these platforms.

(4) **Quantitative program execution performance model for CMP** Based on the research of performance optimization on shared memory systems, a quantitative program execution performance evaluation model for CMP is proposed: CRAM(h). CRAM(h) model extracts the key aspects effecting parallel program performance, such as instruction execution, memory hierarchy and parallelism, uses performance profiler to measure performance parameters, quantifies performance benefits from optimizations, evaluates program performance. Experiments are conducted to evaluate the correctness and effectiveness.

Keywords: Parallel Computing Layered Parallel Computation Model SMP CMP Performance Optimization Performance Evaluation Program Execution Model

插 图

表 格

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文，是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：苗乾坤　　签字日期：2010.6.5

中国科学技术大学学位论文授权使用说明

作为申请学位的条件之一，学术论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权。即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸制论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

☑公开　□保密（___年）

作者签名：苗乾坤　　导师签名：[illegible]

签字日期：2010.6.5　　签字日期：2010.6.5

第1章 绪论

内容提要 并行计算是求解大规模计算问题的有力手段，大规模计算需求推动并行计算往前发展。并行计算把计算问题分解成小的计算任务，同时在多个计算单元上执行各个计算任务。并行计算的研究包括并行计算机体系结构、并行程序设计、并行算法理论和并行应用四个方面。本章首先对并行计算进行概述，介绍并行计算的研究背景和研究现状，包括并行计算机的体系结构、并行编程环境、并行算法理论以及并行计算应用的性能优化；接着介绍国内外并行计算的近年来发展情况以及未来发展趋势，总结当前并行计算研究面临的一些问题与挑战；最后给出本文研究工作的思路、内容、创新之处和组织安排。

1.1 并行计算的基本概念

1.1.1 研究背景

当今，科学研究、国民经济、工程技术和军事国防等领域有诸多具有挑战性的问题[1,2]，如计算物理、计算化学、生物医学、流体力学、桥梁设计、天气预报、油藏勘探、互联网信息处理、工业设计制造、武器模拟、空间技术和地球科学等。并行处理技术对于这些问题的求解至关重要：首先，对于一些大规模的应用问题，单一处理器计算机由于物理速度限制远不能满足需求；其次，一些大型复杂科学计算问题对计算精度要求比较高，同时也意味着大的计算量；第三，大量的科学技术和工程问题，对问题的求解有强烈的时效性要求，超过一定的时间结果就毫无意义。因此，出现了并行计算机和并行计算技术，即把问题分解成若干个尽量相互独立的子问题，然后使用多个计算设备来加快问题求解速度。并行计算由于其强大的计算能力也被称为高性能计算或超级计算，并行计算机又叫高性能计算机或超级计算机[3]。

1.1.2 研究内容

并行计算可分为如下几个步骤[4]：对于一个给定的问题，计算科学家首先将其描述为待求解的数值或非数值计算问题；然后计算机科学家为该计算问题设计一个并行算法，并选择某种并行程序设计语言来编程实现它；最后领域专家/程序使用者在具体的并行机上编译和运行程序，最终求解出此问题。从以上并行计算的一般过程可以看出，并行计算以并行算法为理论基础，以并行计算

机为硬件平台，以并行程序设计为软件支撑，以并行应用为发展驱动。并行计算学科聚集了计算数学家、计算机科学家、软件工程师和应用领域专家等，是个多学科交叉，具有很大研究空间的学科。中国科学技术大学国家高性能中心（合肥）在过去20多年的研究中总结出了"结构-算法-编程-应用"的一体化的并行计算研究方法[5-7]。其中，结构包括并行机硬件设计和制造；算法包括并行算法的设计和分析；编程包括并行编程语言和编程环境的设计；应用包括科学工程应用和其他各种大规模计算应用。一体化的研究方法，为并行计算提供了一个可持续发展的环境，给并行计算的研究人员提供了有益的指导。

1.2　并行计算机体系结构

1.2.1　基本概念

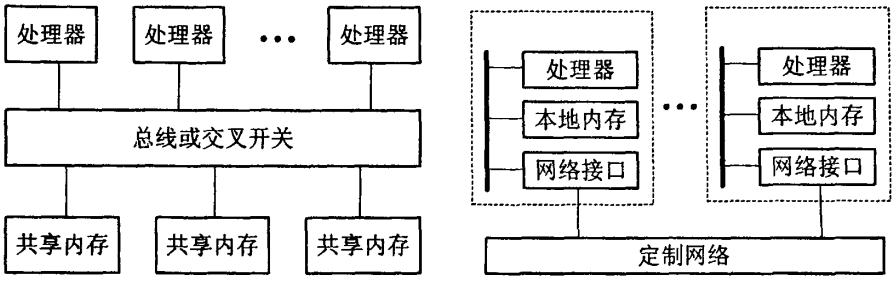
并行计算机[8]就是一组处理单元，通过协作快速解决计算问题的大规模计算系统。与传统单处理器计算机相比，并行计算机拥有更多的处理器和更多的内存，增加了处理单元之间通信和协作的控制部件和协议。并行计算机性能的重要指标就是计算峰值，即每秒钟能完成的浮点运算次数，包括理论峰值和实测峰值。理论峰值是计算机系统理论上能够达到的计算速度。实测峰值是指Linpack测试给出的浮点性能。并行计算机的性能通常以单位时间的浮点运算次数来衡量，比如1 Petaflops等于每秒钟一千万亿次的浮点运算，不同的单位之间换算关系如图1.1所示。并行计算机设计与制造一直保持高速的发展，当今的高性能计算机的计算能力已经突破了每秒千万亿次，研究万万亿次并行计算机的计划已经开始提上议程。

```
1000 Megaflops = 1 Gigaflops
1000 Gigaflops = 1 Teraflops
1000 Teraflops = 1 Petaflops
1000 Petaflops = 1 Exaflops
1000 Exaflops  = 1 Zetaflops
1000 Zetaflops = 1 Yottaflops
```

图 1.1 性能参数换算关系

1972年Flynn根据指令和数据把并行计算机分为四种类型[9]：单指令单数据流(Single Instruction Single Data，SISD)，一条指令序列处理同样的数据，冯·诺依曼体系结构的单处理器计算机都是这类；单指令多数据流（Single Instruction Multiple Data，SIMD），一个指令序列可以同时操作在不同的数据上面，比如数字信号处理和向量运算处理等专用处理器；多指令单数据流（Multiple Instruction Single Data，MISD），没有实际的系统，是为了分类完整而提出来的；

多指令多数据流（Multiple Instruction Single Data，MIMD），不同的指令序列作用在不同的数据上面。这是并行处理系统最常见的结构，现代主流的并行计算机基本都可以划为这一类。MIMD的并行计算机按照存储系统组织方式和编程模型的不同，又可以分为共享存储的多处理机系统和分布存储多处理机系统两种[8]，如图1.2所示。



(a) 共享存储系统　　(b) 分布存储系统

图 1.2 共享存储和分布存储系统结构示意图

1.2.2　共享存储体系结构

共享存储结构的并行计算机是一种紧耦合的系统，多个处理器通过对同一块或多块内存单元的访问进行数据的交换。这一类系统中任何处理器均可访问任何存储单元和I/O设备，具有单一的地址空间。根据物理存储器是否均匀共享，共享存储又可分为均匀存储和非均匀存储，前者所有处理器访问任何存储单元需要相同的时间，后者把存储器分为本地存储和远程存储两种，访问本地存储器时速度比较快，而访问远程存储器时速度比较慢。共享存储系统中，多个处理器同时对共享内存访问，会引起严重的存储竞争和较长的延迟，影响共享存储系统的可扩展性。对称多处理机（Symmetric Multiprocessor，SMP）和片上多核处理器（Chip Multi-Processor，CMP）属于均匀共享的并行系统。分布共享存储的多处理机（Distributed Shared Memory，DSM）通过系统硬件和软件把物理上分布的局部存储，组织成一个具有单一地址编程空间的共享存储器。

1.2.3　分布存储体系结构

分布式存储结构的并行计算机是一中松散耦合的系统，通过互联网络，按一定拓扑结构（如线性阵列、环形结构、Mesh结构和超立方结构等）将每个处理单元连接起来，每个处理单元都有各自的局部存储器，所有的局部存储器构成了整个地址空间。每个处理器可以访问自己的局部存储器，不能访问其他处理器的局部存储器，处理器之间只能通过互联网络进行显式的消息传递，来完

成通信和数据同步。由于高性能的网络可以连接成千上万的结点，因此分布存储具有良好的可扩展性，这类系统的规模可以很大，其总结点数可能达到成千上万。当前世界上排名靠前的超级计算机，大多都是基于小规模共享存储系统，如多核处理器，搭建而成的分布式集群系统。大规模并行处理机（Massively Parallel Processor，MPP）和工作站集群（Cluster of Workstations，COW）都属于分布式存储的并行机。

1.2.4　主流的并行计算机体系结构

随着并行计算机的发展，目前对称多处理机、多核计算机和集群架构并行机逐渐占据了主导地位。当前并行计算可以挖掘多个层次的并行性，包括传统的指令级并行、单处理器同时多线程级并行、多核处理器多线程级并行、对称多处理机内多个处理器级并行和集群内多个节点间并行，如图1.3所示。在最新一期高性能计算机TOP500[10]排行榜上，大部分的并行计算机都是使用多核处理器搭建的集群系统，这种架构也代表了未来高性能并行计算机的发展潮流。本文研究的目标体系结构为共享存储结构，即对称多处理机和多核处理器，为下一步研究基于这两种架构的大规模集群系统打下基础。

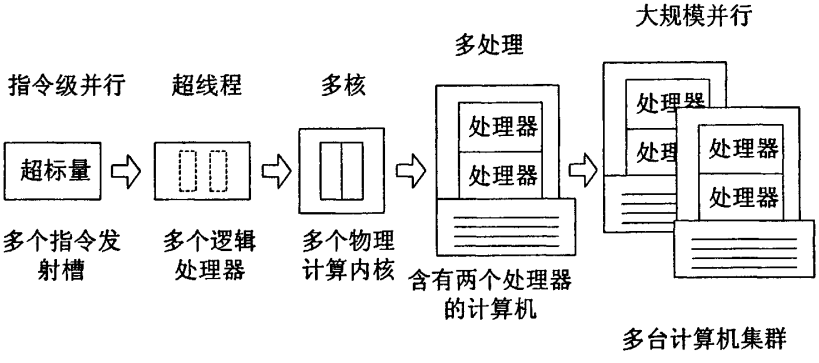


图 1.3 并行计算机中多种并行级别

1.3　并行算法理论

1.3.1　基本概念

算法是求解问题的方法和步骤，并行算法就是并行计算过程的描述，是利用并行计算技术求解问题的方法和步骤，是指在各种并行计算机上求解问题和处理数据的算法[11]。根据计算类型的不同，并行算法可以分为数值并行算法和非数值并行算法，前者研究基于代数关系运算的数值计算问题，主要包括矩阵

运算、方程组求解和数字信号处理等，后者研究基于比较关系运算的符号处理问题，主要包括图论问题、数据库操作和组合优化等；根据并行进程执行的先后顺序，分为同步并行算法和异步并行算法，前者是指进程间存在先后关系，需要互相等待执行的算法，后者是指各进程间不需要互相等待，各进程的执行顺序对结果没有影响；根据并行算法使用的存储模型，可以分为共享存储并行算法和分布存储并行算法，顾名思义就是说分别针对共享存储系统和分布式存储系统设计的算法；根据并行性的来源可以分为数据并行算法和任务并行算法，前者是按数据进行划分，然后每块数据交给一个线程去处理，各线程执行的代码基本相同，只是处理的数据不同，后者是按计算任务进行划分，每个任务交给一个线程去处理，关注被执行的计算任务，例如搜索树对各分支可以并行进行搜索。

1.3.2　并行计算模型

计算模型是为了计算目的将真实体（计算机软/硬件）进行一定的抽象而成的。并行计算模型是并行计算中软硬件之间的桥梁，并行计算模型[11,12]是并行算法设计者所看到的参数化的并行机，是提供给编程者的计算机软/硬件接口，是程序执行时的系统软/硬件支撑环境。并行计算模型提供少数能反映并行机计算特性的可以定量测量或者计算得到的参数，定义了其上的计算行为和成本函数，以此来进行算法复杂度分析。根据并行计算模型的发展历史，可以把并行计算模型分为三代[12]。早期的并行机为共享存储器的SIMD和MIMD的计算机，相应地，提出了以计算为核心的所谓第一代并行计算模型PRAM和APRAM等，并在其上开发了不少优秀的并行算法。后来出现了分布存储的大规模并行机MPP，相应地提出了以网络通信为核心的所谓第二代并行计算机模型BSP、LogP、NHBL等。最近分布共享的集群并行机成为主流并行机，CPU和主存之间的速度差异越来越大，存储系统逐渐成为影响系统性能的主要瓶颈，相应地，提出了以存储访问为核心的所谓第三代并行计算机模型UMH、Memory-LogP和DRAM(h)等。另外，在第三代并行计算机模型的基础上，又提出了同时考虑层次存储和层次并行的所谓第三代半并行计算模型HPM。并行计算模型的演变历史，体现出了模型先以CPU计算为中心，再以网络通信为中心，最后逐步过渡到以存储访问为中心。

1.3.3　并行算法设计技术

并行算法经过多年发展，已经总结出一些基本的设计技术[11]。总结起来有：划分法（Partitioning）是最自然朴素的并行设计方法，它是将一个计算任务根据数据或者任务分解成若干个规模大致相等的子任务而并行求解；分治

法（Divide-and-conquer）是将问题分而治之，大的问题划分成若干个规模比较小的相同或者相似的子问题，先并行分别求每个子问题的解，进而由子问题的解构造原问题的解；流水线法（Pipelining）是一种基于空间并行和时间重叠的问题求解技术，把计算任务按照数据流动的方向分成一系列子任务，前一个子任务执行结束，它的输出直接作为下一个子任务的输入，下一个子任务可以立即开始执行，这一过程形成了一个流水线，不同的线程可以同时针对多个输入处理这些不同的子任务；随机法（Randomization）是一种不确定性算法，在算法设计步中引入随机性，从而可望得到平均性能良好、设计简单的并行算法；平衡树法（Balanced Tree）、倍增法（Doubling）和破对称法（Symmetry Breaking）等都是针对待求解问题本身的特点而采用的一些有效设计方法；迭代法（Iteration）是求解诸如线性方程组之类问题的常用数值求解方法。

1.3.4 并行算法性能度量

设计好的并行算法需要进行性能分析，进而评价算法的好坏。因此，需要有一些被普遍接受的评价指标、方法和工具。

研究并行计算的主要目的就是缩短计算时间，加速计算过程，增大问题规模，所以执行时间和加速比是并行计算最重要的性能指标。在一定意义上，加速比和执行时间是等效的，因为加速比大的执行时间也比较少。在无重复操作的假定下，并行程序的执行时间T_p是指算法从开始执行到执行结束所消耗的时间，主要包括三部分：计算时间、并行开销花费、通信同步时间，可以概括为下面的公式：

$$T_p = T_c + T_o + T_s \tag{1.1}$$

其中，T_c为计算时间，T_o为并行开销，T_s为同步时间。加速比S_p是指对于一个计算任务，其串行算法执行时间和并行算法执行时间的比值：

$$S_p = T_1 / T_p \tag{1.2}$$

其中，T_1表示最优串行算法执行时间，T_p表示并行算法使用p个处理器的执行时间。加速比在最理想情况下等于所使用的处理器数目p，这时候称为线性加速比。但是由于并行额外开销和计算中不可并行部分在所难免，严格的线性加速比是很难达到的，超线性加速比更是难上加难，只在某些特殊的情况下会出现超线性加速现象，例如并行搜索算法中。加速比越趋于所使用的处理器数目，说明并行算法的越好。主要有三种加速比性能定律：适用于固定计算负载的Amdahl定律（Amdahl's Law）[13]，适用于可扩放性问题的Gustafson定律（Gustafson's Law）[14]和适用于存储受限的Sun和Ni定律（Sun and Ni's Law）[15]。

并行算法效率$E_p = S_p/p$，为加速比和处理器数的比值。效率可以衡量并行计算机中处理器的利用率。如果在整个并行算法执行过程中，多数的处理器均处于活跃状态，那么并行算法的效率就比较高。因此优化多个处理器的负载均衡，是提高并行算法效率的关键。并行算法的可扩放性也是评价算法的另一个重要性能指标。随着计算负载的增加和机器规模的增大，也会带来额外开销的增加，因此会降低处理器利用率。人们通过考察并行系统和算法的可扩放性，来判断不断增加的处理器的计算能力是否被有效利用。可扩放性是指在确定的应用背景下，算法性能随着处理器数的增加而按比例提高的能力。主要有以下三种可扩放性度量方法，即等效率、等速度和平均延迟。

1.4 并行编程环境

1.4.1 基本概念

并行算法需要通过并行编程模型表达，才能够在并行机上执行。并行编程模型和语言一直以来受到了广泛的研究[16]，不同时期出现了各种各样的并行编程环境，新的并行编程语言和环境还在层出不穷，目前还没有一个成熟的并行编程解决方案。历史上，并行编程模型的设计主要有以下三种方法[17]:

1. 利用或扩展现有的串行语言，通过库函数的形式支持并行；
2. 扩展现有的编译器，通过编译制导的方式，指导编译器编译成可以并行执行的程序；
3. 设计新的并行编程语言和编程环境；

本节根据不同的计算机体系结构讨论三种典型的并行编程模型[18]，介绍它们的特点以及典型的编程语言，包括数据并行模型（初衷是针对SIMD并行机）、共享变量模型（初衷是针对共享存储并行机）和消息传递模型（初衷是针对分布式存储并行机）。

1.4.2 数据并行模型

在使用数据并行的程序中，有一个进程执行，具有单一控制逻辑，一个语句可以作用在不同数据变量上，每条语句后均有一个隐式同步语句，所有变量驻留在单一地址空间内，程序员不需要显式的分配数据。典型的支持数据并行模型的语言和环境有:高性能Fortran (High Performance Fortran)。

1.4.3 共享变量模型

在使用共享变量模型的程序中，有多个进程同时执行，每个进程有自己的

控制逻辑，执行不同的代码，多个进程异步执行，需要显式的同步来确保正确的执行顺序，有一个单一的全局名字空间，数据驻留在单一共享地址空间中，无需显式数据分配，通信和同步通过读写共享变量来实现。典型的共享变量模型的编程语言和环境有：Raw Threads和OpenMP。

1.4.4 消息传递模型

在消息传递模型的程序中，有多个进程同时执行，每个进程有自己的控制逻辑，执行不同的代码，操作自己的私有数据，多个进程异步执行，需要显式的同步来确保正确的执行顺序，不同节点上的进程具有私有地址空间，需要数据交换时可以通过网络传递消息进行通信，程序员必须显式地创建进程，为进程分配数据。典型的消息传递模型的编程语言和环境有：MPI、PVM和Erlang。

1.4.5 新型的并行编程语言

目前针对并行编程的研究还在继续，工业界和学术界还没有一个较成熟的解决方案。新的编程语言、编程框架和编程思想还不断涌现[19]。Cray公司的Chapel语言[20]、IBM公司的X10[21]和Sun公司的Fortress[22]是新型编程语言的代表，这三种编程语言设计的时候同时考虑了软件开发效率和执行性能，支持超大规模并行程序的设计和开发。MapReduce[23]技术随着Google在大规模数据处理领域的成功使用，逐渐变的流行起来。事务内存[24]作为解决并行处理中内存冲突的方法，近年来被广泛研究。事务内存用基于事务的方法对内存操作进行封装，提供了一种在多核处理器上实现数据共享访问和程序并发执行的方法，可以克服传统基于锁的并发机制中潜在的死锁、护航、优先级反转等问题。投机多线程技术[25]是一种适合多核处理器上串行程序自动并行化的新的编译技术，核心思想是自动分析线程划分方案，并推测执行，挖掘潜在的并行执行机会。

1.5 并行应用性能优化技术

1.5.1 并行应用的现状

从诞生以来，并行计算在各领域中获得了广泛应用，成为材料、物理、化学、医药、生物、气象、能源、游戏、电信、教育、国防等众多领域必须的技术。并行计算技术还在不断发展，未来应用方式也将多元化，应用面会更广，对计算的需求量会更大。可以说，并行应用对计算能力的追求是永无止境的。自上世纪七十年代以来，并行计算机的峰值速度一直在提高，并行计算机的计算能力经历了每秒百万次，每秒十亿次，每秒万亿次，到现在，具有每秒千万

亿次计算能力的并行计算机已经问世[10]。但是，目前并行计算机的实际应用水平还比较滞后，实际应用中的计算性能远远低于硬件峰值性能，因此，迫切需要研究并行应用程序的性能优化技术，充分发挥并行计算机的硬件计算能力。本小节介绍应用性能优化的基本概念和常用的工具，总结了一些常见的优化问题、方向和技术。

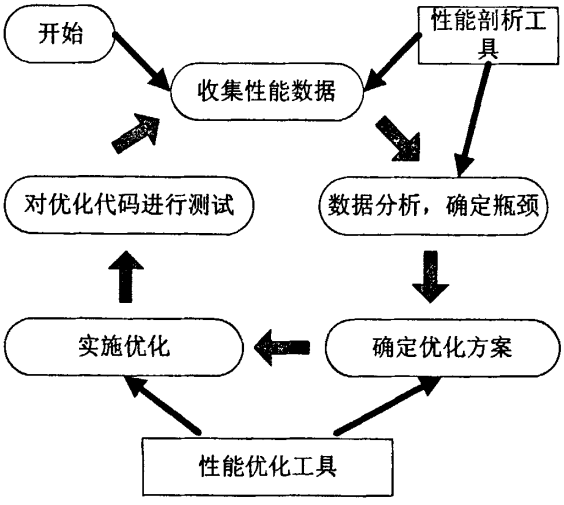


图 1.4 软件优化过程

1.5.2　性能优化过程

如图1.4所示，软件性能优化[26]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。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。

1.5.3 性能优化工具

基本的性能优化工具是：性能调优工具和性能分析工具。对一些常用的计算问题，人们已经编写了高效的函数库，我们只要根据接口直接调用就可以立即获得性能的提升，比如MKL、IPP、FFTW和gotoblas等。现代的处理器和编译技术取得了很大的进步，可以自动的采用处理器特性和优化策略生成高效的代码。性能分析工具用来收集程序运行时的性能数据，为分析程序行为，进行性能优化提供参考，主要的分析工具：Microsoft提供的性能监视器PERFMON.EXE、Intel的Vtune性能分析器和GNU gprof。

1.5.4 性能优化方法

程序的性能取决于程序中的所有指令在处理器上的执行时间。性能优化的本质是减少和消除程序中无用的操作，让处理器的每个部件都保持忙碌状态。性能优化的目的是提供给处理器足够多准备好的指令，好的混合指令，减少处理器等待内存的时间，使用高效的指令，消除无用的指令。在并行计算机上，程序的性能主要由两方面决定：单节点性能和多节点性能，以下从串行优化和并行优化两个角度来考虑程序的性能优化。

单处理内核上的性能主要取决于存储系统性能和指令执行性能。内存速度要比处理器速度慢的多，人们设计了多级存储系统来缓解这一问题，但是存储性能依然是瓶颈所在。不幸的是，多核处理器中内存对性能的影响更加突出，当处理器变的更快时，对数据的需求也增大，而内存带宽等资源并没有增加，内存速度也没有显著改善，处理器被强迫停顿等待数据从主存传输到处理器。直接内存读写、减小高速缓存失效率、乱序执行、软硬件数据预取、计算和访存重叠等技术可以改善内存性能。存储系统性能优化包括：优化局部性减少Cache缺失、优化数据传输的存储带宽、降低Cache不命中延迟和减少访存数量。指令执行优化是发掘指令级并行度，增大可并行执行指令的数量，减少指令执行时的相关性。优化存储访问也可以减少指令等待数据的时间，提高指令执行速度。主要有循环展开[27]、软件流水[28]、数据预取[29]和函数内联[30]等技术。循环由于其重复性，经常是程序中最常见的热点。常见的循环优化方法有消除数据相关、循环分块[31,32]、循环分配和融合、循环剥离、循环展开和合并、循环交换等技术。现代编译器有一系列优化循环的方法，能够自动地使用循环变换，因此手工优化前需要先确定编译器有没有自动优化该循环。算法的选择是决定程序运行速度最关键的因素。不同时间复杂度的算法在执行时间上往往会有数量级上的差距，一个好的算法可能比其他任何优化都有效。

随着多处理器计算机、超线程处理器和多核处理器的普及，开发应用并行

性，同时执行多个任务，变得非常重要。优化并行程序，实际上是要最大限度挖掘计算问题的并行性，使得计算行为尽可能与体系结构特性相匹配，合理安排并行任务的执行，消除不必要的消息传递和数据同步开销。编写并行程序时，为了能够高效地利用多处理器、超线程技术和多核处理器，需要关注以下问题：尽可能挖掘程序中的并行性；实现多线程负载均衡；最小化通信同步开销；消除假共享；使用处理器关联等技术。在后续章节中，本文将会详细介绍这些优化技术，展示如何使用这些技术优化程序性能。

1.6 并行计算的发展现状

1.6.1 国外的并行计算发展

高性能计算机的发展水平，代表了一个国家的科技创新能力和经济发展水平，对科学和经济发展有着重大的战略价值，世界各国都大力发展高性能计算机。美国在1993年提出了推动并行计算研究的高性能计算和通信计划(High Performance Computing and Communication, HPCC)，在1995年美国能源部启动了名为加速战略计算计划（ASCI）的高性能计算机研制计划，在1998年CIC蓝皮书中列出了发展每秒千万亿次超级计算机的计划，在2001年美国国防先进研究项目局提出了HPCS(高效能计算机系统)计划，目标研制适应高端国家安全和工业应用需要的新一代经济可行的高效能超级计算机，在2007年，美国能源部提出了建造百万万亿次超级计算机计划。欧盟第七框架计划（FP7）投入大量资金研究千万亿次高性能计算机，计划在欧洲建立3至4个千万亿次高性能计算设施。2004年5月，欧洲启动了DEISA计划，建立了一个欧洲主要国家的超级计算中心联盟，提供建立在现有的国内服务基础上的欧洲超级计算服务。2007年4月，欧盟14国启动了欧洲先进计算合作伙伴（PRACE）计划，打算合作建立欧洲超级计算研究的基础设施。除了美国和欧洲，日本也是高性能计算机研制的强国。2002年，NEC公司研制成功了地球模拟器（Earth Simulator）超级计算机，以每秒35万亿次的速度，在TOP500排行榜上连续3年排名第一。2005年7月，日本宣布启动下一代超级计算机计划（通用京速计算机）。预计通用京速计算机整个建设工程将于2012年6月完工,作为下一代超级计算机的通用京速计算机每秒能进行1京(1万万亿) 次浮点运算，将应用于地球环境和灾害等预测、新型半导体材料和新药研发等领域。全球TOP500超级计算机排行榜诞生于1993年，其目的是对全球范围内性能最强的500套超级计算机进行排名，以系统Linpack测试值为基准进行排名，每年发布两次，以此为基础跟踪和监测高性能并行计算领域的最新动态和趋势。在2009年11月发布的第34届TOP500超级计算机排行榜上，排名前十的超级计算机情况列在表1.1中。其中，安放在美国橡树岭国家实

验室，由Cray公司设计制造的美洲豹（Jaguar），以每秒1759万亿次的实测性能和2331万亿次的峰值性能占据榜首。之前是IBM制造的走鹃（Roadrunner）排名第一，走鹃也是第一台突破每秒千万亿次（Petaflops）的超级计算机。总上榜计算机数量按国家分布如表1.2所示，可以看出美国牢牢保持霸主地位，英国、德国、法国、中国和日本紧跟其后。本届TOP500排行有五套系统峰值性能超过了每秒千万亿次，比半年前上次排行增加两套，千万亿次已经成为大势所趋，可以预见会有更多的千万亿次超级计算机出现。全部500台系统的总性能为27.6Pflops，而半年前和一年前分别为22.6Pflops和16.9Pflops。TOP500的进入门槛已经达到20Tflops，此次排名中最后一位的系统在半年前的排行榜上位列第336位。可以看出，全球高性能计算发展势头强劲，系统性能节节攀升。

表 1.1 TOP500中性能排名前10的计算机情况

排名	计算机	国家/时间	制造商	CPU数	峰值/实测值(TF)	功耗(千瓦)
1	Jaguar	美国/2009	Cray	224162	2331.00/1759.00	6950.60
2	Roadrunner	美国/2009	IBM	122400	1375.78/1042.00	2345.50
3	Kraken XT5	美国/2009	Cray	98928	1028.85/831.70	N/A
4	JUGENE	德国/2009	IBM	294912	1002.70/825.50	2268.00
5	天河一号	中国/2009	NUDT	71680	1206.19/563.10	1484.80
6	Pleiades	美国/2008	SGI	56320	673.26/544.30	2348.00
7	BlueGene/L	美国/2007	IBM	212992	596.38/478.20	2329.60
8	BlueGene/P	美国/2007	IBM	163840	557.06/450.30	1260.00
9	Ranger	美国/2008	Sun	62976	579.38/433.20	2000.00
10	RedSky	美国/2009	Sun	41616	487.74/423.90	N/A

表 1.2 TOP500中总上榜计算机数量前10的国家

排名	国家	上榜数量	比例	总实测值（TF）	总峰值（TF）
1	美国	277	55.4	16416.1	24187.6
2	英国	45	9.0	1569.1	2457.5
3	德国	27	5.4	2288.8	2933.7
4	法国	26	5.2	1214.3	1827.0
5	中国	21	4.2	1379.9	2536.4
6	日本	16	3.2	994.2	1308.6
7	加拿大	9	1.8	439.8	661.9
8	澳大利亚	8	1.6	219.9	266.6
9	新西兰	8	1.6	233.2	422.7
10	俄罗斯	8	1.6	646.2	822.7

1.6.2　国内的并行计算发展

我国高性能计算机也一直以很高的水平发展，取得了很多令人瞩目的成就。我国的863、973等全国科技计划和历次五年计划都把高性能计算机研制

和应用放在重要地位。我国先后研制了曙光、银河、神威等高性能计算机。2004年6月，曙光公司研制的曙光4000A，运算能力达到峰值速度每秒11万亿次计算和实测速度每秒8万亿次计算，在当时全球TOP500超级计算机中排名第10。2008年11月，曙光5000A以峰值速度230万亿次，实测性能180万亿次的成绩再次跻身世界超级计算机前十。第34届TOP500排行榜上，国防科大研制的我国首台千万亿次超级计算机“天河一号”名列世界第五，亚洲第一。这是自1993年以来，我国超级计算机的历史最好成绩。“天河一号”超级计算机峰值速度和实测速度分别达到每秒1206.19万亿次和563.1万亿次。我国成为美国之外，又一个有能力研制千万亿次超级计算机的国家。随着曙光4000A、曙光5000A和天河一号超级计算机先后占据排行榜前列，标志着我国超级计算机研制能力已经达到世界先进水平。采用国产高性能龙芯通用处理器芯片和其他国产器件、设备和技术的KD系列万亿次高性能计算机的研制成功[33]，标志着我国高性能计算机国产化、个人化的重大突破。

1.7 并行计算存在的问题和不足

随着微处理器技术的不断革新、高速互联结构的进步以及软件技术不断突破，并行计算机硬件结构经历了PVP、SMP、MPP、COW等，计算能力已经达到了每秒千万亿次，相应的软件系统如并行操作系统、并行编程语言和并行编译器等也不断更新。并行计算的最终目的是应用，为了更好地求解应用问题，不仅要看并行机的硬件峰值，还需要关注其实际应用性能。超级计算机的硬件水平不断攀升，但是软件应用远远没有跟上，理论峰值和实际性能差距明显，峰值转换率低，实际并行应用在并行计算机上的性能远远低于硬件峰值性能，差距很大。大多数应用使用的处理器数目远远小于系统可用处理器个数，实际性能在理论峰值的20%左右徘徊，有些一应用仅达到峰值性能的5%-10%左右[34]。因此如何充分发挥并行计算机的计算能力，提高并行应用的效率是当今并行计算的重要研究问题[35,36]。并行应用的效率不高，并行软件严重滞后于硬件发展，主要原因可以概括为以下几点：

1. 并行计算缺乏成熟的理论支持。与串行计算相比，并行计算没有一个成熟的并行计算模型。串行计算得益于冯·诺依曼模型对串行计算机良好刻画，其上算法的设计和分析比较容易，而并行计算机由于体系结构的多样性，并行计算模型一直在发展变化，没有一个统一成熟的可以精确刻画并行计算行为的计算模型。

2. 并行编程难度大。目前，并行程序员需要掌握并行计算知识，对体系结

构，编译原理有一定了解，才可以写出高效的并行程序。无疑，这样并行软件的开发效率是很低的。需要提供更好的并行编程、调试工具，简化并行程序的编写，提高普通程序员并行编程的生产率，实现并行编程由计算专家到普通程序员的普及。

3. 并行应用程序优化比较复杂。影响并行程序执行性能的因素很多也很复杂，包括硬件和软件的诸多方面，以及程序本身的特性。机器硬件因素主要有处理单元参数、存储系统参数和互连网络参数等；软件因素包括编译器，操作系统、线程调度和同步等；程序本身因素包括：算法、并行性、访存行为和通信行为等。

4. 现有并行程序难以在不同体系上高效迁移。并行体系结构多样，针对一个体系结构优化的并行程序到另一个平台上运行时，并不能取得最优的性能。已有程序在新的硬件和软件环境下，难以发挥最好的性能。由于并行体系结构日新月异的发展，不同时期编写的程序针对不同的体系结构做了相应的优化。我们不可能在新的并行硬件环境下，完全淘汰之前编写的程序，因此需要做适当的调整以使已有的并行程序可以在新的硬件环境下发挥较高的性能。

综上，在追求高性能计算机系统规模不断增大，峰值速度持续提高的同时，并行应用开发和并行软件优化还需要更深入的研究，进一步关注系统的实际运行性能，提高并行计算机的利用率、可编程性和易维护性，以充分发挥并行系统的硬件优势。

1.8 论文研究思路、内容和成果

前面介绍了并行计算的研究背景、研究现状、新的发展趋势以及当前存在的主要问题。并行程序的实际执行性能是衡量并行计算机效用的主要指标，也是并行计算需要大力解决的问题。由于并行体系结构的多种多样，设计并行算法需要与具体的并行机硬件相匹配，才能发挥最好的性能。提高并行计算机实际应用性能，涉及到针对应用的特性和并行机硬件特性，设计高效的并行算法，优化程序在实际机器上的执行。本文研究思路是沿着基于共享存储的并行程序性能优化这一主线，研究共享存储系统上并行应用程序执行模型和优化技术，主要包括分层并行计算模型、并行程序性能优化和定量的并行程序执行模型。通过对应用和硬件特性进行建模，给出程序性能的衡量指标，从理论上分析和确定影响程序实际运行性能的瓶颈，然后利用并行程序优化技术消除或减弱这些性能瓶颈，最大限度的挖掘硬件能力，加快应用程序的执行速度，缩短计算

时间。

根据以上研究思路，本文从理论和应用两方面，研究共享存储系统上并行应用程序性能优化：一方面研究并行计算模型，回顾随着并行体系结构不断变化的三代并行计算模型，提出分层的并行计算模型，即把并行计算模型分为并行算法设计模型、并行程序设计模型和并行程序执行模型。本文重点研究其中一层：共享存储系统上的并行程序执行模型。程序执行模型与程序性能优化密切相关：程序执行模型可以指导程序性能优化方法和过程；程序性能优化方法反过来为程序执行模型提高思路和借鉴。因此，另一方面研究典型的并行应用程序在共享存储系统上的性能优化方法，通过实验测量和分析优化的结果，揭示影响程序性能的主要因素，为并行程序优化提供一个思路，同时有助于并行执行模型的建立，主要包括：MPI基本消息传递库在对SMP上的优化；计算生物学应用MFold程序在SMP上的优化；基于内容的图像检索系统（Content-based Image Retrieval，CBIR）在多核处理器上的优化。

本文主要取得了以下研究成果：

1. 分层的并行计算模型。深入研究了并行计算模型的演变过程，示例了不同并行计算模型上并行算法的设计方法，针对当前单一并行计算模型表达能力不足和实用性较差，提出并行计算模型进行分层研究的思想，改变传统单一的并行计算模型，按并行计算各阶段把并行计算模型分为并行算法设计模型、并行程序设计模型和并行程序执行模型三个层次，指出并行执行模型研究比较少并且难度大，是分层并行计算模型的研究重点。

2. 对称多处理机上MPI通信性能优化。研究了MPI在共享存储系统上消息通讯的设计思想，提出了一种共享存储系统上消息通讯的改进机制，并在此基础上实现了共享存储系统上基本的消息传递库，实验表明点对点通讯和集合通讯性能都取得了很大的提高，实际应用中的通讯性能也显著提高。

3. 对称多处理机上Mfold性能优化。优化Mfold在共享存储上的性能，减少了通信中的数据复制开销，减少了通信启动延迟时间。研究已有的针对分布式存储系统使用MPI编写的并行程序，如何高效地在共享存储上执行，取得令人满意的性能。

4. 多核处理器上CBIR性能优化。优化基于内容的图像检索系统在多核系统上的性能，给出了一套完整的应用软件性能优化方法和步骤，得出最后一级Cache（LLC）和前端总线带宽对于多核处理平台上应用程序性能有重要影响。研究目前已成为主流计算平台的多核处理器上的并行和优化方法，为更好的利用更大规模的多核处理器提供借鉴。

5. 定量化的并行程序执行模型。提出了面向片上多核处理器的定量化程序执行模型CRAM(h)，利用性能剖析器收集程序执行时的性能数据，进而对程序执行过程进行建模，定量地分析程序执行行为，指导程序性能优化方向和过程。

1.9 论文组织结构

论文共分7章，具体内容组织如下:

第一章为绪论，介绍并行计算的研究背景，包括并行计算的发展历史、并行计算机硬件体系结构、典型的并行程序设计模型和并行程序性能优化方法和步骤；介绍当前国际和国内并行计算的研究现状、新的趋势以及存在的问题，指出并行应用的性能优化是当前并行计算研究的重要问题；最后，介绍本文研究思路和内容，总结了研究成果，给出论文的组织结构。

第二章提出分层的并行计算模型，介绍了并行计算模型的基本概念，回顾了历史上并行计算模型的演化过程，总结了不同时期三代并行计算模型的特点，使用N体问题示例了各种并行计算模型上并行算法设计方法和性能分析过程；针对计算模型发展中遇到的问题，提出并行计算模型分层的研究方法，给出了各层模型的特点和研究内容。

第三章研究对称多处理机上MPI通信性能优化，首先介绍MPI通信协议在对称多处理机上的实现，分析存在的效率问题；接着提出了优化的通信实现方法，并理论分析其性能；最后通过实验测量优化带来的性能提升，对实验结果进行了详细的分析和讨论。

第四章研究对称多处理机上Mfold的并行优化，首先介绍RNA二级结构预测的基本概念和预测软件Mfold中的动态规划串行算法，分析算法中的数据依赖关系；接着使用对角线法实现串行算法的并行化，并且使用进程间通讯技术进一步优化并行算法的性能；最后实验测量并行Mfold算法的性能，分析讨论优化技术对性能的影响。

第五章研究多核系统上基于内容的图像检索系统的并行优化，首先介绍基于内容的图像检索（CBIR）的研究背景、多核处理器技术和OpenMP编程模型；接着设计了一个CBIR系统框架，给出了各模块实现方法；然后通过挖掘系统内线程级并行度、数据级并行度和指令级并行度，优化系统性能；最后在两台多核系统上进行了实验分析。

第六章提出定量化的并行程序执行模型CRAM(h)，首先总结已有的性能评测方法；接着研究当前影响并行程序性能的主要因素；然后对指令执行时间、存储访问时间和并行执行时间进行建模，得到CRAM(h)模型，使用性能剖析器

测量模型中应用软件和系统硬件相关参数；最后使用模型对矩阵相乘进行了实验分析，验证模型的可用性和精确性。

第七章为总结与展望，对全文的研究内容、成果和创新点进行总结，指出进一步可以进行的工作。

1.10 本章小结

本章首先介绍了并行计算的基本概念和研究背景，包括并行计算机硬件体系结构、并行编程环境、并行算法的概念和设计方法、并行应用的发展水平和并行程序性能优化技术；然后，介绍了当前国内外并行计算近年来的发展情况、新的趋势以及面临的问题，指出并行软件发展水平远远落后于并行硬件的发展水平；最后给出了本文的研究思路、内容、创新之处及组织结构。

第2章 分层并行计算模型

内容提要 并行计算模型是并行计算中软硬件之间的桥梁。本章对并行计算模型的研究做一个回顾，介绍并行计算模型的基本概念、发展现状，总结历史上不同时期典型的并行计算模型，根据发展历史把模型分为三代，讨论不同模型的适用范围，总结比较它们的特点，通过一个应用实例N体问题的求解算法，研究不同计算模型上并行算法的设计方法，给出这些模型上并行算法的设计模式，分析不同模型上算法的复杂度，比较各个模型上算法设计风格以及算法性能的差异。通过对已有模型的分析和比较，可以看出并行计算模型沿着丰富、强化单一模型的路线发展。单一模型变得越来越复杂，导致模型的不实用和不可操作性；本章研究并行计算模型的分层，根据并行计算过程，把并行计算模型分为并行算法设计模型、并行程序设计模型和并行程序执行模型，分别给出了各层模型的特点和研究内容。

2.1 并行计算模型相关工作

2.1.1 基本概念

计算模型是为了计算目的将真实体(计算机软/硬件) 进行一定的抽象而成的。通过建立清晰的确定的模型，才可能分析计算中所需的资源，如执行时间、空间开销，讨论算法和计算机的瓶颈。串行计算中，随机存取计算机RAM（Random Access Machine）模型[37]取得了很大的成功。RAM模型反映了采用冯·诺依曼结构的串行计算机的基本特性，代表了串行计算的主要特征，它认为每个计算操作和读写操作都可以在单位时间完成。

并行计算模型[11,12]是算法设计者所看到的参数化了的并行机，是提供给编程者的计算机软/硬件接口，是程序执行时的系统软/硬件支撑环境。并行计算模型是算法设计者、程序设计者和系统运行者在实现并行计算时所看到的虚拟并行计算机，是他们共同执行任务的桥梁。传统的并行计算模型一般指的是并行算法设计模型，为并行算法的研究者提供一个独立于具体并行机体系结构的抽象的并行机。并行计算模型从并行机中抽取若干能够反映计算特性的可计算或测量的参数，按照模型所定义的计算行为构造成本函数，从而可以被用来进行并行算法复杂度的分析。一般模型中通常定义了机器参数、计算行为和开销函数，它们被称为计算模型三要素。由于并行体系结构的多样性，构造一个真正

可用的并行计算模型具有很大的挑战性，迄今为止，并没有一个像串行RAM模型那样的统一的并行计算模型来准确描述并行机的体系结构，来指导并行算法的设计，多年来国内外并行计算领域的研究者一直在探索。

并行计算模型跟着并行机硬件体系结构不断发展变化，并行性和局部性一直是影响并行机系统的主要因素，计算模型也主要抽取计算机的这两大特性。根据并行计算　型的发展历史，可以把并行计算模型分为三代[12]。早期的并行机为共享存储器的SIMD和MIMD的计算机，相应地，提出了以计算为核心的所谓第一代并行计算模型。后来出现了分布存储的大规模并行机MPP，相应地提出了以网络通信为核心的所谓第二代并行计算机模型。最近分布共享的并行机成为主流并行机，以及CPU和主存之间的速度差异越来越大，存储系统逐渐成为影响系统性能的主要瓶颈，相应地，提出了以大规模并行和存储访问为核心的所谓第三代并行计算机模型。目前还没有一个成熟实用的模型准确刻画并行算法性能，如何从错综复杂的因素中抽取影响性能的关键要素，是并行计算模型研究的难点。本章通过对一个计算科学中的实际问题（N体问题）设计并行算法，展示不同并行计算模型上的并行算法设计方法和性能分析方法。在以下各小节中，首先给出N体问题的描述和串行算法，然后分别给出各种模型上的并行算法。

算法 2.1: N体问题求解的串行算法

输入: 空间中N个粒子的状态信息，包括初始速度，位置等信息

输出: 经过一个时间步后所有N个粒子的新的状态信息

```
1  读入N个粒子的初始信息;
2  for (i = 1 ... N) do
3  |  for (j = 1 ... N) do
4  |  |  if (i ≠ j) then
5  |  |  |  计算粒子j对粒子i的作用力，并且累加
6  |  |  end
7  |  end
8  end
9  for (i = 1 ... N) do
10 |  根据牛顿定律和粒子i的受力情况，更新粒子i的状态信息
11 end
```

2.1.2 N体问题及其串行算法

N体问题可以描述如下：在一定的物理空间中，分布有N个粒子，每对粒子间都存在相互作用力（如万有引力，库仑力等）。它们从一个初始的状态开始，每隔一定的时间步，由于粒子间的相互作用，粒子的状态会有一个增量，需要对粒子的加速度、速度和位置信息进行更新。算法2.1 给出N体问题的串行算法。在算法2.1中需要计算$N(N-1)$次受力，故此算法的时间复杂度为$O(N^2)$。

N体模拟问题在天体物理、分子动力学等很多方面都有重要的应用。一般模拟的粒子的规模都很大，一个体系中可以包含数百万乃至上千万的粒子，直接计算的话$O(N^2)$的量级对于任何高性能的单个处理器都是一个难以突破的瓶颈。实际应用中人们一般采取并行编程的方法，把计算任务分配到多个处理器上并行完成。

2.1.3 共享存储模型

2.1.3.1 PRAM模型

PRAM[38,39]模型是串行RAM模型的一个并行扩展，是一种理想的并行计算模型，属于共享存储的同步计算模型。PRAM模型中包含若干个独立的RAM处理器和一个全局的共享存储器，这些处理器功能相同，具有算术运算和逻辑运算的功能，全局的共享存储器容量无限大，任何时刻每个处理器都可以在单位时间内访问全局存储单元，各处理器同步地协同完成计算任务，都按照存储器数据读取、处理器执行计算、存储器数据回写的步骤执行。PRAM模型的两大特点是共享与同步。PRAM模型使用简单，易于并行算法的分析和表达，许多诸如处理器间通信、存储管理以及进程同步等并行算法的细节均隐含在模型中。在PRAM 模型中并行机被理想化了，单位时间内存存取、计算和访存隐式同步以及忽略同步开销。使用PRAM模型进行算法分析时，只考虑算术运算执行时间，不把存储访问开销计入算法的时间复杂度。

P个处理器各自负责计算N/P个粒子的受力以及对这些粒子的状态进行更新，由于PRAM模型中采用共享存储，任何时刻每个处理器都可以从共享存储中读取数据，因此不需要考虑通信的问题，具体算法如算法2.2所示。算法2.2把原来串行算法的$O(N^2)$的计算量平均分配给P个处理器，总的执行时间为$O(N^2/P)$。

在PRAM模型基础上增加对存储器同时读和同时写的限制后，PRAM模型又可以分为三种：不允许同时读和写的PRAM-EREW；允许同时读但不允许同时写的PRAM-CREW；既允许同时读又允许同时写的PRAM-CRCW。根据写操作的执行者和数据，又可以进一步把PRAM-CRCW 细分为：CPRAM-CRCW，

算法 2.2: PRAM模型上的N体问题求解的并行算法

输入: 空间中N个粒子的状态信息，包括初始速度，位置等信息

输出: 经过一个时间步后所有N个粒子的新的状态信息

```
1  每个处理器读入N/P个粒子的初始信息;
2  for (all P_i, where 0 ≤ i ≤ P − 1) do
3  |  for (j = i × N/P ... (i + 1) × N/P) do
4  |  |  for (k = 1 ... N) do
5  |  |  |  if (j ≠ k) then
6  |  |  |  |  计算粒子k对粒子j的作用力，并且累加;
7  |  |  |  end
8  |  |  end
9  |  end
10 end
11 for (all P_i, where 0 ≤ i ≤ P − 1) do
12 |  for (j = i × N/P ... (i + 1) × N/P) do
13 |  |  根据牛顿定律和粒子j的受力情况，更新粒子j的状态信息;
14 |  end
15 end
```

所有的处理器可以同时写相同的数据；PPRAM-CRCW，优先级最高的处理器执行写操作；APRAM-CRCW，所有的处理器可以自由执行写操作，存储器状态为最后一个执行写操作的处理器写的结果。

PRAM同步性决定了所有的指令均按锁步方式操作，这种方式是很费时的；假定对存储器的访问没有存取竞争和带宽限制是不现实的，所有操作均取单位时间是不实际的。下一节介绍一种对它的推广和改进模型APRAM。

2.1.3.2 APRAM模型

APRAM[40]模型是一个异步模型，包含了P个处理器，每个处理器都有其局部存储器、局部时钟和局部程序，处理器间通过共享存储单元交换信息，无全局时钟，所有的处理器独立执行各自指令，每条指令可以在非确定但有限的时间内完成，利用显式同步路障来处理依赖关系。在APRAM中，计算是由一系列用同步路障分开的全局相组成，指令分为四类，分别为全局读、局部操作、全局写和同步。在各全局相内，每个处理器异步地运行其局部程序，每个局部程序中的最后一条指令是一条同步指令；每个处理器可以异步地读取和写入全局

存储器，全局读写的开销记为d，但在同一相内，不允许两个处理器访问同一单元。同步障是计算中的一个逻辑点，在该点每个处理器均需等待别的处理器到达后才能继续执行其局部程序。

算法 2.3: APRAM模型上的N体问题求解的并行算法

输入: 空间中N个粒子的状态信息，包括初始速度，位置等信息

输出: 经过一个时间步后所有N个粒子的新的状态信息

```
1  每个处理器处理N/P个粒子，读入N/P个粒子的初始状态信息；
2  每个处理器将其上N/P个粒子写入到共享单元SM中；
3  Barrier;/*实施路障同步*/;
4  for (all P_i, where 0 ≤ i ≤ P − 1) do
5  |  for (j = 1 ... N/P) do
6  |  |  for (k = 1 ... P − 1) do
7  |  |  |  for (l = 1 ... N/P) do
8  |  |  |  |  u = [(i + k)%P] × (N/P) + l
9  |  |  |  |  计算P_i中粒子j和共享单元中粒子u的作用力，并且累加；
10 |  |  |  end
11 |  |  end
12 |  |  Barrier;/*实施路障同步*/;
13 |  end
14 end
15 for (all P_i, where 0 ≤ i ≤ P − 1) do
16 |  计算P_i中N/P个粒子间的作用力，并且累加；
17 end
18 for (all P_i, where 0 ≤ i ≤ P − 1) do
19 |  for (j = 1 ... N/P) do
20 |  |  根据牛顿定律和粒子j的受力情况，更新粒子j的状态信息；
21 |  end
22 end
```

每个处理器局部存储中保存N/P个粒子的信息，通过全局写操作，P个处理器复制一份自己局存中的粒子信息到共享单元中，在计算时需要用到的其它处理器上的信息可以通过全局读从共享单元中读取。为防止多个处理器同时读同一个共享单元，每个处理器对共享单元中的粒子读时采取不同的顺序，实现方法见算法2.3。算法2.3中每个处理器中的计算时间仍为$O(N^2/P)$，在计算某一

粒子的受力时需要的$N-1$个粒子信息有$(P-1)\times N/P$个要从共享单元中读取，需要的时间为$d\times(P-1)\times N/P$。因此算法中的时间为$O(N^2/P+d\times N)$。

APRAM模型比PRAM模型更接近于实际的并行机，但是APRAM模型仍然是一个共享存储模型，不适用于分布存储的MIMD机型。

算法 2.4: BSP模型上的N体问题求解的并行算法

输入: 空间中N个粒子的状态信息，包括初始速度，位置等信息

输出: 经过一个时间步后所有N个粒子的新的状态信息

```
每个处理器处理N/P个粒子，读入N/P个粒子的初始信息；
for (all P_i, where 0 ≤ i ≤ P − 1) do
    计算P_i内部粒子间的作用力；
    P_i把其内N/P个粒子的状态信息发送给其右邻居；
    Pi接收其左邻居发送过来的粒子信息；
end
Barrier；
for (all P_i, where 0 ≤ i ≤ P − 1) do
    for (j = 1 ... P − 2) do
        Pi计算其内粒子和收到的粒子间的作用力，并且累加；
        Pi把其接收到粒子的状态信息发送给其右邻居；
        Pi接收其左邻居发送过来的粒子信息；
        Barrier；
    end
end
for (all P_i, where 0 ≤ i ≤ P − 1) do
    Pi计算其内粒子和收到的粒子间的作用力，并且累加；
    for (j = 1 ... N/P) do
        根据牛顿定律和粒子j的受力情况，更新粒子j的状态信息；
    end
end
```

2.1.4　分布存储模型

2.1.4.1　BSP模型

BSP（Bulk Synchronous Parallel）[41]模型的目的提供一种可以作为高级语言目标机器又可以被大多数硬件结构有效实现的并行计算机体系结构模型。

BSP模型是一种分布存储的MIMD模型，把并行计算机抽象为三个独立的模块：处理器/储存器模块，统称为处理单元，执行运算和存储访问操作；选路器，执行处理单元之间点对点传递消息；路障同步器，执行以时间间隔L为周期的同步操作。相应地，BSP模型中定义了三个参数：P（处理器/存储器数）；g（选路器的带宽因子，即连续发送或接收消息的时间间隔）；L（全局同步周期）。BSP模型的计算由一系列周期为L的超级步组成，每个超级步后要进行显式的同步。每个超级步可以分为有序的三个部分：本地计算、全局通信和路障同步。在每一超级步开始阶段，每一处理器/存储体部件可以进行使用本地可用数据的计算，这些数据来自本地计算或者上一超级步中通信操作获得的数据；计算完成后进行处理器间数据通信，通信以点对点的方式进行；在每L个时间单位周期之后，所有处理单元进行路障同步，执行一次全局的检查以确保所有的部件都完成了一个超级步中的计算和通信操作。上述执行过程如图2.1所示。

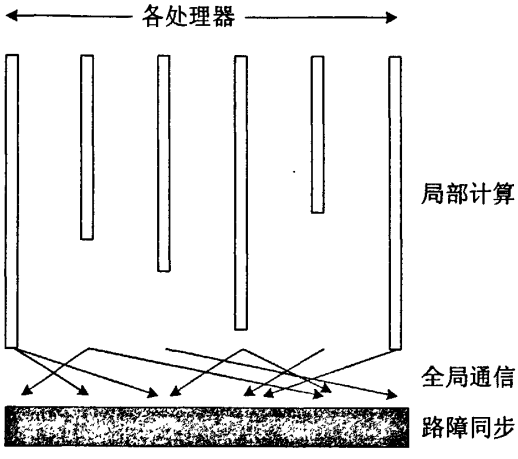


图 2.1 BSP模型中超级步的计算过程示意图

BSP模型通过限制每一超级步内所允许发送或接收的最大消息个数为$h = L/g$，来加入带宽的限制，假设P台处理器同时传送h个字节的数据，则$g \cdot h$就是通信代价。

BSP模型把处理器和选路器分开，强调了计算和通信的分开；选路器支持点到点的消息传递，不支持组合和广播功能，由此隐藏了网络的具体拓扑结构，简化了通信协议；采用硬件同步路障来实现全局同步，比较容控制同步的粒度；一些重要的算法可直接用BSP模型实现，在PRAM模型下的算法可以容易地移植到BSP模型；BSP模型可以有效地在若干互连网络技术上实现。

BSP模型上N体问题并行算法如算法2.4所示。每个处理器负责处理N/P个粒子,初始时每个处理器上保存有N/P个粒子的信息。首先计算每个处理器内部

粒子间的相互作用力，接着发送本地粒子信息到右邻居，并且接收其左邻居发送过来的粒子信息。在后面的每个超级步内，每个处理器按如下的方式进行计算和通信，每个处理器计算收到粒子和本地粒子间的相互作用力，然后发送接收到粒子信息到其右邻居，接收其左邻居发送的粒子信息。这样经过$P-1$个超级步后，每台处理器上的粒子信息正好可以传递给所有的处理器，完成所有粒子间的相互作用力计算。这样每个超级步由$(N/P)^2$个计算，N/P个粒子信息的发送和N/P个粒子信息的接收，以及开销为B的路障同步组成，那么可以取$L=(N/P)^2+2\times N/P\times g+B$。

该算法在每个超级步内，两台处理器间的粒子间相互作用力计算时间为$(N/P)^2$，通信时间为$2\times N/P\times g$，每次需要B的路障同步时间，总共需要的超级步数为$O(P)$，所以算法的执行时间为:$O(P\times((N/P)^2+2\times(N/P)\times g+B))=O(N^2/P+2\times N\times g+B\times P)$。

BSP模型也有许多缺点，在一个超级步中发送的消息，即使网络延迟比超级步短，也只能在下一个超级步中使用，BSP模型中的全局同步路障是特殊的硬件支持的，这在很多的并行机中没有相应的实现。LogP模型对BSP的这些缺点做了改进。

2.1.4.2　LogP模型

LogP[42]模型是一个分布存储的，点到点通信的异步并行计算模型。LogP模型有4个参数：（1）L（最大通信延迟）：源处理器与目的处理器之间进行消息通信所需要的等待延迟；（2）o（通信开销）：处理器准备发送或准备接收每个消息的时间开销，这段时间内处理器不能执行其它操作；（3）g（通信间距）：一台处理机连续两次发送或者连续两次接收消息的最小时间间隔，其倒数即为处理器的通信带宽；（4）P（处理器/存储模块数）：处理器的个数。LogP模型对网络容量进行了限制，同一时刻任何处理器最多可以接收或发送$[L/g]$条消息，消息长度超过网络容量，则处理器等待。在LogP模型下两个处理器间传递大小为M的信息包的开销，可计算如下：$o+L+M\times g+o=2\times o+L+M\times g$。

该模型上设计的算法和BSP模型上相似，只是算法中不再有超级步的概念，所有的进程异步的执行，通过消息传递显式地同步，不像在BSP在每个超级步中，所有的进程需要在超级步的最后进行同步，处理器接收到消息后可以立即在后面的计算中使用，充分利用了处理器的计算资源，具体算法如算法2.5所示。

所有$O(N^2)$的计算平均分配到P个处理器上执行，因此计算时间为$O(N^2/P)$。每个处理器需要进行$P-1$次消息传递，每次传递N/P个粒子的信息。处理器传递大小为N/P的信息包的开销为$2\times o+L+(N/P)\times g$，而每个处理器需要和其

算法 2.5: LogP模型上的N体问题求解的并行算法

输入: 空间中N个粒子的状态信息，包括初始速度，位置等信息

输出: 经过一个时间步后所有N个粒子的新的状态信息

```
1  每个处理器处理N/P个粒子，读入N/P个粒子的初始信息；
2  for (all P_i, where 0 ≤ i ≤ P − 1) do
3  |  计算P_i内部粒子间的作用力；
4  end
5  for (all P_i, where 0 ≤ i ≤ P − 1) do
6  |  P_i把其内N/P个粒子的状态信息发送给其右邻居；
7  |  Pi接收其左邻居发送过来的粒子信息；
8  |  Pi计算其内粒子和收到的粒子间的作用力；
9  end
10 for (all P_i, where 0 ≤ i ≤ P − 1) do
11 |  for (j = 1 ... P − 1) do
12 |  |  Pi把其接收到粒子的状态信息发送给其右邻居；
13 |  |  Pi接收其左邻居发送过来的粒子信息；
14 |  |  Pi计算其内粒子和收到的粒子间的作用力，并且累加；
15 |  end
16 end
17 for (all P_i, where 0 ≤ i ≤ P − 1) do
18 |  for (j = 1 ... N/P) do
19 |  |  根据牛顿定律和粒子j的受力情况，更新粒子j的状态信息；
20 |  end
21 end
```

它$P-1$个处理器进行通信，通信开销为$(P-1)\times(2\times o+L+(N/P)\times g)$，$P$个处理器总的通信开销为$P\times(P-1)\times(2\times o+L+(N/P)\times g)$，算法总的执行时间为$O(N^2/P+(P-1)\times(2\times o+L+(N/P)\times g))$。

LogP模型充分揭示了分布存储的并行机的性能瓶颈在于通信。它用L，o，g三个参数刻画了通信网络，但掩盖了网络的拓扑结构、选路算法和通信协议等具体细节，没有考虑不同大小消息长度的影响，也没有考虑通信时的网络拥塞和资源竞争。LogGP[43]模型增加了一个参数G，来体现长消息对通信性能的影响。LoGPC[44]模型进一步扩展，加入了网络冲突对通信性能的影响。

2.1.4.3 NHBL模型

随着网络技术和单个工作站性能的提高，机群系统（COW）成为越来越流行的并行计算平台。这种计算环境不同于以前的SMP和MPP等大型并行机，主要表现在它的异质性和非独占性等特性。这使得前面的并行计算模型不能准确地反应这类系统的计算行为。NHBL模型[45]正是针对机群系统得特性提出的并行计算模型。NHBL模型认为机群系统中的每个工作站计算速度是有差别的，并且整个系统资源不是被某一个并行计算任务独占，同时会有其它的用户计算在执行，并行计算任务有可能被暂时挂起，而且每个工作站上的其它用户计算量也各不相同。NHBL中的计算不在是原有串行计算时间的$1/P$，而是为无其它用户负载情况下的计算时间和其它进程抢占的计算时间的总和。NHBL中的通讯采用LogGP模型。该模型上设计的算法和算法2.5基本相同，只是算法分析时要考虑实际的程序运行环境，把其它用户负载对并行算法的影响考虑进来，除此之外算法设计和分析都跟LogP上类似，这里不再给出详细的设计和分析过程。在NHBL基础上，有人提出了考虑网络竞争的NHCBL模型[46] 以及扩展通信开销为发送和接收两部分的ENHBL模型[47]。

2.1.5 考虑存储访问的模型

由于处理器和内存之间越来越大的速度差异，访存开销变得不可忽略。有几个并行计算模型考虑改变RAM模型[37]的单位时间存储访问假定，在模型中引入对存储层次性能的分析。考虑存储层的并行计算模型，设计算法时存储访问不再是单位时间，不同数据的访问开销不同。因此算法需要考虑存储访问对性能的影响，设计高存储访问性能的算法。把非均匀存储访问开销引入到并行计算模型中来，大大增加了模型分析的难度。因为程序中访存数量众多，模式也多种多样，使用模型精确分析存储行为变得非常复杂，对于一些具有复杂访存行为的程序往往代价很高，一般凭经验粗略分析存储性能。以存储为核心的模型随着发展，对存储系统的刻画越来越细致，同时模型分析难度也越来越大，本小节仅介绍各模型的特性，不再给出N体问题具体的算法实现和分析。

2.1.5.1 HMM模型和BT模型

两个早期串行模型HMM（Hierarchical Memory Model）[48]和BT（Hierarchical Memory Model with Block Transfer）[49] 是由Alok Aggarwal等提出来的，考虑了不同存储单元性能的差异。HMM模型假定有k级存储层次，每级有2^k个存储单元。对地址为a的存储单元的访问开销为$f(a)$，$f(a)$是a的单调上升函数。BT模型考虑了数据访问时块传输的性能优势，对一个以地址a开始的大小为b的数据块的访问，需要的开销为$f(a)+b$。这两个模型的并行版本P-HMM 和P-BT，可

以通过分别复制p份串行模型得到。

2.1.5.2　UMH模型和PMH模型

UMH（Uniform Memory Hierarchy）[50]模型反映了串行计算机存储器和通信的诸多限制，考虑了存储系统的分层，认为对每层存储单元中数据的访问开销是存储层次参数的函数。UMH模型将计算机存储器视为存储容量逐渐增大的一组存储模块，相邻两级存储层次之间通过总线相连，相邻两层之间数据以固定大小的块为单位在总线上传输，不同层次的总线是独立的，数据可以在不同总线上同时传输。离处理器越远，存储层次的容量越大，数据块长度越大，单个块传输耗时越长。UMH忽略的存储访问中的空间和时间局部性，在该模型中数据的每次访问总是从存储层次的底层向高层传递，没有考虑数据在较高层次中可能已经存在，可以重复利用。

PMH（Parallel Memory Hierarchy）[51]模型，可以看作是UMH的并行版本。PMH使用统一的机制实现处理器间通信和存储层次间的数据传递，把并行计算机系统看作是存储模块和处理器组成的树，根节点和中间节点均为存储器，叶子节点为处理器，每个子节点有唯一的通道和它的父节点连接，数据以块的形式在子节点和父节点之间传输，所有的通道可以同时处于活动状态，但是同一数据在同一时刻只能在一个通道传输。PMH中每个存储器模块用四个参数刻画特性：单数据块大小、数据块个数、子节点数和数据传输延迟。含有p个处理器的PRAM模型是PMH模型的一个特例，可以被看作是只含两层，根节点代表整个全局存储器，p个处理器为叶子节点，数据块为单位大小，个数为整个存储容量的大小，传输时间为单位时间。使用树结构来抽象并行计算机系统是在PRAM模型和任意图模型之间的一个折中。

2.1.5.3　DRAM(h)模型

DRAM(h)[52]模型抽象并行计算机为通过互联网把p个处理单元连接起来，不考虑具体网络拓扑结构，每个处理单元有本地存储器，存储器为多层次存储结构，各处理单元之间通过点对点消息传递进行通信，并且把远地存储看作本地存储的扩展，提出了存储复杂度的概念，把数据的空间和时间重用对存储系统性能影响考虑到模型中来，认为数据的访问时间取决于每层命中次数和每层的访存延迟。DRAM(h)模型把RAM中只有一层单位时间开销的存储系统扩展为h层存储，距离处理器的远近决定了某一存储层次的访问速度快慢。算法刚开始执行的时候，所需的数据存放在离处理器最远的存储层次，当计算需要的时候逐层按块传输到离处理器最近的一层，在此过程中，每一层都将会保留该数据的一份拷贝，直到计算结束或者由于某一层容量不足，被存储控制系统按照

某种替换算法，用其他数据替换出去。随后再次访问曾经被访问过的数据，可以直接从距离处理器最近的存储层次中开始数据传输，而不是像UMH模型总是需要从最远的一层开始数据传输。邻近层次间数据以块为单位进行数据传递，对一个数据的访问会导致该数据所在的整个数据块被从最远的存储层次取到最近的存储层次中，不同层次间块大小是不同的，往靠近处理器的方向逐层递减或者不变，下一次对该块数据块中的其他数据访问，根据情况也不一定需要从最远的存储层次开始传输。每次存储访问的开销由数据所驻留的离处理器最近的存储层次数和数据重用的能力决定。数据所在的层次里处理器越近，数据传输开销越小；数据的重用性越好，数据传输开销也越小。对于远程存储器的访问认为是本地存储访问的扩展，把远程存储器看作整个存储层次的一级，并用LogP模型来分析单次传递开销。DRAM(h) 模型鼓励用户优化数据在不同存储层次间的移动，减少访问存储系统的开销。该模型没有将磁盘访问纳入存储层次，忽略了对TLB访问的分析，这在某些情况下会影响分析的精度。但即使这样，使用DRAM(h)来分析算法复杂度已经相当困难，仅用于访存模型单一或比较简单计算内核程序，对于大规模的实际应用程序分析起来比较繁琐。怎样简化分析方法且不要丢失太多的精度，是个值得研究的问题。

2.1.5.4　HPM模型

HPM（Hierarchical Parallelism and hierarchical Memories）[53]模型抽象了并行计算机的层次并行和层次存储两个特性，定义了并行函数描述系统的多层并行性，定义了存储函数描述存储层次特点，定义了绑定关系给出了并行和层次存储之间的连接关系。HPM模型把一般并行计算模型的单机RAM处理器扩展为ERAM处理器，加入了层次存储和处理器内部并行的概念。并行函数递归定义了整个并行系统是由一些列规模小的并行系统组织成的树状结构。存储函数定义了广义的层次存储结构，每层存储层次由一般意义上的本层存储结构和基于该层的通信网络构成。绑定函数给出了各并行层次上子系统之间基于层次存储进行的信息交换的内在联系，定义了子系统之间的同步组织方式并形成层次关系树。

2.2　分层的并行计算模型

2.2.1　单一模型存在的问题

根据前文对各种模型的介绍、分析和比较，可以看出并行计算模型沿着丰富、强化单一模型的路线发展。随着并行计算机体系结构飞速发展，为了使计算模型能够反映体系结构的变化，人们不断的向该单一模型中加入旨在反映机

器特性的新的参数，调整计算行为，修改开销函数。利用并行计算所能获得的性能是以下因素的综合体现：算法、实现、编译器、操作系统、处理器体系结构和网络互连技术等，因此实际的计算行为相当复杂。一味追求单一模型的功能强和多目标，致使单一模型越来越复杂，最终导致模型不实用和不可操作。单一模型中，反映机器不同特性的参数过多，会使单一模型描述越来越困难，最终导致成本函数过于复杂而无法求解。当在单一模型中又考虑到机器的底层特性和不同的硬件并行度时，模型很难建立。对算法设计者而言，单一模型太复杂，设计算法时考虑的因素过多，影响设计低时空开销的优秀算法。此外，传统的并行计算模型，主要是为算法研究者开发的，比较抽象和理论，只关心并行算法的设计与分析，不考虑算法的具体实现和执行，在实际并行应用执行时的性能分析中难以使用。

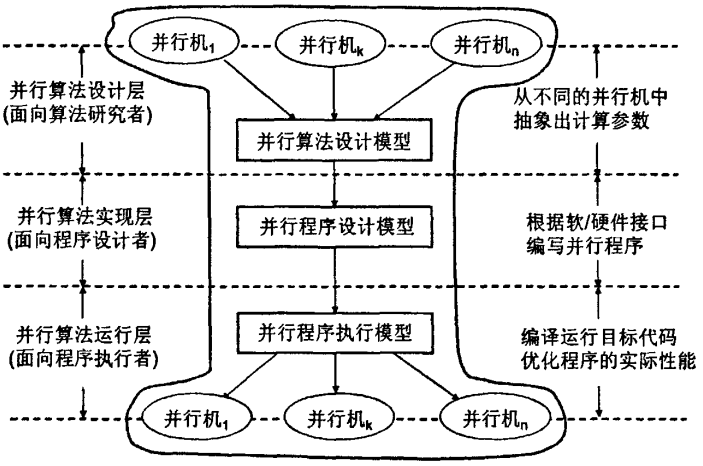


图 2.2 分层并行计算模型示意图

2.2.2　分层模型的概念

并行计算的一般过程如下，从给定的计算问题开始，首先用伪代码设计并行算法，然后用高级语言实现算法，最后编译成机器代码在实际并行机上运行。整个并行计算过程表现出来的性能取决于一系列的因素，算法的伪代码数和通信次数，决定了算法的时空复杂度和通信复杂度；程序设计语言、编译器、运行环境的选择决定了每条源级语句转换成机器指令的数量和优化的程度；操作系统、处理器、存储系统、体系结构决定了每条机器指令实际执行的速度，I/O系统决定了整个程序I/O操作的数量。为了获得在实际并行机上高的编程效率和高的运行效率，并行计算的各个阶段都要在各自模型的限制和约束下来进行。因此，并行计算模型下一步研究应该分为并行算法设计、并行程序设计和

并行程序执行三个层次来考虑[54]。通过分层，在并行计算的各个阶段，模型的职能各有侧重，分工明确，目标单一，从而适合不同阶段的设计人员关注主要的问题。按照分层的观点，并行计算模型可分为：并行算法设计模型、并行程序设计模型和并行程序执行模型。

并行算法设计模型，面向并行算法研究者，是算法设计者和计算机体系结构家之间的桥梁，从算法的角度，将不同的并行机抽象为一种通用虚拟并行机，算法设计者在其上设计和分析并行算法。并行程序设计模型，面向并行程序设计者，是程序设计者与计算机软/硬件的接口，从编程的角度指导程序员，按照程序的执行流程，选用某种并行语言，正确地编程实现某并行算法。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，将不同的并行语言实现的程序，在具体的并行机上编译、优化和运行并行程序。

分层计算模型可将单一模型中的功能按要求分配到模型不同的层次中，缓解了单一计算模型的精确性与可使用性之间的矛盾。分层后，各层次模型职能不同，目标单一，各负其责，易于设计与实现。分层并行计算模型中的并行算法设计模型和并行程序设计模型目前相对比较成熟，从而可集中精力重点研究并行程序执行模型。三层并行计算模型功能如图2.2所示。三层并行计算模型从几何形状上看，呈现哑铃形状：从不同的并行计算机来(抽象计算参数建立模型)，经过统一的加工后（用语言编程实现算法），又回到不同的并行计算机中去(运行代码，求解问题)。以下各小节分别给出每个模型的定义、参数、行为描述以及当前已有的相关模型。

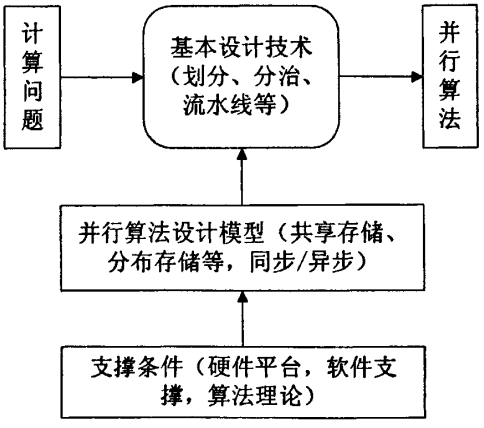


图 2.3 并行算法设计模型示意图

2.2.3　并行算法设计模型

算法设计模型需要反映机器硬件关键特性同时又必须非常简单易于算法描述和表达，是算法设计者与机器结构家之间的桥梁，是面向算法研究者的。算法设计模型重点关注算法的设计原理，确保设计出来的算法的正确性和较低的时间、空间复杂度。算法设计模型可归纳出如下的三要素：机器参数（抽象出的CPU，Memory，I/O网络参数等）、执行行为（算法的同步，异步执行等）、成本函数（算法的复杂度函数，它是机器参数的函数）。并行算法设计模型的功能特性如图2.3所示。长期以来计算机科学家大多都是研究并行算法的设计模型，前文详细介绍了在不同时期提出的各种适用当时并行机体系结构的模型。

并行算法设计模型是并行计算理论主要的研究内容之一，理论性较强，所以历史上特别是对第一代并行计算模型，理论计算机科学家们做出了诸多的贡献。随着并行机的发展，不少计算机结构科学家，参与了所谓第二代并行计算模型的研究。到了第三代并行计算模型的研究，涉及到了并行机底层的很多硬件知识，所以参与研究该模型的科学家越来越广泛。并行算法设计模型的出现，首先使得并行算法的研究，不仅仅再限于针对某台具体的并行计算机专门设计和分析并行算法，而更重要的是，可在一类抽象的并行机上研究算法，从而使得并行算法本身更趋普适化，可广泛适用于某一类并行机。其次，在并行算法设计模型上设计出的并行算法，可以利用数学工具严格定量的分析其性能，这样使得并行算法学科本身更加定量化更趋成熟。最后，因为有了统一的并行算法设计模型，这样在其上设计出的很多不同的并行算法，可以根据算法的复杂度来相互比较优劣。

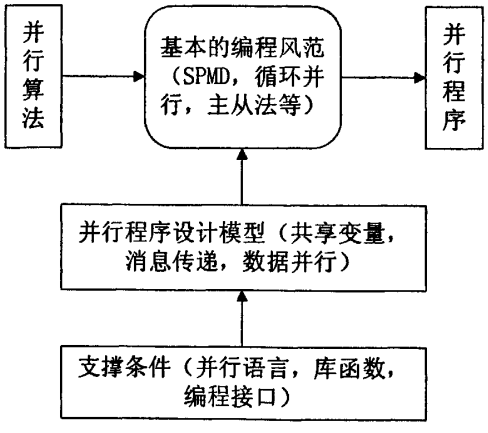


图 2.4 并行程序设计模型示意图

2.2.4　并行程序设计模型

并行算法设计出来后，人们需要考虑如何正确、方便、快速的在某种并行程序设计模型下用某种编程语言实现之。并行程序设计模型也可以称为并行程序编程模型，是提供给程序员使用的，为程序员提供了一些计算机软/硬件的编程接口。程序设计模型主要确保并行算法使用某种程序设计语言可以正确的在并行计算机上编程实现，在此基础上还应能满足：高性能、可移植性和可编程性。并行计算的快速发展，使得多种体系结构并存，不同体系结构差别很大，高性能总是要利用到具体体系结构特性，而可移植性要求一台机器上编写的程序可以在其他并行机上执行，因此高性能和可移植性之间存在矛盾。可编程性要求并行程序设计模型应该尽量缩减与传统串行编程模型的差异，让串行程序设计人员可以快速的掌握并行程序编写的方法，降低大规模并行程序设计的难度，提高并行程序生产力。可编程性要求语言抽象尽可能简单，这又与高性能和可移植性矛盾。

程序设计模型需要有自己的高级语言或者对当前语言进行扩展，用语言来对程序员进行约束，让他们把算法在该语言下实现。并行程序设计模型的功能特性如图2.4所示。本文第一章中介绍了各种体系结构上适用的编程模型。当今流行的并行程序设计模型主要有大粒度的进程级的消息传递模型，如MPI；中、细粒度的线程级的共享变量模型，如OpenMP、Pthread；细粒度进程级的数据并行模型，如HPF。

并行计算机体系结构的快速发展，多核桌面计算机的普及，更大规模并行计算机的设计与研制，给并行程序设计模型的研究带来新的机遇和挑战。并行性已经广泛存在于个人桌面计算机，要充分利用这些计算资源，就需要设计更加易于使用的并行编程模型，来满足普通程序员开发并行程序的需求。由于并行程序要比串行程序复杂的多、并行执行的不确定性、易出错性，并行程序设计一般由并行领域的专业人员来完成，普通程序员进行并行编程较困难。多个线程同时运行时，通常有对共享数据的访问，会带来数据竞争的问题，最终的执行结果跟多个线程对共享数据区读写的先后顺序有关。另一方面并行程序执行时，每次执行的指令流的顺序是有差别的，执行中遇到的错误往往很难重现，这就导致了对并行程序执行的分析、调试、优化非常困难。一般来说不同的并行算法会有一些共同的结构，可以抽取出这些结构，把它们放到编程模型中。这样程序员编程时可以集中在应用问题本身的实现，考虑如何把算法映射到适合的编程模型，其他并行处理相关的细节，诸如通信如何完成、任务怎么映射到各个线程、如何进行同步等，不需要在编程时显式考虑，利用编程模型提供的一些接口，并行相关细节都交给运行时环境去处理。一些常见的并行编程风

范有fork/join、pipeline、work-farms、meshes等模型。这些模型可以解决很大范围的问题，并且它们都是经过仔细检验和多次验证，可以保证开发出来的并行程序的正确性。模型中定义了程序的控制流，所有并行相关的部分都放在编程模型中解决，应用相关的一些代码就可以直接使用已有的串行程序，程序员只需要把这些应用相关的代码填入到控制流的各个阶段，这样一个并行程序就可以快速的设计出来了。近年来研究者们基于并行编程模型的思想开发了许多简化并行程序设计的支持系统，如MapReduce[23]，Dryad[55]，Hadoop[56]。这些系统都在保证程序正确性的前提下，尽量使得程序员从高层入手，不需要有并行编程的经验，就可以很容易的编写出高效的可并行执行的程序。目前这些系统支持的应用还有一定的局限性，只有能和系统提供的编程模型匹配的应用才可以很好的在以上系统上实现，主要应用在诸如信息检索、数据挖掘等领域。随着研究的深入，它们支持的编程模型也将越来越多。同时，这些缺陷也是新的并行编程模型研究的机遇和挑战。

并行编程语言是并行程序设计模型的一个重要的组成部分。面对多核甚至众核技术的迅猛发展以及千万亿次并行机时代的到来，学术界和工业界对高生产率并行编程语言需求迫切，目前新一代并行编程语言的研究正日渐升温。Stanford大学开发的Sequoia面向Cell多核处理器和机群系统，提供对存储层次进行程序设计的方法，通过把存储层次暴露给程序员，允许他们显式的对数据在各级存储中进行分配，在保证高可移植性的前提下，取得了很好的性能和效率。美国HPCS（高生产率计算系统）项目资助三种新型并行编程语言，IBM的X10、Sun的Fortress以及Cray的Chapel。它们都属于研究型语言，目前只是小范围内使用，还没有对当前并行编程和并行软件开发产生革命性的影响。它们从降低并行编程难度出发，力争提高并行软件的生产率，同时还要提供高性能、方便移植和健壮性的支持。随着时间的推移，CPU和存储器性能差异会日益加大，存储层次也会变得更加的复杂，这些新出现的编程语言跟以往最大的不同是提供了对存储层次的显式控制，允许程序员在编程语言里对存储局部性进行描述，明确指出数据所在的位置，试图解决目前存储系统的性能瓶颈问题。

2.2.5　并行程序执行模型

编写好的并行程序需要在实际并行计算机上执行，才能得到计算问题的结果。并行程序的执行过程包括：高级语言编写的程序编译成机器语言，运行时资源分配和线程调度，数据在各级存储系统中的移动，每条指令在实际机器上的执行过程，以及指令流水线的动态执行过程。这些因素在使用算法设计模型进行算法设计和使用程序设计模型编写程序时，不需要考虑或者有些时候没法考虑。人们在设计算法和编写程序时并不清楚最终程序会在什么样的机器上执

行，而且往往会遇到为一台机器编写的代码需要移植到其他体系结构特性相差很大的机器上，所以很难取得最好的性能。

并行程序编写好之后仅仅是一个开始，需要花大量的时间来对它进行优化才能取得满意的性能，因此需要使用并行执行模型来指导如何在实际的机器上进行性能调优。并行程序执行模型将具体的并行计算机抽象成若干个影响程序执行性能的因素，从而可以使程序运行者针对具体的机器环境，进行程序的优化工作。并行执行模型使用执行时间、加速、效率和可扩展性等标准来衡量程序性能。并行程序执行模型通过对程序执行环境进行抽象来得到一个面向程序执行者的简洁的统一模型。执行模型定义了低层次的系统结构的编程抽象，表示了程序在一个并行计算机上的真实的执行行为。程序执行模型是编译器设计人员与系统实现人员之间的接口，编译器设计人员决定如何将一种高级语言程序按某种程序执行模型转换成一种机器代码；系统实现人员则决定该程序执行模型在具体目标机器上的有效实现。程序执行模型的适用性决定并行计算机是否以最低的代价提供最高的性能。根据程序执行模型进行调整的并行程序，可以在一台具体的并行计算机上获得最优的性能。并行应用程序去匹配这个模型，匹配的程度越高应用程序可以取得的性能就越好。

影响并行程序执行性能的因素很多也很复杂，包括硬件和软件的诸多方面，如缓存共享、处理器连接方式、通信的次数和每次通信的数据量、同步开销，多个处理器之间的负载平衡等等，这些都是串行程序不需要考虑的，也使得并行程序的性能优化更加复杂。

并行程序执行模型中，需要考虑如下机器硬件因素对程序执行性能的影响：（1）中央处理器性能参数：时钟频率，功能单元执行速度，寄存器数量，片内缓存大小，指令流水线，指令多发射，指令和数据预取等；（2）存储系统性能参数：存储系统的层次，每级存储系统的容量，块大小，延迟，带宽等；（3）互连网络性能参数：存储一致性协议，网络的延迟，带宽等；（4）输入/输出系统性能参数：磁盘的容量、速度以及其他的I/O设备的性能等。

并行执行模型中，需要考虑如下应用程序和运行时软件环境相关问题：（1）并行性级别问题：例如，任务级并行（考虑任务划分、映射、调度），数据级并行（考虑数据划分，迭代和循环分解），指令级并行（考虑如何高效发挥向量流水线，多发射，执行预取的效率等）；（2）线程问题：如何来进行线程调度，如何分配任务到细粒度的线程，尽量使得负载平衡，线程间通讯和同步；（3）存储系统问题：对共享地址空间的支持，存储一致性的支持，通过分块、数据重新组织调整提高程序的空间局部性和时间局部性等；（4）性能剖析问题：相应工具的在线剖析，垃圾收集，动态编译，自适应多版本执行，实现反馈式的性能优化，提供一些简单的自动优化。

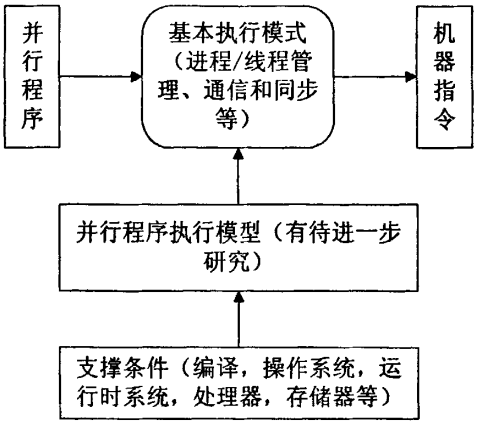


图 2.5 并行程序执行模型示意图

线程问题是并行程序相比串行程序所独有的，用线程来表示并行执行的任务，定义了各线程间数据同步和数据通信的方式。在并行程序设计模型中定义了一些线程的实现接口，在程序实际运行时并行程序执行模型通过运行时环境和硬件执行部件来具体负责创建线程、维护线程、销毁线程以支持这些接口。近年来研究者提出使用投机多线程（Speculative multithreading）和事务内存（Transaction Memory）来处理多线程并发的行为，解决多线程程序的相关性及同步问题，把同步控制的压力由程序开发人员转换到了编译运行时系统的设计人员。使用这两种执行模型时，程序员使用的编程模型就是传统的串行编程模型，编译器和运行时系统来进行并行任务的划分及执行。采用这种方式的并行应用程序没法用并行算法设计模型和并行编程模型来分析，因为直到执行前它们的设计跟串行程序无异；也没法用串行程序的分析方法来分析，因为实际执行的时候它是并行执行的。因此，我们需要在并行程序执行模型中考虑这种应用的行为，对应用的执行性能进行分析。图2.5给出并行执行模型相关的一些组成部分以及功能特性等。

2.2.5.1　分层模型特点比较

并行算法设计模型，研究时间比较长，有丰富的研究成果，在三层模型提出过程中，其它两层的模型定义和概念参考了并行算法设计模型。表2.1中总结、比较了各层模型的特点。

2.3　本章小结

本章回顾了并行计算模型的研究状况，按照并行计算模型的演变历史，把并行计算模型分为三代。从并行计算模型的发展过程可以看出，模型先

表 2.1 分层模型的对照比较

	并行算法设计模型	并行程序编程模型	并行程序执行模型
面向对象	算法设计者	编程人员	程序运行者
作用	算法设计者和 机器设计者 之间的桥梁	程序设计者与 计算机软/硬件 之间的接口	编译设计者与 系统实现者 之间的接口
关注点	算法正确性 和复杂度	编程正确、 高效、方便	优化执行性能
要素	计算参数 计算行为 开销函数	编程模式 接口和库函数	执行行为 性能指标
方法学	划分、分治 流水线等	SPMD、循环并行 主从、Fork/Join	测量/分析
复杂度	算法步数	高级语句条数	机器指令条数
支撑条件	算法理论 硬件平台 软件支撑	并行语言 工具和接口	编译器、OS 运行时系统 硬件结构
现有模型	PRAM、APRAM BSP、LogP DRAM(h)、HPM	MPI、OpenMP Pthread、HPF	有待进一步工作

以CPU计算为中心，再以网络通信为中心，最后逐步过渡到以存储访问为中心。一味追求单一模型的功能强和多目标，并行计算模型变得不实用和不可操作，现今对并行计算模型进行分层是势在必行的。按照并行计算模型的功能和使用的对象，本章提出把并行计算模型分为三个层次（并行算法设计模型，并行程序设计模型和并行程序执行模型），认为三层是充分和必要的。分层模型中，各模型职能不同，目标单一，易于设计和实现。并行算法设计模型和并行程序设计模型，目前相对比较成熟，而并行程序执行模型尚处于酝酿阶段，有待进一步研究与开发。并行执行模型关注点在并行程序的性能，相应的研究应该从并行程序的性能优化入手，而性能优化应首先从编译系统、操作系统和运行时系统等方面入手，兼顾考虑处理器、存储器和输入/输出设备等因素。

第3章 SMP系统上消息传递优化技术

内容提要 共享存储系统是并行计算的一个重要平台。对称多处理机（SMP）是一种共享存储的并行计算机。MPI是一种广泛使用的并行编程接口。并行处理中通信往往成为系统性能的瓶颈，改善和优化消息通信的性能具有重要的意义。但是当前SMP系统上MPI的实现并没有完全利用共享存储的硬件特点，导致通信性能不高。本章分析MPICH实现中的通信过程，在MPICH中完成一次消息通信，需要两次的数据复制。本章提出了一个新的基于共享内存消息传递协议实现方法，新方法中仅需一次数据复制，极大地优化了通信性能。通过实验比较了MPICH与新方法的通信性能，结果显示新的方法有更低的延迟，对于点对点通信，新方法比MPICH快15倍；对于集合通信新方法快大概300倍；对于实际应用程序性能也有显著的提升，NPB IS基准测试性能提升了1.71倍。

3.1 研究背景

3.1.1 MPI消息传递协议

MPI(Message Passing Interface)标准[57]是当今最流行的基于消息传递的并行编程标准，MPI吸收了许多消息传递接口的优点，具有很好的可移植性和可靠性。1992年，为了建立一个高标准并具有可移植性的消息传递库，由主要的并行机厂商、软件开发商、研究机构和高校成立了MPI论坛。MPI论坛先后于1994年和1997年发布了MPI-1和MPI-2两个标准。MPI可以在多种并行系统上运行，不仅适用于具有分布式内存的大型机、工作站集群，也支持共享存储的对称多处理机和多核系统等，理论上使用MPI编写的并行程序可以在任一并行计算机上不加修改的执行。历史上来看，MPI在分布式存储并行机上取得了极大的成功，有大量地用MPI编写的并行应用程序。

一个MPI程序通常需要一组库、头文件和编译、运行、调试环境的支持。MPI库包括进程初始化和终止的函数，点到点通信函数和几何通信函数，其中最基本的6个库函数为：MPI_Init、MPI_Comm_size、MPI_Comm_rank、MPI_Send、MPI_Recv和MPI_Finalize，用这几个基本函数就可以完成一个完整并行程序的编写，除此之外还有库函数提供更强大的功能。MPI程序开始时使用MPI_init指明并行开始，MPI_Finalize指明并行结束。并行执行过程中，维护几个活跃进程，每个进程有一个唯一的ID号，每个进程可以执行不同的操作，拥有独立的地址

空间，不同进程间通过消息传递函数实现数据传递和同步。

MPI具有众多优点，扩展性好，适用于大规模的并行系统；绑定了编程语言，与操作系统和硬件特性无关，被所有的并行计算机和操作系统支持，可移植性较好；进程管理方便；通信函数丰富，可根据需要选用合适的通信函数，使用方便，通信效率高。

3.1.2　SMP系统介绍

共享存储系统是并行计算中很重要的硬件结构，其中具有代表性的就是对称多处理机SMP（Symmetric Multi-Processor）。SMP在服务器领域广泛应用，集成了若干个同质的商用处理器，通常具有外置高速缓存，利用高速系统总线连接共享内存，实现通信，通信的性能和可靠性都非常高。SMP系统配置简单，现代操作系统可以直接支持SMP，无需用户进行特殊配置，操作系统会均衡调度处理器上任务执行。SMP受总线带宽和存储竞争限制，存储带宽往往成为系统性能的瓶颈，可扩展性较差，很难看到超过32个CPU的SMP系统。SMP也常常作为一个子模块，用来构造更大规模的并行机系统。SMP一般采用均匀访存模型（Uniform Memory Access，UMA），任意处理器对内存的访问时间是一样的，也有少量大规模SMP采用非均匀访存模型（Non-Uniform Memory Access，NUMA）。

SMP具有以下特征：（1）对称性：所有处理器都是同样配置，对存储器和I/O设备访问是对称的；（2）全局共享地址空间：所有处理器单元共享同一个存储器，具有统一的地址空间；（3）高速缓存一致性：通过高速缓存一致性硬件来保证多处理器之间数据的一致性；（4）低通信延迟：通过读写共享内存来实现处理器之间通信，通信代价低。

针对共享存储系统，有比较成熟的编程模型如OpenMP[58]和Pthreads[59]。但是仍然需要研究使用MPI编写可以在共享存储系统上运行的并行程序。首先，历史上有许多优秀地使用MPI编写的针对分布式系统的应用程序，需要把它们移植到共享存储系统上，在做尽量小的修改的同时，最大限度地挖据性能；其次，MPI标准有很好的兼容性，比较成熟，在并行计算领域取得了很大的成功，当前大部分的并行计算机都支持MPI，使用MPI编写的程序可移植性好，程序员对MPI也相对熟悉，他们愿意使用MPI编写并行程序。

3.1.3　MPI在SMP系统上缺点

通信系统是并行计算机系统的一个关键部分，各处理器之间的数据交换都是通过它来完成的。根据第一章的介绍，并行计算时间包括了计算时间和通信时间，如果通信时间在整个并行计算过程中占了很多比例，那么整个计算过程

的加速比就会非常低，系统效率也会比较低。高效的数据通信是MPI编写的并行应用程序在共享存储机器上取得高性能的关键。

本小节，分析MPI的一个流行实现MPICH中的通信协议实现方法。MPICH实现了一个用于共享存储系统的驱动程序ch_shmem[60,61]。由于并行机体系结构差异比较大，硬件配置千差万别，MPICH对不同的体系结构抽象了各种消息驱动程序，如各种系统普遍适用的ch_p4，针对集群系统的ch_p4mpd，针对网格的globus，针对共享内存系统的ch_shmem。出于兼容性、通用性、可靠性和安全性的考虑，例如不同机器可以使用的共享内存大小不同，MPICH中消息传递驱动ch_shmem并没有针对共享存储做深入优化，导致MPI通信协议在共享存储系统上效率并不高。ch_shmem实现中，每个进程有一个独立的队列，用来接收其他进程发送的消息，队列的大小为当前通信域中进程的个数。队列放置在共享内存中，两个进程之间的消息传递通过读写该通信队列来实现，通信域中的所有进程都可以访问该共享队列的不同部分，即多个进程可以写共享队列（任何进程向共享队列所有者发送消息），只有一个进程可以读共享队列（队列所有者）。接收者和发送者之间使用锁来实现对共享队列的互斥读写。消息发送者从私有地址空间复制数据到共享队列的尾部，然后消息接收者从共享队列头开始复制数据到私有地址空间。通常需要锁来保证对系统共享缓存区的顺序访问，防止多个发送进程同时向共享队列追加数据或者接收进程在发送进程还未完成数据发送就开始读消息，保证对共享队列写的时候没有其他进程对共享队列操作，但是锁的开销比较大。

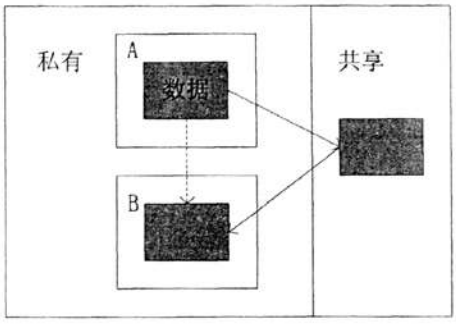


图 3.1 MPICH中消息传递示意图

两个进程间消息传递的典型过程如图3.1所示，实线代表物理的消息传递路径，虚线代表逻辑的消息传递路径。假设进程A打算向进程B发送一个消息。预先系统会分配一个共享的缓冲区和一个同步标志，来辅助消息传递过程。共享的缓冲区用来进行数据的中转，同步标志用来指示对于共享缓冲区的读写是否完成。消息传递开始前，A先检查同步标志，查看系统缓冲区是否就绪允许A写

数据，如果已经就绪，A把数据从自己的缓冲区复制到共享缓冲区，数据复制完毕后，A设置同步标志，表示复制结束，B可以从共享缓冲区复制数据；如果共享缓冲区还没就绪，A选择一个退避策略，休眠等待一段时间后，被重新唤醒然后重新尝试数据复制。另一方面，进程B通过探测同步标志来检查共享缓冲区中数据是否已经复制完毕，一旦发现同步标志被设置，B开始从共享缓冲区中复制数据到本地缓冲区，复制完毕后B清除同步标志。至此，一次消息传递结束，可以开始新的消息传递。

以上的消息传递过程效率比较低，主要有以下几个原因：

(1) 每次消息传递需要两次数据复制过程，两次数据复制并不是必需的，增加了存储系统的负担，导致高的通信延迟。

(2) 需要开辟额外的系统共享缓冲区，通过它来辅助消息传递，进程的私有数据可以复制到该缓存区，也可以从该缓冲区复制到私有缓冲区。这增加了内存的消耗，在共享存储系统上，内存和Cache容量以及存储带宽都会限制通信的可扩展性。

(3) 发送者需要先把数据全部复制到共享缓冲区中，复制完成后，接收进程才能开始复制数据到私有缓冲区。对于长消息，接收进程需要等待较长的时间，消息复制过程才能开始。

(4) 在共享缓冲区不可用时，退避策略会让进程等待一定的时间后，再次去查询共享缓冲区是否就绪，如果一直不可用，以后每次等待的时间会逐渐增大到某一值。假设某一时刻数据已经就绪，但是还没有达到该次等待的时间长度，那么进程会一直等到满足该次等待时间长度后，才会检测到数据已经就绪。共享缓冲区中数据准备好到进程检测到数据就绪这段时间就是不必要的等待开销，会显著影响短消息传递的性能。

(5) 在集合通信中，不必要的数据复制和高的同步开销对通信性能的影响会更加突出。

3.1.4 相关工作

当前已经有一些工作，考虑共享内存系统上的MPI通信优化。William Gropp[62]研究了在NEC SX-4 机器上使用系统提供的test-and-set原子操作代替使用System V锁，使用两两进程间通信槽代替通信队列，在一定程度上减少了同步开销，但是仍没有避免两次内存复制的开销。TMPI[63]实现了一个线程级的MPI版本，TMPI使用多个线程而不是多个进程执行并行任务，线程更轻量级，上下文切换开销小、速度快，对于非独占的环境非常高效。中科院计算所马

捷[64]研究了基于SMP的机群系统上的多重通信协议，支持异构的网络环境，包含了共享内存的通信协议和一种半用户级通信协议，研究了机群中节点间和节点内的通信性能，考察了通信协议对安全性、可靠性、可移植性的影响，其中共享内存的通信协议是使用的基于系统共享缓冲区的流水线技术来加快长消息传递。本文工作跟以上的差别在于，主要考虑单进程独占处理器情况下，对性能要求比较高，安全性和可移植性要求较低的应用，以性能为主要目的，降低应用程序的通信开销，加快执行速度。

3.2 SMP系统上通信优化技术

一般情况下，每个进程有自己的局部数据区，不同进程间数据是相互隔离的，共享内存机制允许多个进程对同一块内存区域的访问。进程间可以通过共享内存机制交换数据。为了解决MPI在共享存储机器上实现的问题，本节设计了一个新的基于共享内存的通信协议，使用进程间通信机制（IPC/shm）创建进程级的共享内存块，来进行消息传递，从而进程间消息传递仅需一次数据复制；从考虑性能最大化角度考虑，采用自旋等待策略来实现同步，最小化进程等待时间。

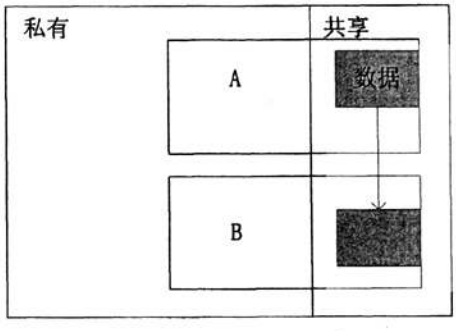


图 3.2 优化的消息传递示意图

3.2.1 单次数据复制技术

上一节讨论的MPI消息传递实现中，两次的数据复制是不必要的，如果消息发送者需要保存发送的数据仅一次的数据复制就可以了，甚至某些情况下不需要数据复制，如图3.2所示。当进程间有数据要传递时，预先使用进程间同步机制申请一块共享内存，需要传递的数据直接存放在这块内存中，其他进程都可以访问该共享内存块，可以读取其中的数据或者复制到自己的地址空间中。当所有的进程共享同一个数据结构，不同进程写该数据结构的不同部分，计算时会用到其他进程之前写入的数据，这时不需要进行数据复制，在一个进程写

入数据之后，其他所有进程能立即看到写入的数据，可以直接读取该进程已写入的数据而不需要数据复制操作。由于不需要额外的系统共享内存支持，同时仅需0次或1次数据复制操作，以上的数据传递过程极大减少了通讯延迟。

3.2.2 自旋等待同步策略

并行执行的多个进程间通信时，进程互相依赖是难以避免的。为了保证通信过程中不同进程间消息发送和接收顺序的正确性，需要一个同步机制。同步要求限定了发送者把数据准备就绪之后，接收者才能开始数据接收过程。在共享存储系统上最常用的实现同步的方法是锁和信号量。然而不幸的是，一方面锁和信号量带来的开销都非常大，影响整体通信性能；另一方面，锁没有被释放前，其他进程采用退避策略来探测资源是否可用。根据前文讨论，进程在等待时间消耗完之后才会获知数据已经就绪，因此会延缓消息传递的启动时间。

本章考虑最大化单个应用程序在SMP机器上的通信性能，假定一台机器上在某一时刻只有一个应用程序的多个进程在执行，该应用程序独占所有的计算资源和存储资源。在这种情况下，为了最小化同步时间，采用简单的自旋等待（spin-wait）策略，代替保守退避策略来进行数据同步，是最佳的方法。自旋等待是一个密集型循环，进程始终保持活跃状态，并消耗处理器资源。进程反复地测试同步标志，在一个循环中自旋，处于忙等状态，一旦同步标志状态发生变化，它可以及时探测到，因此可以立刻开始数据传递，没有任何延迟。这种同步策略非常适合于一个处理器上只有一个进程的情况，可以减少数据同步的开销。如图3.3给出了一个实现。

```
while ( sync_var != predefine_value)
{
        __asm PAUSE;
        sleep(0);
}
```

图 3.3 自旋等待实现示例代码

多个进程同时执行自旋等待循环时，由于循环执行很快，会产生大量的读请求，这些请求的顺序是无序的。当处理器检测到一个线程的写操作对象，为其他线程正在进行中的读操作的对象，处理器必须保证内存访问顺序合法。为了保证大量内存操作合法的顺序，处理器在退出等待循环时会受到严厉的惩罚。这是因为一般指令执行比分支判断要快，处理器总是会进行分支预测，然后尝试执行，在等待循环中，处理器会预测循环退出条件不满足。前若干次循环，处理器预测总是正确的；但是当某时刻同步变量满足了循环退出条件，会产生

预测错误，那么之前所有探索执行的指令都要取消，因此就浪费了大量的系统资源。通过在循环中加入PAUSE指令，提示处理器这是一个循环等待，处理器遇到这个指令的时候，就不会大量预取指令执行，减少了系统资源占用，避免了大量预取指令清空带来的惩罚开销，可以显著的提升性能。加入sleep(0)，可以使得在有其他线程等待执行的时候，让出处理器；如果没有其他等待执行的线程，自旋等待循环就会始终执行。这样处理器仅占用其他线程空闲的时间来自旋等待，不会一直霸占全部的处理器资源，可以更好地避免处理器资源浪费。虽然CPU负载仍表现出100%，但是整个系统的运行流畅，不会拖慢处理器的响应速度。

3.2.3　消息传递性能分析

下面分别给出传递大小为n个字节的消息时，原来的消息传递实现和新的优化实现方法的通信开销，分别用$T_{original}$和$T_{optimized}$标识。MPICH中使用两次数据复制实现的消息传递开销为：

$$T_{original} = 2T_{datacpy} + T_{syn} \tag{3.1}$$

优化的单次数据复制实现的消息传递开销为：

$$T_{optimiezed} = T_{datacpy} + T_{alloc} + T_{free} + T_{spin} \tag{3.2}$$

优化后不需要数据复制实现的消息传递开销：

$$T_{optimiezed} = T_{alloc} + T_{free} + T_{spin} \tag{3.3}$$

其中，$T_{datacpy}$表示一次数据复制的时间，T_{syn}表示未优化的通信同步时间，T_{alloc}和T_{free}分别代表共享内存分配和释放的时间，T_{spin}表示使用自旋等待策略的同步时间。一般共享内存分配和释放的时间与数据复制时间相比可以忽略不计。因此优化后通信性能要比优化之前有很大的性能提升。

3.2.4　实验与分析

本小节通过实验来研究新的优化通信协议的性能，采用通信延迟来评价通信性能的好坏。通信延迟是指消息从发送者开始发送到接收者完成接收花费的时间。本节测试大小在0字节到1M字节的不同消息的通信延迟，0字节大小的通信延迟反映了通信的启动和结束开销。本节列出的结果都是多次测量结果的平均值。

3.2.4.1　实验平台配置

实验用的系统为一个16路的对称多处理机[65]，运行Suse Linux 9.0操作系统，共有16个x86处理器，运行主频为3.0GHz，有四级Cache，每4个处理器为一

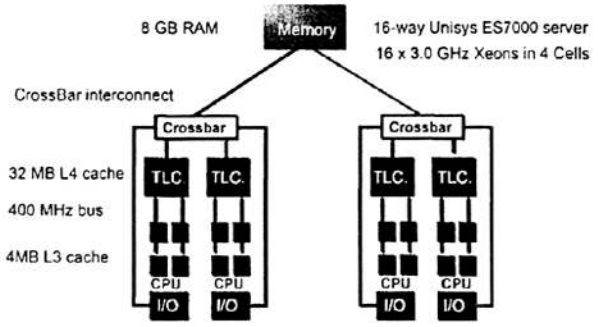
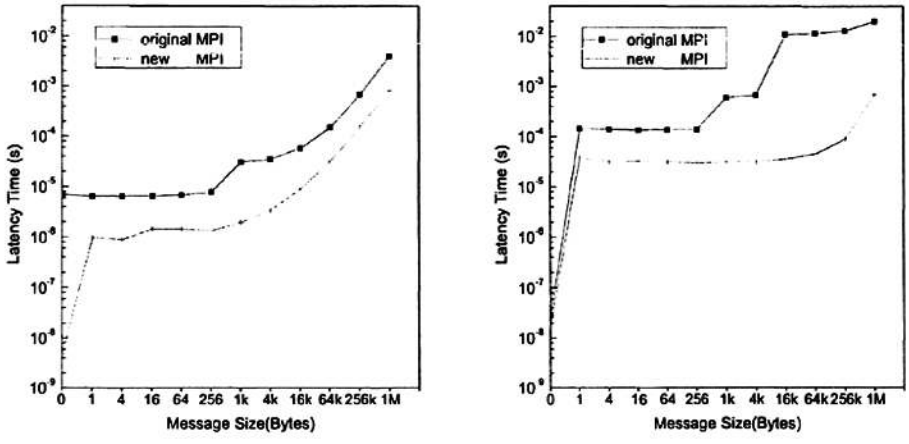


图 3.4 16路Intel Xeon对称多处理机体系结构图

个socket，一个socket共享一个32MB的四级Cache，每两个socket通过交叉开关与主存相连，系统结构如图3.4所示。实验中使用的软件配置如下，MPI使用的是MPICH-1.2.5.2（采用ch_shmem驱动程序），gcc3.3。每个进程绑定在一个特定的处理器上，防止进程在不同的处理器之间迁移影响通信性能。



(a) 点对点通信　　(b) 集合通信

图 3.5 优化前后消息传递性能比较

3.2.4.2 实验结果及分析

点对点通信在一对进程之间传输数据，本节选用标准MPI_Send/Recv进行点对点通信测试。在实际的应用程序中，集合通信的使用频率比较高，集合通信涉及大量的进程，这些进程之间传递的数据量也都比较大，集合通信的性能对整体通信性能有着很大的影响。本文选择了MPI_Bcast、MPI_Gather、MPI_Scatter、MPI_Alltoall和MPI_Reduce来研究分析优化前后集合通信的性能，

给出MPI_Bcast的结果作为集合通信的代表。MPI_Bcast是root进程将一条消息广播发送到组内的包括自己在内的所有进程。

实验结果如图3.5所示，图3.5(a)给出了优化前后点对点通信的性能比较，图3.5(b)给出了广播通信的性能比较，图中Original和New用来标识优化前和优化后的通信性能。可以看出优化后的通信性能远远高于未优化的版本。对于优化后的通信实现，各种消息长度的点对点通信和集合通信的延迟均比优化前要低。在点对点通信中，对于短消息优化后的版本相比于MPICH有6倍左右的性能提升，对于长消息最高有15倍的性能提升；在集合通信中，短消息有大概5倍的性能提升，当消息长度大于1KB时，取得了上百倍的性能提升。消息大小在$1B$到$4KB$之间的广播延迟基本没有变化，这是因为通信组内的其他进程可以同时从root进程的发送缓冲区中读数据，然后复制到各进程本地缓冲区。数据的复制时间是*memcpy*()的执行时间，对于各种不同短的消息*memcpy*()执行时间差别不大。当消息大小逐渐增大，前端总线带宽竞争激烈，有限的带宽限制了通信性能，导致通信延迟随着消息大小而增加。在未优化的MPI消息传递中，当消息长度大于预先申请的共享系统缓冲区的大小时，系统缓冲区不能一次容纳全部的消息，整个消息会被分割成若干块，每次传输一块。因此在图中，可以看到未优化的集合通信延迟呈现阶梯上升的趋势。对于优化后的版本来说，整个消息不受长度的影响，接收者都可以直接从发送者缓冲区中复制，因此优化后的集合通信性能对消息的大小并不是很敏感。

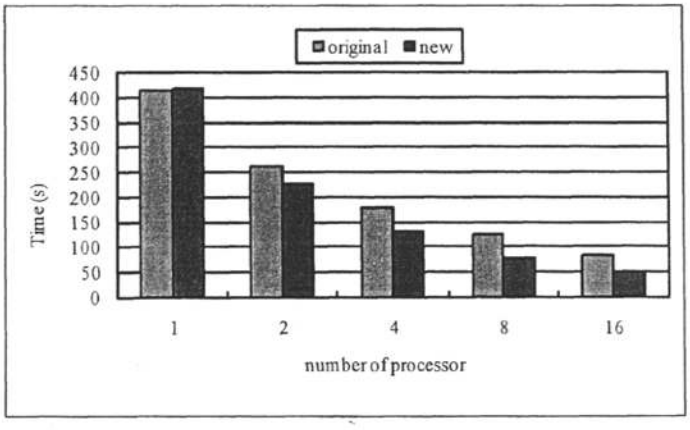


图 3.6 NPB IS基准测试优化效果

3.2.5　实际应用的优化

本小节使用经过优化的MPI通信函数库，优化NPB（NAS Parallel Benchmarks）[66]中的IS（Integer Sort）基准测试，以分析优化的MPI通信库在实际

应用中的表现。NPB是并行计算机性能测试的一个常用benchmark，其中包含了8个计算流体力学应用领域的软件，能够反映一般应用程序的实际性能。除IS外，其他几个基准测试程序使用Fortran语言编写，本文仅实现了C语言版本的优化MPI库，因此只对IS进行了实验测试。IS对大量的整数进行排序，主要测试整数运算性能和集合通信的性能。IS中主要的通信操作为全局交换Alltoall，对通信延迟非常敏感。图3.6中给出IS使用C类规模数据集时的实验结果。从图中可以看出随着处理器个数的增加，使用new mpi的并行IS，比使用original mpi的并行IS，性能提升越来越显著，当使用16个处理器时取得了1.71倍的性能提升。使用一个处理器时new mpi版本时间稍微有增加，是因为对于共享内存的维护会带来少量的额外开销，而只使用一个处理器时并没有消息传递，未能体现出优化MPI通信库的优势。

3.3 本章小结

对称多处理机（SMP）是一种重要的共享存储并行计算平台。本章首先介绍SMP系统的特点，指出使用MPI在SMP系统上并行编程的必要性；接着，分析MPICH中基于共享内存的通信驱动程序的实现方法，得出两次消息复制和其中的同步策略限制了通信性能；然后，提出了SMP系统上消息通信的优化方法，包括单次消息复制技术和自旋等待同步策略，最后，通过实验测量研究新通信协议的性能，实验结果显示，对于点对点消息传递，优化的MPI通信库性能提升了15倍；对于集合通信，优化后MPI通信性能提升了300倍左右；对于NPB中的IS基准测试，当使用16个处理器时，使用优化MPI通信库的并行IS性能提升了1.71倍。从实验结果可以看出优化的MPI通信方法对于减少处理器间的通信时间，提高并行程序的性能是十分有效，特别是当处理器个数增加时，新的方法能明显提高整个程序的性能。由此得出本章提出的针对共享系统上的MPI程序的优化方法，具有广泛的可用性，能显著提升实际应用程序的性能，可以把已有的针对分布式系统编写的基于MPI的并行程序，方便、高效地转换到共享存储系统上来运行。

第 4 章　SMP系统上Mfold的并行优化

内容提要　RNA二级结构预测是计算生物学中的一个重要问题。Mfold是一个广泛使用的RNA二级结构预测软件。Mfold使用一个动态规划算法来搜索具有最小自由能的二级结构，该动态规划算法是整个程序中最耗时的计算部分。本章研究Mfold程序在SMP系统上的并行化及其优化，来缩短RNA二级结构的预测时间，研究共享存储系统上应用程序的性能加速方法。首先，采用对角线法来并行Mfold中的动态规划算法，并利用MPI编程实现；接着根据共享存储系统特性，消除并行Mfold中的数据传递开销，优化并行Mfold 在共享存储系统上的性能；最后通过实验来测量并行Mfold的性能，比较未优化和优化两个版本的加速比，使用性能分析工具收集性能相关数据，分析性能优化对程序执行行为的影响。

4.1　串行Mfold算法介绍

4.1.1　RNA二级结构预测

RNA在生物体中有着重要功能和结构作用，既是信息序列又是功能序列。RNA分子的功能是与其结构密切相关的。通过确定RNA的结构可以对其功能有全面理解。RNA的结构分为一级结构、二级结构和三级结构三种，一级结构是由4种核苷酸碱基排列而成的线型序列，二级结构是RNA序列折叠形成的平面结构，三级结构是二级结构中子结构扭曲构成的空间结构。RNA的功能是由它的三级结构决定的，而三级结构又是由二级结构决定的。确定RNA的二级结构对于预测三级结构、研究其功能有着重要的意义。RNA以单链的形式存在，通过自身折叠形成交替出现的茎和环的二级结构，连续的碱基配对形成茎，连续的不匹配碱基形成环，其中环根据形态的不同又进一步分为发夹环、突环、内部环和多分支环等。通过确定这些茎和环来确定RNA 序列的二级结构。RNA的二级结构可以形式化定义如下：对于一个RNA序列$S = s_1 s_2 \ldots s_n$，二级结构可以表示为一系列基本对$S = \{(s_i, s_j) | 1 \leq i < j \leq n\}$，同时$S$满足：

(1) $j - i > 3$

(2) 二级结构中的两个不同的基本对(s_i, s_j)和(s'_i, s'_j)满足s_i, s_j, s'_i, s'_j为不同的核苷酸，即一个碱基不可能与两个或两个以上的碱基配对。

RNA二级结构的稳定性主要由碱基配对形成的堆叠力和环产生的作用力决定。RNA二级结构预测就是找出一个给定RNA序列最稳定的二级结构。最小自由能模型认为稳定的二级结构具有最低的自由能，二级结构的自由能是组成结构的各个独立环的能量总和，环的能量是由环本身决定的，与结构中的其他部分没有关系。

目前，实验的方法如核磁共振、X射线晶体衍射和化学方法等，虽然比较精确和可靠[67]，但是非常耗时和昂贵，每次只能测定一条RNA序列，而且需要大量相似序列的支持。因此，有必要借助于计算的方法，加上对已有结构的认识，精确地预测一个给定RNA序列的二级结构[68]。

4.1.2　Mfold中的串行预测算法

Mfold[69]是一个在实际中被广泛使用的预测RNA二级结构的软件，它以分子热力学原理为基础，采用热力学模型计算满足最小自由能的二级结构。Mfold中考虑二级结构由以下几个元件组成：发夹环、堆叠区、突起环、内部环和多分支环。Mfold利用动态规划的方法来寻找具有最小自由能的结构，把RNA二级结构最小自由能分解为各个元件的自由能总和，通过求解较短子序列的最优二级结构，进而递归求解更长序列的最优二级结构。对于长度为n的RNA序列，算法的时间和空间复杂度分别为$O(n^3)$和$O(n^2)$。在实际预测时候，在一台单处理机上，计算一个比较长的RNA序列的二级结构需要几个小时乃至数天。当有大量的RNA序列时，这样的计算速度是不能接受的。因此并行和优化Mfold在多处理机上的性能非常重要，通过并行计算来加速Mfold的计算速度。

对于每对(s_i, s_j)，其中$1 \leq i < j \leq n$，算法计算哪一种环和外部的基本对(s_i, s_j)具有最小自由能。算法中用到的4个数组W，V，VBI和VM，来保存某一子序列的结构能量。本小节给出动态规划算法的简要递推公式，详细的算法可以参考相关文献。整个动态规划算法的计算过程可以用以下递推公式来表示[70]。子序列$s_1 s_2 \cdots s_i$的最小自由能用$W(i)$表示，定义如下：

$$W(i) = \min\{W(i-1), \min_{1<j\leq i}\{W(j-1) + V(j,i)\}\} \tag{4.1}$$

$W(n)$表示整个序列的最小自由能。

$V(i,j)$表示子序列$s_i \cdots s_j$的最小自由能：

$$V(i,j) = \min\{eH(i,j), eS(i,j) + V(i+1, j-1), VBI(i,j), VM(i,j)\} \tag{4.2}$$

公式4.2中，$eH(i,j)$为包含碱基对(s_i, s_j)的发夹环的能量，$eS(i,j)$为(s_i, s_j)和(s_{i+1}, s_{j-1})同时配对时堆积的能量，$VM(i,j)$为(s_i, s_j)以多分支环结尾时候的

最小自由能，$VBI(i,j)$为包含碱基对(s_i,s_j)的子序列$s_i\cdots s_j$以突起或者内环结尾时候的最小自由能：

$$VBI(i,j)=\min_{\substack{i<i'<j'<j\\ i'-i+j-j'>2}}\{eL(i,j,i',j')+V(i',j')\} \tag{4.3}$$

能量函数$eL(i,j,i',j')$是包含外部碱基对(s'_i,s'_j)和内部碱基对(s_i,s_j)的突起或者内环的能量。

图4.1显示了计算的依赖关系。矩阵中的深色元素值的计算依赖于浅色元素的值。当计算$V(i,j)$的值时，依赖下标为$(i',j')|i'\geq i\&j'\leq j$的元素值。数组$V$和$VBI$沿着从左至右，从下往上的顺序计算填充。

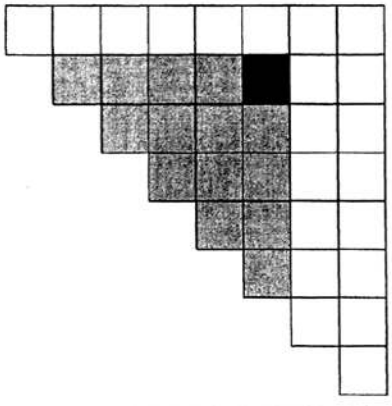


图 4.1 Mfold中动态规划算法依赖关系图

4.2　并行Mfold及优化

4.2.1　并行Mfold实现及分析

根据上一小节能量矩阵的依赖性关系，本节采用对角线法来并行整个计算过程，对角线法是并行动态规划算法一个比较常用的方法之一。在对角线法中，如图4.2所示计算沿着对角线方向，从左往右，从下往上计算。每条对角线分成若干个数据块，这些数据块均匀地分配给所有的处理器，每个处理器大概处理$1/p$的数据块。整个计算过程分成若干个阶段，每阶段不同的进程处理不同的数据块，所有进程每阶段的本地计算需要其他进程上一阶段计算的结果，在一个阶段的末尾，不同进程间需要数据同步，相邻进程间需要数据通信传递下一阶段所需的数据，每个进程需要把本地计算的结果发给root进程，root进程汇总各进程数据作为整个计算的结果。MPICH中各进程间的消息传递，通过MPI_Send/Recv来实现。根据上一章的讨论，MPICH在共享系统上消息通信的

性能比较低。MPICH实现的共享存储系统上的驱动程序考虑了不同系统上的通用性，因此实现的时候性能会有所下降。各进程间通信是通过对系统共享缓冲区的读写来完成的。发送进程把数据写入共享缓冲区，然后通知接收进程数据就绪，接收进程把数据读取到私有存储空间。以上的通信过程有诸多的缺点，会带来性能损失：

(1) 两次的数据复制，通信开销比较大，数据复制还会增加存储系统负担，影响整个应用的Cache命中率，占用前端总线带宽。

(2) 需要额外的共享系统缓冲区作为数据中转站，另一方面每个进程中都保存了一份各矩阵的复本，因此总体上消耗了过多的内存，导致Cache中频繁的数据换入换出，过多的内存消耗也会影响系统的整体性能。

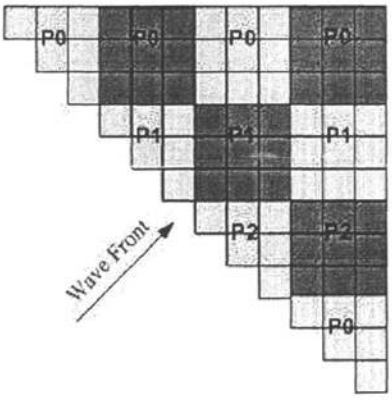


图 4.2 对角线法并行化Mfold

4.2.2　并行Mfold的优化

鉴于上一小节的分析，可以得出通过减少总的内存消耗，采用合理的数据放置策略，有利于整体并行性能的提高。本小节研究如何优化内存使用和数据同步，进一步加速并行Mfold在SMP系统上的性能。

在共享内存系统中，利用进程间通信机制（IPC/shm），可以创建进程级的共享内存，相关的矩阵都可以存放到共享内存中，所有的进程都可以访问该共享内存，图4.3给出了共享内存分配的实现过程。使用共享内存存放矩阵数据，只需要保存一份数据就可以，而不是像上一小节中每个进程都在私有地址空间中保存一个矩阵数据的复本。总体上会大量减少内存的使用，带来Cache性能的提高。例如对一个1000 × 1000规模的矩阵，其中保存的数据为双精度浮点数，总共消耗8MB的内存空间，假设在一个8路SMP机器启动8个进程，未经优化的

并行程序中总共会有64MB的内存消耗，而把矩阵存放在共享内存中的优化后的并行程序仅消耗8MB的内存空间。

```
void * shm_alloc(int key, int size)
{
    int shmid;
    if((shmid = shmget(key,size,IPC_CREAT|IPC_EXCL|0666))<0)
        if(errno == EEXIST)
            shmid = shmget(*key,*size,0);
    return shmat(shmid,NULL,SHM_REMAP);
}
```

图 4.3 共享内存分配的代码实现

矩阵存放在共享内存中，消息交换可以用对所需数据的直接读取来实现。在并行Mfold中，每个进程处理矩阵的不同部分，没有交集，一个进程需要另外一个进程上一阶段计算的结果，在一个计算阶段的末尾会有一个路障操作，在开始下一计算阶段前所有的进程该阶段的计算已经完成，计算结果对其他进程可见，确保了下一阶段需要使用的数据都已经准备好，保证了整个计算过程的正确性。由于直接对共享内存中的矩阵进行操作，计算中无需数据传递，极大降低了维护各进程间数据一致性的开销。

共享存储系统上，多个进程同时操作一个数据结构有可能会引入假共享。这是因为，不同的内存单元可能恰好存放在同一个缓存行内。从逻辑上来看，这些进程并不共享这些内存单元，但是缓存行是内存读取的最小单位，物理上来说这些进程共享这些缓存行内的数据，这种现象称为假共享。根据缓存一致性协议，每个进程更新自己的缓存行后，其他处理器内核上对应缓存行被标记为无效，这些共享的缓存行会不停地被踢出和传入处理器内核。因此假共享会严重影响共享系统上缓存性能。假共享在一级、二级高速缓存中都存在。通常解决数据假共享的方法主要有两个：（1）仔细分配内存或通过数据填充，使得不同的线程使用的数据在不同的缓存行；（2）每个线程对全局数据创建私有拷贝，先使用私有拷贝进行计算和更新，在全部计算完成后再写到全局数据。在本小节实现中，多个进程共享同一个数据结构，在分块的边缘有可能会有假共享产生，通过设置合理的分块大小和数组边界对齐到缓存行，让两个相邻块中的数据不会分布在同一个缓存行中，由此避免了假共享。

默认情况下，在共享存储系统上，操作系统会采用时间片轮转的策略来调度进程在每个核上执行。默认的调度策略会照成频繁的上下文切换，增加缓存失效，使得缓存数据在两个核之间来回传送，浪费总线带宽。如图4.4所示，进程在两个处理器之间来回迁移，照成Cache性能降低。使用处理器关联技术，调度进程在特定的处理器上执行，避免进程迁移，进而提高性能。在优

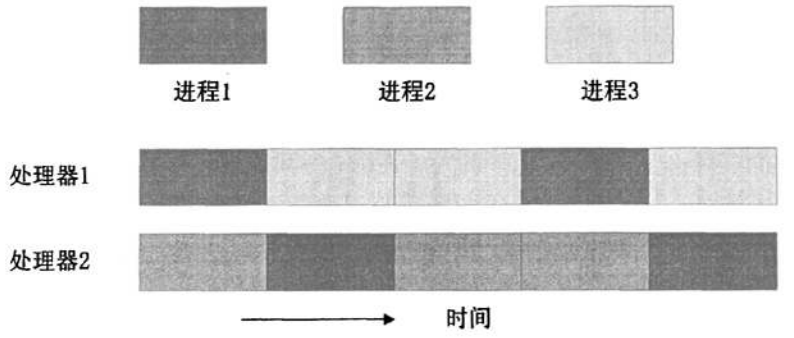


图 4.4 进程在两个处理器上迁移示意图

化的并行Mfold中，每个进程绑定在一个处理器上，阻止操作系统调度进程在不同的处理器上执行，减少进程迁移时带来的上下文切换的开销。处理器关联技术，一般有下面几种好处：（1）把有数据共享的进程调度到共享Cache的一些处理器上，相互之间会预取数据到Cache中，可以减少Cache强制失效；（2）把对总线带宽需求高的进程调度到不共享前端总线的处理器上，可以充分利用存储带宽；（3）把对内存需求大的进程调度到不共享Cache的处理器上，可以减少Cache容量不足带来的损失。

4.3　实验结果及性能分析

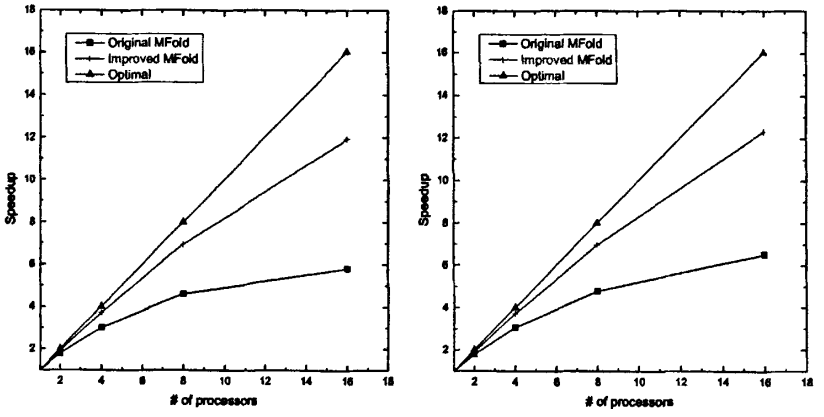
本节通过实验来研究并行Mfold的性能，对比未经优化的并行Mfold和优化后的并行Mfold的性能差异，给出对于不同长度RNA序列，并行预测算法的性能加速比，分析了两种并行实现的性能细节。下文分别用未经优化的并行Mfold和优化后的并行Mfold来指代两个并行版本。

4.3.1　实验配置

实验中的硬件环境和软件环境与上一章中一致。Mfold为一个开源的软件，本文采用的版本为3.1.2。实验中使用的RNA序列片段取自Genbank的网站。对于不同的RNA片段，并行Mfold的性能差异不大，本文选取两个不同长度的RNA片段给出实验结果和分析，长度分别为2500和3500。所有给出的结果都是多次测量取平均值。

4.3.2　结果与分析

实验结果如图所示，图4.5中比较了未经优化的并行Mfold加速比、优化过并行Mfold加速比与理想加速比。理想加速比是指使用n台处理器可以取得n倍的性能加速。根据实验结果，当使用16台处理器时，对于长度为3500的RNA序列，



(a) length = 2500　　(b) length = 3500

图 4.5 优化前后并行Mfold的加速比分析

优化后的并行Mfold性能比未经优化的并行Mfold提高了95%，加速比由6.52 提高到12.31。从图中可以看出优化后的并行版本，随着处理器数目的增多，系统性能可扩展性较好。

当使用处理器数目比较少时，在未经优化的并行Mfold执行时间中，计算时间占据主导地位，计算部分可扩放性比较好，取得了较好的加速比；当处理器逐步增多时，通信时间逐渐增多，削弱了计算时间的主导地位，成为系统性能的瓶颈，影响了加速比的提高。因此，虽然处理器数量增加了很多，但是加速比并没有同比例提高。在经过优化的并行Mfold中，消除了数据在不同进程间的传递，因此经过优化的并行Mfold在使用较多数量的处理器时，依然可以取得较高的加速比。

优化后的并行Mfold中仅保留一份矩阵数据，四级Cache可以容纳整个能量矩阵，数据访问基本都可以在Cache中命中，极大减少了对内存的访问，带来存储系统性能的提高，进而也提升了整个应用的性能。与之相比，未经优化的并行Mfold中，每个进程保留一份私有数据复本，通过消息传递来进行数据同步，这些数据复本不可能同时在四级Cache中保存，数据访问无法全部在Cache中满足，需要访问主存来取数据，从而影响整个应用的性能。

下面使用性能剖析器收集程序运行时的性能数据，详细分析程序的执行行为，利用收集的数据计算各级Cache失效率和前端总线带宽。表4.1给出了一级、二级和三级Cache在使用不同数量处理器时的失效率和前端总线带宽，由于没有相应的硬件计数器，四级Cache的失效率无法获得。从表4.1中可以看出，优化后的各级Cache失效率都高于未经优化的版本，前端总线带宽也消耗的更多。这主要是由于Cache失效率和总线带宽都是整个执行阶段的度量，而优化后的并行

表 4.1 优化前后Cache缺失率和总线带宽利用情况比较

	Original			Optimized		
	1P	4P	16P	1P	4P	16P
L1 Cache Miss Rate	6.51%	5.62%	4.47%	6.42%	5.98%	5.50%
L2 Cache Miss Rate	11.55%	9.72%	11.69%	11.06%	10.71%	12.04%
L3 Cache Miss Rate	0.79%	2.72%	8.79%	0.91%	2.80%	11.38%
FSB Bandwidth (MB/s)	4.76	44.28	484.85	4.82	55.13	824.44

表 4.2 优化前后Cache缺失绝对数量比较

	Original			Optimized		
	1P	4P	16P	1P	4P	16P
L1 Misses	1.42E+10	1.53E+10	1.78E+10	1.41E+10	1.46E+10	1.48E+10
L2 Misses	1.64E+09	1.73E+09	2.08E+09	1.56E+09	1.66E+09	1.78E+09
L3 Misses	1.31E+07	3.90E+07	1.83E+08	1.22E+07	3.08E+07	1.03E+08

版本需要往处理器传输的数据量大大减少，各级Cache缺失的绝对数量减少，总的执行时间也较相应减少，因此统计的Cache失效增大，前端总线带宽也增大，但远没有达到前端总线的容量25.6GB/s，因此前端总线带宽不是两个并行版本的性能瓶颈。表4.2中给出了各级Cache缺失的绝对数量，从表中可以看出优化后的并行Mfold与优化前相比，各级Cache缺失量大大减少，并且随着使用处理器数量的增加，缺失数增幅远小于优化前的版本。

表 4.3 优化前后执行时间组成比较

	Original			Optimized		
	1P	4P	16P	1P	4P	16P
W_	53.17%	48.51%	41.04%	53.38%	51.11%	45.12%
Fill_ij__	28.05%	25.67%	19.07%	27.75%	26.35%	23.22%
Fce	12.53%	11.94%	8.03%	12.50%	11.88%	10.18%
Erg3	5.35%	5.02%	3.49%	5.51%	5.23%	4.49%
MPI_	0%	8.51%	25.64%	0%	3.75%	9.21%

表4.3给出了两个并行版本中执行时间的组成。并行Mfold中执行时间主要分散在少数几个函数中，如W_、Fill_ij、Fce、Erg3和MPI相关函数。经过优化的并行Mfold中，MPI相关的函数的执行时间相比优化前大量减少，说明相关的优化技术减少了数据同步开销，这也是优化版本性能提升的主要原因。

4.4　本章小结

了解RNA序列的二级结构，对于理解RNA的功能具有重要的意义。Mfold是生物信息领域广泛使用的预测RNA二级结构的软件。本文首先研究了Mfold中使用的动态规划算法，分析了算法中的数据依赖关系；接着使用MPICH实现了并

行版本的Mfold；然后研究使用进程间通信技术，进一步优化Mfold在SMP系统上的性能潜力；最后通过实验测量并行版本的加速比，分析优化前后性能差异的原因，实验表明未经优化的并行版本可扩展性较差，使用16个处理器时仅取得6倍左右的加速，而优化版本达到了12倍加速，使用性能剖析器测得的结果表明，优化版本减少了Cache缺失的绝对数量，降低了数据传递开销，是取得较好加速比的主要原因。

进一步的研究，可以开发更多的科学和应用程序，提升它们在共享存储上的性能；设计一个源到源转换工具，自动地把已有的MPI程序转换成可以在SMP系统上高效运行的程序；比较经过优化的MPI程序与共享存储系统上专有编程模型OpenMP、phtread等编写的并行程序的性能。

第 5 章　CMP系统上基于内容的图像检索系统的并行优化

内容提要　随着互联网和多媒体技术的发展，图像信息日益丰富，高效准确的图像检索技术是查找有用信息的关键技术。基于文本的图像检索系统不能适应海量图像信息检索要求，促使人们研究基于内容的图像检索技术。基于内容的图像检索系统是一件非常耗时的工作，主要是由于图像数量庞大和检索系统算法的复杂性引起的。多核技术的发展，意味着处理器能有更强大的性能，为加速基于内容的图像检索系统提供了可能。目前大多数的程序还是单线程的，很难从多核CPU获益，因此，必须进行并行编程，开发并行性，才能有效利用多核CPU。本章首先介绍了基于内容的图像检索的背景和基本概念、当前主流的多核技术，以及OpenMP编程接口，接着介绍了一个基于内容的图像检索系统实现QBE，给出了基本框架和各模块串行算法；在此基础上研究在多核系统上CBIR系统的并行实现，以及优化技术，通过实验测量并行CBIR 系统分别在8核和16核两个多核系统上的性能，比较两个多核系统上的性能差异，分析导致差异的原因，最后使用性能剖析器进行详细的可扩放性和存储系统性能分析，确定影响并行CBIR系统整体性能的因素，为多核系统上应用程序的并行优化提供借鉴。

5.1　研究背景

5.1.1　基于内容的图像检索背景

随着数字化图像采集设备（摄像机、照相机、扫描仪）和大容量存储器的普及应用，每天都会产生数以万计的图像和视频，包括许多类型的图片和视频，如科学，医学，地理，生活等。这些数字图像中包含了海量的、无序的、分散的有用信息，无法直接有效地访问和利用。图像检索技术[71]研究把这些海量的图像信息进行有效的索引和分类，进而便于人们快速而准确地浏览、搜索和管理自己感兴趣的图像。

当前，大规模商用的图像检索引擎都是基于文本关键词的图像搜索（Text-Based Image Retrieval，TBIR），如谷歌图片搜索和百度图片搜索。基于关键词的图像检索系统，是建立在图片视觉内容的文字信息描述或图像元数据（大小、格式、色调等）的前提下，把对图像的检索转换成对图片文字描述信息的检索，进而可以利用当前相对成熟的文本搜索技术实现图像检索。基于关键词的图像

检索技术有着其本质的缺点，当处理海量图像数据的时候缺点更加明显。首先，对于图片的文本标注基本都是人工方式进行输入的，手动图片内容标注是一件既耗时又枯燥的工作；其次，图片内容丰富，仅用几个关键词或少数几句话，很难充分表达清楚整幅图像的内涵，会出现词不达意的情况；第三，个人对于图像有主观的理解，对同一幅图像不同的人给出的描述可能会相差很大，主观因素会影响文本信息描述的准确性；第四，基于文本的检索方法无法精确定义查询内容。

随着图像数量的爆炸式增长，上述基于文本的图像检索中的问题更加突出。为了克服基于文本的图像检索技术的局限性，在上世纪90年代早期，研究人员开辟了图像检索的另外一个方向，提出了基于内容的图像检索技术（Content-based Image Retrieval，CBIR）[72]。基于内容的图像检索技术利用图像的本身底层视觉内容，如颜色、纹理、形状、位置等，代替文字说明来描述图像信息，使用图像处理技术、模式识别技术和计算机视觉技术，建立图像特征库，使用数据库技术、人机交互技术和信息检索技术实现相似图像检索。CBIR区别于传统的基于关键词的检索技术，运用了图像理解技术，提供了一种从大规模图像数据库中，根据需求自动进行检索的方法，把人从繁重枯燥的图片内容标注中解放出来，图像的本身底层视觉内容具有较强的客观性，减少了人为主观因素的影响。CBIR提出以后，得到了很多学者和机构的重视，近年来取得了长足进步，在互联网、图像数据库、医疗图像处理等领域得到广泛应用，其中比较著名的系统有IBM的QBIC[73]，哥伦比亚大学的Visualseek[74]，MIT的PhotoBook[75]，UIUC大学的MARS[76]等。当前出现了一些基于内容的商用搜索引擎，如google similar-images[77]、picitup[78]和TinEye[79]。这些系统目前还处于发展阶段，给基于内容的图像检索注入了新的活力。但CBIR技术现在还存在着不少问题，其系统功能和效率都很有限，迄今为止，还没有一个成熟的CBIR系统为用户提供可靠方便的图像检索服务，工业界和学术界都还需深入探索和研究。本章主要研究利用当前流行的多核处理器，通过并行计算技术来加速CBIR系统的性能，研究多核系统上应用程序的性能优化方法，CBIR系统涉及到的图像处理、数据索引、模式匹配和结果评价等技术不是本章的研究重点。

5.1.2 多核技术介绍

CBIR系统通常涉及到比较大的数据量和计算量，因此需要功能强大的处理器支持。自上世纪70年代初，Intel发布世界上第一颗微处理器4004以来，处理器性能遵循着摩尔定律不断提高[80]。传统处理器性能提升方法是发掘指令级并行性，指令级并行性是指处理器同时发射和执行多条指令的并行能力，主要包

括三个方面：深度流水线（Super Pipeline）、超标量（Super Scalar）和存储系统优化。

(1) 深度流水线技术，不断提高时钟频率，增大单位时间内的时钟周期数，把指令的执行分成若干个阶段，各阶段按照流水线的方式执行，一个时钟周期内不同指令的各阶段可以在不同的部件上执行，整体上看相当于同时执行多条指令。流水线技术通过利用指令重叠，让处理器单位时间内可以执行更多的指令，从而提高运行速度，流水线级数越多，重叠执行的执行机会就越多，提高主频，意味着单条指令执行时间较短，这也是微处理器厂商一直以来的杀手锏。处理器的主频在微处理器厂商不断推动下，到2005年，微处理器进入3GHz时代。

(2) 超标量多发射技术，超标量技术是在单个时钟周期内同时发射多条指令到多个功能部件，让指令重叠执行，从而每个时钟周期内可以执行更多的指令，通常采用分支预测技术、猜测乱序执行和动态调度技术等，来发掘可能更多并行执行的指令。

(3) 存储系统优化技术，存储层次的分层设计，采用多级缓存结构，缩小处理器和主存之间的速度差异，降低了数据延迟，减少了指令停顿时间。

指令相关会造成流水线停顿，指令间相关性较多的程序，超标量处理器无法找到足够的指令填充全部指令发射槽，就造成了水平浪费（Horizontal Waste）；如果由于访存指令会暂停执行，某些时钟周期可能会出现空置的指令发射槽，就造成了垂直浪费（Vertical Waste）。图5.1展示了现代处理器中，可能造成的指令发射槽垂直浪费和水平浪费情况，横向代表处理器多个发射槽，纵向表示时间发展，不同的填充纹理代表不同线程的指令正在执行，空白块代表该指令发射槽浪费[81]。

挖掘数据级并行性是另外一种提升处理器性能的技术。现代处理器都支持SIMD（Single Instruction Multiple Data，单指令多数据）指令。SIMD指令是处理器提供数据级并行度的基本技术，同一时刻一条指令可以同时操作多组数据进行计算。现在的处理器都支持SIMD指令，如Intel的MMX、SSE2、SSE3、SSE4、SSE5、AVX指令集，AMD的3DNOW!，IBM的AltiVec指令集，具有多个执行部件。SIMD指令执行时，多个执行部件同时进行访存获得所有操作数，然后对一组数据同时进行向量运算，每组数据由多个操作数组成，所有的操作数执行相同的操作。SIMD指令特别适合多媒体应用中的向量运算。SIMD指令根据多媒体应用中存在大量数据并行性和处理器具有多个运算部件的特点，利用数据级并行性，使用一条指令完成多个一般指令的操作，从而可以减少总

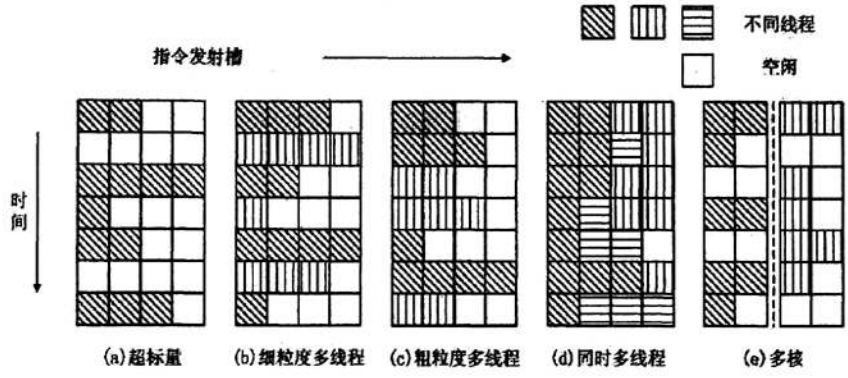


图 5.1 超标量、多线程、SMT、多核处理器中指令发射槽填充情况

的指令数，实际是一种向量运算的形式，能有效提高目前流行的多媒体处理、浮点运算、整数运算速度，提高了程序的性能，降低了功耗，提高了资源利用率。图5.2展示了一个典型的SIMD操作，原来需要四条指令完成的计算，使用一条SIMD 指令就完成了，从而获得了数据级并行。

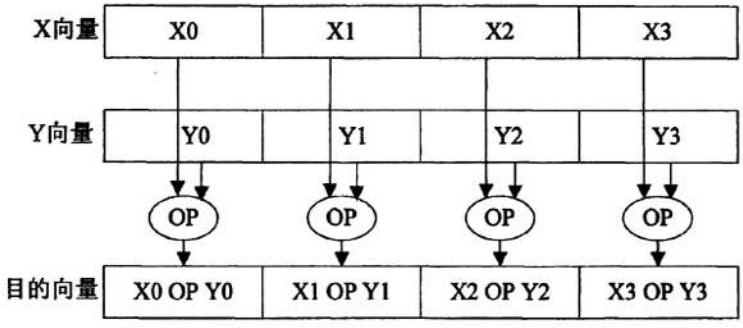


图 5.2 SIMD模型

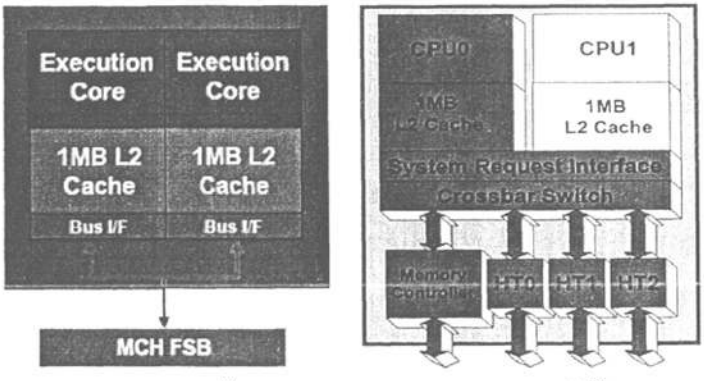
由于制造工艺、能耗和稳定性的限制，传统改善处理器性能的技术已经走到尽头[82]。为了进一步提高处理器性能，不仅需要关注单线程的指令级并行性，还需要开发线程级并行性。人们认识到开发线程级并行性是处理器发展的下一个重要方向，据此相继开发了超线程技术和多核技术。研究人员提出了多种多线程处理器技术，如多线程处理器（Multithreaded Processor）、同时多线程处理器（Simultaneous Multithreading，SMT）和单片多核处理器（Chip MultiProcessor，CMP）。

多线程处理器，支持多个线程轮换执行，能够快速地进行上下文切换，保存和恢复每个线程的局部状态，如寄存器信息，程序计数器PC值等。单一个线程由于等待存储访问、IO操作或指令相关而停顿时，可以切换到另外一个线程执行，保存每周期都有指令发射，从而避免了处理器闲置，消除了垂直浪费。

根据粒度不同，又可以分为粗粒度多线程处理器和细粒度多线程处理器。粗粒度多线程处理器只有当执行线程停顿时，才进行上下文切换；细粒度多线程处理器，在每个时钟周期都进行线程切换，图5.1（b）和（c）分别展示了两种处理器指令发射槽利用情况。

同时多线程处理器[83]，虽然单线程处理器中指令发射槽每周期都有指令发射，但是同一时刻只有一个线程的指令执行，仍然在一个时钟周期中无法填满整个指令发射槽，不可避免的有水平浪费存在。同时多线程处理器通过在超标量处理器上增加一些硬件控制，通过从多个活跃线程中选择指令执行，允许一个时钟周期内发射多个线程的指令到功能部件上执行，尽可能提高功能部件的利用率，可以同时减少水平和垂直浪费，如图5.1（d）所示。Intel通过超线程技术来实现同时多线程，把一个单核物理处理器模拟成两个逻辑核，可以并发执行两个线程，进而减少了处理器闲置时间，提高处理器性能。使用超线程技术的两个逻辑处理器，共享执行单元、缓存和总线接口，交替执行指令。Intel表示，超线程技术在只增加5%的芯片面积的情况下，可以带来20%左右的性能提升。

多核处理器也叫做片上多处理器（Chip Multi-Processor，CMP）[84,85]，在单个芯片上集成两个甚至更多个处理器内核，可以实现在不同核上同时运行不同线程的指令序列，如图5.1（e）所示。芯片内集成多个内核，每个内核拥有独立的执行单元、指令流水线、寄存器和一级Cache，内核间通信具有更高的通信带宽和更短的通信延迟，在挖掘多线程应用程序并行性方面具有天然优势，各线程可以分配给一个物理内核去执行，从而成倍提高芯片整体计算能力。CMP采取的是任务划分的方式，当没有足够的任务时，会带来一定的性能损失。传统的程序都是串行的，必须改写成并行程序，才能从硬件革新中获益。



(a) Intel双核　　(b) AMD双核

图 5.3 Intel和AMD的双核处理器结构图

CMP单个内核设计简单，全局信号较少，资源竞争少，具有设计简单、扩展性好、功耗低、通信延迟低等特点。多核处理器以其众多优势，被工业界和学生界广泛接受和推广，已经成为微处理器的主流结构。2001年IBM发布了第一款用于服务器领域的通用双核处理器Power4[86]，每个处理器都集成了两个64位的1GHz+的PowerPC核心。之后主要的微处理器厂商，如Intel[87]、AMD、Hp和Sun等都已经发布了自己的多核心芯片。图5.3中展示了Intel和AMD于2005年发布的的两款双核处理器Pentium D和Opteron的结构。2010年四核的处理器将成为主流，未来将会有更多核的处理器出现。2007年初在IEEE国际固态电路会议（ISSCC）上，Intel展示了尚处于实验室研究阶段的80核北极星（Polaris）处理器，浮点运算能力达到1.01TFlops（每秒万亿次）。虽然Polaris只是Terascale工程下的挖掘浮点能力的研究原型，但研发技术是适合主流产品的，为未来计算世界勾画了一副美好的画面，展现了多核处理器的发展前景非常广阔。

多核处理器，较之以前的单核处理器，能带来更多的性能。多核结构具有良好的性能潜力和实现优势：

(1) 多核结构将芯片划分成多个处理器核来设计，每个核都比较简单，有利于优化设计。

(2) 多核结构有效地利用了芯片内的资源，能够有效开发程序的线程级并行性，带来性能的成倍提升。

(3) 处理器核之间的互连缩短，提高了数据传输带宽，有效地共享资源，功耗也会有所降低。

通过性能优化和并行计算技术充分发挥现有通用多核处理器的计算能力，以提高当前基于内容的图像检索系统的响应速度。多核处理器，提供了三种并行度：指令级并行度、数据级并行度和线程级并行度：

(1) 使用流水线、超标量技术，提供指令级并行度，同时可以执行多条指令。

(2) 使用SIMD技术，提供数据级并行度，每条指令可以操作多个数据。

(3) 集成多个核，提供线程级并行度，每个核独立处理一个线程的计算任务。

5.1.3 OpenMP编程模型

OpenMP[58]是目前共享存储系统上并行编程的工业标准，是实现程序并行化的一种强大、有效、方便和简单的方式，它主要包括如下一些部分：一组制导语句（Pragmas）、环境变量和运行时类库。编译制导语句用于线程创建、调

度和同步；环境变量用于运行时的行为控制；运行库用于设置和查询线程属性。目前gcc编译器、Intel C++编译器和Visual Studio 2008均支持OpenMP标准。

程序员只需在串行代码中添加少量的OpenMP制导语句，然后编译器会根据制导语句把串行代码自动转换成多线程执行的代码，增量式的开发易于实现串行程序到并行程序的转换，减轻了程序员的负担。目前大多数平台的编译器都已经支持OpenMP，如果编译器不支持OpenMP，程序中的制导语句将会被视为注释。OpenMP是一种高层编程模式，程序员使用制导语句告知编译器哪里可以并行、使用几个线程、调度策略，余下的细节交给编译器和运行时环境来处理，把程序员与底层并行细节隔离开来，这些细节包括：并行任务划分、线程管理、数据共享与同步等。因此，程序员只需对串行程序做少量修改，就能从多核多线程处理器获得较高的性能。

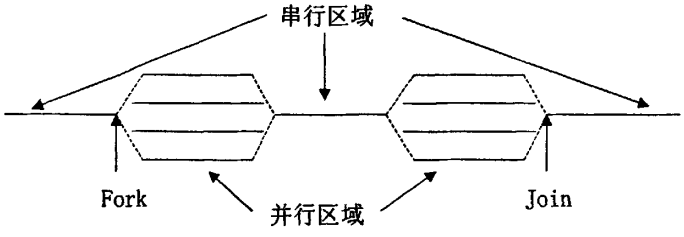


图 5.4 OpenMP的创建合并模型

如图5.4所示，OpenMP中使用了一个简单有效的创建-合并模型支持并行执行[88]。OpenMP程序由一个单线程开始执行，该线程被称为主线程。主线程是串行执行的，当遇到一个并行区的制导语句时，主线程创建一个包括自己在内的工作线程组，该线程组内的所有线程协作执行并行制导语句作用的并行结构中的代码，线程组内的线程访问同一个全局共享存储器，数据有共享和私有两种，共享数据可以被所有线程访问，私有数据只能被其拥有线程访问。执行过程中系统调度多线程执行。在并行结构结尾处，有一个隐式的同步路障，并行结构计算结尾时合并中间结果，已经执行结束的线程在路障处等待其他线程，所有的线程都执行完成到达这个隐式路障后，在并行结构的结尾处执行合并操作，之后只有主线程继续执行后续代码。程序中可以有多处并行制导语句，程序执行时也会相应地创建和合并多次。

5.2　基于内容的图像检索系统实现

本章首先实现了一个基于内容的图像检索系统QBE（Query By Example），在此基础上多方面优化其在多核系统上的性能。QBE使用示例图片代替文本关键词表达查询意图，示例图片可以更精确直观地描述用户想要查询的图片。

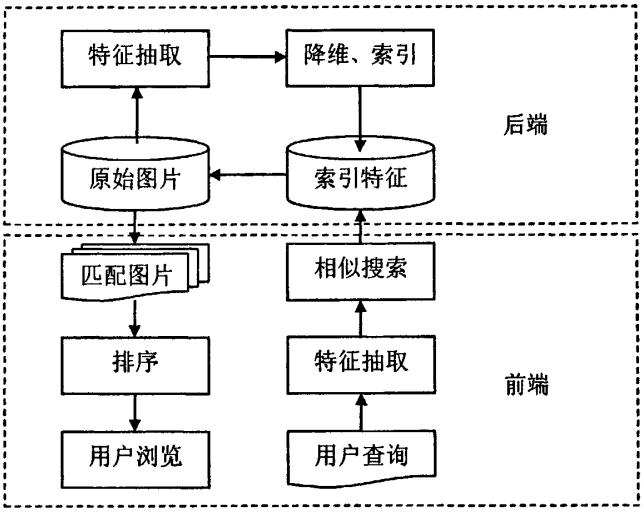


图 5.5 QBE系统框架图

图5.5给出了QBE系统的框架图。整个图片查询系统分为前端用户界面和后端处理系统两个部分。后端处理系统自动地抽取原始图片的视觉内容为一系列特征向量，各特征向量保存为一个特征数据库。视觉内容为图片本身低级的视觉表征，包括颜色、纹理和形状等。QBE使用Gabor[89]和Canny[90]边缘梯度直方图两个纹理特征。这两个特征有着较强的区分能力，实际使用中性能比较好。前端用户界面是用户和后端系统交互的桥梁。当进行图像检索时，用户提交一副具有代表性的图片描述想要查询的内容，系统计算查询图片的特征向量，然后基于Gabor特征的k近邻搜索技术被用来快速地在特征数据库中查找k个相似特征向量，接着计算查询特征向量与这k个特征向量的相似度，相似度是根据Gabor特征和Canny边缘梯度特征综合计算得到的值，两种特征在相似度计算中权重相等，最后QBE返回一些与查询图片视觉上相似的结果图片，并按相似度大小排序。下面几小节将详细介绍查询过程中的各部分。

5.2.1　特征抽取

特征抽取是用来获得图片视觉特征的方法，是基于内容的图片检索技术的基础。如何描述图像的视觉特征，关系到整个系统检索效果和效率。一副图像的视觉特征主要包括：颜色、纹理和形状。通过对这些特征的提取，组成一个特征向量来描述一副图像，在之后的检索中，查询图片也同样被表示成向量，这样图片相似性的比较就等价为对应特征向量间距离的比较。Gabor滤波和Canny边缘特征在实际中区分度强，效果好，QBE中选用这两种特征来描述查询图像。QBE实现中使用了一组6个不同方向、5个不同尺度的Gabor滤波器组，

组合起来共有30个滤波器。滤波结果的平均值和标准差组成Gabor特征的特征向量。QBE中根据方向和幅度计算边界上点的梯度直方图得到一个图像的特征向量表示。

5.2.2　高维向量降维

由于图像的数量巨大以及每个图像的特征向量维数比较高，图像间相似度的计算代价很大。降维是解决维度灾难的一个有效途径[91]。在尽可能多地保存原始特征向量信息，减少精度损失的前提下，把特征向量的维数进行压缩，使用降维后的特征向量进行相似度计算，从而减少了计算量，提高检索系统的整体速度。QBE系统中采用主成分分析法[92]把特征向量投影到一个新的低维空间中，在新的空间中计算距离和相似度。主成分分析法是一个精度损失少，效率高的降维方法，在图像压缩、模式识别等领域广泛应用。主成分分析中，计算主要消耗在矩阵向量乘的计算上。在QBE的实现中，Gabor原始特征向量为240维，降维后变为159维；Canny边缘梯度原始特征向量为320维，降维后变为289维。

```
for(i = 0; i < Query_size; i++)
    for(j = 0; j < Database_Size; j++)
    {
        Dis[i][j] += Dis(query[i],database[j]);
        if(dis[i][j] < curMax)
            Insert(j);
    }
```

图 5.6 候选图像集合选取的伪代码实现

5.2.3　候选图像集合构造

为了加快整个图像检索系统的响应速度，首先根据Gabor特征快速过滤掉大部分不相似的图像，得到一个大小为k候选相似图像集，相似度的计算和结果的排序都是针对该候选相似图像集。这样会减少不必要的计算，减少系统的整体响应时间。候选图像的过滤需要用到k近邻查询技术，已经有不少相关的研究成果如kd-tree[93]，R-tree[94]，LSH[95]等。这些技术对维数低的向量比较有效（20维以内），当维数逐渐增高时，所有的方法都退化为最原始的蛮力搜索。Gabor特征经过降维后，维数依然高达159维，这些方法都没有特别好的效果。QBE中使用了易于实现的线性搜索方法。搜索过程中维护一个大小不超过k的有序队列，当搜索到一个与查询向量间的距离小于当前队列中的最大距离的特征向量时，把当前队列中距离查询向量距离最大的特征向量丢弃，插入刚刚找到

的特征向量，更新优先队列。所有数据库中特征向量都遍历结束后，优先队列中的特征向量对应的图像为候选图像集。以上过程的实现代码如图5.6所示。

5.2.4　相似度匹配和结果排序

在基于内容的图像检索中，两个图像之间通过计算特征向量的距离，来评价它们之间的相似度。特征向量的相似性匹配，通常采用空间向量模型，即把特征向量看作是空间中的点，把这些点之间的距离作为向量间的相似度，距离越近，特征向量越相似，系统返回给用户的结果是按照相似度从大到小的顺序来排序。QBE中采用欧式距离来计算空间中点的距离。例如图片Ii和Ij的视觉特征向量分别表示为：$Vi = (v_{i1}, v_{i2}, \ldots, v_{id})$，$Vj = (v_{j1}, v_{j2}, \ldots, v_{jd})$。其中$d$表示特征向量的维数。它们之间的相似度可以按下面的公式计算：

$$D(Vi, Vj) = \sqrt{\sum_{k=1}^{d}(v_{ik} - v_{jk})^2} \tag{5.1}$$

计算查询图像与候选图像集合中的图像之间的相似度，按照相似度大小排序，从候选集合中选择相似度最高的若干张图像作为查询结果。相似度是把Gabor特征和Canny边缘梯度特征计算得到的相似度按对应权值累加得到的值，QBE当前实现中两种特征在相似度计算中权重相等。

5.3　挖掘CBIR的多级并行性

从第5.1.2小结的介绍可知，多核处理器提供了三种并行度，通过开发应用不同层次的并行性，可以利用潜在的并行机会提高应用软件的性能。基于内容的图像检索系统存在多种层次的并行性，可以从以下三个层次挖掘CBIR系统的并行性，即指令级并行性（Instruction-Level Parallelism，ILP）、数据级并行性和线程级并行性（Thread-Level Parallelism，TLP）。

5.3.1　优化指令级并行性

提高指令级并行度的关键是消除指令间的相关性和减少指令等待数据时间。通过循环展开，把循环体展开若干次，指令的混合度更好，可以消除循环间数据相关，可以提供更多的指令让编译器调度，增大指令并行执行的机会，减少水平浪费。分支会打乱指令执行的顺序，影响指令的并发执行。处理器通常采用分支预测技术，来选择可能的分支，一旦预测错误，会带来严重的性能惩罚。通过消除不必要的分支和采用一些易于预测的实现技术，可以减少指令执行因为分支而停顿或则由于执行了错误的分支受到的惩罚。降低存储开销，可以减少指令等待所需数据的时间，有效地降低垂直浪费，下面介绍采用循环分块技

术优化QBE 中Cache访问性能的具体实现。

在候选图像选取阶段，处理一个查询时，所有的数据库中的特征向量都会被依次装入Cache，使用一次后由于映射冲突或者Cache容量不足而被替换出Cache，处理其他查询时，会重复这一过程，同一特征向量会被多次载入和踢出Cache，Cache的局部性很差。QBE使用数据分块技术来提高Cache利用率。数据库中的特征向量被划分成一系列的块，每块数据可以存入最后一级Cache，多个查询同时在一个数据块上做搜索，处理完一个数据块后转移去处理另一个数据块，直到所有的数据块都被处理过后，多个查询过程结束。数据分块后，每个数据块在Cache中会被多次复用，极大提高了Cache性能。数据分块的伪代码如图5.7所示。

```
for(k = 0; k < Num_Block; k++)
    for(i = 0; i < Query_Size; i++)
        for(j = idx; j < idx + blocksize; j++)
        {
            dis[i][j] += Dis(query[i],database[j]);
            if(dis[i][j] < curMax)
                Insert(j);
        }
```

图 5.7 使用分块技术的候选图像集合选取实现

5.3.2　优化数据级并行性

图像特征提取阶段，有大量的向量的操作，数据级并行丰富，使用SIMD指令可以很好地加速特征提取过程。下面使用两个单精度浮点四维向量的点积来说明SIMD过程。对于$X \cdot Y = x_0y_0 + x_1y_1 + x_2y_2 + x_3y_3$，一般采用C语言实现如图5.8所示，需要执行4个乘法指令，3个加法指令。

```
float dot4(float x[4], float y[4])
{
        return x[0]*y[0] + x[1]*y[1] + x[2]*y[2] + x[3]*y[3];
}
```

图 5.8 两个四维向量点积的C语言实现

而采用SIMD intrinsics可以实现如图5.9所示。其中_m128为打包的数据结构，可以保存128位的数据，因此可以顺序放置4个单精度浮点数。语句$Z = _mm_mul_ps(X,Y)$执行X和Y两个向量中对应位置元素的乘法操作，结果为$z_0 = x_0 * y_0$，$z_1 = x_1 * y_1$，$z_2 = x_2 * y_2$，$z_3 = x_3 * y_3$。语句$Z = _mm_hadd_ps(X,Y)$执行X和Y向量的水平累加，计算结果为$z_0 = x_0 + x_1$，

```
_m128 dot4SIMD(_m128 X, _m128 Y)
{
    _m128 Z = _mm_mul_ps(X,Y);
    Z = _mm_hadd_ps(Z,Z);
    Z _mm_hadd_ps(Z,Z);
    return Z;
}
```

图 5.9 两个四维向量点积的SIMD实现

$z_1 = x_2 + x_3$，$z_2 = y_0 + y_1$，$z_3 = y_2 + y_3$。因此dot4SIMD的计算结果向量Z中$z_0 = x_0 * y_0 + x_1 * y_1 + x_2 * y_2 + x_3 * y_3$保存了点积的结果。因此，使用3条指令完成了C语言实现版本中7条指令完成的计算任务。特征提取中有大量类似的操作，都可以借助SIMD指令来实现数据级并行性，加快执行速度。

5.3.3 优化线程级并行性

线程级并行性为同一时刻执行多个线程的并行能力，是指在多核处理器上，利用多线程技术，把任务分解到多个计算内核上去执行，同时有多个指令序列在独立的硬件上面并行执行，从而加快了计算速度。对于一个串行程序，需要把它转换成并行程序，才能有效利用多个计算内核的计算能力。下面介绍挖掘QBE系统中线程级并行性的方法。

首先，系统可以同时并行处理多个查询请求，这是一种粗粒度的任务级并行方案。不同用户的查询是独立的，相互之间没有关联，多个独立的查询具有自然的并行性，每个查询可以分配给一个内核去执行。服务端程序有足够的查询需要处理，系统的整体吞吐量比较高，因此可以从这种并行模式中获得极大的益处。当同一时刻收到的查询请求较少时，有些处理器内核没有任务可做，这时粗粒度的并行方案效率会比较低。其次，对于每个查询请求，可以挖掘更细粒度的并行性。一方面，通常一副图像内有上万个像素，把图像划分若干块，每块分配给一个处理器内核去处理。另一方面，图像数据库中有大量的图像，需要计算查询图像与数据库中图像之间的相似度，这些计算可以并行地执行。细粒度并行可以加速某一个特定的查询，可以减少一个查询的响应时间。细粒度并行也有一些缺点，如算法设计时需要考虑数据和任务划分，处理数据依赖关系和数据同步，并行额外开销比较大。最后，还可以采用更细粒度的并行，图像的特征向量通常是高维的，向量相关的运算可以并行处理，但是这种更细粒度的并行开销会很大，一般用数据级并行来实现。

通过以上分析，可以看出QBE系统中包含丰富的并行性，粗粒度和细粒度并行有各自的优缺点。QBE系统主要面向服务器端应用，因此主要采用了易于实现的粗粒度并行方式。下面给出使用OpenMP编程模型，并行化QBE系统中关

键模块的方案。

特征提取（FE）。这个模块同时处理多个查询图像的特征向量提取。所有的查询可以均匀地分配到所有的可用计算内核，比如有p个处理器内核，那么每个内核分配$Query_size/p$个查询。使用OpenMP的parallel for制导语句把多个查询并行化，伪代码如5.10所示。

```
#pragma omp parallel for
{
    for(i = 0; i < Query_Size; i++)
    {
        Gabor_Feature(i);
        CEH_Feature(i);
    }
}
```

图 5.10 并行特征抽取的伪代码实现

候选图像过滤。使用与特征提取相同的粗粒度并行技术，并行计算不同查询请求的候选图像数据库。如果使用细粒度并行，不同的线程共享有序队列，更新时会产生数据冲突，需要使用锁来保证正确地更新，这会降低整体的可扩放性，而粗粒度并行方案中，不同线程间没有数据共享，避免了处理数据冲突的开销。

矩阵操作。特征向量降维中大量用到矩阵向量操作。Intel的数学函数库MKL提供高度优化和线程化的矩阵处理函数，如dgemm等。MKL对当前多核x86平台进行了深入而全面的优化，并且将会不断支持未来的计算平台，因此使用MKL可以自动地从最新处理器架构中获得最大的性能。QBE中使用MKL来实现矩阵运算的隐式并行。

整个系统并行化之后，采用负载均衡、消除同步开销、处理器关联等优化技术，进一步最大化QBE系统的并行效率，提高并行QBE系统的整体性能。

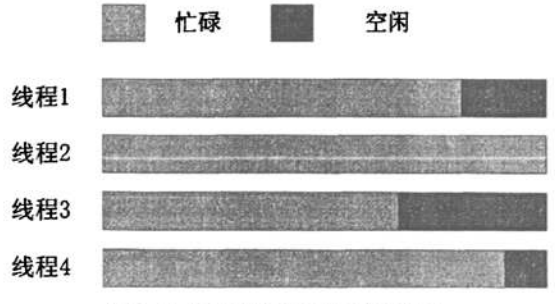


图 5.11 多线程负载不均衡示意图

在并行系统中，如果一个处理器分配的负载比其他处理器多，那么它将消耗更多的执行时间，在计算的最后阶段只有这些少数处理器在进行运算，其他处理器处于闲置状态，这样将会严重的限制并行系统的整体性能，如图5.11所示。并行系统中需要通过平衡所有任务的负载，使得每个并行线程获得大致相同的负载，以保持所有的处理器忙碌。QBE面向服务器端应用，具有丰富的并行性，使用简单的静态调度既可以均匀地分配任务，又可以最小化调度开销，在实际中取得了比较好的性能。

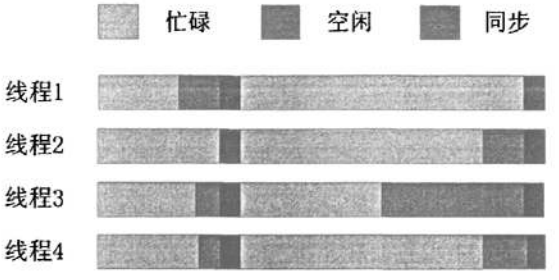


图 5.12 多线程同步操作示意图

程序运行时，多个线程间不是独立的，执行到一定阶段需要进行数据通信和同步操作。如图5.12所示，同步会造成某些线程停下来等待另一个线程的执行，等待过程中闲置的处理器浪费了计算资源，因而会降低系统并行度。同时同步操作会带来额外的开销。通过更有效的任务划分，设置合理的同步点，减少通信次数，传输长数据包，能够减少同步带来的性能损失。QBE中采用了粗粒度并行，进程间相关性少，无需特别地同步操作。内存分配和释放操作是由操作系统控制的，使用锁来维护同一时刻多个进程对内存的申请和释放操作，锁限制了系统的并发。在优化时，减少频繁地内存申请和释放，每个进程维护自己的内存池，减少系统调用的发生，消除了内存管理的同步开销。

在特征提取阶段，不同的线程处理不同的查询，线程之间没有数据共享，每个线程对前端总线带宽要求比较高，调度线程在不同的socket上执行有助于取得较好的性能。在候选图像集选取阶段，不同的查询都需要处理同一个特征数据库，多个线程之间共享特征数据，此时调度线程在同一个socket上执行有助于提高Cache性能。

5.4　实验及分析

5.4.1　实验配置

本章使用了两台不同配置的多核系统测量CBIR的性能，一台为Xeon E5450系统，含有8个处理器内核，另一台为Xeon E7310系统，含有16个处理

器内核，两台机器的详细硬件参数如表5.1所示。软件配置方面，两台系统均运行Windows Server 2003，使用Intel C/C++编译器9.1。实验中用到的图像数据库中包含94,028副不同分辨率的彩色图像，共处理80个查询请求。

表 5.1 两个多核系统硬件配置情况

	Xeon E5450	Xeon E7310
CPU type	Dual-socket Quad-core	Quad-socket Quad-core
Core Speed	3.0GHz	1.60GHz
L1 data cache	32KB	32KB
L2 cache	4 × 6MB	8 × 2MB
RAM	8GB	8GB
FSB speed	1333MHz	1066MHz
Bandwidth	21.328GB/s	34.112GB/s

5.4.2 系统响应速度优化结果

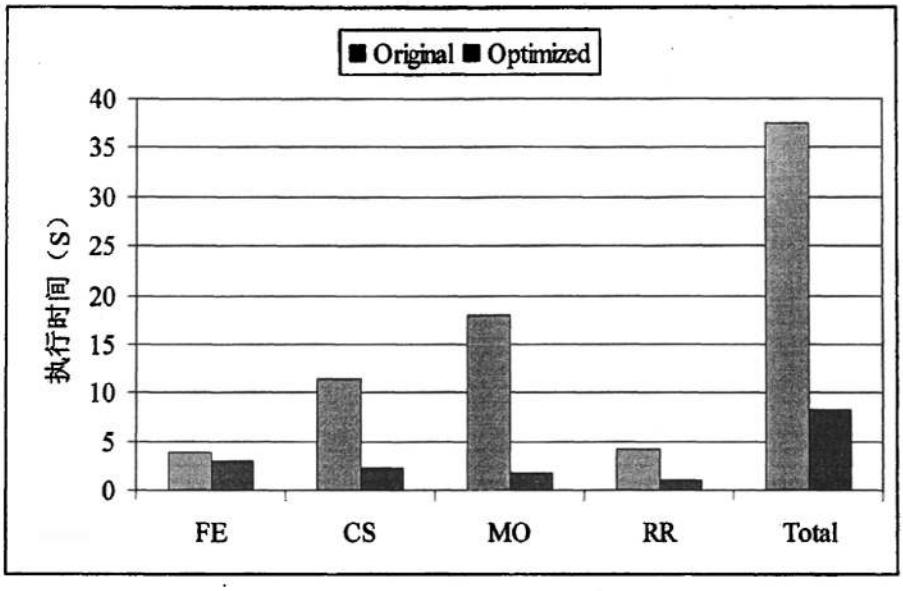
在以后的性能分析中，总的执行时间分成四个部分讨论，即特征提取（FE）、候选图像搜索（CS）、矩阵运算（MO）和结果排序（RR）。优化指令级并行和数据级并行是在串行代码上做的优化，称之为串行优化；线程级并行是使用OpenMP并行化，为通常意义上所说的并行，因此与之相关的优化称之为并行优化。

图5.13给出了两个系统上串行优化前后的各模块执行时间的比较。从图5.13(a)上可以看出，在8核系统上串行优化总共取得了4.6倍的加速。优化前平均每秒可以处理2.13个查询，优化之后平均每秒可以处理9.71个查询。从图5.13(b)上可以看出，在16核系统上可以得到类似的结果，串行优化后获得3.4倍加速，优化前平均每秒可以处理1.47个查询，优化之后平均每秒可以处理5.0个查询。

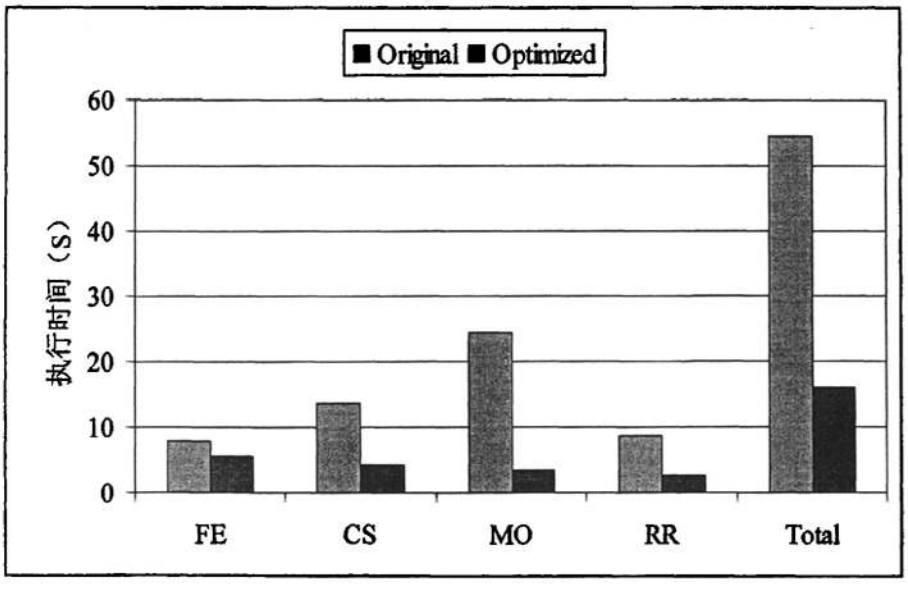
更进一步，经过并行优化后，QBE系统在8核系统上可以取得平均每秒处理54.6个查询的速度，在16核系统上可以取得平均每秒处理48.5个查询的速度。总的来说，经过串行和并行优化后，整体上取得了大概30倍左右的性能加速。

5.4.3 性能差异分析

图5.14中给出了两个系统上执行时间的对比，从图中可以看出8核系统上的执行时间要比16核系统上的执行时间少。性能的差异主要是机器硬件性能的不同引起的，两台系统有不同的计算能力和存储结构。一方面，8核系统的处理器主频为3.0GHz，16核系统的处理器主频为1.6GHz，8核系统单位时间内可以执行更多的指令。另一方面，8核系统二级Cache比16核系统大，图5.15给出了两个



(a) 8-core E5450 System



(b) 16-core E7310 System

图 5.13 两个多核系统上串行性能优化结果

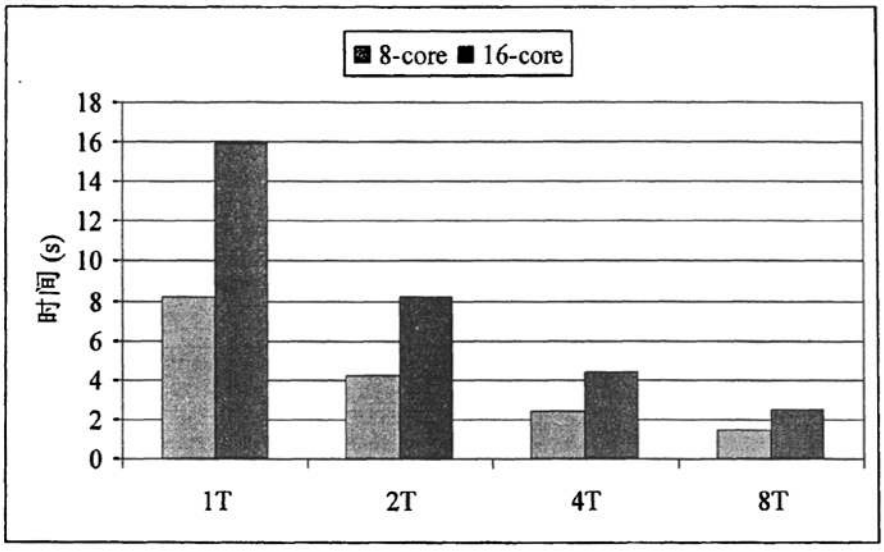


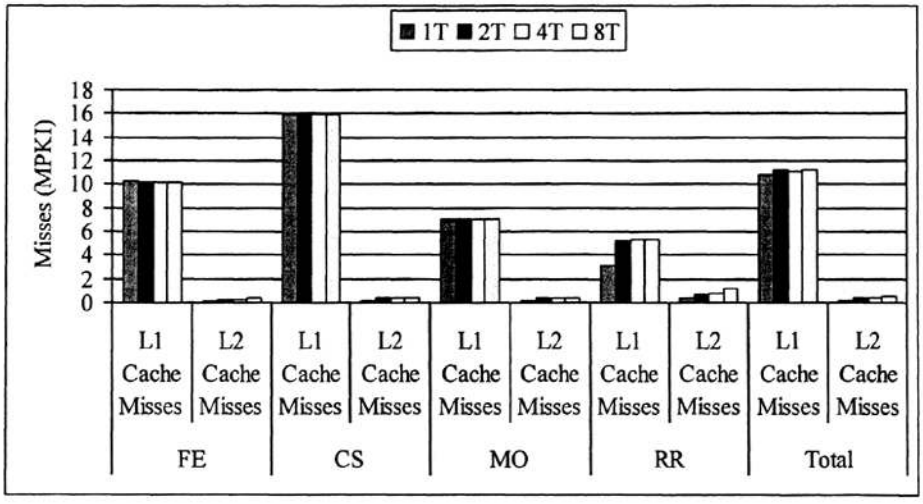
图 5.14 两个系统上执行时间对比

系统上每千条指令Cache失效率，可以看出8核系统上一级和二级Cache的失效率要远远16核系统上低，尤其是二级Cache的失效率差别更加明显，二级Cache失效会导致到主存中取数，主存的数据延迟比Cache要大两个数量级。以上两个原因解释了两个系统上性能差异的原因。

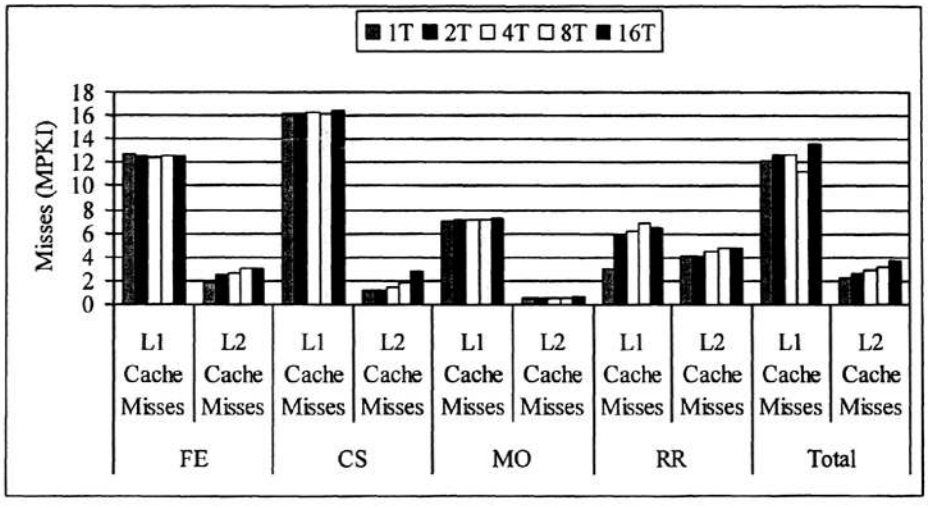
5.4.4　加速比分析

图5.16给出了两个系统上的并行加速比。依启动一个线程时的执行时间为基准，计算启动2、4、8、16 个线程时系统的加速比。图5.16(a)显示8核系统上使用2个线程可以近似可以取得线性加速，使用4个线程可以取得的加速比为3.4，使用8个线程可以取得的加速比为5.6。随着线程数的增加，性能获益越来越小。16核系统上取得相似的结果，图5.16(b)显示16核系统上使用两个线程取得加速比为1.9，使用4个线程可以取得的加速比为3.6，使用8个线程可以取得的加速比为6.4，使用16个线程可以取得的加速比为9.7。系统的整体性能加速基本与使用的处理器核数成正比，但与线性加速还有一定差异。

使用性能剖析器Intel Thread Profiler[96]和Intel VTune Analyzer[97]详细地分析影响QBE扩展性的因素，下面的分析是基于16核系统的，对8核系统有类似的结果。图5.17给出了执行时间的分解比例，从图中可以看出并行执行时间在总执行时间中占主导，随着线程数增多，并行开销，负载不均衡会有少量增加。非并行区比例的增加，是因为随着线程数增加，并行部分的执行时间绝对量大大减少，并行区执行时间在总执行时间中的比例也相应缩小，事实上非并行



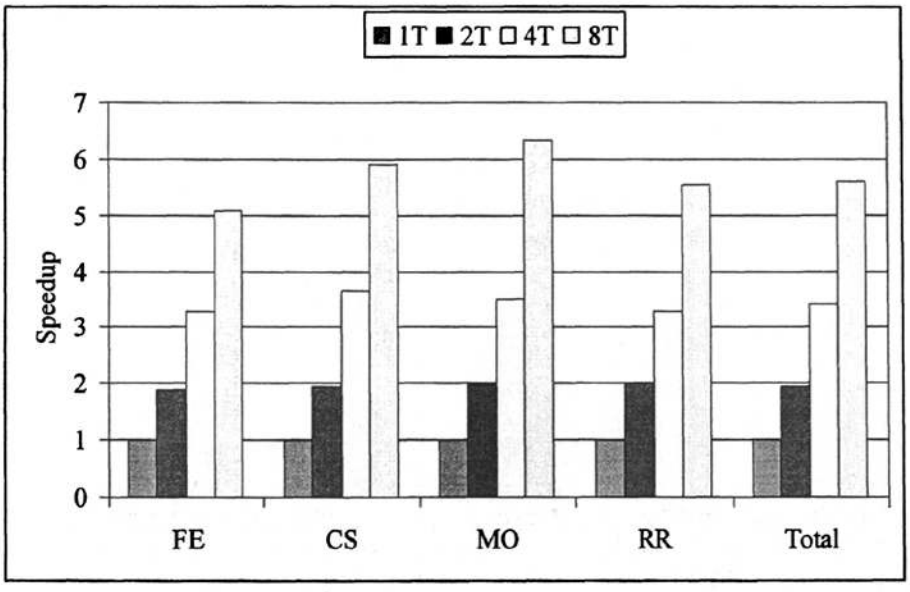
(a) 8-core E5450 System



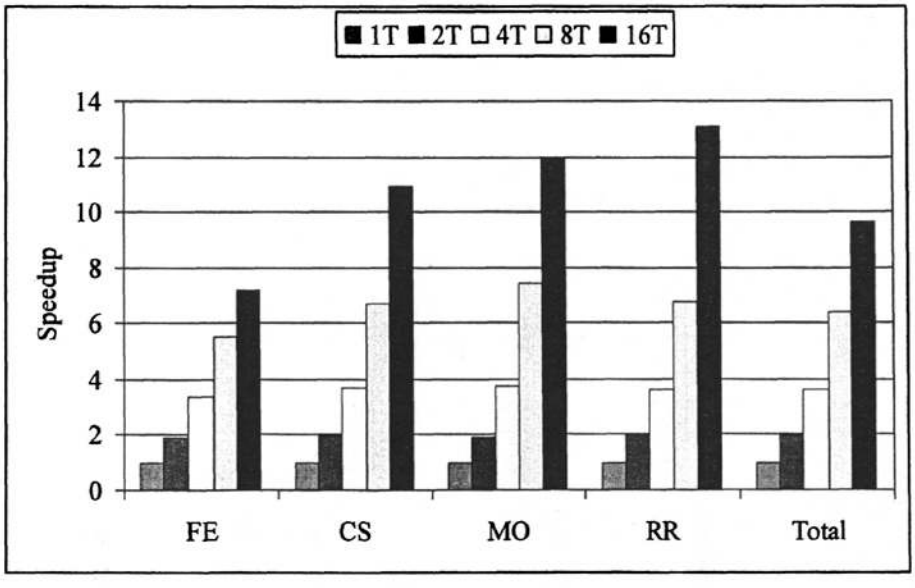
(b) 16-core E7310 System

图 5.15 两个系统上各模块千条指令Cache缺失率比较

区执行时间的绝对量并没有太大变化。根据Amdahl定律，非并行区限制了整个QBE系统的可扩放性。图5.15中显示了随着线程数增加，二级Cache失效率也相应提高。二级Cache性能的降低也在一定程度上限制了QBE系统的可扩放性。图5.18 给出了前端总线带宽利用率随线程数的变化情况。图中可以看出，前端总线的带宽利用率随着线程数增加，基本上线性地增长，使用16个线程时，前端总线数据传输带宽达到了7.7Gb/s。FE模块对带宽需求比较高，使用16个线程时带宽需求为9.8Gb/s，如此高的带宽需求限制了FE模块的可扩放性，使用8个



(a) 8-core E5450 System



(b) 16-core E7310 System

图 5.16 两个系统上各模块加速比对比分析

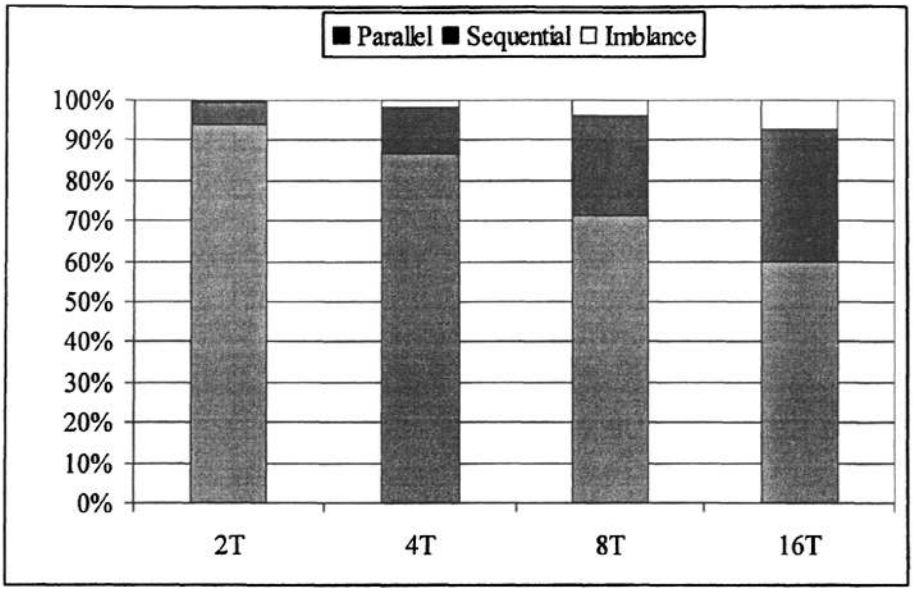


图 5.17 16核系统上运行时间各部分比例图

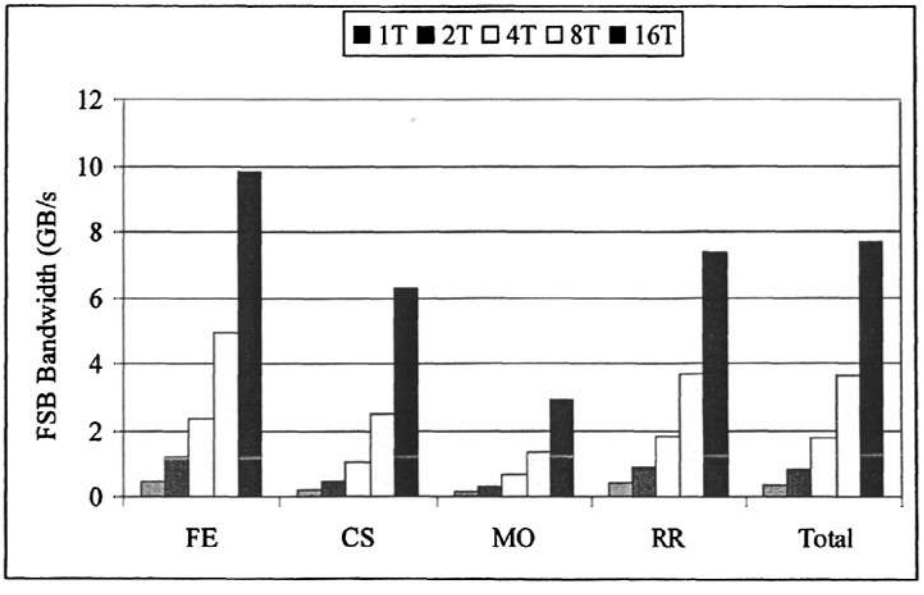


图 5.18 16核系统上各模块带宽使用情况

线程时加速比为5.5，使用16个线程时加速比为7.2。因此，可以看出高的带宽需求一定程度上影响了系统的性能加速比提高。

5.5　本章小结

技术进步令图像采集和存储变得非常方便，导致图像信息呈爆炸式的增长。如何从海量图像信息中，查找所需的图像信息，进一步利用这些信息，变得越来越困难。传统的基于文本关键字的图像检索技术已逐渐不能满足需要，基于内容的图像检索方法（CBIR）是解决海量图像检索的有效方法。但是由于CBIR对计算的需求很大，实际中CBIR系统的性能不高。

目前多核已经成为主流计算平台，单个芯片内集成的计算内核数不断增加。多核技术的发展，使得在通用处理器上加速CBIR系统性能成为可能。利用多核处理器，可以加速CBIR系统的响应速度，提高单位时间内可处理的查询数。本章设计了一个基于内容的图像检索原型系统，以此为基础，研究系统在一个8核系统和一个16核系统上CBIR的并行及优化技术，显著优化了系统的指令级并行性、数据级并行性和线程级并行性。CBIR经过优化，在8核系统上使用8个线程可以取得25.7倍加速，在16核系统上使用16个线程可以取得33.0倍加速。相关技术是多核系统上有代表性的优化方法，对于其他应用程序同样适用，可以最大化它们在多核系统上的系统；最后，比较了两个系统上CBIR的性能差异，给出了合理的解释，确定了影响多核系统加速比的主要因素，揭示多核系统上程序性能主要受限于存储系统性能、总线带宽和各级并行性。

第 6 章　面向CMP的定量化程序执行模型CRAM(h)

内容提要　定量的程序性能评测，可以提供更加详细、精确的程序执行信息，有益于更好地性能优化。本章首先介绍性能评测的基本概念和意义，回顾一些常用的定量性能评测方法，有分析、测量和模拟，以及多种方法的综合；接着，总结影响程序执行性能的关键因素，包括处理器、存储器、并行性等方面；然后，使用分析和测量结合的方法，研究基于共享存储的定量化程序执行模型，提出一个适用于片上多核系统的CRAM(h)模型，分析CMP系统上程序的执行行为，确定影响程序性能的主要因素，定量地对CMP系统上程序执行性能进行建模；最后，测量模型中的相关参数，使用模型来分析矩阵相乘的四种不同实现方式的执行时间，以此来验证模型的准确性和可用性。

6.1　程序性能评测

6.1.1　研究背景

性能评测是并行计算领域的一个重要研究方向，是在一个给定的并行计算机系统上，评估一个应用程序的执行性能。对程序进行精确性能评测具有非常重要的意义，并行系统的性能评测在并行算法设计、程序执行性能优化和体系结构设计中发挥着重要的作用。性能评测一方面基于应用需求，指导和分析体系结构设计；另一方面基于体系结构特性，指导和分析具体应用的性能优化。一个科学合理的评测方法和标准，有助于提高并行应用程序性能，发挥并行硬件的计算能力，提高系统利用率。

性能评测具体可以应用在以下几个方面：1）系统选择：通过性能模型指导选择适合具体应用的机器[98]，最大化其执行速度；2）系统设计：使用性能模型评测未来体系结构的性能，获取性能信息，指导新体系结构的设计；3）应用优化：性能模型可以揭示性能瓶颈，指导特定机器上性能优化；4）应用设计：性能模型评估实际、在建甚至未来体系结构上，应用的运行时间，指导应用开发过程，考虑各种针对性的优化实现。

对应用进行性能评测是一件极具挑战性的工作。现代计算机系统的软硬件趋于复杂，例如软件规模越来越大、行为多种多样；硬件集成众多复杂的技术，陆续引入了流水线、分支预测、超标量、超线程、多核等技术，精确程序性能评测的难度越来越大。

6.1.2　相关工作

定性的性能分析[99]是程序优化过程中最常用的性能评测方法，比如应用在频繁模式挖掘[100]和图片视频等特征提取[101]的优化中。通过定性分析可以得到瓶颈所在、优化的方向和手段。但是，定性的分析不能反映出各种优化对程序执行具体产生多大影响。定量的性能评测，可以提供给程序员更加详细的信息，对程序的行为有更加清楚的描述，为进一步优化其他情况下程序性能，如改变输入数据集、变换目标机器等，提供借鉴。因此，如何定量地对程序执行性能进行评价，是当前性能评测的一个重要研究方向。当前定量的性能评测主要有三类方法：基于分析的方法（Analytical-based modeling）、基于测试的方法（Measurement）和基于模拟的方法（Simulation）[102]。

基于分析的方法，通过对源程序静态分析，结合目标体系结构特性，使用计算机系统参数和程序特性建立数学模型，找出影响程序性能的主要因素，利用数学的方法分析程序的可能执行性能，常用来在算法设计阶段分析程序的时空复杂性。对于简单的情况非常有效，速度快，准确度可以接受。对于复杂的大规模应用，分析方法就不太适用，精确性难以保证。本文第二章介绍的计算模型相关的工作，就是通过理论模型分析算法的性能。这些模型粗略抽象计算机系统的参数，描述算法的行为，可以用来分析算法的复杂度。程序的实际执行行为非常复杂，通常同一复杂度的算法在实际机器上执行时候表现出极大的性能差异，因此算法设计模型对于实际程序的执行就显得很不精确，在程序执行行为分析、程序性能优化的时候就显得力不从心。

基于测量的方法，在程序执行时收集性能数据，然后统计分析，使用一些评价标准评估性能。通过测量一些关键的计算内核在各种机器上的性能，或使用性能剖析器收集性能数据，可以了解整个程序性能。基准测试一般是由一些经常使用的计算核心模块组成，常用的并行基准测试有Linpack[103]和NPB（NAS Parallel Benchmarks）[66]。性能剖析器主要有Intel公司的Vtune[97]，美国田纳西大学开发的PAPI （Performance Application Programming Interface）[104]。通过测量应用在实际机器上的性能，从而得到各种性能指标，这是最直接和基本的方法，也是目前使用最多的方法，评测数据非常精确，缺点是只能评价已运行的程序。

基于模拟的方法，使用模拟器（simulator）仿真实际硬件，让程序在模拟器上执行，记录和统计各种事件，收集性能数据，然后通过分析可以确定性能问题。模拟方法比较灵活，允许变换程序和机器参数，数据准确性高，但是模拟执行速度比较慢，在实际硬件上执行1秒的程序，放到模拟器上运行可能需要1000秒或者更多的时间，让人难以接受。实际中，常常模拟分析小规模的计

算问题，近似大规模的问题；或者，模拟分析程序有代表性的代码段，近似整个程序[105]。常用的模拟器有Simplescalar[106]。

各种性能评测方法都有其优缺点，分析模型对应用程序性能进行理论分析，分析速度非常快，但是不能够获取性能行为的细节，这一类模型需要用户手工建立，限制了可用性；测量可以评测已有程序的行为，验证和参数化性能模型，对一些度量标准比较有效，但是灵活性不好，可测标准比较少，局限于现有的程序和系统配置；模拟方法可以获得性能执行的细节行为，可以自动对一个程序建模，但是对于大规模并行应用来说代价庞大，需要大量的模拟时间和内存消耗。研究者们开始研究综合两种技术，进行性能评测。

重用距离分析[107]，把分析方法和模拟方法结合到一起，分析程序访存局部性。对于给定的数据访问流，重用距离为两次同一数据访问之间，所访问的不同数据的数量。重用距离是程序的固有属性，独立于硬件参数，与缓存大小、替换策略和存储系统结构无关。重用距离分析可以精确地预测基于LRU替换策略的全相连缓存的行为，评测程序访存性能。重用距离理论上很漂亮地描述了程序的访存行为，它是以数据访问流为基础进行分析的，数据访问流需要通过模拟器来收集。

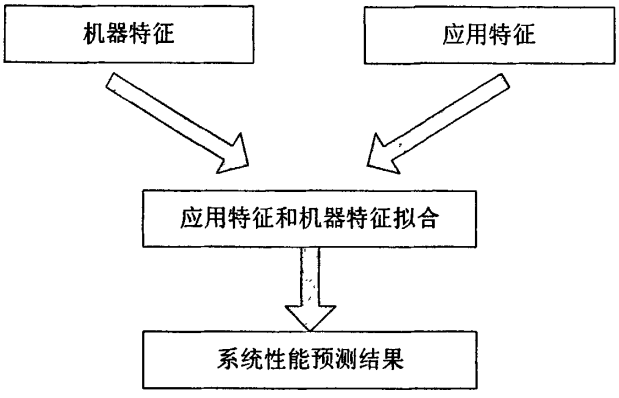


图 6.1 PMaC性能预测框架

PMaC把测量方法和模拟方法结合到一起。美国圣地亚哥数据中心（SDSC）提出一个自动化的性能预测技术PMaC（Performance Modeling and Characterization）[108-110]，根据应用软件特征、输入数据和目标硬件机器特征，来预测执行时间。如图6.1所示，性能评测分为三个部分：刻画机器特征，刻画分析应用特征，把两种特征拟合得到性能预测结果，并开发了一些列的工具来辅助预测。PMaC模型中，使用MultiMAPS 和MPI乒乓测试获取机器特征；使用PEBIL[111]、MPIDTrace分析存储和通信行为，获取应用特征；然后使用模拟器PMaC Convolver和PSiNS Simulator[112]，把应用特征和硬件特征进行拟合，获得系统的性

能。PMaC是一种测量和模拟综合的方法，测量硬件和软件关键的特征，然后通过轻量级的模拟器来评估程序运行，速度比单纯的模拟方法有很大的优势。软件特征与机器特征分别独立，可以收集A机器上的程序的特征，对B机器上的程序性能进行预测分析。

本文采用分析和测量相结合的技术，建立片上多核处理器（CMP）的定量程序执行模型CRAM(h)，使用性能剖析器测试模型参数，从而分析程序在多核系统上的性能，可以指导程序性能优化，评测程序执行性能。

6.2　多核系统关键因素分析

6.2.1　处理器指令执行

程序性能的持续提高，得益于处理器性能的提升，随着处理器的更新换代，软件无需改动自然获得性能的提升。处理器性能提升的手段主要是，制造工艺的进步，使的芯片内集成度的增加；体系结构的进步，不断提高时钟频率，使用超标量技术，优化指令执行；使用存储层次技术优化存储访问。其中，时钟频率的提高带来的性能提升占80%左右。调查表明，1990-1999这十年中，处理器主频每年提高60%，2000-2004频率提高逐渐力不从心，每年40%的速度提高，到了2005年只有20%左右。2004年，Intel被迫放弃了更高主频4GHz处理器的研发，预示着摩尔定律的终结，也预示着微处理器领域一个新时代的到来。理想情况下4核4发射的超标量多核处理器一个时钟周期可以发射16条指令，但是由于指令相关、资源冲突、分支预测失败和数据等待等因素的影响，总是会使流水线停顿和指令发射槽空闲，实际单个时钟周期指令发射数低于理想情况下处理器可发射数。

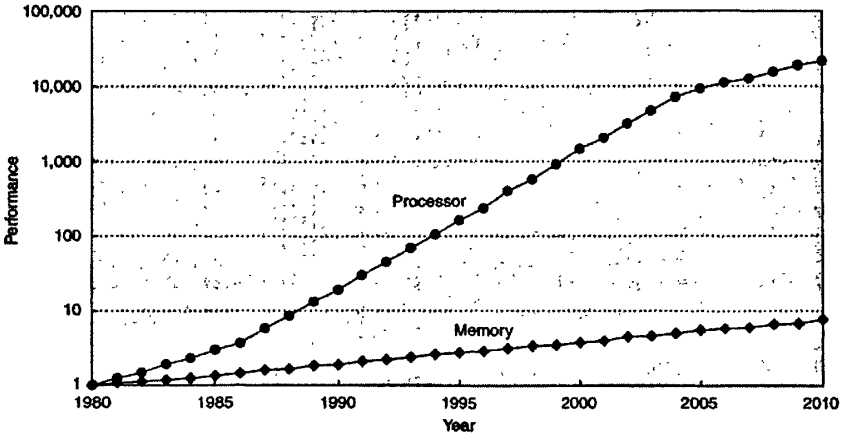


图 6.2 CPU和内存速度增长差异

6.2.2 存储器访问

一直以来处理器速度提高和存储器速度提高严重失衡，处理器和存储设备速度差异越来越大[113]。图6.2展示了一直以来处理速度和存储速度的变化趋势，比较了它们之间的差异。从图中可以看出，处理器性能和存储器性能逐年拉大，计算机系统逐渐由早期的处理器受限，过渡到存储器受限。计算机发展的早期，处理器速度比较慢，程序执行的性能受限于处理器。目前，系统设计人员从硬件和软件两个方面来改进存储系统的性能，主要有多级存储系统、指令调度（Instruction Scheduling）[114]、支持数据预取（Software Prefetching）[115]和提高数据局部性等技术。

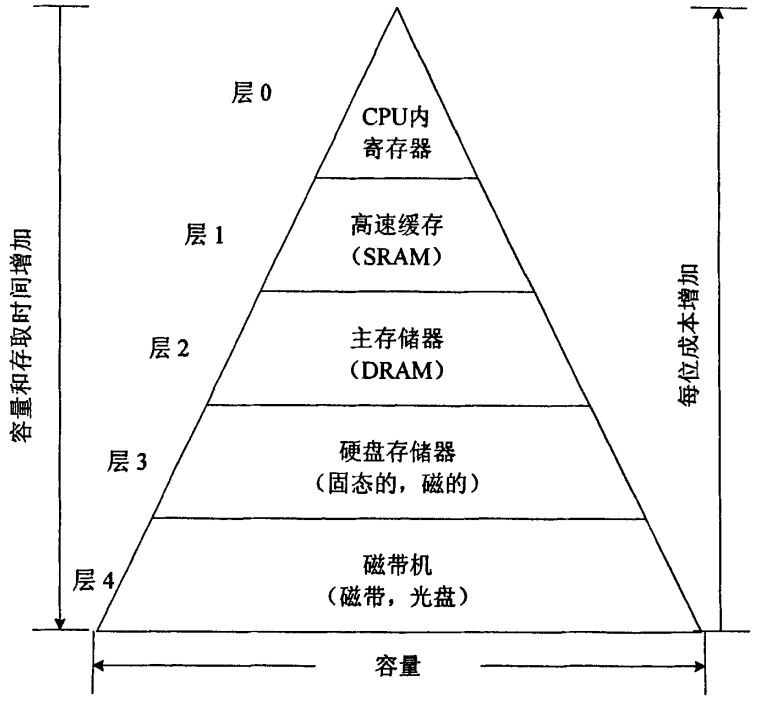


图 6.3 多级存储层次结构

计算机使用者总是希望无限大、快速的存储系统。存储层次结构是一个经济有效的解决方案，利用了局部性原理和不同存储器性价比的差异。图6.3展示了一个典型的多级存储层次结构。不同存储设备的速度、容量、价格差异很大，容量小的存储设备，访问速度可以很快，每字节的成本也高；容量大的存储设备，访问速度慢，价格便宜。

存储系统从高到低，存储设备变得更大、更慢和更便宜。各级存储层次典型性能参数如图6.4所示。最高层是少量的寄存器，CPU可以在一个时钟周期内访问它们，接下来是一个或几个基于SRAM的高速缓存存储器，CPU可以在几

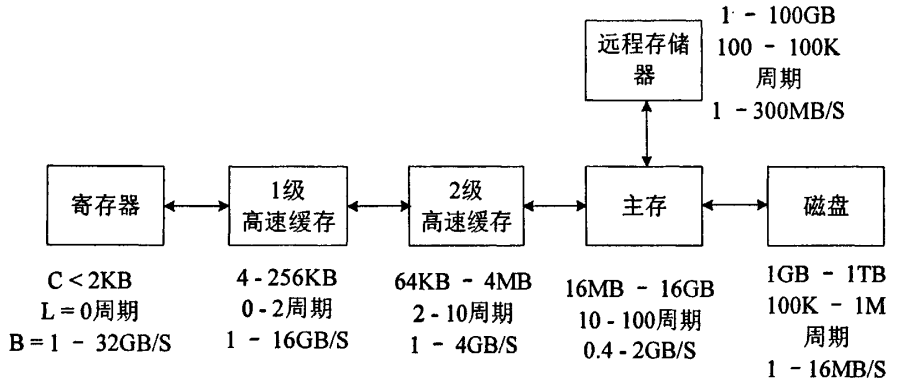


图 6.4 各层存储层次典型参数

个时钟周期内访问它们，再下面是一个大的基于DRAM的主存，CPU可以在几十或者几百个时钟周期内访问它们；接下来是大容量，速度慢的本地磁盘；最后有些系统还有速度更慢容量更大的磁带或者与远程计算机上的磁盘相连。多级存储层次提供了一个单字节价格与磁盘同样便宜，访问速度与最快的寄存器相当的存储系统。

数据一开始保存在离处理器最远的存储层次中，计算开始后，数据沿着存储层次往处理器传送，在各层间按块传送，不同层次间块大小可能不等，存储系统控制数据在各级存储层次中保存的位置，以及某一层次数据满时，采用某种替换策略把一个数据块替换出去，数据一旦取到高层存储器中，再次访问时，无需最底层存储器开始传送，可以直接从高层存储器开始往处理器传输数据，一旦数据取到处理器，处理器立即开始执行计算操作，在此之前处理器可以执行其他无需数据等待的指令或处于停顿状态。

编译器设计人员提出了众多软件优化技术来更好的利用存储层次，主要分为两类：循环变换和数据变换，包括以下技术：各种循环相关的优化，如：循环展开、循环分割、循环合并、循环不变量外提等；减小内存消耗以使得数据可以大部分放到速度更快的存储层次中；数据重用，把将会再次访问的数据尽可能更长时间保存在高层存储层次中，按照寄存器、高速缓存、内存、磁盘的优先级放置经常使用的数据。循环展开、调整数据访问模式等技术已经成功地用在数值计算库和编译器优化中，如LAPACK，MKL，ICC等。

6.2.3　并行性

片上多核处理器（CMP）在一个芯片内集成多个处理器内核，提供了丰富的计算资源，多个线程可以在不同核上并行执行，提升系统总体计算能力，但是，另一方面多核处理器也引入了一些并行相关的效率问题。通过前面三章的

并行程序性能优化技术的研究，可以得出并行程序性能一般受限于较少的并行度、共享资源竞争、同步开销、负载不均衡和通信开销。

根据Amdahl定律，应用中的不可并行部分是影响系统性能的关键因素。当不可并行部分占很大比例时，并行带来的性能提升非常有限。例如，如果不可并行部分占10%，那么当处理器采用10个核来计算时，可以取得5.26倍的加速，当处理器核数增加到100个时，加速比仅增加到9.17左右。处理器核数增大了10倍，加速比增加2倍不到。当处理器核数趋于无穷时，也仅能取得10倍的加速。

多个处理器或者计算内核共享存储器，当它们同时对共享存储器进行存取访问时，多个线程竞争前端总线和共享的Cache，带来访存延迟的增加，性能会有明显的降低[116,117]。未来处理器核心数量更多，对总线带宽和Cache的竞争会更加激烈，导致存储访问延迟严重影响程序性能。

为了确保多线程，内存访问顺序的正确性，必须使用同步操作。同步操作会带来两个性能问题：一是同步开销不可忽视，过多的同步操作会带来严重的额外开销；另一个是，多线程同步会造成某些线程停下来等待另一个线程的执行，等待过程中闲置的处理器浪费了计算资源。

一个并行程序通常由多个进程或线程组成，并行程序的执行时间取决于执行时间最长的进程或线程。在并行系统中，如果一个处理器分配的负载比其他处理器多，那么它将消耗更多的执行时间，在计算的最后阶段只有这些少数处理器在进行运算，其他处理器处于闲置状态，这样将会严重的限制并行系统的整体性能。

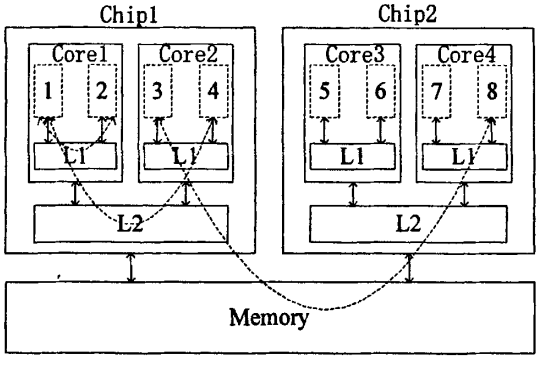


图 6.5 不同层次通信方式示意图

多线程并行执行时，需要通过数据共享完成通信。多核系统中不同核间的数据共享是非均匀的，即两个线程间数据共享的开销，根据数据存储的物理位置不同而不同。图6.5展示了经过不同层次存储系统的通信情况。同一个核上的

线程，如采用超线程技术的内核上的两个线程，通过共享的一级Cache进行通信，通信延迟为1到2个时钟周期；对于不在同一个核上，但在同一个芯片上的不同核上的两个线程，通过共享的二级Cache进行通信，通信延迟为10到20个时钟周期；不在同一芯片上的两个线程之间，通过共享的内存进行通信或通过Cache一致性协议，通信延迟一般为上百个时钟周期。

6.3　定量化的执行模型CRAM(h)

从上面的分析可以看出，程序性能同时和计算机硬件和应用程序相关，可以从两方面考虑，单机性能和并行性能。单机性能主要取决于指令执行速度、程序中指令数以及指令等待数据的时间。指令执行速度跟硬件参数有关，包括处理器、存储系统、总线等；程序中总的指令数与应用程序本身相关，包括算法和各种优化技术的影响；指令等待时间主要来自指令相关和数据延迟，与硬件和程序同时相关，包括乱序执行、寄存器重命名、分支预测、存储系统硬件参数、程序访存模式、指令调度、数据软硬件预取等。

性能模型实际是计算机硬件特性和应用程序特性的函数。计算机的一组硬件参数，用$M = m_1, m_2, \cdots, m_n$表示，应用程序特性用$A = a_1, a_2, \cdots, a_n$来表示，$m_i$和$a_i$分别体现了并行机第$i$个模块和相对的应用特征。并行系统的性能可以表示为$P = F(M, A)$，考虑不同硬件模块和应用特征的差异，可以把映射函数进一步分解为$F = f_1, f_2, \cdots, f_n$。$P$可以表示如下：$P = f(f_1(m_1, a_1), \cdots, f_n(m_n, a_n))$。性能模型使用方法如下，选择关键硬件性能参数和程序应用特征，构造合理的映射函数，建立性能执行模型，测量模型中的参数，计算得到性能评测结果，验证模型并加以修正。

本小节从指令执行、访存行为和并行性三个方面，研究片上多核系统（CMP）上程序的执行性能，建立定量性能模型CRAM(h)，以RAM模型为基础，引入h层存储访问层次，利用性能剖析器来收集性能模型所需数据，进而利用该性能模型定量地对程序执行行为进行分析。

6.3.1　执行时间模型

程序的性能可以用程序的执行时间来度量，执行时间是由应用程序特性和硬件参数两方面决定的。程序的特性主要包括程序中总的指令数、计算操作数量和访存行为。硬件参数包括指令执行的速度、存储层次的层数及延迟。对于一个应用程序，在某台具体的计算机系统上的执行时间可以用下面的公式来计

算：

$$\begin{aligned} T &= m/s + M(1/s + T_{mem}) \\ &= (m+M)/s + M \cdot T_{mem} \\ &= T_c + T_m \end{aligned} \tag{6.1}$$

m是访存无关的指令数，M是有访存行为的指令数，总的指令数为$m+M$，s为处理器的速度，即处理器主频，与时钟周期呈倒数关系，T_{mem}为平均访存时间，计算时间$T_c=(m+M)/s$，总的访存时间$T_m=M\cdot T_{mem}$。公式6.1中，m和M可以通过性能剖析器获得，T_{mem}在下一小节给出计算方法。

在实际程序执行中，不同类型的指令执行速度也是有差别的，比如乘法和除法总是比加法和减法更耗时，乘方和平方根运算需要更多的时间。因此，为了更精确描述不同类型指令的执行性能，在公式6.1中加入对不同类型指令的支持，如公式6.2所示：

$$T = \sum m_i/s_i + M \cdot T_{mem} \tag{6.2}$$

公式6.2中，m_i和s_i分别代表了不同类型指令的数量和执行速度。公式6.2鼓励用户在设计算法时更多地采用简单的指令和操作，比如乘方运算可以使用一系列乘法操作来替代。

6.3.2　平均访存时间模型

一条有访存行为的指令的访存延迟，与该指令所需数据在哪一级存储层次命中以及该层的访存延迟相关。程序整体的访存开销，为所有的访存指令总的访存时间，为程序在每级存储器延迟的总和。CRAM(h)模型通过计算平均访存时间，来描述程序在具有h级存储层次的CMP系统上的访存开销，总的访存开销等于访存相关指令数量与平均访存时间的乘积。平均访存时间可以用公式6.3来表示：

$$T_{mem} = \sum_{i=1}^{h} P_i t_i = \sum_{i=1}^{h} (n_i/N) t_i = \sum_{i=1}^{h} n_i t_i / N \tag{6.3}$$

t_i为单个数据在第i层存储层次上的访存延迟，n_i 代表第i层总的访问次数，等于第i层load和store指令数之和，N是程序中所有访存相关指令的个数，P_i为数据在第i层存储层次命中的概率，$P_i=n_i/N$。一般寄存器的访存时间计入指令执行时间。一个具有L1缓存、L2缓存和主存的计算机系统来说，平均访存时间为：

$$T_{mem} = \frac{n_{l1}t_{l1} + n_{l2}t_{l2} + n_m t_m}{N} \tag{6.4}$$

6.3.3　并行性相关参数

本小节从计算资源、存储竞争和负载分布的角度，研究执行模型中并行相关部分。

多核系统中，假设有n个计算内核，它们共享最后一级Cache和内存，应用程序中可并行部分比例为$f(f <= 1)$。

负载均衡是并行计算中的一个重大问题，是系统中的任务数以及每个任务的大小的函数。假设n个并行执行的任务，各自的执行时间为$t_1, t_2, \cdots, t_n$，它们一起并行执行时候的时间t_p可以表示为：

$$t_p = Max(t_1, t_2, \cdots, t_n) \tag{6.5}$$

引入一个降速因子$\alpha(\alpha >= 1)$来描述负载不均衡，α为n个线程并行执行任务的实际时间与串行执行该任务的时间的$1/n$的比值。

多核处理器中，通信通过读写共享Cache或内存来实现，因此通信开销可以计入各线程访存时间。多个线程对同一共享内存交叉访问，竞争总线带宽和Cache，线程越多竞争就越激烈，会严重影响程序的性能[118]。引入并行访存竞争指数$\beta(\beta >= 1)$，描述竞争带来延迟的增加。β为数据访存在当前资源竞争情况下的访存时间与无竞争时需要的访存时间比值。

综上，对于一个串行执行时间为T的应用，在具有n个计算内核的CMP上的执行时间T_n可以描述如下：

$$T_n = (1 - f)T + \alpha\beta fT/n \tag{6.6}$$

在这个公式下，执行模型鼓励并行度高、数据竞争小、负载均衡的程序实现。

6.3.4　性能剖析技术

现代处理器都内建了一些硬件计数器，用来检测和统计各种事件的发生，获得性能相关的数据，如时钟周期数、执行指令的类型和数目、Cache缺失和分支事件等，并记录下来，由于记录的是程序执行时的即时信息，准确性比较高，而且速度快，系统开销比较小。人们开发了一些性能剖析器，用来帮助读取硬件计数器的值，并且给出更高层次的性能数据，帮助用户理解程序行为。性能剖析技术在性能分析领域被广泛应用，利用性能剖析器可以收集程序运行时的性能数据。计算机体系结构设计者使用性能剖析器来评估新的体系结构上程序的性能，指导体系结构的设计；软件开发人员使用剖析器来确定程序中最耗时部分，分析软件的性能瓶颈，指导程序性能优化；编译器设计人员使用剖析器来验证指令调度算法、分支预测算法等的性能，指导新的编译优化技术

开发。性能剖析器通过采集硬件计数器的值和插入监测代码等方法，来收集程序执行时的性能数据，这些数据反映了程序执行时的行为，可以用来帮助用户定位程序中执行最慢的部分，给出性能不好的原因。VTune是Intel开发的一个针对Intel处理器的性能剖析器。本文利用VTune收集CRAM(h)模型中所需的性能数据和参数，如：程序执行的时钟周期数、指令信息、缓存失效和命中信息等。

6.4　模型验证

本小节通过实验来验证CRAM(h)模型的准确性和可用性。首先，介绍实验的硬件环境，接着介绍使用的程序，最后描述存储层次参数采集的方法，以及利用性能模型分析具体程序执行行为，对模型进行验证。CRAM(h)模型中考虑了并行对程序性能的影响，但是由于并行相关参数还没有一个完善的测量和分析方法，本小节仅验证CRAM(h)中的串行部分，并行部分的验证留待进一步工作中解决。

6.4.1　实验环境

实验硬件平台为Intel Conroe处理器，该处理器为双核处理器，频率为2.4GHz，每个核有一个32KB的一级Cache，两个核共享4MB的二级Cache，一级和二级Cache的缓存行大小均为64个字节，系统总的内存为2GB，前端总线频率1066MHz，总线带宽容量为8.5GB/s。实验的软件环境为Windows Server 2003，Intel C/C++ Compiler 9.1，打开了/O2编译选项，使用Intel Vtune来测量相关性能模型中的参数和性能数据。

6.4.2　存储系统参数

本小节通过实验来测量各级存储系统的延迟，测量用的方法来自Calibrator。假设load和store操作有相同的延迟，通过赋值操作来测量load和store操作的平均开销。如图6.6所示，在循环中执行一系列赋值操作，然后测量不同存储层次的执行时间，计算所有存储操作执行的平均时间，来获得每个load或store操作的执行时间。根据Cache容量和Cache行大小，选择合适的数组大小和访问步长，可以控制数据访问在某一层Cache命中或者不命中，假设某级Cache的容量为C_i，行长为B_i。

(1) 如果数组可以在某一级Cache中全部放下，那么对该级Cache的访问，无论步长多少，均不会出现数据不命中的情况，每次访问的执行时间即为该级Cache的延迟。

(2) 否则如果数组大小超过了某一级Cache的容量，那么对该级Cache的访问将出现数据不命中的情况，不命中的频率与访存步长和Cache行的大小相关。相邻两层Cache之间数据传输的基本块为一个Cache行，如果访存步长大于等于一个Cache行的大小，那么每次数据访问都不命中；如果访存步长小于一个Cache行的大小，那么第一次访问数据Cache行会产生一次Cache不命中，其后对该Cache行中其它数据的访问都会在Cache中命中，因此每$line_size/stride$次数据访问将会有一次不命中发生。某一级Cache不命中惩罚为数据在该级Cache不命中的访存开销与在该级Cache命中的访存开销之差。

```
double a[N], b[N];
for(j = 0; j < n; j++)
{
    for(i = 0; i < N; i+ = stride)
        a[i] = b[i];
}
```

图 6.6 测量存储层次参数使用的代码.

利用上面的相关知识，通过控制访存步长和数据大小来测量目标硬件系统的存储层次参数。目标机器存储层次分为五层：寄存器、一级Cache、二级Cache、主存和磁盘，寄存器延迟可以计入计算时间，本节假设数据一开始都存放在内存中，暂不考虑磁盘的访问开销。使用如图6.6所示的测量代码，N是数组的大小，n是赋值重复的次数（为了多次测量求平均值），访问步长s表示相邻两个赋值操作数的间距，通过调整N、n和s来测量不同存储层次的访存延迟。表6.1总结了各种情况下的存储访问失效情况和访问延迟。其中，B_1为一级缓存行的大小，C_1 为一级Cache的容量，B_2为二级Cache行的长度，C_2为二级Cache的容量，C_m为内存容量，T_{l1}为一级Cache访问的延迟，T_{l2}为二级Cache的访问延迟，T_m为内存的访问延迟。

6.4.2.1 L1 Cache延迟测量

当a和b的总大小小于一级Cache的容量时，数据第一次装入Cache后，数据总是会在Cache中命中，以后的访问都不会有缺失，每次访问的时间就是一级Cache的读写延迟。实验用的处理器的一级Cache大小为$32KB$，行长为$64B$，因此选取N为1024，n为1024×1024，步长设为1，由此数组a和b的元素访问都可以在一级Cache中命中，此时数据访问开销即为T_{l1}。

表 6.1 存储层次访问情况

数组长度	步长	说明	访问延迟
$N <= C_1$	$1 <= s <= N/2$	经过强制失效后 全部在L1中命中	T_{l1}
$C_1 < N <= C_2$	$B_1 <= s <= N/2$	在L1中全部失效 L2中全部命中	$T_{l1} + T_{l2}$
$C_2 < N <= C_m$	$B_2 <= s <= N/2$	在L1和L2中全部失效 在主存中命中	$T_{l1} + T_{l2} + T_m$

6.4.2.2 L2 Cache延迟测量

当数组a和b的大小都大于一级Cache的容量，但是比二级Cache容量小时，一级Cache不能全部命中数据。一开始，数据被从内存取到二级Cache，然后以后的数据访问不需要到内存中取数，全部会在二级Cache 中命中。由于数组大小比一级Cache 大，因此数据在一级Cache中会发生冲突失效，发生失效后，需要到二级Cache中取数据。二级Cache的延迟为从二级Cache中传输一个Cache行所需的时间。目标机器一级Cache 大小为$32KB$，二级Cache大小为$4MB$，行长为$64B$，因此，N设为64×1024，n设为16×1024，步长为$8 \times 8 = 64B$，所以对a和b的访问，会在一级Cache中全部失效，在二级Cache中命中，因此数据访问延迟为$T_{l1} + T_{l2}$。

6.4.2.3 内存延迟测量

当数组a和b的总大小大于二级Cache的容量时，L1和L2不能容纳全部的数组元素，有些数据访问需要访问主存。实验中我们发现内存对步长短和长有不同的延迟，这是因为现在的测量方法没有考虑内存中存储体的影响。对内存的顺序访问，会分配到不同的体，以此获得高性能，随着步长增大，内存访问会映射到同一存储体，引起体冲突，增大内存访问延迟。与测量二级Cache延迟方法类似，可以选取$N = 16 \times 1024 \times 1024$，$n = 1024$，步长为$64B$和$1024B$，分别测试短步长和长步长时候的二级Cache失效延迟T_m。这种情况下数据访问开销为$T_{l1} + T_{l2} + T_m$。

测量结果如下表6.2所示：

表 6.2 存储层次延迟

	Time (ns)
L1 latency (t_{l1})	0.42
L2 latency (t_{l2})	1.86
Memory latency with small stride(t_m)	20.25
Memory latency with large stride(t_m)	32.20

6.4.3　矩阵相乘的局部性分析

矩阵相乘是数值计算领域经常使用的计算核心。本节利用矩阵相乘作为应用实例来验证本章所提出的定量的性能执行模型CRAM(h)。对于矩阵相乘不同的实现形式，下文分别用MM1、MM2、MM3、MM4来表示。

本节考虑两个大小为$n \times n$的矩阵A和B相乘，其结果矩阵为$C = A * B$，即$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$。四种矩阵相乘实现具有相同的时间复杂度$O(n^3)$，但是它们表现出不同的内存访问模式，由此导致了总的执行时间的差异。因此可以看出同一复杂度的算法，由于访存方式的不同，它们的执行时间差异很大，存储系统性能和程序的访存模式对程序的执行时间有极大的影响。在下面的实现中，矩阵均为行优先存储的，即同一行中的数据元素在内存中是顺序存储的，每一行第一个数据紧跟着它前一行最后一个数据开始存储。

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        for(k = 0; k < n; k++)
            C[i][j]+ = A[i][k] * B[k][j];
```

图 6.7 MM1实现的伪代码

MM1实现的伪代码如图6.7所示。在这个实现中，矩阵A中第i行元素为连续访问，且在第二层循环中被重用了n次，根据时间局部性和空间局部性原理，可以假设矩阵A大部分可以在一级Cache中命中。矩阵B的访问空间局部性比较差，每个$B[k][j]$的访问距上次访问都比较远（n个元素的距离）。B中的数据元素仅仅在最外层循环中重用，所以对B中数据的访问大部分在内存中命中。矩阵C在最内层循环中重用，$C[i][j]$基本上可以在寄存器中命中。综上，MM1的实现中，共有n^3次对矩阵A的load操作，n^3次对矩阵B的load操作，n^2次对矩阵C的load操作和n^2次对矩阵C的store操作。

```
for(i = 0; i < n; i++)
    for(k = 0; k < n; k++)
        for(j = 0; j < n; j++)
            C[i][j]+ = A[i][k] * B[k][j];
```

图 6.8 MM2实现的伪代码

MM2实现的伪代码如图6.8所示。在这个实现中，矩阵A的元素$A[i][k]$在最

内层循环中被重用，A中的元素大部分都可以在寄存器中命中。$B[k][j]$被连续访问，空间局部性比较好。当$B[k][j]$从二级Cache传输到一级Cache或者从主存传送到二级Cache的时候，和$B[k][j]$同在一个Cache行的元素都会一起传输到一级或者二级Cache。这样，未来对该Cache 行的访问都会在离CPU近的一级或者二级Cache中命中。当矩阵B的大小比一级Cache大时，一级Cache不能同时容纳B的全部元素，矩阵B的部分访问会到二级Cache甚至主存中去取数。$C[i][j]$在第二层循环中重用n次，当n不是非常大时，矩阵C一行数据总的大小不超过一级缓存容量，第i行所有数据可能会全部存放在一级Cache中。综上，MM2的实现中，共有n^2次对矩阵A的load操作，n^3次对矩阵B的load操作，n^3次对矩阵C的load操作，n^3次对矩阵C的store操作。对以上的load和store操作，A基本可以在寄存器中命中，B主要在一级或者二级Cache中命中，C主要在一级Cache中命中。

```
for(j = 0; j < n; j++)
    for(k = 0; k < n; k++)
        for(i = 0; i < n; i++)
            C[i][j]+ = A[i][k] * B[k][j];
```

图 6.9 MM3实现的伪代码

MM3实现的伪代码如图6.9所示。在这个实现中，$A[i][k]$是按列顺序来访问，在内存中的访问步长为n，对矩阵A访问的空间局部性非常差。同时，$A[i][k]$在最外层循环重用，所以时间局部性也非常不好。对$B[k][j]$的访问为非连续访问，访问步长为n，但B中的元素在最内层循环中被重用，所以基本上可以在寄存器中命中。$C[i][j]$在第二层循环中被重用，同时对于$C[i][j]$的访问为非连续访问，所以对$C[i][j]$存取性能比较差。综上，共有n^3 次对矩阵A的load操作，n^2次对矩阵B的load操作，n^3次对矩阵C的load操作，n^3次对矩阵C的store 操作。对于以上的load和store操作，A大部分访问可以在主存中命中，B基本上可以在寄存器中命中，C可以在Cache或者主存中命中。

最后，MM4的实现伪代码如图6.10所示。在这个实现中，$A[i][k]$是连续内存访问，$B[k][j]$是间隔n的不连续内存访问，$C[i][j]$ 在最内层循环中重用，对C中元素的访问基本可以在寄存器中命中。综上，共有n^3次对矩阵A的load操作，n^3次对矩阵B的load操作，n^2次对矩阵C的load操作，n^2次对矩阵C的store操作。以上load和store操作中，A大部分在Cache和内存中命中，B大部分在内存中命中，C在寄存器中命中。

```
for(j = 0; j < n; j++)
    for(i = 0; i < n; i++)
        for(k = 0; k < n; k++)
            C[i][j]+ = A[i][k] * B[k][j];
```

图 6.10 MM4实现的伪代码

表6.3总结了以上四种矩阵相乘实现的计算行为和访存行为，列出了每种实现形式需要花费的load操作数、store操作数和计算操作数，同时也给出了矩阵A，B，C主要命中的存储层次。本文使用VTune来收集性能数据。在后面的

表 6.3 四种矩阵相乘实现各种操作数量分析结果

	MM1	MM2	MM3	MM4
Load	$2n^3+n^2$	$2n^3+n^2$	$2n^3+n^2$	$2n^3+n^2$
Store	n^2	n^3	n^3	n^2
Computation	$2n^3$	$2n^3$	$2n^3$	$2n^3$
Register	C	A	B	C
Cache	A	B,C	/	A
Memory	B	/	A,C	B

分析中，计算两个1024 × 1024大小的矩阵相乘。收集的数据列在表中。每一项

表 6.4 四种矩阵相乘实现各种操作数量测量结果

	MM1	MM2	MM3	MM4
CPU_CLK_UNHALTED.CORE	2.22E+10	5.85E+09	4.78E+10	1.75E+10
INST_RETIRED.ANY	8.75E+09	1.08E+10	1.10E+10	8.72E+09
INST_RETIRED.LOADS	2.15E+09	2.15E+09	2.15E+09	2.15E+09
INST_RETIRED.STORES	1.30E+06	1.07E+09	1.07E+09	1.27E+06
INST_RETIRED.DOUBLE	2.15E+09	2.15E+09	2.15E+09	2.15E+09
L1D_REPL	1.09E+09	1.35E+08	2.80E+09	1.22E+09
L2_LINES_IN.SELF.ANY	2.50E+08	1.18E+08	3.74E+08	1.74E+08

代表一个性能数据，具体含义描述如下。CPU_CLK_UNHALTED.CORE代表程序总共执行的时钟周期数，是程序执行时间的度量。INST_RETIRED.ANY代表程序总共执行的指令数，INST_RETIRED.LOADS代表程序总共执行的load指令的条数，INST_RETIRED.STORES代表程序总共执行的store指令的条数，INST_RETIRED.DOUBLE代表了程序中总共执行的浮点运算指令的条数。L2_LINES_IN.SELF.ANY统计二级Cache分配的缓存行数，代表有多少次主存访问，即模型中的n_m。L1D_REPL统计一级Cache分配的缓存行数，代表有多少次二级Cache访问，即模型中的n_{l2}。一级Cache访问次数n_{l1}，是总的load和store指

令数，因为有load和store指令就至少产生一级Cache的访问。通过对比表6.4和把$n = 1024$带入表6.3的结果，可以得出实际测量的数据与分析得到的数据是基本一致的，包括load和store指令数，计算指令数，存储访问行为，如图6.11所示，其中MM1和MM4由于写操作数比较少，写操作数量在图中没有显示出来。

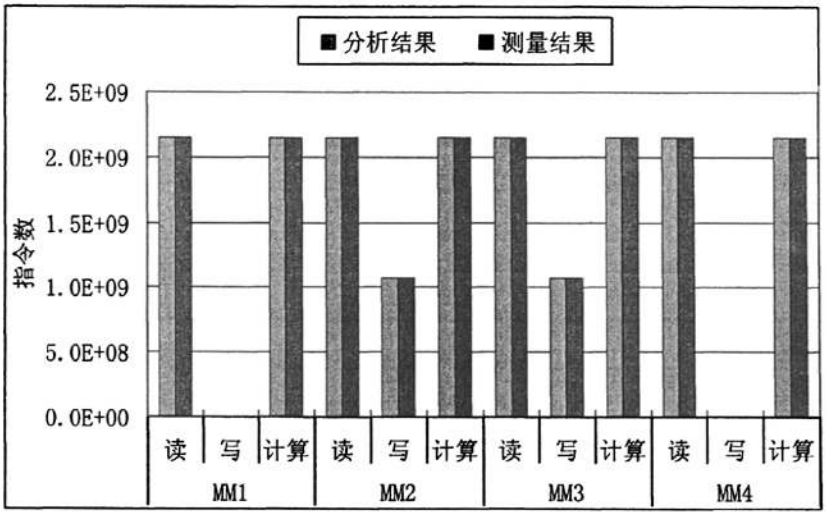


图 6.11 各种操作数测量和分析结果比较

6.4.4　性能模型的评价

本小节通过把实测性能参数，代入定量化性能执行模型CRAM(h)中，把模型分析的结果与实际的程序执行时间进行对比，进而来验证模型的准确性。实际执行时间是指程序执行过程中消耗的时间；时钟周期时间是计算测量的消耗的时钟周期数与处理器主频的比得到的执行时间；模型分析时间是把测量的模型参数带入模型开销函数公式后得到的执行时间。在CRAM(h)模型中，程序的执行时间有两部分组成：指令无延迟时候处理器的执行时间和处理器等待数据准备的时间。当前，本文不对不同类型指令做区分。目标机器是一个4发射超标量处理器，在一个时钟周期内可以同时发射执行4条指令，指令的执行时间可以用下面的方法来计算$number_of_instructions/CPU_frequency/4$。总的存储系统数据访问时间可以利用测量的存储层次参数以及性能剖析器收集的性能数据，根据模型计算得到。图6.12和表6.5比较了各种实现形式的实际执行时间、时钟周期时间和模型分析时间。

从图6.12和表6.5可以看出，除了MM2模型分析时间和实际执行时间差别比较大之外，其他三种实现形式模型和实际执行时间非常接近。在MM2中三个矩阵都有非常好的时空局部性，数据访问均为连续访问，这种情况下现代处理器

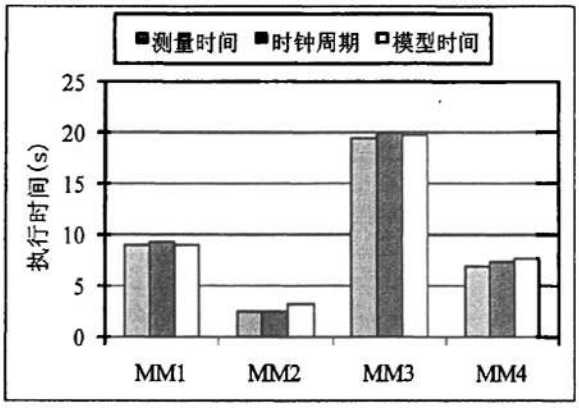


图 6.12 测量时间、时钟周期时间和模型分析时间对比

表 6.5 测量时间、时钟周期时间和模型分析时间对比

	MM1 (s)	MM2 (s)	MM3 (s)	MM4 (s)
Actual	8.946	2.464	19.375	6.876
Clocks	9.260	2.400	18.948	7.307
Model	8.907	3.242	19.774	7.611

和编译器都可以很好地对实现数据预取，进而实现计算和访存的重叠。而本章的研究中，并没有把计算和访存重叠情况纳入到模型中来，所以带来了评估上的偏差。

可以通过对本章模型修正，来加入计算和访存重叠。实际上计算时间T_c不可能省略，有重叠只是会隐藏访存延迟，减少存储访问时间T_m，因此修正后，$T = T_c + (1-\lambda)T_m$，λ为计算和访存之间的重叠因子，且$0 \leq \lambda \leq 1$。

6.5　本章小结

定量性能评测对于提高并行应用程序的效率，充分发挥并行计算机的计算能力，缓解峰值和实测性能差异具有非常重要的意义。性能评测可以确定性能瓶颈，揭示影响系统性能的主要因素，指导程序和体系开发，提高求解实际问题的能力。本章首先回顾性能评测相关的工作，总结常用的性能评测方法，提出建立性能执行模型来评测CMP系统性能，性能模型有其他方法没有的优点，对系统设计者和使用者都有指导意义；接着，详细分析了影响片上多核系统性能的关键因素，当前主流的并行机体系结构为基于多核处理器搭建的集群结构，本章研究片上多核系统的性能评测方法，从而为进一步研究大规模多核集群的性能评测提供借鉴，分析表明处理器性能、存储器性能以及并行相关的性能对于CMP系统的整体性能至关重要；然后，提出一个把分析和测量相结合的定量

的程序执行模型CRAM(h)，来评测CMP系统的性能，CRAM(h)模型是RAM模型针对CMP系统的扩展，改变单一存储访问开销的假定，加入h级存储层次，各级存储层次存储访问参数不同，数据根据局部性原理会在各级存储层次复用，综合对共享存储系统性能优化的研究，模型相应加入反映程序并行度、资源竞争和负载不均衡的参数，模型中相应参数通过性能剖析器测量获得；最后，使用数值计算中常用的计算内核矩阵相乘来对模型进行验证分析，实验测量了矩阵相乘算法的四种不同实现形式，通过比较实际测量以及模型计算得到执行时间比较，得出CRAM(h)模型分析得到结果与实际执行情况基本一致，分析时间与实际测量的执行时间非常接近。

但是程序的执行行为是非常复杂的，还有一些因素CRAM(h)模型中没有考虑到，在将来的工作中，会把其他一些因素加入到模型中，在保持模型简单实用的前提下进一步提高模型的精确度，主要有以下待考虑的影响性能的因素：

当前本文的模型中没有考虑TLB对程序性能的影响，TLB缓存了页表数据，支持快速的虚实地址转换，TLB不命中会导致访问主存中的页表，带来性能的开销。

现在的计算机结构中，处理器指令执行和存储访问在一定程度上可以重叠，从而隐藏数据访问时间，当前模型中没有考虑这一点，导致实验中MM2模型计算得到的结果与实测执行时间相差较大，因此，模型中应该加入对计算和访存重叠的分析。

第7章　总结与展望

本章总结全文的研究工作、成果和创新点，并对进一步工作进行了展望。

7.1　本文总结

人们对计算速度永无止境的追求，推动着并行计算技术高速发展。经过三十多年的发展，并行计算在理论、硬件、编程、应用等方面取得了长足的进步。并行计算的理论基础并行计算模型和并行算法取得了许多优秀的研究成果；并行计算机的运算能力已经突破了每秒千万亿次；并行编程模型逐渐统一标准，确定了适用于分布存储消息传递标准MPI和适用于共享存储的标准OpenMP；并行计算已经广泛应用在科学与工程的各个领域，在推动社会进步上发挥着重要的作用。但是，在看到以上成果的同时，我们也要认识到当前并行计算新的发展与新的挑战。虽然具有千万亿次峰值计算能力的超级计算机已经研制成功，但是并行应用水平没有跟上，科学和工程应用的实际性能一直处于较低的水平。因此，并行应用开发和优化还需要深入探索和研究，以充分发挥并行计算机的硬件优势，有效利用并行计算机的计算资源。基于共享存储系统搭建大规模并行计算机，是并行体系结构发展的主流趋势。本文研究基于共享存储的计算模型和性能优化技术，主要研究内容和成果总结如下：

(1) 分层的并行计算模型

随着并行体系结构飞速发展，单一并行计算模型不断被加入新的参数以强化其功能，致使单一模型越来越复杂。本文提出分层的并行计算模型，根据并行计算的三个阶段，把并行计算模型分为并行算法设计模型、并行程序设计模型和并行程序执行模型三个层次。分层计算模型将单一模型中的功能按要求分配到模型不同层次中，每个层次重点关注一个计算阶段，缓解了单一计算模型的精确性和可用性之间的矛盾。并行算法设计模型，面向并行算法研究者，指导并行算法的设计和时间、空间和通讯复杂度分析；并行程序设计模型，面向并行程序设计者，从并行编程角度指导程序员正确地编程实现并行算法；并行程序执行模型，面向并行程序运行者，指导并行程序在具体的并行机上编译、优化和运行。

(2) SMP系统上MPI基本通信库的性能优化

SMP系统是一种重要的共享存储并行机，其上的MPI通信实现的性能不高，限制了其上并行应用整体性能的可扩展性。通过分析MPI在SMP系统上通信

驱动程序的实现方法，得出通讯性能主要受限于两次消息复制开销和使用的同步策略延迟了消息传递的启动。本文提出一种SMP系统上通信优化方法，利用进程间通讯技术分配共享内存来存放待传递消息，消除不必要的消息复制，使用自旋等待同步策略完成进程间同步，在较小影响程序的可移植性和兼容性前提下，极大地提高了SMP系统上的MPI通信性能。实验结果显示，对于点对点消息传递，优化的MPI通信库性能提升了15倍；对于集合通信，优化后MPI通信性能提升了300倍左右；对于NPB中的IS基准测试，当使用16个处理器时，使用优化MPI通信库的并行IS性能提升了1.71倍。

(3) **SMP系统上RNA二级结构预测软件Mfold的性能优化**

RNA序列二级结构预测有助于揭示RNA序列的功能，是计算生物学中的一个重要研究内容。Mfold是一个广泛使用的预测RNA二级结构的软件，其中串行算法的时间复杂度为$O(n^3)$，对于大规模RNA序列二级结构的预测，其求解速度让人无法忍受。本文首先提出了并行Mfold算法，利用对角线法挖掘Mfold中动态规划算法的进程级并行性，并使用MPI在SMP 系统上进行了实现；接着根据共享内存系统的特点，消除了各进程间数据传递过程，通过共享内存来实现数据同步。实验结果显示，在装配16个处理器的SMP系统上，优化前并行Mfold获得了6.52倍的加速比，经过优化后的并行Mflod减小了总的内存消耗，获得了12.31 倍的加速比，比优化前性能提高了95%。使用性能剖析器测量执行时间的主要组成成分，可以看出MPI相关操作的执行时间占总执行时间的比例，由优化前的25.64%下降到优化后的9.21%，证实了相关优化的有效性。

(4) **基于内容的图像检索系统在多核处理器上的性能优化**

基于内容的图像检索系统（CBIR）是解决海量图像检索的有效方法，但是由于系统本身计算需求较大，实际系统的性能不高。多核技术的繁荣发展，可以挖掘系统的各级并行性加速其执行速度。本文搭建了一个基于内容的图像检索系统，在此基础上研究系统在两台不同配置多核计算机上的性能优化方法，给出了一套比较完整的多核系统上软件性能优化步骤及典型的优化技术和工具。分别优化了CBIR系统的指令级并行性、数据级并行性和线程级并行性。实验数据表明，相关优化技术使CBIR系统在8核系统上获得了26倍左右的加速比，在16核系统上获得了33倍左右的加速比从而显著提高了查询的响应速度。通过使用性能剖析器，对系统执行过程进行分析，测量各级Cache失效率和前端总线带宽，得出了多核系统上CBIR的性能主要受限于Cache容量以及前端总线带宽，为以后多核系统设计、程序性能优化方向提供了借鉴。

(5) 面向CMP系统的定量程序执行模型

结合共享存储系统的性能优化技术，本文提出了面向片上多核系统的定量的程序执行模型CRAM(h)。CRAM(h)模型使用分析和测量相结合的方法，扩展RAM模型，引入存储层次和并行性，对片上多核系统上程序执行进行性能评测，对程序执行行为进行建模，分析指令执行、存储访问、并行执行等行为，使用程序中指令数、访存指令数、存储层次延迟、存储访问分布、可并行部分比例、存储竞争降速因子和负载不均衡降速因子等几个参数，评估并行程序的执行时间。CRAM(h)模型使用性能剖析器测量和计算模型中的参数，定量地分析优化技术对于指令执行时间和存储访问时间的影响，提供给程序员更加清晰的程序执行行为，确定程序性能瓶颈，指导更深入的性能优化，评估各种优化技术的有效性。使用数值计算中常用的计算内核矩阵相乘来对模型进行验证分析，实验测量了矩阵相乘算法的四种不同实现形式，通过比较实际测量以及模型计算得到执行时间，CRAM(h)模型分析得到的执行时间与实测时间基本吻合，验证了模型的精确性和准确性。

7.2 进一步工作

并行计算仍在不断发展中，并行计算中仍存在很多问题，研究和应用上都有巨大的发展空间，本文仅对其中一小部分内容进行了初步的研究，在本文工作的基础上，还可以进行如下更广泛深入的研究工作:

(1) 研究并行程序执行模型的细化和改进

分析程序的执行行为并进行精确地建模，可以优化体系结构设计和应用程序的执行性能。程序的执行行为相当复杂，涉及的面很广，需要从体系、编译、操作系统等角度综合考虑。下一步，可以扩展和改进本文提出的面向CMP系统的CRAM(h)程序执行模型，研究如何加入TLB性能模型、内存页面管理模型、磁盘访问的性能模型、异构多核的线程模型和节点间通信的性能模型，研究并行执行模型的验证方法，研究全自动化的模型参数提取工具，探索千万亿次大规模并行计算机的执行模型。

(2) 研究适合多核平台的高性能并行编程模型

并行编程模型为分层并行计算模型的重要组成部分。多核技术的发展推动了并行编程的普及，并行计算平台门槛降低，为了能够有效利用多核处理器的并行计算能力，完成并行编程由计算专家到一般程序员的普及，下一代并行编程模型需要重点研究两个问题：降低并行编程难度和提高编程模型对体系结构的描述能力。这两个问题在一定程度上是相互矛盾的，一方面并行

计算的普及要求所有软件开发人员掌握并行编程技术，需要降低并行编程难度，对程序员隐藏并行体系结构和并行编程的细节，使得程序员集中在应用本身的实现；另一方面提高编程模型对体系结构的描述能力，可以针对不同的硬件系统显式地表达并行性，根据存储参数显式指定数据的访存行为，进而使得程序与硬件紧密结合，发挥最大的性能，但同时也增加了编程模型的复杂性，降低了可编程性。新的并行编程模型需要消除生产率和性能之间的壁垒，对并行编程多方探索，确立一个成熟的并行编程方案。

(3) 研究应用程序在更大规模多核系统上的性能优化技术

虽然目前并行计算机硬件峰值速度达到每秒千万亿次，但是并行应用程序的性能远远没有达到峰值速度，仅仅维持在20%左右。本文的研究对于共享存储的SMP和CMP上程序性能优化方法进行了探索，总结了一些典型的优化技术和优化步骤，在此基础上可以研究更大规模的并行多核集群系统上的应用软件性能优化技术，使用几万甚至几十万的计算内核来加速应用软件性能；总结不同类型应用的性能瓶颈和优化重点，抽取对求解时间和求解规模有重大影响的核心子程序，设计广泛适用的高效基础算法库和并行算法程序库；研究软件优化的自适应技术，根据硬件配置、输入数据特点，自动调整软件的参数，尽可能地获取最大性能。

(4) 研究并行计算中的能效问题

人们在不断追求计算机系统性能的同时，也越来越关注并行计算机的能效问题，提出了绿色计算的概念。绿色计算强调在计算机系统设计和软件开发过程中，关注计算机系统运行时的能量消耗，优化系统的性能瓦特比。降低系统运行时的功耗，可以减少系统运行和维护成本，提高稳定性和可靠性。因此，在并行计算机研制和并行软件开发过程中，需要研究降低能耗的技术，把能耗比作为一个计算机系统的一评价指标，分别研究硬件低功耗技术和软件低功耗技术，实现绿色计算。

(5) 研究硬件和软件全部国产化的高性能并行计算机

并行计算广泛应用在关系到国计民生的各个领域。并行计算领域的核心部件和关键技术，被国外少数国家的公司和企业垄断。国产龙芯处理器、KD系列万亿次并行计算机等相继研制成功，表明我国完全有能力采用国产部件和自主研发技术，设计生产高性能计算机。下一步，我们需要紧跟世界上并行计算的发展趋势，设计研制硬件和软件全国产化的千万亿次乃至万万亿次的并行计算机，提升我国科技竞争力，打破国外垄断，维护国家安全，为国民经济建设和科学研究服务。

参考文献

[1] 周毓麟and 沈隆钧. 高性能计算的应用及战略地位. 中国科学院院刊, 14(3): 184–187, 1999.

[2] 陈国良. 并行计算一结构· 算法· 编程(修订版). 高等教育出版社, 2003.

[3] F.J. Harry and G. Alaghband. *Fundamentals of Parallel Processing*. 北京: 清华大学出版社, 2004.

[4] G.L. Chen, G.Z. Sun, Y.Q. Zhang, and Z.Y. Mo. Study on Parallel Computing. *Journal of Computer Science and Technology*, 21(5):665–673, 2006.

[5] 陈国良, 孙广中, 徐云, and 龙柏. 并行计算的一体化研究现状与发展趋势. 科学通报, 54(8):1043–1049, 2009.

[6] 孙广中, 陈国良, 徐云, 郑启龙, and 吴俊敏. 并行计算系列课程教学团队建设. 计算机教育, (15):42–45, 2008.

[7] 陈国良, 孙广中, 徐云, and 吕敏. 并行算法研究方法学. 计算机学报, 31(9): 1493–1502, 2008.

[8] 陈国良, 吴俊敏, 章锋, and 章隆兵. 并行计算机体系结构. 高等教育出版社, 2002.

[9] M.J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[10] TOP500 Site. http://www.top500.org/. 2009.

[11] 陈国良. 并行算法的设计与分析(修订版). 北京: 高等教育出版社, 2002.

[12] Y. Zhang, G. Chen, G. Sun, and Q. Miao. Models of Parallel Computation: A Survey and Classification. *Frontiers of Computer Science in China*, 1(2):156–165, 2007.

[13] G.M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 483–485. ACM, 1967.

[14] J.L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31 (5):532–533, 1988.

[15] X.H. Sun and L.M. Ni. Scalable Problems and Memory-bounded Speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37, 1993.

[16] 陈国良. 并行算法实践. 高等教育出版社, 2004.

[17] J.R. McGraw and T.S. Axelrod. *Exploiting Multiprocessors: Issues and Options*, volume 6. Addison-Wesley Pub. Co., 1988.

[18] 陆鑫达. 并行程序设计. 北京: 机械工业出版社, 2005.

[19] K. Yelick. Language Innovations for HPCS. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, page 119, 2005.

[20] D. Callahan, BL Chamberlain, HP Zima, C. Inc, and WA Seattle. The Cascade High Productivity Language. In *Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 52–60, 2004.

[21] P. Charles, C. Grothoff, V. Saraswat, and et al. X10: An Object-oriented Approach to Non-uniform Cluster Computing. In *Proceedings of the 20th annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 519–538. ACM, 2005.

[22] E. Allen, D. Chase, J. Hallett, and et al. The Fortress Language Specification Version 1.0 beta. Technical report, Sun Microsystems Inc., 2007.

[23] J. Dean and S. Ghemawat. Map Reduce: Simplified Data Processing on Large Clusters. *Communications of the ACM-Association for Computing Machinery-CACM*, 51(1):107–114, 2008.

[24] M. Herlihy and J.E.B. Moss. Transactional Memory: Architectural Support for Lock-free Data Structures. In *Proceedings of the 20th annual International Symposium on Computer Architecture*, pages 289–300, 1993.

[25] Z.H. Du, C.C. Lim, X.F. Li, and et al. A Cost-driven Compilation Framework for Speculative Parallelization of Sequential Programs. *ACM SIGPLAN Notices*, 39(6):71–81, 2004.

[26] R. Gerber, A.J.C. Bik, K.B. Smith, and X. Tian. *The Software Optimization Cookbook: High-performance Recipes for IA-32 Platforms*. Intel Press, 2006.

[27] V. Sarkar. Optimized Unrolling of Nested Loops. *International Journal of Par-*

allel Programming, 29(5):545–581, 2001.

[28] V.H. Allan, R.B. Jones, R.M. Lee, and S.J. Allan. Software Pipelining. *ACM Computing Surveys (CSUR)*, 27(3):367–432, 1995.

[29] B. Rychlik, J. Faistl, B. Krug, and J.P. Shen. Efficacy and Performance Impact of Value Prediction. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pages 148–154, 1998.

[30] H. Aydin and D. Kaeli. Using Cache Line Coloring to Perform Aggressive Procedure Inlining. *ACM SIGARCH Computer Architecture News*, 28(1):62–71, 2000.

[31] G. Pike and P.N. Hilfinger. Better Tiling and Array Contraction for Compiling Scientific Programs. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–12, 2002.

[32] M.D. Lam, E.E. Rothberg, and M.E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–74, 1991.

[33] 张俊霞, 张焕杰, and 李会民. 基于龙芯2F 的国产万亿次高性能计算机KD-50-I 的研制. 中国科学技术大学学报, 38(1):105–108, 2008.

[34] 侯晓吻, 张林波, and 张云泉. 万亿次机群系统高性能应用软件运行现状分析. 计算机工程, 31(22):81–83, 2005.

[35] L. Oliker, A. Canning, J. Carter, J. Shalf, and S. Ethier. Scientific Computations on Modern Parallel Vector Systems. In *Proceedings of the IEEE Conference on Supercomputing*, pages 1–10, 2004.

[36] J. Carter, L. Oliker, and J. Shalf. Performance Evaluation of Scientific Applications on Modern Parallel Vector Systems. pages 490–503. Springer, 2006.

[37] S.A. Cook and R.A. Reckhow. Time Bounded Random Access Machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973.

[38] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In *Proceedings of the 10th annual ACM Symposium on Theory of Computing*, pages 114–118. ACM, 1978.

[39] L.M. Goldschlager. A Universal Interconnection Pattern for Parallel Computers.

Journal of the ACM (JACM), 29(4):1073–1086, 1982.

[40] R. Cole and O. Zajicek. The APRAM: Incorporating Asynchrony into the PRAM Model. In *Proceedings of the first annual ACM Symposium on Parallel Algorithms and Architectures*, pages 169–178, 1989.

[41] L.G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103–111, 1990.

[42] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken. LogP: Towards a Realistic Model of Parallel Computation. *ACM SIGPLAN Notices*, 28(7):1–12, 1993.

[43] A. Alexandrov, M.F. Ionescu, K.E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into The LogP Model — One Step Closer Towards a Realistic Model for Parallel Computation. In *Proceedings of the seventh annual ACM symposium on Parallel Algorithms and Architectures*, pages 95–105. ACM, 1995.

[44] C.A. Moritz and M.I. Frank. LoGPC: Modeling Network Contention in Message-passing Programs. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and Modeling of Computer Systems*, pages 254–263, 1998.

[45] 计永昶and 丁卫群. 一种实用的并行计算模型. 计算机学报, 24(4):437–441, 2001.

[46] 赵琛. 一种改进的NHBL并行计算模型及其性能评测. 硕士学位论文, 2007.

[47] 许入文. NHBL并行计算模型的扩展及其性能验证. 硕士学位论文, 2008.

[48] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proceedings of the 19th annual ACM symposium on Theory of computing*, pages 305–314. ACM, 1987.

[49] A. Aggarwal, A.K. Chandra, and M. Snir. Hierarchical Memory with Block Transfer. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 204–216, 1987.

[50] B. Alpern, L. Carter, E. Feig, and T. Selker. The Uniform Memory Hierarchy Model of Computation. *Algorithmica*, 12(2):72–109, 1994.

[51] B. Alpern, L. Carter, and J. Ferrante. Modeling Parallel Computers as Mem-

ory Hierarchies. In *Proceedings of Programming Models for Massively Parallel Computers*, pages 116–123, 1993.

[52] Y.Q. Zhang. DRAM (h): A Parallel Computation Model for High Performance Numerical Computing. *Chinese Journal of Computers*, 26(12):1660–1670, 2003.

[53] X. Qiao, S. Chen, and L.T. Yang. HPM: a Hierarchical Model for Parallel Computations. *International Journal of High Performance Computing and Networking*, 1(1):117–127, 2004.

[54] 陈国良, 苗乾坤, 孙广中, 徐云, and 郑启龙. 分层并行计算模型. 中国科学技术大学学报, 38(7):841–847, 2008.

[55] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 59–72. ACM, 2007.

[56] A. Bialecki, M. Cafarella, D. Cutting, and O. O' Malley. Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware. *Wiki at http://lucene. apache. org/hadoop*, 2005.

[57] D.W. Walker and J.J. Dongarra. MPI: a Standard Message Passing Interface. *Supercomputer*, 12:56–68, 1996.

[58] *OpenMP Architecture Review Board, OpenMP Application Program Interface v2.5.* http://www.openmp.org/drupal/mp-documents/spec25.pdf.

[59] B. Nichols, D. Buttlar, and J.P. Farrell. *Pthreads programming.* O'Reilly Media, 1996.

[60] W. Gropp and E. Lusk. A High-performance MPI Implementation on a Shared-memory Vector Supercomputer. *Parallel Computing*, 22(11):1513–1526, 1997.

[61] B.V. Protopopov and A. Skjellum. Shared-memory Communication Approaches for an MPI Message-Passing Library. *Concurrency: Practice and Experience*, 12(9):799–820, 2000.

[62] W. Gropp and E. Lusk. A High-performance MPI Implementation on a Shared-memory Vector Supercomputer. *Parallel Computing*, 22(11):1513–1526, 1997.

[63] H. Tang, K. Shen, and T. Yang. Compile/run-time Support for Threaded MPI

Execution on Multiprogrammed Shared Memory Machines. In *Proceedings of the 7th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pages 107–118. ACM, 1999.

[64] 马捷. 基于*SMP*结点的机群通信系统关键技术的研究. 2001.

[65] Y. Chen, Q. Diao, C. Dulong, C. Lai, and et al. Performance Scalability of Data-mining Workloads in Bioinformatics. *Intel Technology Journal*, 9(12):131–142, 2005.

[66] D.H. Bailey, E. Barszcz, Barton, and et al. The NAS Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[67] B. Fuertig, C. Richter, J. Woehnert, and H. Schwalbe. NMR Spectroscopy of RNA. *Chembiochem*, 4(10):936–962, 2003.

[68] P.P. Gardner and R. Giegerich. A Comprehensive Comparison of Comparative RNA Structure Prediction Approaches. *BMC Bioinformatics*, 5(1):140–157, 2004.

[69] M. Zuker, D.H. Mathews, D.H. Turner, et al. Algorithms and Thermodynamics for RNA Secondary Structure Prediction: A Practical Guide. *RNA Biochemistry and Biotechnology*, 70:11–44, 1999.

[70] 谭光明, 冯圣中, and 孙凝晖. RNA二级结构预测中动态规划的优化和有效并行. 软件学报, 17(7):1501–1509, 2006.

[71] R. Datta, D. Joshi, J. Li, and J.Z. Wang. Image Retrieval: Ideas, Influences, and Trends of The New Age. *ACM Computing Surveys*, 40(2):Article 5:1–60, 2008.

[72] M.S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based Multimedia Information Retrieval: State of The Art and Challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2(1):1–19, 2006.

[73] M. Flickner, H. Sawhney, W. Niblack, et al. Query by Image and Video Content: The QBIC System. *Computer*, 28(9):23–32, 1995.

[74] J.R. Smith and S.F. Chang. VisualSEEk: A Fully Automated Content-based Image Query System. In *Proceedings of the fourth ACM international conference on Multimedia*, pages 87–98. ACM Press New York, NY, USA, 1997.

[75] A. Pentland, RW Picard, and S. Sclaroff. Photobook: Content-based Manip-

ulation of Image Databases. *International Journal of Computer Vision*, 18(3): 233–254, 1996.

[76] T.S. Huang, S. Mehrotra, and K. Ramchandran. Multimedia Analysis and Retrieval System (MARS) Project. In *Proceedings of 33rd Annual Clinic on Library Application of Data Processing-Digital Image Access and Retrieval*, pages 260–265, 1996.

[77] Google Corporation. *Similar Images Search Engine.* http://similar-images.googlelabs.com/, 2010.

[78] Picitup. *Picitup Visual Image Search.* www.picitup.com/picitup/index.jsp, 2010.

[79] TinEye. *TinEye Reverse Image Search.* www.tineye.com, 2010.

[80] G.E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, 1965.

[81] J.L. Lo. *Exploiting Thread-level Parallelism on Simultaneous Multithreaded Processors.* PhD thesis, University of Washington, 1998.

[82] M.S. Hrishikesh, S.W. Keckler, D. Burger, V. Agarwal, W. Lin, and S.K. Reinhardt. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 248–259, 2000.

[83] D.M. Tullsen, S.J. Eggers, and H.M. Levy. Simultaneous Multithreading: Maximizing On-chip Parallelism. In *Proceedings of the 22nd annual International Symposium on Computer Architecture*, pages 392–403. ACM, 1995.

[84] K. Olukotun, B.A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The Case for a Single-chip Multiprocessor. *ACM SIGPLAN Notices*, 31(9):2–11, 1996.

[85] L. Spracklen and S.G. Abraham. Chip Multithreading: Opportunities and Challenges. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 248–252, 2005.

[86] V. Scott, B. Stephen, Peter F., and et al. The POWER4 Processor Introduction and Tuning Guide. *IBM Redbooks*, 2001.

[87] R.M. Ramanathan and T. Evangelist. Intel Multi-core Processors: Leading the Next Digital Revolution. *Technology*, page 1, 2005.

[88] M.J. Quinn and 陈文光. *MPI 与OpenMP* 并行程序设计. 北京: 清华大学出

版社, 2004.

[89] T.S. Lee. Image Representation Using 2D Gabor Wavelets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(10):959–971, 1996.

[90] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[91] Bohm C. Berchtold, S. and H.P. Kriegel. The Pyramid-technique: Towards Indexing Beyond the Curse of Dimensionality. In *Proc. ACM SIGMOD Int. Conf. on Management of Data, Seattle*, pages 142–153, 1998.

[92] I.T. Jolliffe. *Principal Component Analysis*. Springer New York, 2002.

[93] J.L. Bentley. K-d Trees for Semidynamic Point Sets. In *Proceedings of the 6th annual Symposium on Computational Geometry*, pages 187–197. ACM Press New York, NY, USA, 1990.

[94] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. *ACM SIGMOD Record*, 14(2):47–57, 1984.

[95] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions Via Hashing. In *Proceedings of the 25th Internation Conference on Very Large Data Bases*, pages 518–529, 1999.

[96] Intel Corporation. *Intel Threading Analysis Tools*. http://www.intel.com/software/products/Threading/, 2010.

[97] Intel Corporation. *Intel VTune Performance Analyzer*. http://www.intel.com/software/products/VTune/, 2010.

[98] Y. Yan, X. Zhang, and Y. Song. An Effective and Practical Performance Prediction Model for Parallel Computing on Non-Dedicated Heterogeneous NOW. *Journal of Parallel and Distributed Computing*, 38(1):63–80, 1996.

[99] 乔香珍. 并行计算性能的“双流”分析. 计算机科学, 28(10):7–12, 2001.

[100] L. Liu, E. Li, Y. Zhang, and Z. Tang. Optimization of Frequent Itemset Mining on Multiple-core Processor. In *Proceedings of the 33rd international conference on Very Large Data Bases*, pages 1275–1285, 2007.

[101] Q. Zhang, Y. Chen, J. Li, Y. Zhang, and Y. Xu. Parallelization and Performance Analysis of Video Feature Extractions on Multi-Core Based Systems. In *Proceedings of International Conference on Parallel Processing*, 2007.

[102] D.J. Lilja. *Measuring computer performance.* Cambridge University Press, 2000.

[103] J.J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: Past, Present and Future. *Concurrency and Computation Practice and Experience*, 15 (9):803–820, 2003.

[104] P.J. Mucci, S. Browne, C. Deane, and G. Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[105] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder. Using SimPoint for Accurate and Efficient Simulation. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):318–319, 2003.

[106] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure For Computer System Modeling. *Computer*, 35(2):59–67, 2002.

[107] C. Ding and Y. Zhong. Predicting Whole-program Locality Through Reuse Distance Analysis. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation*, pages 245–257. ACM, 2003.

[108] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A Framework for Performance Modeling and Prediction. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 21–21, 2002.

[109] A. Snavely, N. Wolter, and L. Carrington. Modeling Application Performance by Convolving Machine Signatures with Application Profiles. In *Proceedings of the 2001 IEEE International Workshop on Workload Characterization*, pages 149–156, 2001.

[110] D.H. Bailey and A. Snavely. Performance Modeling: Understanding The Past and Predicting The Future. In *Processings of the 11th International Euro-Par Conference on Parallel Processing*, pages 185–195. Springer, 2005.

[111] M.A. Laurenzano, M.M. Tikir, L. Carrington, and A. Snavely. PEBIL: Efficient Static Binary Instrumentation for Linux. In *Proceedings of the International Symposium for Performance Analysis of Systems and Software (ISPASS)*, pages 175–183, 2010.

[112] M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Pro-*

ceedings of the 15th International Euro-Par Conference on Parallel Processing, pages 135–148. Springer, 2009.

[113] J.L. Hennessy, D.A. Patterson, D. Goldberg, and K. Asanovic. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2003.

[114] J.F. Collard and D. Lavery. Optimizations to Prevent Cache Penalties for the Intel Itanium2 Processor. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 105–114, 2003.

[115] A.H.A. Badawy, A. Aggarwal, D. Yeung, and C.W. Tseng. Evaluating the Impact of Memory System Performance on Software Prefetching and Locality Optimizations. In *Proceedings of the 15th International Conference on Supercomputing*, pages 486–500. ACM, 2001.

[116] M. Kondo, H. Sasaki, and H. Nakamura. Improving Fairness, Throughput and Energy-Efficiency on a Chip Multiprocessor Through DVFS. *ACM SIGARCH Computer Architecture News*, 35(1):31–38, 2007.

[117] L. Liu, Z. Li, and A.H. Sameh. Analyzing Memory Access Intensity in Parallel Programs on Multicore. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 359–367. ACM, 2008.

[118] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting Inter-thread Cache Contention on a Chip Multi-processor Architecture. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 340–351, 2005.

致 谢

经过五年的课题研究，毕业的日子越来越近了。回顾在科大的九年学生生涯，感慨万千，借此机会向一直以来给予我指导、关心、支持和帮助的老师、同学和亲人们致以衷心的感谢！

感谢我的导师陈国良教授。您把我领进了并行计算的大门，指导我如何做研究。您对我的学习、科研和生活都给予了大量的关爱。从论文选题到论文完成，您一直都给予了悉心的指导和帮助，您对论文中字句的反复斟酌和修改，让我非常感动。您渊博的知识、严谨的研究态度，高超的学术水平，一门心思搞科研的精神，对国家和民族的责任感，时刻感染着我，激励着我，使我没有丝毫的马虎和懈怠，没有虚度光阴。您的谆谆教诲和言传身教，将使我在以后的生活和工作上终身受用。

感谢实验室的孙广中老师。您是我的良师益友，年轻有活力，科研作风踏实，学术造诣深厚，领导并行与分布式计算组的研究课题。您带领着我做科研，提供各种研究条件和交流机会，您虽然自己工作繁忙，但总是会抽时间与我一起讨论课题中遇到的困难并提出建议。您总是非常认真地帮忙修改润色读博期间发表的论文，对本论文选题和写作都给予了非常有价值的建议，倾注了大量的心血。

感谢科大计算机学院和国家高性能中心（合肥）的许胤龙老师、卢贤若老师、徐云老师、郑启龙老师、卢建平老师、李春生老师和赵莉莉老师多年来在学习、工作和生活上给予的帮助。您们为我提供了良好的学习环境和实验条件，当有事情需要麻烦您们时，您们总是热心地帮忙处理，给予了极大的支持。

感谢中国科学院软件所的张云泉老师。您平易近人，科研思维敏锐，在并行计算模型和数值计算领域有非凡的研究功底。在软件所交流的几个月里，您从工作和生活上给予我很大的帮助。您毫无保留的把您对并行计算模型的独特见解与我分享，多次向我推荐计算模型领域最新的科研动态和研究成果，对我的博士论文方向提供思路和建议。

感谢英特尔中国研究中心的单久龙研究员和陈玉荣研究员。您们深厚的技术功底和勤恳的工作作风深深影响了我，帮助我在科研和技术上不断提高，感谢您们对我的精心指导和无私帮助，从您们身上我学到了许多有益的知识和宝贵的经验。感谢英特尔中国研究中心提供交流和实习机会，良好的工作氛围，一流的工作环境让我受益匪浅。

感谢国家高性能中心（合肥）并行与分布式计算实验室的李晖，李世胜、

张琦、龙柏、方维、吴超、齐鸣、吕强、王录恩、廖银、袁晶、闾明。我们在实验室朝夕相处，开讨论班，交流学术思想，畅谈科研体会。感谢自然计算与应用实验室的陈天石，同处一室的一段时间里，感受到了你对科研的热爱与执着，深深佩服你的勤奋与在演化算法领域取得的成绩。感谢国家高性能中心的赵裕众，一起在实验室通宵写论文的日子，让人终生难忘。感谢你们一直以来给予我的无私帮助，与你们的交流和讨论丰富了我的知识，开拓了我的研究视野，对于我的研究水平，科研态度，研究方法大有帮助，与你们在一起的日子给我留下了美好的回忆。

感谢PB01011本科，SA05011硕士和BA07011博士的所有同学，我们一起享受了纯真、快乐、绚丽的学校生活。我们从全国各地来，一起学习，一起游戏，一起搓饭，毕业后大家各奔东西，但是忘不了教室图书馆里大家自习的身影，忘不了楼道里打牌的聒噪，忘不了球场上胜利的欢呼。花样年华，有你相随，同窗之情，一生常在！

感谢我的家人，一直以来在生活和精神上对我细致的关怀和照顾，让我能够在一个宽松的氛围中不断进步，能够集中精力在科研学习上，感谢你们默默为我做的一切。

九年的时光转瞬即逝，我的学生生涯即将结束，在科大经历的一切仍历历在目，其中欢乐与悲伤相伴、精彩与平淡共存。这段经历是我一生最宝贵的财富，在此谨以此文献给母校，中国科学技术大学！

攻读博士学位期间发表的学术论文与参加的科研项目

发表的学术论文

[1] **Qiankun Miao**, Guangzhong Sun, Jiulong Shan, and Guoliang Chen, "Parallelization and Optimization of Mfold on Shared Memory System", *Parallel Computing*, 已录用, 2010.2.

[2] **Qiankun Miao**, Yurong Chen, Jianguo Li, Qi Zhang, Yimin Zhang, and Guoliang Chen, "Parallelization and Optimization of a CBVIR System on Multi-core Architectures", in *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing(IPDPS2009)*, Rome, Italy, May 23rd-29th, 2009, pp.1-8.

[3] **Qiankun Miao**, Yunquan Zhang, Guangzhong Sun, and Guoliang Chen, "Quantitative Performance Evaluation for Program Optimization by Profiling", in *Proceedings of the Inaugural Symposium on Parallel Algorithms, Architectures and Programming*, Hefei, China, Sep. 16th-18th, 2008, pp.144-162.

[4] 陈国良, 苗乾坤, 孙广中, 徐云, 郑启龙, "分层并行计算模型", 中国科学技术大学学报, 38(7):841-847, 2008.

[5] **Qiankun Miao**, Guangzhong Sun, Jiulong Shan, and Guoliang Chen, "Single Data Copying for MPI Communication Optimization on Shared Memory System", in *Proceedings of The 7th International Conference on Computational Science (ICCS2007)*, Beijing, China, May 27-30, 2007, pp.700-701.

[6] 苗乾坤, 孙广中, 李涛, 陈国良, "若干并行计算模型上的N体问题求解算法", 计算机工程与应用, 43(10):52-54, 2007.

[7] Zhang Yunquan, Chen Guoliang, Sun Guangzhong, and **Miao Qiankun**, "Models of parallel computation: A survey and classification", *Frontiers of Computer Science in China*, 1(2):156-165, 2007.

参加的科研项目

1. 国家自然科学基金重点基金项目"当代并行机的并行算法应用基础研究"（编号：60533020）

2. 中国科学院研究生创新基金“多核结构下并行计算模型和编程模型”

3. 中国科学技术大学研究生创新基金“基于内容的图片检索系统在多核架构上的实现和优化”（编号：KD2008062）