UCMLib:一种多核多线程编程库

杨际祥
(重庆交通大学数学与统计学院 重庆 400074)

摘 要 多核并行编程的开发效率和加速比是影响多核进一步发展的两个重要问题。针对这两个问题,设计并实现了一个轻量级的多核多线程库(UCMLib)。该库基于任务原语概念,提供了数据并行性和任务并行性两种表达逻辑并行性的模式;对多线程编程的复杂性进行了封装和抽象,为开发者提供了高级的编程方法而不必显式地考虑锁和竞争,并降低了并行编程难度以提高开发效率。UCMLib的任务调度器基于对任务队列和工作者线程的有效构建和管理来提高并行程序的加速比。性能测试表明,当计算规模增大时,UCMLib在数据并行性与任务并行性两方面获得了比TPL库略优的加速比。最后给出了可能的性能改进方法以及需要进一步研究的问题。

关键词 多核多线程,数据并行性,任务并行性,任务调度器
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.4.038

UCMLib:A Multi-core Multithread Programming Library

YANG Ji-xiang
(School of Mathematics and Statistics,Chongqing Jiaotong University,Chongqing 400074,China)

Abstract The radical shift in microprocessor architectures to multi-core designs and their further successful development require high productivity and speedup for multi-core programming. Aiming at the two goals,we focused on designing and implementing a lightweight multi-core parallel programming library called UCMLib supporting high level programming in C# for .NET. The library was built upon task primitive concept,supporting two models for expressing logical parallelism,namely data parallelism and task parallelism. The library simplifies parallel programming by encapsulating complexity for obviating the need for programmer to worry explicitly about locking and competing,thus increasing productivity. The speedup techniques surround efficient construction and management of tasks queues. The measured performance shows good parallel speedups comparable to that of Microsoft TPL. Lastly,we also suggested possible improvements and some issues to be further studied.

Keywords Multi-core multithread,Data parallelism,Task parallelism,Task scheduler

1 引言

在过去的几十年里,处理器速度一直按照摩尔定律发展。长久以来,人们习惯性地认为手中的软件会随着新一代处理器频率的提高而自然地运行得更快。然而,这一发展趋势由于处理器在性能提高上受到诸如功耗问题、散热问题以及针脚等的物理限制而可能不复存在了。处理器性能的提高不再单一地依赖于时钟频率的提高,而是要发挥多核的线程级并行性。目前,双核和更多核的处理器已经遍布到桌面PC机、工作站及服务器中。微处理器的性能增长在未来至少5年的发展时间内将来自于使用多核处理器技术的线程级并行的发展[1]。

并行计算的主要特点是与应用领域紧密相关,其尽管在这些领域已取得了许多显著的成绩,但仍处于发展过程中的初级阶段。要加速这个发展过程,往往希望能够在多核系统上运行这些应用,获取平台提供的潜在性能,并且不牺牲可编程性。然而,这种想象包含了许多尚未解决的挑战问题,如软件的可扩展性、可编程性和开发线程级并行性等,所有这些问题需要在软件中得到解决。为了充分利用多核多线程的高度并行性,开发新体系结构提供的潜在的计算能力,需要克服比过去更为复杂、更具有挑战性的并行软件设计问题,甚至需要重新设计应用程序、库、算法以及新的编程语言[1]。

在过去的三十多年中并行软件的发展十分缓慢,远远落后于并行机器的发展,导致的一个事实就是最初硬件的性能和可扩展问题即将由并行软件的性能和可扩展问题所替代,软件已成为制约并行计算应用发展的一个关键问题。此外,多核的普及使得这些过去只是并行计算特定领域人员面临的软件挑战问题转变为每个软件开发者所必须面临的挑战问题,即并行软件的可编程性与开发效率问题[1]。

为了提高并行软件的开发效率或性能,出现了多种支持多核计算的库、系统以及语言(见表1)。其中,Wool库[2,3]提供了一个轻量级(与进程相比,线程在创建、维护和管理方面给操作系统带来的负担都要轻很多,开销少,因此是轻量级的)的C语言编程接口,对于不相关的任务并行性可获得与

到稿日期:2015-03-27 返修日期:2015-08-11 本文受重庆市自然科学基金(KJ1400316),河北省自然科学青年基金(F2015407039)资助。
杨际祥(1975-),男,博士,副教授,主要研究方向为并行计算、数据挖掘与机器学习,E-mail:jixiang_yang@126.com。

TBB[4-6]相当的性能，开发效率低。与产品 Cilk++[7,8]、库 MCSTL[9]、Wool 和 OpenMP[10]一样，Java Fork-Join Framework[11,12]本质上不支持任务抽象概念，限制了它们的使用方式。Cilk++与 Java Fork-Join Framework 局限于 fork-join 并行编程模式，并且后者可能导致死锁[13]。TPL[13-15]基于任务原语概念、C#语言和 work-stealing 任务调度器，在开发效率和并行性能两者间获得了一个较好的折衷。

表 1　几种运行时系统

库/系统名称	任务原语概念	编程语言
Wool[2,3]	不支持	C
TBB[4-6]	支持	C++
Cilk++[7,8]	不支持	C++
MCSTL[9]	不支持	C++
OpenMP[10]	不支持	C++/Fortran
Fork-Join Framework[11,12]	不支持	Java
TPL[13-15]	支持	C#

本文工作重在提供一个开发效率和并行性能综合性能较优的库 UCMLib。该库与 TPL 类似，基于任务抽象原语概念和 work-stealing 任务调度器，获得了与 TPL 相当的性能；当计算规模增大时，实验测得的加速比略优于 TPL。

2　UCMLib 运行时库

图 1 给出了 UCMLib 库的主要组成元素，该库与 TPL 类似，基于任务抽象原语概念和 work-stealing 任务调度器[1]。与标准线程处理和其它的高级库比较，UCMLib 的最大优点在于其简单性，不同的任务间能够相互交换数据、协调事件和同步，程序员在进行并行程序的开发时，需要考虑的是“任务”而非“线程”级别的抽象，仅需要规定好目标计算机将需执行什么样的任务即可，线程的同步与创建等问题则由 UCMLib 自动管理。

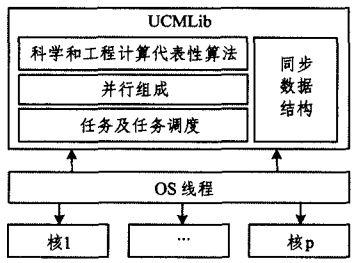

图 1　UCMLib 主要构件

在 UCMLib 中，提供了如图 2 所示的 3 个并行构件（具体包括的构件为：信道、邮箱和分派），只需要通过这 3 个并行构造就可以编写任意复杂通信和同步模式的并发程序。提供的构成元素数据结构包括一些常用的数据结构，这些数据结构是线程安全的，如常用的堆栈和队列等都实现了线程的安全访问，即程序员在编写并行程序时，如果需要访问它们，不必显式地操作锁。调度器负责任务的执行操作、暂停以及创建操作等管理工作，它连续监视排队等待执行的任务数量、当前正在执行任务的线程数量、任务阻塞的数量以及系统中的可用核数。分析所有信息后，调度器决定何时有必要确保负载均衡、最优的线程数量以及程序是可响应的。调度器与线程池确保线程在需要时是启动的，不再需要时是停止的。

在图 2 中，每个任务具有一个集成的邮箱，其它任务可以通过该邮箱放入任意类型的数据而无需考虑同步或锁，邮箱通过缓存允许任务之间实现一对一的通信。信道是对低级同步机制的抽象，为多个任务交换数据提供了一个极其简单的方式，而无需关心传统的如锁、互斥、信号、关键区或条件变量等问题。分派构件使得多个任务之间易于协调通信，在确保多个任务通信公平性的同时，能够确保优先通信的实现，消除了饿死的风险，也不会使得因等待特定事件的发生而浪费 CPU 周期。一个使用任务、信道、分派和邮箱的 UCMLib 程序结构如图 2 所示。在几个构成元素中，任务调度器是核心，与线程池相集成，负责将任务分派给线程池中的各个线程执行。线程位于线程池内，为了提高线程执行任务的效率，将任务调度器集成到一个线程池中。

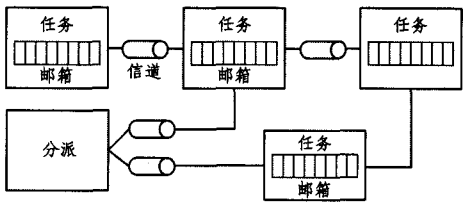

图 2　UCMLib 程序并行构件（任务、信道、分派和邮箱）

线程池通常用一组队列来实现，线程能够被插入至这些队列并从这些队列中取出来执行。它可能是单个全局队列或者是一个分布式队列的集合，典型的是每个线程一个队列，如图 3 所示。全局队列很简单，但每个线程或核访问相同的线程队列时有不利之处，可能会增加通信和引起处理器争用队列访问的情况。对队列的修改（添加线程或删除线程）必须要互斥，从而进一步增加了冲突，导致了操作的串行化。如果线程任务的粒度不大，使得线程或核每访问一次队列取走一个线程，工作一小段时间便再一次访问队列，当线程数增加时，全局队列会很快成为性能的瓶颈。

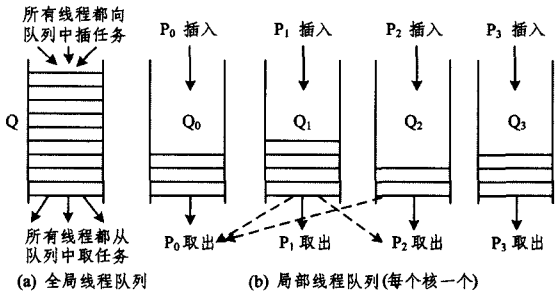

图 3　用队列组来实现动态线程池

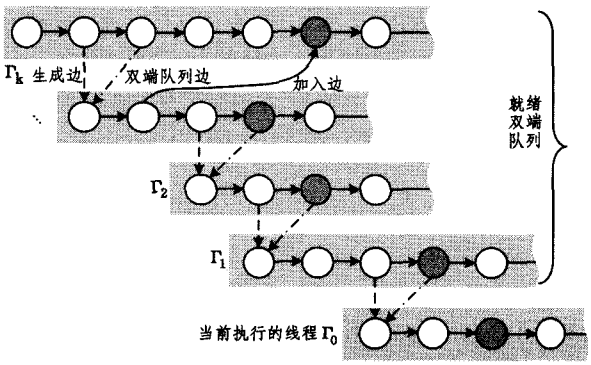

图 4　处理器就绪双端队列结构

图 4 给出了一个处理器就绪双端队列结构，每个线程中的深灰色指令表示该线程当前的就绪指令。在该图中，线程

Γ_k 最后生成一个子线程，当前只有该线程是处于工作状态中的，图中长划线点线为双端队列边。设 G 表示原始的有向无环图，即有向无环图由计算的指令为其顶点，连续边、生成边和加入边为其边。增广的有向无环图 G' 为原始图 G 加之某些新的边。对于每个指令集 u、v 和 w，(u,v) 为一个生成边，且 (u,w) 为一个连续边，双端队列边 (w,v) 被放置到 G' 中。G' 以及双端队列边主要用于分析。

对于局部的分布式队列，每个线程最初在它本地的队列中会分得一些任务。这种初始的分配可以用某种智能化方法得到以减少线程间通信，这样可以提供比自调度和全局队列更多的控制。一个处理器尽量从它自己的队列中移出并执行线程任务，如果它创建了线程，会将它们插入本地队列，当本地队列中没有线程与任务时，它就查询全局队列来从中获得任务，否则就向其它本地或局部队列请求线程与任务。由于任务窃取隐含有通信，还可能产生竞争，在实现窃取过程中就提出了若干有趣的问题。例如，如何极小化窃取、从哪儿窃取、每次窃取多少任务和哪些任务等等。窃取还引入了重要的终止检测问题：如果所有线程或任务都可能产生能动态插入队列中的其他线程或任务，那么如何确定何时不再窃取线程与任务并假定所有线程与任务都做完了？在实践中，简单的启发式算法能工作得很好，而一个完善的方案是相当复杂的，而且会引起很多通讯，本文使用简单的贪婪调度来获得一个实际高效的公平调度。即使在不可预测且环境条件不清楚的条件下，动态技术通常也能给出好的负载均衡，但它们所导致的并行性管理代价要高得多。如果希望显式地控制某个线程与任务必须在某个处理器上执行，动态任务技术也是不适合的，可能增加通讯，也可能会不利于对数据局部性的利用。因此，只要对一个应用和环境能提供好的负载均衡效果，静态技术通常是可取的。

任务调度器连续监控系统中排队等候执行的任务数量、当前正在执行任务的线程数量、被阻塞的任务数量以及可用的核数。分析完所有的信息后，UCMLib 调度器根据需要确定是否介入，以确保负载均衡、最佳的线程数以及程序是可响应的。调度器与线程池相互协作，在需要时启动线程，不再需要时停止线程。

与 Windows 中的线程调度器不同的是，UCMLib 任务调度器是非抢占式的。这意味着一个任务一旦执行便不会被 UCMLib 调度器中断，即使其它任务正在等待执行中。由于不提供固定的时间片给任务执行，由 UCMLib 调度器执行的调度本质上是不公平的。由于公平问题，运行过程中完全有可能会绕过任务调度器。然而，在绝大多数现实世界问题中，该任务调度器运行良好。

3 实验

本实验在四核 PC 机上进行，具体配置：Intel(R) Core (TM)2 Quad CPU Q6000 @2.40GHz，8GB 内存，L2 Cache 4MB，L1 Cache 32KB；Windows Server 2003，Microsoft Visual Studio 2010。

计算实例为 Strassens 矩阵乘(整数)，分别在 TPL 和 UCMLib 环境中采用数据并行和任务并行两种方法来实现，并比较两种实现方法的计算结果，如图 5 和图 6 所示。表 2 给出了 UCMLib 调度器监视到的数据变化情况和调度器的活动记录。

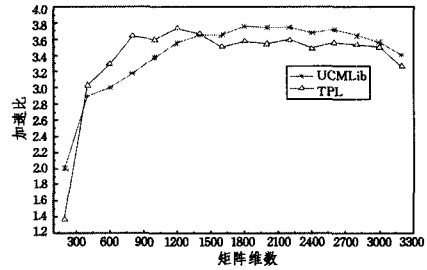

图 5 标准矩阵乘的计算性能比较(数据并行)

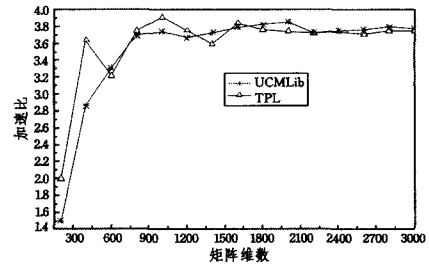

图 6 Strassen 矩阵乘计算性能比较(任务并行)

表 2 在一个四核机器上调度器在各种情况下的活动

核数	任务数	线程数	阻塞任务数	调度器活动
4	4	4	0	无
4	2	2	0	无
4	4	2	0	启动 2 个线程
4	2	4	0	停止 2 个线程
4	10	4	3	启动 3 个线程
4	40	7	0	停止 3 个线程
4	10000	4	0	无

表 2 展示了一个四核机器上的任务调度器在各种情况下的运行状态，其中，任务数一列表示待执行和当前正在执行的任务数；线程数一列表示当前正在运行的线程数；阻塞任务数表示在一个操作上被阻塞的任务数量，当一个任务等待其它任务的数据时可能产生阻塞；调度器活动一列说明了调度器在分析获得全面情况后采取何种活动。观察表 2 可明显得到，在有任务可做的前提下，任务调度器努力将活动的线程数调整为核数；如果在执行期间某些任务被阻塞，则启动新线程，执行处于等待状态的任务，让核保持忙碌状态，不至于空闲下来；当运行的非阻塞线程数超过系统中的核数时，一部分线程被停止执行并被返回到线程池中。

结束语 与 TPL、Wool 与 MCSTL 等编程库一样，本文旨在给出一种多核 PC 环境下的多线程编程库，即 UCMLib。该库提供了一个简单且性能较优的多核编程方法和一个最基本的并行编程环境。作为库的核心，给出了基于双端队列的调度器。当计算规模增大时，实验测得的加速比略优于 TPL。客观来说，TPL 库为其预览版本。本文目前提供了一个 UCMLib 的基础版本，且与 TPL 相比，仍然有待进一步改进，包括：(1)考虑数据局部性和多核存储体系结构局部性；(2)小规模并行性能的改进；(3)可扩展性，即研究在更多的核上的性能。

参 考 文 献

[1] Yang Ji-xiang, Tan Guo-zhen, Wang Rong-sheng. Some key is-

sues and their research progress in multicore software [J]. ACTA Electronica Sinica, 2010, 38(9): 2140-2146(in Chinese)
杨际祥,谭国真,王荣生. 多核软件的几个关键问题及其研究进展[J]. 电子学报, 2010, 38(9): 2140-2146

[2] Karl-Filip F. Wool-A work stealing library [J]. ACM SIGARCH Computer Architecture News, 2009, 36(5): 93-100

[3] Vikranth B, Rajeev W, Raghavendra R C, et al. Topology Aware Task stealing for On-Chip NUMA Multi-Core Processors[J]. Procedia Computer Science, 2013, 18(1): 379-388

[4] Kim W, Voss M. Multicore desktop programming with Intel Threading Building Blocks [J]. IEEE Software, 2011, 28(1): 23-31

[5] Koutny T. Experience with Lamport Clock Ordered Events with Intel Threading Building Blocks in a Glucose-Level Prediction Software[C]// Proceedings of International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO'14). Washington: IEEE CS Press, 2014: 515-526

[6] Navarro A, Asenjo R, Corbera F, et al. A case study of different task implementations for multioutput stages in non-trivial parallel pipeline applications [J]. Parallel Computing, 2014, 48(8): 374-393

[7] Leiserson C E. The Cilk++ concurrency platform[J]. The Journal of Supercomputing, 2010, 51(3): 244-257

[8] Acar U A, Charguéraud A, Rainey M. Scheduling parallel programs by work stealing with private deques [J]. ACM SIGPLAN Notices, 2013, 48(8): 219-228

[9] Singler J, Sanders P, Putze F. MCSTL: the multi-core standard template library[C]// Proceedings of 13th International Euro-Par Conference (Euro-Par'07). Heidelberg: Springer-Verlag, 2007: 682-694

[10] OpenMP home page. The OpenMP API specification for parallel programming [OL]. (2015-2-3) http://www.openmp.org

[11] Lea D. A Java fork / Join framework [C]// Proceedings of the ACM 2000 conference on Java Grande. New York: ACM Press, 2000: 36-43

[12] Tardieu O, Wang H C, Lin H B. A work-stealing scheduler for X10's task parallelism with suspension [J]. ACM SIGPLAN Notices, 2012, 47(8): 267-276

[13] Leijen D, Schulte W, Burckhardt S. The design of a task parallel library[C]// Proceeding of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'09). New York: ACM Press, 2009: 227-241

[14] Qiu J, Scott B, Seung-Hee B, et al. Performance of Windows Multicore Systems on Threading and MPI[C]// Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10). Washington: IEEE CS Press, 2010: 814-819

[15] Mackie R I. Dynamic analysis of structures on multicore computers—Achieving efficiency through object oriented design[J]. Advances in Engineering Software, 2013, 66(12): 3-9

(上接第 181 页)

参 考 文 献

[1] Floyd R W. Assigning Meanings to Programs[C]// Schwartz J T, ed. Proceedings of Symposium on Applied Mathematics. 1967: 19-32

[2] Zhou Chao-chen. Introduction to Formal Semantics[M]. Changsha: Hunan Science and Technology Press, 1985(in Chinese)
周巢尘. 形式语义学引论[M]. 长沙:湖南科学技术出版社, 1985

[3] Hoare C A R. An Axiomatic Basis for Computer Programming [J]. Communications of The ACM, 1969, 12(10): 576-580, 583

[4] Apt K R. Ten Years of Hoare's Logic: A Survey Part-I[J]. ACM Transactions on Programming Languages and Systems, 1981, 3(4): 431-483

[5] Jones C B, Roscoe A W, Wood K R, et al. Reflections on the Work of C. A. R. Hoare[M]. Springer-Verlag, 2010

[6] Winskel G. The Formal Semantics of Programming Languages: An Introduction[M]. MIT Press, 1993

[7] Wang Zhi-jian, Fei Yu-kui, Lou Yuan-qing. Software Component Technology and Its Application[M]. Beijing: Science Press, 2005 (in Chinese)
王志坚,费玉奎,娄渊清. 软件构件技术及其应用[M]. 北京:科学出版社, 2005

[8] Yan Shi-jian, Wang Jun-xiang, Liu Xiu-ying. A Foundation Course for Probability Theory (Second Edition)[M]. Beijing: Science Press, 2009(in Chinese)
严士健,王隽骧,刘秀英. 概率论基础(第二版)[M]. 北京:科学出版社, 2009

[9] Ding Wan-ding. A Summary Measure Theory[M]. Hefei: Anhui People's Publishing House, 2005(in Chinese)
丁万鼎. 测度论概要[M]. 合肥:安徽人民出版社, 2005

[10] Yan Jia-an. Measure Theory Handout (Second Edition)[M]. Beijing: Science Press, 2004(in Chinese)
严加安. 测度论讲义(第二版)[M]. 北京:科学出版社, 2004

[11] Chung K L. A Course in Probability Theory (Third Edition) [M]. Academic Press, 2001

[12] Hailperin T. Probability Logic[J]. Notre Dame Journal of Formal Logic, 1984, 25(3): 198-212

[13] Wang Guo-jun, Wang Wei. Logical Metric Spaces [J]. Acta Mathematica Sinica, 2001, 44(1): 159-168(in Chinese)
王国俊,王伟. 逻辑度量空间[J]. 数学学报, 2001, 44(1): 159-168

[14] Wu Xin-xing, Hu Guo-sheng. Trustworthiness Measurements of Real-time Web Services[C]// 2014 International Conference on E-Commerce, E-Business and E-Service (EEE 2014). 2014

[15] Wu Xin-xing, Hu Guo-sheng, Chen Yi-xiang. Studies on Measurements of Component Approximate Matching[J]. Computer Science, 2014, 41(5): 190-195(in Chinese)
吴新星,胡国胜,陈仪香. 构件近似匹配的度量研究[J]. 计算机科学, 2014, 41(5): 190-195

[16] Wu Xin-xing, Hu Guo-sheng, Chen Yi-xiang. Quantification and Conformance of Web Service Degraded Substitution[J]. Computer Science, 2015, 42(2): 81-85, 94(in Chinese)
吴新星,胡国胜,陈仪香. Web 服务降级替换的一致性问题及量化研究[J]. 计算机科学, 2015, 42(2): 81-85, 94