支持推测并行化的多核事务存储体系结构研究

王耀彬◎编著

科学出版社

支持推测并行化的多核事务存储体系结构研究

王耀彬　编著

科学出版社
北京

内 容 简 介

随着多核平台的普及，如何利用多核加速串行应用的执行已成为当前的热点研究问题。利用事务存储技术解决并行程序正确性维护给并行编程带来的复杂性和对性能的制约问题，已成为学术界和工业界的共识。本书集中反映了作者多年来在多核处理器体系结构研究方面的最新成果和数据，主要涉及线程级推测并行性的判定准则、研究方法和剖析机制；桌面应用、多媒体应用和高性能计算应用的推测并行性剖析；同时支持线程级推测和事务存储语义的多核事务存储处理器体系结构、编程环境、硬件模拟环境设计；在线剖析指导的推测多线程动态优化、分析模型等方面的研究内容。

本书可供从事计算机系统结构、并行程序设计环境与工具、高性能计算的科研人员、[illegible]技术人员、管理人员参考，也可作为本科生和研究生的教学参考用书。

图书在版编目(CIP)数据

支持[illegible]并行化的多核事务存储体系结构研究 / 王耀彬编著. —
北京：科学出版社，2014.6

ISBN 978-7-03-042225-5

Ⅰ.①支… Ⅱ.①王… Ⅲ.①串行处理-研究 Ⅳ.①TP302.1

中国版本图书馆 CIP 数据核字（2014）第 244663 号

责任编辑：杨 岭 孟 锐 / 责任校对：董素芹
责任印制：余少力 / 封面设计：墨创文化

科学出版社出版
北京东黄城根北街16号
邮政编码：100717
http://www.sciencep.com

成都创新包装印刷厂印刷
科学出版社发行 各地新华书店经销
*
2014 年 11 月第 一 版 开本：B5 (720×1000)
2014 年 11 月第一次印刷 印张：11.75
字数：240 千字
定价：56.00 元

前　言

随着多核平台的普及，如何利用多核加速串行应用的执行已成为当前的热点研究问题。而传统的显式锁同步机制自身就有高复杂性、易错性和性能保守等天然缺陷，这从根本上限制了并行程序的可扩展性和编程效率，也限制了对多核资源的充分利用。为了开发更多的多核结构上可利用的线程级并行性，利用事务存储(Transactional Memory，TM)技术解决并行程序正确性维护给并行编程带来的复杂性和对性能的制约问题，已成为学术界和工业界的共识。

多核结构上的程序并行化方法必须允许程序员在两个相互竞争的目标：并行编程的生产力和并行程序的执行效率之间取得平衡。达到这个平衡关键取决于两个相互矛盾的编程抽象方法：①提高底层体系结构的抽象层次，避免要求程序员了解复杂的体系结构细节，从而提高程序员并行编程的生产力；②暴露更多的底层硬件细节给程序员和编译器，以便他们能根据应用的特征和结构设计的参数编程，获得更高的性能/功率比。目前在这两种编程方法的折中问题上还缺乏统一的意见。但是，也有一些基本认识达成了一致：①并行编程模型的发展应从过去的以硬件、应用和形式化方法为中心转变到以人为中心，应将编程模型设计成适合构造高效能的体系结构，易于实现程序的调试和维护的模型；②并行编程模型必须独立于处理器的个数，不限制计算任务的映射和自由调度；③并行编程模型应支持丰富的数据类型和数据大小；④并行编程模型应支持已经证明是有效的并行化和同步方式。

本书集中反映了作者多年来对多核事务存储体系结构研究的最新成果和数据，从有效开发应用中的线程级并行性入手，着眼于高效能、易编程和可兼容这三个目标，通过软硬件协同的优化方式对支持推测并行化的多核事务存储体系结构展开深入研究，使之既能提高多核芯片片上计算资源的有效利用率，又能有效降低并行编程难度，平滑移植传统应用软件。这些研究得到了国家自然科学基金(61202044，61303127)，国家“973”计划项目(2011CB302501)，国家“863”计划基金资助项目(2012AA010902，2012AA010303)，国家科技重大专项基金资助项目(2011ZX01028-001-002-3)和国家发改委项目(13zs0101)的资助，中国科学技术大学安虹教授进行了悉心指导，研究生梁博、刘圆、郭锐、李凌、赵旭剑等参加了部分研究工作并作出了积极贡献，在此一并致谢。

全书共分 15 章，主要由西南科技大学的王耀彬编著。参加编写的人员还有：刘志勤(第 2 章)，梁博(第 3 章和第 4 章)，梁竹(第 5 章)，喻琼(第 6

章)，付婕(第7章)，李凌(第12章)，赵旭剑(第13章)，刘涛(第14章)，刘圆(第8章~第10章，第15章)，郭锐(第11章)。王耀彬和唐苹苹完成了全书统稿和审校。

由于多核技术和产品发展迅猛，作者学识和经验有限，时间仓促，不足之处在所难免，敬请同行专家和读者的谅解和批评指正。

编著者

2013年12月

目　录

第1章 绪 论

1.1 引言

1.1.1 研究意义

随着多核芯片(chip multi-processor, CMP)时代的到来，如何将传统上难以并行化的串行程序线程化执行以加速单个程序的执行，同时也为片上越来越多的计算核心提供更多的可并行执行的计算任务以提高片上资源的利用率，已成为学术界和工业界共同关注的热点研究问题。

提高硬件的应用性能和降低软件的编程复杂度一直是计算机体系结构领域的两难问题。与所有的新型体系结构的机器一样，多核芯片也有两个重要的问题需要解决：一是如何充分利用新结构的新优势有效提升传统结构上不能有效支持的应用的性能；二是如何做到软硬件兼容，即如何实现各种应用软件的平滑移植。

从提升性能的角度来看，一方面，多核芯片拥有更多的计算资源和更快的访存速度，通过在多个单核上并行执行程序，可以开发出串行代码中潜在的线程级并行性，使得在单核芯片时代性能提升已达极限的串行代码重新获得了加速的可能；另一方面，随着半导体工艺逼近纳米量级，出于对功耗、线延迟、设计和验证复杂性、电路不可靠性等方面的考虑，未来的片上单核设计会日趋简单，使得片上单核的性能将比传统意义上的单处理器性能有所下降，串行程序的性能将无法再由片上单核保证，只能通过多个片上单核并行执行的方式加速。

从软件兼容的角度来看，大量的传统应用都是串行程序，要想将当前应用广泛的成熟软件推倒重来几乎是不可能的，因此多核芯片必须要能加速传统的串行应用，实现各种应用软件的平滑移植。

再从工业界的发展进程来看，超微半导体公司(Advanced Micro Devices, AMD)于2006年率先发布了共享片上内存控制器的双核Opteron通用处理器，而行业巨头Intel也于2006年相继推出共享片上二级Cache的双核、4核通用处理器。从2007年开始，多核芯片产品就层出不穷，包括Intel的80核处理器Polaris、

AMD 的 4 核 Opteron、IBM 的 Power 6 和 Cell 以及 P. A. Semi 的 PA6T－1682M 等。但是由于 Polaris 处理器的核数并未得到充分利用，其用途也仅限于科研而未能实现市场化，Intel 公司在 2010 年年底之前推出采用 32nm 工艺的 32 核通用处理器 Gulftown，而且要重点解决如何充分利用多核计算资源和功耗、成本等问题。由此可以看出工业界对于如何充分利用多核资源也有着非常迫切的需求。

因此，利用多核加速串行程序的执行不仅具有重要的研究意义，而且具有很高的实际应用价值。

利用多核加速串行程序的最大难点在于如何将计算和访存依赖不规则的串行程序并行化，开发出在多核上可利用的线程级并行性。而要解决如何对串行程序线程化则需要考虑两方面的问题：一方面是线程划分机制，包括程序的哪些部分可以作为推测线程，推测线程在执行的哪个时刻应该被激发，哪个时刻应该结束；另一方面是线程的执行机制，在推测线程执行过程中，如何保存它产生的结果而不影响处理器间的共享状态，线程间如何同步和通信，以及推测线程如何验证、提交和回退等。

从软件方面，即线程划分时的并行性开发这个角度来看，共享存储的多核结构为利用应用中的线程级并行性加速串行程序的执行带来了巨大的机遇，但同时也向大多数程序员提出了将大量现有的串行程序线程化的难题。传统的线程划分方法为显式地制导线程级并行性的开发，如 OpenMP。这种方法完全依赖程序员，难度大且易出错，具有很大的局限性：用户制导只能开发出并行度有限、粒度较粗的线程级并行性，对于存在复杂的数据依赖关系的代码难以由人工实现细粒度的线程化。从编程效率来看，一方面，锁是不可组合的，不利于软件的模块化；另一方面，与程序顺序相关的错误难以发现和调试。从保障程序的正确性角度来看，锁机制的使用十分复杂，稍有不当就会造成优先级倒置甚至死锁等问题。

从硬件方面，即线程执行时的运行时性能这个角度来看，为了维护程序在运行时的存储一致性，各个线程之间必须采用显式同步的方法来保证多个线程并发执行时的程序语义正确。而基于锁实现的显式同步存在很多问题。从性能的角度来看，锁粒度的选择对性能影响很大。一方面，粗粒度锁可以同步大量的数据，但是粒度过大，会造成互不依赖的代码串行执行，降低程序执行的并发度，而在实际应用中，为了保证并发程序的正确性和设计的简单性，常常不得不保守地采用粗粒度锁；另一方面，细粒度锁由于执行频繁会带来大量额外的开销，也会损伤性能。

为了开发更多的多核上可利用的线程级并行性，解决并行程序正确性维护给并行编程带来的复杂性和对性能的制约问题，学术界从不同的角度分别提出了线程级推测（thread-level speculation，TLS）和事务存储（transactional memory，

TM)两种技术。TLS 旨在打破线程间依赖对线程并行执行的限制，增加程序并行执行的机会。当编译器或者程序员无法确定线程候选者间完备的依赖关系时，不需要采用保守的策略放弃并行或加入大量冗余的同步保护机制，可以无视可能存在的依赖直接并行化，串行语义的维护由运行时支持推测执行的硬件机制保证，从而具有最大程度地挖掘程序中的并行性的技术潜力。TM 旨在为显式的锁同步机制寻找替代方案，通过运行时系统提供的隐式同步机制，实现无锁的共享存储编程。由于不需要请求锁和解锁，所以这也是一种非阻塞的同步方式，既解决了锁机制中存在的死锁、优先级倒置等正确性问题，也解决了锁粒度可能对性能造成的影响。TM 由系统自动维护来自多个线程或执行单元的并发操作对共享存储结构状态修改时的语义一致性。TLS 和 TM 的共同点在于都能够降低并行编程的难度，增加线程并行执行的机会；在硬件方面也提出了类似的支持推测访问、数据缓存、冲突取消的要求。

为了将 TLS 和 TM 技术的研究推向实用，还需要改进以下技术层面上的不足。

在编程模型方面，为了减少算法或程序设计人员的负担，应将尽可能多的指示程序并行执行的程序结构由转换机制(如编译器和运行时系统)插入，而不应由程序员手工去做。这就意味着模型应该尽可能隐藏以下编程细节：①程序到并行线程的分解；②线程到处理器的映射；③线程间的同步。从而为用户提供一种简洁的描述程序并行的手段，使模型易于学习和理解，否则，多数的软件开发者就会不愿意使用它。TLS 和 TM 技术相结合能够很好地符合上述三个要求，但现有的研究工作大都是将 TLS 和 TM 技术分开进行的，很少将二者有机地结合起来考虑，达到上述目标。

在体系结构方面，需要研究能否提出统一的软硬件协同支持的实现方案，将传统的共享存储并行编程模型与支持 TLS 和 TM 的并行编程模型有机地结合起来。即需要研究如何用同一套硬件机制同时有效地实现对这两种技术的支持，并且能够满足大规模并行对可扩展性的需求。需要研究多阶段的并行编程、编译和优化策略。通过离线剖析和静态编译的方法，帮助程序员在并行编程的初始阶段只关注如何找到应用中固有的可利用的线程级并行性而不必关心硬件实现对性能的影响，从而提高并行编程的生产力；再通过在线剖析和动态编译的方法将与应用和底层硬件相关的性能特性在程序运行时动态地暴露给程序员和编译系统，动态实现线程优化分解和线程到处理器的映射，从而在并行编程的生产力和并行程序的执行效率之间取得合理的平衡。

综上所述，为了开发更多的多核结构上可利用的线程级并行性，利用 TM 技术来解决并行程序正确性维护给并行编程带来的复杂性和对性能的制约问题，已成为学术界和工业界的共识。TM 旨在为显式的锁同步机制寻找替代方案，通过运行时系统提供的隐式同步机制，实现无锁的共享存储编程。TM 由系统自动

维护来自多个线程或执行单元的并发操作对共享存储结构状态修改时的语义一致性，降低了并行编程的难度，增加了线程并行执行的机会。而利用 TLS 技术寻找和定位程序中的可并行区域，将其作为线程划分的软件支持手段来配合 TM 技术这一技术方案也得到了越来越多的关注。

1.1.2　传统方法的局限性

沿用传统的线程级并行性开发方法主要有两条技术途径：程序员手工线程化和程序自动线程化。手工线程化主要依靠程序员人工识别出程序中较独立并且容易并行的区域，完成线程化相关工作；而自动线程化则主要依靠软件工具，如编译器、二进制翻译器等，对程序的特征进行自动分析并对其进行线程化。以上两种方法都有一定的局限性。

手工线程化方法通常采用共享存储的并行编程模型和语言，如 OpenMP，由程序员手工并行化，显式地制导线程级并行性的开发，使用锁和同步变量来实现线程间的同步。这种方法的优势在于程序员了解程序设计的思路，能够很好地把握在哪儿进行线程化工作可以更好地提升性能；但是对于怎么样进行线程化工作则存在先天的缺陷——习惯了串行编程控制流思维的程序员必须要跳出思维定式，转到以类数据流执行方式的多线程编程思维上来，精准地识别依赖并且进行准确的同步操作。这种方法难度大且易出错，具有很大的局限性：①并行度的开发问题，用户制导只能开发出并行度有限、粒度较粗的线程级并行性，对于存在复杂的数据依赖关系的代码难以由人工实现细粒度的线程化；②使用锁同步机制的问题，基于锁实现的显式同步存在很多问题，首先是正确性问题，锁机制使用复杂，稍有不当就可能会导致死锁、优先级倒置等问题；其次是性能问题，锁粒度的选择对性能影响很大，粗粒度锁可以同步大量的数据，但是粒度过大，会造成互不依赖的代码串行执行，降低并行度；而细粒度锁由于执行频繁而带来大量额外开销，也会损伤性能，而且放置细粒度锁使并行程序的设计与调试变得异常艰难。

总之，程序员在进行手工线程化工作时，由于线程之间存在数据依赖，为了维护程序并行执行时的正确性，需要加入同步来限定线程的执行顺序。在这个过程中要精准地识别依赖非常困难，而且锁同步机制的使用很容易出错，往往很难保证编写的并行程序的正确性；而要完成一个负载平衡、同步正确、通信开销合理、资源竞争较少的并行编程工作是极富挑战性的。因此手工线程化工作对于程序员，特别是缺乏并行编程经验的新手，是非常困难的。

自动线程化方法则通常依靠并行编译器自动地将串行程序线程化。这种方法的优势在于自动工具在进行烦琐的重复性工作时相对于人的方便有效性，但是其最大的缺陷就在于编译器没有人“聪明”，在依赖分析和程序变换方面存在

很多问题：①依赖分析的准确性问题，在对代码进行依赖分析时，如遇到指针别名等复杂情况，编译器几乎不可能对需要“在哪儿”进行并行和同步得出准确的分析；②依赖分析的开销问题，在分析如含有多个数组引用的多层嵌套循环等复杂结构时，用当前所知最佳的 Omega 测试进行分析时，其开销在最坏情况下的时间复杂度会达到指数级(Pugh，1992)，此等情况下的巨大开销是不能承受的；③代码变换的开销问题，即使依赖关系能够完全确定，按照依赖合理调整程序的执行顺序也非常困难。程序变换需要对代码进行重新划分，并在合适的位置插入同步。虽然代码变换的方式有限，但划分点和插入点的选择却非常多，而由各种选择组成的选择空间范围之大超乎想象。将此问题抽象成在一个可变换空间搜索最优解的问题，其时间复杂度和工作难度几乎是不可接受的。而以最少的通信和同步开销将串行代码划分成多个并行任务已被证明是一个 NP(non-deterministic polynomial)完全问题(Sarkar et al.，1986)。

上述约束，使得自动线程化的工作只能针对某一类特殊应用进行特定优化，才能取得明显的加速效果。而在面对大多数传统应用时，由于分析能力不足，自动线程化工作只能采取一些保守的同步手段保证正确性。但是由此带来的大量伪依赖会对线程执行时的并行性产生不必要的约束，大大损害后续并行的效果。由此可知并行编译器由于不了解程序设计的思路，仅凭代码分析得到的信息识别依赖和变换程序，在充分开发并行性这个问题上受到的限制比程序员更多。同时并行编译器还面临着在编译阶段缺乏输入数据信息等一系列问题，因此开发一个实用的自动线程化工具也是非常困难的。

总之，这种方法要求并行编译器必须仔细地处理线程间大量的模糊依赖关系，而为了保证正确性，分析能力不足的编译器只能采用相对保守的并行化和同步策略，从而极大地影响了程序的并行化效果。实践证明，除了少数科学计算程序，编译器自动线程化不适用于开发大量通用代码中的并行性。

1.2 推测并行技术简介

传统的线程级并行性开发方法不仅编程复杂易错，而且并行化效果不佳，因此在单核时代并行化只是少数高级程序员在高端领域需要考虑的问题，但随着多核平台的普及，对并行应用的需求与日俱增，串行程序和并行执行的矛盾越来越突出。为了解决并行程序正确性维护给并行编程带来的复杂性和对性能的制约问题，学术界分别从打破线程间依赖限制的角度和替代显式的锁同步机制的角度提出了两种线程级推测并行技术——TLS 技术和 TM 技术。如前面所述，这两种技术拥有很多共同点。加上 TM 技术具有自动维护共享存储空间一致性的特点，而 TLS 技术则可以较好地解决基于事务存储的线程划分问题。因此，两种技术从各自不同的发展轨迹自觉地统一到了将两者结合的技术方案上来：

利用TM技术解决线程执行时的一致性维护和性能制约等问题，而TLS技术则作为辅助手段，解决线程划分时的依赖限制等问题。

1.2.1　TLS技术简介

TLS(也称为推测多线程，speculative multithreading)(Prabhu，2005；Martínez et al.，2002；Krishnan et al.，1999；Krishnan et al.，1998)技术使在多核上加速传统上难以手工或自动并行化的串行程序成为可能。它借鉴了超标量处理器挖掘指令级并行的思路，将推测执行、顺序提交这两个超标量的核心思想从指令这个粒度平移到线程这个更大的粒度上来。其基本思想是将串行程序划分成若干代码片段，在片上多个处理器核上推测地并行执行来提高性能。

这种方法的关键是放松了线程间的依赖关系对程序并行执行的限制，对于在编译时不能静态确定的模糊依赖关系，或者可能存在的少量部分依赖关系，先假定它们不存在，依据一定的线程划分策略从串行程序中选择一些适当的可以并发执行的区域(如循环结构或子程序结构)，放到多个不同的处理器核上推测地并行执行，充分挖掘程序的并行潜力。在这种程序执行模型中，只有一个线程是非推测执行的，其余都是推测执行的。只有非推测执行的线程才能修改体系结构状态和写回结果。当这个非推测执行的线程执行完毕时，逻辑意义上与之相邻的下一个线程变成新的非推测线程。TLS技术需要由硬件保存推测执行线程的状态和运行结果，检测线程的推测执行是否出现误推测(miss-speculation)，即发生了写后读(read after write，RAW)相关。一旦出现误推测，就要取消相关推测线程执行的状态和结果，并重新执行。由于推测执行线程的取消和重新执行开销非常大，所以发生线程误推测的概率对系统整体性能将产生严重的影响。

TLS技术的特性在于：①优势，放松串行代码并行化时的严格约束，增加了代码的并行机会；②相同点，线程划分的机制大多源于控制流图信息(循环、子程序或代码注释)；线程执行的机制基本基于两点，即缓存推测写结果、维护一致性；③区别，各种实现机制各不相同，采用软硬件机制的各种方案各有优劣。

从解放程序员和兼容二进制代码的角度来看，利用自动化工具采用软件手段可以较好地全局掌握代码的动态运行特征，专注于最大化代码并行机会，比较适合于线程划分阶段；而从线程执行检测的效率和当前硬件发展的趋势(片上晶体管数目不断增加)来看，将串行语义的维护交由硬件负责，采用一些可以自动维护一致性的技术来支持硬件实现，做到有效控制硬件开销，可以实现比较合理的线程执行检测机制。而通过合理的软硬件协同方案，让两者各司其职而有效结合，才能够最有效地挖掘程序的并行潜力。

当然，TLS技术也不是万能的，目前也存在一些困难：①技术的适用面，TLS技术最适用的是代码中存在不易识别的依赖关系的领域，这样可以采用乐观