·体系结构与软件技术· 文章编号: 1000-3428(2015)11-0077-07 文献标识码: A 中图分类号: TP311

# HL-TLS: 支持热点的线程级猜测编译实现

金 跃,李春强,尚云海,卢永江

(浙江大学超大规模集成电路设计研究所,杭州 310027)

**摘 要:** 猜测并行化编译,即线程级猜测(TLS)编译,可将原来顺序运行的程序并行化。但由于猜测数据的不确定性引起的数据管理开销过大,以及猜测线程失败引起的线程回滚开销,使得并行后的执行性能较低。针对上述问题,提出一种HL-TLS并行化编译优化框架。HL-TLS能有效地标记并行化的循环体为热点循环体,采用对最高层次热点循环体进行更激进的并行化的方式提高性能,而对非热点循环体采用保守的顺序执行以减少开销。实验结果表明,使用HL-TLS编译优化框架,实验程序的执行效率可以提高20%。

**关键词:** 并行计算; 多线程; 猜测执行; 线程级猜测并行; 热点循环; 动态转换执行机制

**中文引用格式:** 金 跃,李春强,尚云海,等. HL-TLS: 支持热点的线程级猜测编译实现[J]. 计算机工程,2015,41(11):77-83.

**英文引用格式:** Jin Yue, Li Chunqiang, Shang Yunhai, et al. HL-TLS: Compiling Implementation of Thread Level Speculation Supporting Hot Spot[J]. Computer Engineering, 2015, 41(11): 77-83.

# HL-TLS: Compiling Implementation of Thread Level Speculation Supporting Hot Spot

JIN Yue, LI Chunqiang, SHANG Yunhai, LU Yongjiang

(Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China)

**【Abstract】** Thread Level Speculation (TLS) compiling can effectively improve the parallel efficiency. But the overheads, caused by the management of the speculative data and the failure of speculative thread's rollback, decreases the improvement of the parallel performance. Aiming at the too big overhead of data management and thread rollback, the Hot Loops-TLS (HL-TLS) framework is proposed. HL-TLS marks the loops which can be efficiently paralleled as HL, using a more eager parallel way on HL to improve performance, while using conservative sequence way on non-HL to reduce the overheads. Experimental result shows that HL-TLS improves 20% performance.

**【Key words】** parallel computing; multi-thread; speculative execution; Thread Level Speculation (TLS) parallel; Hot Loops (HL); dynamic transformation execution mechanism

**DOI:** 10.3969/j.issn.1000-3428.2015.11.014

## 1 概述

循环体的并行化是当今多核系统中提升计算机性能的一个编译研究热点。线程级猜测(Thread Level Speculation, TLS)技术能有效提高并行化效率。TLS技术在假设不存在数据依赖的条件下,延后检查数据冲突,积极地对程序进行并行化,从而提高程序执行的性能。使用硬件TLS技术[1-3],虽然取得较好的效果,但是往往代价昂贵,并且设计复杂。软件TLS编译技术[4-6]相比硬件来说,有更好的可扩展性,但存在资源开销大的问题,如猜测线程间通信的开销、猜测数据管理一致性的开销等,特别是当猜测线程执行时如果检测到数据冲突发生,往往需要将整个线程回滚(Rollback)[6]到安全点,整个程序才能继续保持数据一致性。当猜测错误达到一定比例时,线程回滚形成的CPU资源浪费会导致TLS猜测并行化执行时间比顺序执行时间更长。

HL-TLS(Hot Loops-TLS)并行编译优化框架编译出的程序在执行时,编译时插入的桩代码能有效地标记可以并行化执行的热点循环体,并对循环体的并行化执行方式进行动态转换,转换后的热点循环体执行方式根据热点程度不同分为多个层次。在

基金项目: 国家自然科学基金资助项目(61204111); "核高基"重大专项(2010ZX01030-001-001-006)。

作者简介: 金 跃(1990-),男,硕士研究生,主研方向: 编译器优化,并行计算; 李春强、尚云海,硕士; 卢永江,副教授、博士。

收稿日期: 2014-11-06 修回日期: 2014-12-07 E-mail: xuyv@zju.edu.cn

HL-TLS 并行编译优化框架中，第一层次热点循环体先采用顺序提交的 TLS 并行化执行方式; TLS 并行化执行过程中，如果回滚次数低于一定程度，则此循环体被标记为第二层次热点循环体，被动态转换为激进的直接提交( In-Place) [7-9] 的 TLS 执行方式，从而进一步提升性能; 对于 TLS 执行中回滚次数高于一定程度的循环体，将被标记为非热点循环体，被动态转换为顺序执行，从而避免开销过大而产生 TLS 执行效率低于顺序执行的情况。

针对 TLS 编译技术的上述问题，本文提出HL-TLS 编译优化框架，在 TLS 编译技术的基础上，引入热点循环体( Hot Loops，HL) 的概念来解决现有 TLS 编译技术因线程回滚过多导致资源开销过大的问题，并对热点循环体采用更为激进的并行化执行策略，从而提高程序性能。

## 2 相关研究

LRPD test [10] 系统地实现了软件 TLS 并行化技术。在 LRPD test 中，数据依赖在线程执行的最后检测，相比同步并行化技术( synchronization parallel) 减少了等待和执行时检查的时间，从而提高了性能。此后 R-LRPD test [11] 基于 LRPD test 做了改进，并使用了 sliding window 策略，从而控制了线程间通信的开销。

文献 [12-13] 针对循环体进行 TLS 研究。其中文献 [12] 实现了代价驱动的编译器框架，其使用代价描述决定了哪些可以被 TLS 并行，而哪些不适合。

本文的 HL-TLS 框架不同于上文提到的框架。首先，HL-TLS 是一个全新的概念，以上文章都没有针对 HL-TLS 进行研究。其次，虽然 HL-TLS 和文献 [12] 中的 cost-driven compiler 一样是代价驱动的，但是后者是静态计算代价，而在 HL-TLS 是动态计算并决定的。但是以上文献都没有提到动态转换执行机制的技术。

文献 [14-15] 实现了编译器辅助 TLS 并行化，提高了编程效率，但是对性能并没有改进。

文献 [16] 总结了 TLS 并行技术，并从生命周期的角度提出新的分类方法，但是并未提出新的框架，也并未提高性能。

与本文框架最为接近的是文献 [6-7]。文献 [7] 提出了一个轻量级的 TLS 软件并行框架，并验证了性能提升和可行性。文献 [6] 对文献 [7] 进行改进，在数据管理上采用新的数据结构以节省缓存开销。本文 HL-TLS 框架的不同之处在于: ( 1) 采用了新框架，并提出了 TLS 热点循环体的概念，针对 TLS 最高层次的热点循环体，可以采用激进的直接提交的 TLS 策略，进一步提升程序性能; ( 2) 本文采用了执行机制动态转换，程序将使用最适合的执行方式，从而最大程度上保证程序的性能提升。

## 3 TLS 编译技术

TLS 编译技术，是一种并行化计算编译技术。当程序在编译时不确定是否存在数据依赖关系时，编译器往往保守地认为存在数据依赖关系，从而阻碍了程序的并行化编译。但是运用 TLS 编译技术，可以先假设程序中不存在数据冲突，将程序并行化编译以提高性能，而将数据冲突检测延后。

为了解决数据冲突的问题，TLS 编译技术对猜测线程产生的数据进行了一定管理，管理方法采用 2 种方式 [8]:

( 1) 每个猜测线程产生的数据采用专门的缓存( Buffer) 管理，不直接写到内存( Memory) ; 在猜测线程的提交( Commit) 阶段，再将最终数据从缓存中提交到内存中。这种方式称为顺序提交( Serial-Commit) 。

( 2) 每个猜测线程产生的数据直接写到内存( Memory) ，写入前的内存数据备份到专门的缓存( Buffer) 中; 在猜测线程的提交( Commit) 阶段，如果检测到数据冲突，则内存中的数据被取消，采用备份在缓存中的原数据进行恢复。这种方式称为直接提交( In-Place Commit) 。

直接提交相对顺序提交而言，是一种激进的数据管理方法。直接提交假设猜测线程产生的猜测数据是正确的，且不存在将数据从缓存中提交到内存的阶段，因此如果程序中存在的数据冲突较少时，即回滚次数相对较小时，则采用直接提交将比顺序提交可以获得更好的性能。但是一旦数据冲突超过一定比例，直接提交方式的内存恢复的开销将引起性能急剧下降，并抵消并行化带来的性能提升，因而此时性能比顺序提交要差。另外，在 TLS 编译技术中，当程序执行时冲突比例超过一定阈值时，TLS 并行不管采用直接提交或是顺序提交，其执行时间将会超过顺序执行时间。传统的 TLS 编译技术一般假设程序中存在一定比例的数据冲突，因此使用的数据管理方式往往是比较保守的顺序提交方式。

图 1( a) 展示了一个循环体。假设只有在执行时才知道数组成员间的依赖关系，则为了保证程序的正确性，这段程序只能顺序执行，如图 1( b) 所示。但是如果运用 TLS 编译技术，则可以将这段程序并行化执行。刚开始猜测线程执行时形成的数据，称为猜测数据，因为并没有检验其数据一致性，所以猜测数据是不安全的; 当猜测线程执行完毕，如果没有检测到数据冲突，则该猜测线程产生的数据是安全的，此猜测线程产生的猜测数据被确认写入内存，这个过程称为提交; 如果检测到数据冲突，线程需要撤销( Squash) 到安全状态，并进行回滚操作，以保障数据一致性，这个过程称为撤销并回滚。

如图 1( c) 中所示,线程 1 对 array [3] 进行写操作,线程 3 对 array [3] 进行读操作,而且读操作发生在写操作前,即 RAW ( Read-After-Write) 冲突,则线程 3必须回滚到安全点重新执行。

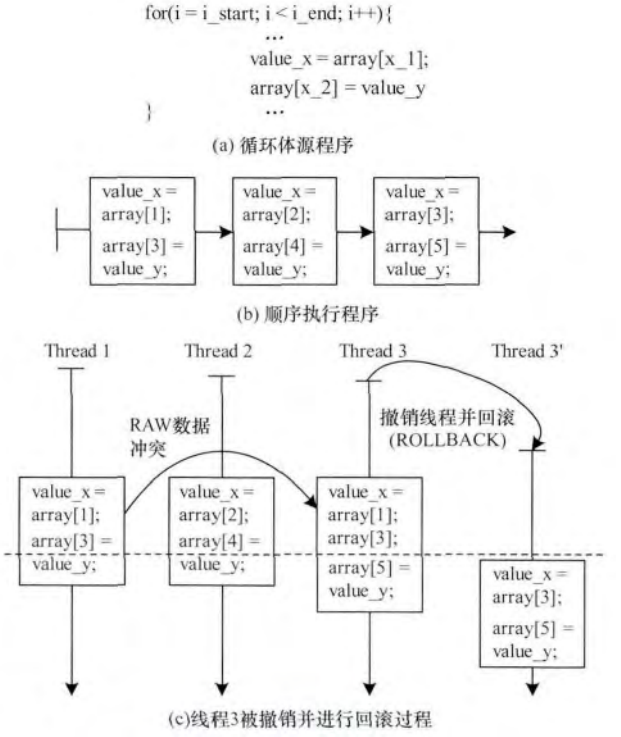



**图 1　线程猜测循环过程**

综上所述,猜测线程的生命周期包括了空闲、猜测化执行、提交或检测到冲突进行撤销并回滚等阶段,如图 2 所示。

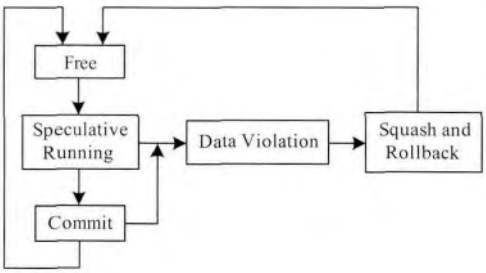



**图 2　线程猜测的生命周期**

## 4　HL-TLS 框架

本文在 TLS 编译技术的基础上提出了 HL-TLS 并行化编译框架: 标记能有效并行化的循环体为热点循环体( HL) ,采用对最高层次的热点循环体进行更激进的并行化的方式提高性能,而对非热点循环体采用保守的执行方式以减少开销。

HL-TLS 编译出的程序,除循环体的线程化部分外,还包含 2 个重要部件: 热点循环体判断部件和执行机制动态转换部件。

热点循环体判断部件的任务是寻找符合热点条件的循环体。热点循环体判断部件将会使用如下 3 个条件:

( 1) 循环体所在的函数被调用多次,超过HL-TLS 定义的临界调用次数——N_HOT_CALL; 循环体顺序执行时,所用的时间超过 HL-TLS 定义的临界执行时间——T_HOT_EXE。

( 2) 循环体 TLS 执行后,冲突比例不超过HL-TLS 定义的临界冲突比例 1——P_ROLLBACK1。

( 3) 循环体 TLS 执行后,冲突比例超过 HL-TLS 定义的临界冲突比例 1——P_ROLLBACK1,但不超过 HL-TLS 定义的临界冲突比例 2——P_ROLLBACK2。

执行机制动态转换部件的任务是根据以上条件决定的循环体热点程度选择该循环体的执行方式: 顺序执行、顺序提交的 TLS 并行化执行和直接提交的 TLS 并行化执行。

循环体在程序执行初期均为非热点循环体,采用保守的顺序执行; 当符合热点条件 1 时,HL-TLS 将其标记为第一层次热点循环体,循环体将采用顺序提交的 TLS 并行化执行; 到达下次执行前,将根据是否符合热点条件 2 来决定是否转换为激进的直接提交的 TLS 并行化执行,如果冲突比例小于P_ROLLBACK1,则 HL-TLS 将其标记为第二层次热点循环体,采用激进的直接提交的 TLS 并行化执行此循环体,以提高执行效率; 如果冲突比例大于P_ROLLBACK1 但是小于 P_ROLLBACK2,即满足热点条件 3,此循环体仍然作为第一层次热点采用顺序提交的 TLS 并行化执行; 如果冲突比例超过P_ROLLBACK2,过多的数据冲突造成的回滚操作将浪费大量 CPU 资源,判定此循环体不适合 TLS 并行化,HL-TLS 将其标记为非热点循环体,此后执行将转为保守的顺序执行方式。

### 4.1　热点判断

如前文所述,热点判断的条件有 3 点。循环体满足条件 1 或者条件 3 时,为第一层次热点循环体,将采用顺序提交的 TLS 方式执行; 循环体满足条件 2时,为第二层次,即最高层次热点循环体,将采用激进的直接提交的 TLS 方式执行,以提高程序性能。如果 3 个条件皆不满足或者 TLS 执行后发现冲突比例超过了 P_ROLLBACK2 时,此循环体将被标记为非热点循环体,其后续的执行方式将使用顺序执行。使用各个条件来判断热点循环体的原因如下:

( 1) 不满足条件 1: 如果循环体所在的函数没有被多次调用( 调用次数小于 N_HOT_CALL) ,循

环体并行化的意义不大，继续延续顺序执行。如果循环体的单次执行时间很短（小于 T_HOT_EXE），则可以判定将来并行化带来的性能提升将被创建线程的开销所抵消，循环体不能被判定为热点循环体，将继续延续顺序执行；

（2）不满足条件 2 或者条件 3：如果冲突比例过高（HL-TLS 中定义为大于 P_ROLLBACK2），则数据冲突而回滚整个线程造成的开销将抵消并行带来的优势，甚至超过了带来的优势，即 TLS 并行化执行时间超过顺序执行时间的情况。

（3）区分第一层次和第二层次热点循环体：如前文所述，如果回滚比例在一定范围内（HL-TLS 中定义为小于 P_ROLLBACK1），则直接提交的 TLS 并行执行方式可以获得比顺序提交的 TLS 并行执行方式更快的执行速度。第二层次热点循环体采用的是直接提交的 TLS 并行执行，其回滚比例小于P_ROLLBACK1；第一层次热点循环体采用的是顺序提交的 TLS 并行执行，其回滚比例大于 P_ROLLBACK1 但小于 P_ROLLBACK2。当回滚比例大于 P_ROLLBACK2，此循环体为非热点循环体。

热点机制的算法实现如下：

```
#entry of the loop_x:
if satisfy condition HOT_1
        if satisfy condition first_HOT_entry
                run_tls( &loop_x)
        else
                if satisfy condition HOT_2
                        run_tls_eager( &loop_x)
                else if satisfy condition HOT_3
                        run_tls( &loop_x)
                else
                        run_seq( &loop_x)
else
        run_seq( &loop_x)
```

判定为热点循环体的优势在于，下一次执行此循环体时，将根据循环体为第一层次或第二层次热点循环体的信息，来选择下一次执行方式。如果此循环体被判定为第一层次热点循环体，则说明此循环体满足了顺序提交的 TLS 并行化执行的前提条件，下次执行则会采用顺序提交的 TLS 并行化执行。如果此循环体被判定为第二层次热点，则说明此循环体用 TLS 来并行化将有极大可能给程序带来性能的提升，此时，可以采用更加激进的策略去 TLS 并行这段程序块，即采用直接提交的数据管理机制。

相反，如果这段循环体没有被判定为第一层或第二层次热点循环体，则此循环体为非热点循环体，只能够保守地对它进行顺序执行。在这种情况下，通常是因为程序执行时遇到了过多的冲突，此时如果仍然采用 TLS 并行化策略，将会导致开销（冲突检测开销、回滚开销）带来的性能下降大过并行化带来的性能提升。此时，相比较现行其他的 TLS 策略，HL-TLS 编译优化框架有利于避免这种情况的发生从而大大减少了不必要的开销。

使用到的宏、数据结构和函数接口如表 1 所示。

**表 1 热点判断部件用到的结构体以及函数接口**

| 结构体/函数接口 | 描述 |
|---|---|
| struct loops_manager loop [N] | 管理每一个循环体，记录执行时的热点、回滚等信息 |
| bool P_HOT ( struct loops_manager * loop) | 判定是否为热点循环体 |
| bool P_SPEC ( struct loops_manager * loop) | 判定下次执行此循环体时，能否进行 TLS 并行 |
| N_HOT_CALL | 调用次数热点条件 |
| T_HOT_EXE | 执行时间热点条件 |
| P_ROLLBACK1 | 回滚比例热点条件 1 |
| P_ROLLBACK2 | 回滚比例热点条件 2 |
| void * run_tls ( struct loops_manager * loop) | 顺序提交的 TLS 并行化策略执行此循环体 |
| void * run_tls_eager ( struct loops_manager * loop) | 用激进化的直接提交的 TLS 并行化策略执行此循环体 |
| void * run_seq ( struct loops_manager * loop) | 顺序执行此循环体 |
| void * rollback ( struct loops_manager * loop,TH * loop_thread) | 检测到冲突时，进行线程回滚，并检测此时回滚数量是否过多；如果过多到影响了并行的性能，则判定后续不再进行 TLS 猜测化执行 |

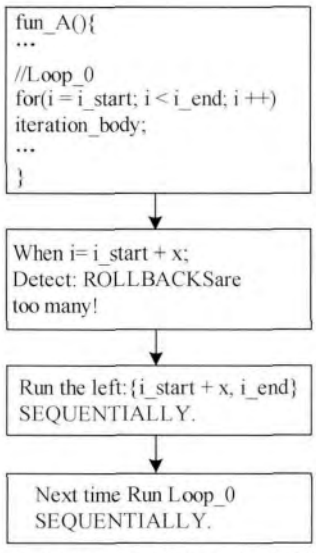
## 4.2 执行机制动态转换

从上节可知，执行机制的动态转换依赖于热点判断部件。对于循环体运用什么方式的执行机制，是由热点判断部件决定的。这种动态的执行方式的转化，解决了因 TLS 并行带来的并行性能低于顺序执行性能的问题；并且因更加激进的 TLS 策略的应用，一定程度上提高了程序的性能。如图 3 所示，在 HL-TLS 框架中的动态转换执行机制分为下面 2 个方面：

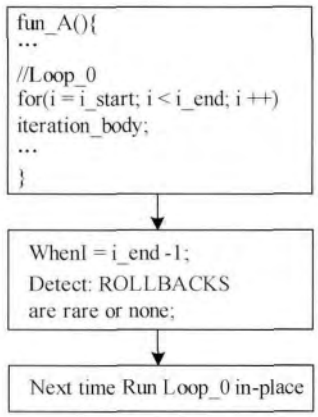
（1）发生在循环体多次调用之间。根据热点判

断部件,可以知道循环体是否被判定为第一层或第二层次热点循环体。如果循环体是第一层次热点循环体,则下次执行循环体时,将用 TLS 并行化机制;如果循环体是第二层次循环体,则使用更加激进的直接提交的 TLS 并行化执行策略;否则,此循环体将使用保守的顺序化执行机制。

(2) 发生在循环体执行时。如前文所述,多次执行循环体时,会根据前几次的执行情况,来指导下一次执行。不仅如此,在循环体执行的时候,也会根据循环体内部前几次循环执行后的数据冲突情况,来指导循环体剩下部分的执行。如果前几次循环出现冲突,那么在剩下的循环体中,出现冲突的可能性就会大大增加。所以,在前几次循环中,数据冲突出现一定的比例之后,就会转而用顺序或同步并行的方法去执行剩下的部分。




(a)发生在循环体内的执行机制动态转换




(b)发生在循环体此次与下次被调用执行之间的执行机制动态转换

**图 3　动态转换执行机制**

### 4.3　程序执行流程

运用 HL-TLS 框架时,编译器对源代码中每个可以 TLS 并行化执行的循环体前插入一段热点判断和执行机制动态转换的代码,并对其进行并行化编译,程序执行时,通过 HL-TLS 的热点判断和执行机制动态转换部件,进行第一层次、第二层次热点的判断,并转换到最适合此循环体的执行方式执行。

(1) 顺序执行程序,跟踪循环体是否满足热点条件 1;

(2) 如果不满足条件 1,则此循环体被判定为非热点循环体,循环体继续采用顺序执行;

(3) 如果满足条件 1,则此循环体被判定为第一层次热点循环体,循环体的下次执行方式为顺序提交的 TLS 并行,TLS 执行后判断是否满足热点条件 2 或者 3;

(4) 如果满足热点条件 2,则此循环体被判定为第二层次热点循环体,此后循环体的猜测并行化执行将采用更为激进的直接提交的 TLS 执行方式;

(5) 如果不满足条件 2,但满足条件 3,则此循环体延续为第一层次热点循环体,此后的执行方式为顺序提交的 TLS 执行;

(6) 如果既不满足条件 2,也不满足条件 3,则此循环体将被判定为非热点循环体,此后的执行方式为保守地顺序执行。

循环体执行的流程如图 4 所示。

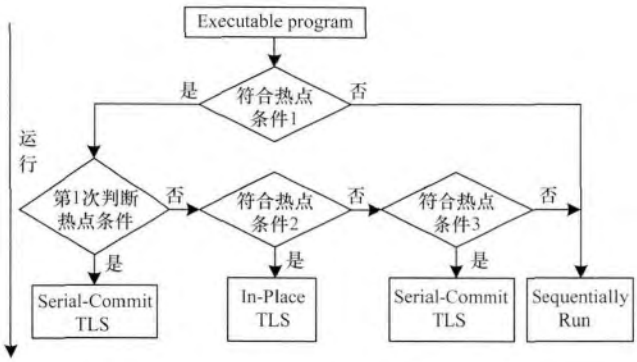



**图 4　HL-TLS 执行流程**

在 HL-TLS 框架中,热点判断部件以及执行机制动态转换的采用,有效减少了因回滚引起的开销,解决了因开销过大而引起的性能降低的问题;而被标记为第二层次热点之后,采用更为激进的直接提交的 TLS 方式执行循环体,在一定程度上提高了其执行的性能。

## 5　性能比较与分析

将 HL-TLS 和 SpLSC[7] 做性能比较。SpLSC 是一个轻量级的 TLS 编译框架,是目前相对实用且性能表现比较好的实现框架。SpLSC 假设程序中会存在一定的冲突比例,采用顺序提交的 TLS 执行方式,

且其执行方式在程序执行时不可改变。相比较 SpLSC 来说,HL-TLS 引进了热点循环体的概念和分析,并运用了执行机制的动态转换,且主要在以下 2 个方面对 SpLSC 体现了优势:

(1) 引进热点循环体判断,能有效减少不必要的开销,特别是避免因回滚过多引起的 TLS 并行性能比顺序性能还低的问题;

(2) 对于最终判定为第二层次热点的循环体,HL-TLS 将采用更加激进的方案进行 TLS 并行,所以在一定程度上能提高加速比。

实验所用的处理器为 Intel Xeon 8 核处理器,频率3.47 GHz,cache 大小为 12 288 KB。所用的编译器为 gcc-4.6.3,使用-O2 选项。实验采用的测试用例来自以下 3 个 benchmark: SciMark2,BYTEmark 和 JOlden。实验结果如表 2 所示,SpLSC 和 HL-TLS 加速比皆为和顺序执行相比较的结果。

表 2 测试用例实验结果

| 测试用例 | 核数 | SpLSC 加速比 | HL-TLS 加速比 |
|---|---|---|---|
| NNBW | 4 | 1.46 | 1.50 |
| | 8 | 1.47 | 1.49 |
| NNFW | 4 | 1.25 | 2.06 |
| | 8 | 1.44 | 2.38 |
| IDEA CIPHER | 4 | 1.95 | 2.44 |
| | 8 | 4.08 | 4.69 |
| IDEA Dekey | 4 | 1.01 | 1.27 |
| | 8 | 1.28 | 1.27 |
| SparseMult | 4 | 1.12 | 1.60 |
| | 8 | 1.28 | 2.14 |

如表 2 所示,4 核上的 HL-TLS 执行速度平均要比顺序执行提高了 79.7%,比 SpLSC 提高了 19.8%。其中 NNFW 性能提升较为明显,比 SpLSC 提升了将近 40%。

8 核上的 HL-TLS 执行速度平均要比顺序执行提高了 152.3%,比 SpLSC 提高了 20.0%。其中 SparseMult 性能提升较为明显,比 SpLSC 提升了将近 40%。而在 IDEA Dekey 中性能略微下降了 0.8%。

从实验结果中可以看到,除了 IDEA Dekey 在 8 核HL-TLS 比 SpLSC 要慢,HL-TLS 的性能均有所提升,不管是在 4 核还是 8 核上,执行速度平均要比 SpLSC 提高了 20%。HL-TLS 性能有所提升的主要原因是 HL-TLS 在执行时识别了循环体热点层次,其后最高层次热点循环体将采用更加激进的直接提交的数据管理方案进行 TLS 并行,因此得到了更多的性能提升。

IDEA Dekey 在 8 核中,HL-TLS 执行速度比 SpLSC 低 0.9%。分析运行结果发现,IDEA Dekey 运行时猜测线程产生的数据有较高的比例是猜测错误的,因此 IDEA Dekey 适合使用顺序提交的 TLS 并行方式。在 IDEA Dekey 中,HL-TLS 没有体现出对 SpLSC 的优势,HL-TLS 热点判断模块产生的额外开销导致了性能略微有所下降。

除了动态地采用直接提交的 TLS 并行策略提升性能,HL-TLS 还将动态地识别那些不适合 TLS 并行化的循环体转换为保守地顺序执行。如表 3 所示,在 EM3D 中,HL-TLS 是 SpLSC 性能的 2 倍左右。其中,SpLSC 性能明显低于顺序执行,且不到顺序执行的 50%。而 HL-TLS 性能接近于顺序执行。SpLCS 引起的回滚数量为: 20(4 核)、60(8 核)。例如,循环体进行 TLS 并行执行时如果数据冲突过多此时线程撤销并回滚的开销已经抵消了并行化带来的优势,HL-TLS 将采用顺序执行,从而避免了如 SpLSC 的性能明显下降的问题。

表 3 SpLSC 中开销过大的情况

| 测试用例 | 核数 | SpLSC 加速比 | HL-TLS 加速比 |
|---|---|---|---|
| EM3D | 4 | 0.41 | 0.87 |
| | 8 | 0.47 | 0.90 |

因此,从总体上看,无论循环体猜测并行执行时回滚次数多少,HL-TLS 都体现出了对 SpLSC 的优势。

## 6 结束语

本文提出一种新的 TLS 编译框架 HL-TLS。该框架有效解决了现有 TLS 编译技术中猜测并行执行回滚开销过大、性能提高有限的问题。HL-TLS 对第二层次的热点循环体采用更加激进的 TLS 并行化策略,从而进一步提高了并行执行的性能。而针对回滚过多影响性能、甚至导致并行性能低于顺序执行的情况,HL-TLS 能够加以识别并动态转换执行机制,从而避免此种问题的发生。实验结果表明,HL-TLS在一定程度上提高了程序的性能,并且解决了开销过大的问题。下一步的研究方向是在 HOT 机制中加入同步并行的 HOT 判断,并在动态转换执行机制中进行同步并行方案,实现顺序、同步并行、顺序提交 TLS 并行和直接提交 TLS 并行执行方式的动态转换,从而进一步挖掘程序的并行化能力。

## 参考文献

[1] Steffan J G, Colohan C G, Zhai A, et al. A Scalable Approach for Thread Level Speculation[C]//Proceedings of the 27th Annual International Symposium on Computer Architecture. New York, USA: ACM Press, 2000: 1-12.

[2] 赖 鑫,刘 聪,王志英. 支持线程级猜测的存储体系结构设计[J]. 计算机工程,2012,38(24): 228-234.

[3] Krishnan V,Torrellas J. A Chip-multiprocessor Architecture with Speculative Multithreading [J]. IEEE Transactions on Computers,1999,48(9):866-880.

[4] Cintra M, Llanos D R. Toward Efficient and Robust Software Speculative Parallelization on Multiprocessors [C]//Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York,USA: ACM Press,2003: 13-24.

[5] Liu W,Tuck J,Ceze L,et al. POSH: A TLS Compiler that Exploits Program Structure [C]//Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York,USA: ACM Press,2006: 158-167.

[6] Yiapanis P, Rosas-Ham D, Brown G, et al. Optimizing Software Runtime Systems for Speculative Parallelization [J]. ACM Transactions on Architecture and Code Optimization,2013,9(4):39-51.

[7] Oancea C E,Mycroft A. Software Thread-level Specula-tion: An Optimistic Library Implementation [C]//Pro-ceedings of the 1st International Workshop on Multicore Software Engineering. New York,USA: ACM Press,2008: 23-32.

[8] Oancea C E, Mycroft A. A Lightweight In-place Implementation for Software Thread-level Speculation [C]//Proceedings of the 21st Annual Symposium on Parallelism in Algorithms and Architectures. New York, USA: ACM Press,2009: 223-232.

[9] Oancea C E,Mycroft A. Set-congruence Dynamic Anal-ysis for Thread-level Speculation [M]. Berlin,Germany: Springer-Verlag,2008: 156-171.

[10] Rauchwerger L, Padua D. The LRPD Test: Speculative Run-time Parallelization of Loops with Privatization and Reduction Parallelization [C]//Proceedings of ACM SIGPLAN'95. New York, USA: ACM Press, 1995: 218-232.

[11] Dang F, Yu H, Rauchwerger L. The R-LRPD Test: Speculative Parallelization of Partially Parallel Loops, TX77843-3112 [R]. College Station,USA: Texas A & M University,2001.

[12] Du Zhaohui,Lim Chu Cheow,Li Xiaofeng,et al. A Cost-driven Compilation Framework for Speculative Parallelization of Sequential Programs [C]//Proceedings of ACM SIGPLAN'04. New York, USA: ACM Press, 2004: 71-81.

[13] Dubey P K,O'Brien K,O'Brien K M,et al. Single-program Speculative Multithreading (SPSM) Architecture [C]//Proceedings of PACT'95. Manchester, UK: [s. n.],1995: 109-121.

[14] Xiang Lingxiang, Scott M L. Compiler Aided Manual Speculation for High Performance Concurrent Data Structures [C]//Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York,USA: ACM Press,2013: 47-56.

[15] Aldea S,Estebanez A,Llanos D R,et al. A New GCC Plugin-based Compiler Pass to Add Support for Thread-level Speculation into OpenMP [C]//Proceedings of EPP'14. Porto, Portugal: Springer International Publishing,2014: 234-245.

[16] 郭　辉,王　琼,沈　立,等. 多核平台上的线程级猜测执行综述 [J]. 计算机科学,2014,41(1):16-21.

编辑　索书志

(上接第 72 页)

## 参考文献

[1] Newman M E J. The Structure and Function of Complex Networks [J]. SIAM Review,2003,45(2):167-256.

[2] Albert R,Barabasi A L. Statistical Mechanics of Complex Networks [J]. Reviews of Modern Physics,2002,74:47-97.

[3] Kernighan B W,Lin S. An Efficient Heuristic Procedure for Partitioning Graphs [J]. Bell System Technical Journal,1970,49(2):291-307.

[4] Girvan M,Newman M E J. Community Structure in Social and Biological Networks [J]. PNAS Proceedings of the National Academy of Sciences,2002,99(12):7821-7826

[5] Newman M E J. Fast Algorithm for Detecting Commu-nity Structure in Networks [J]. Physical Review E,2004,69(6).

[6] Rosvall M, Bergstrom C T. Maps of Radom Walks on Complex Networks [J]. Proceedings of the National Academy of Science,2008,105(4):1118-1123.

[7] 刘　微. 基于共享邻居数的社团结构发现算法 [J]. 计算机工程,2011,37(6):172-174.

[8] Xuan V N,Julien E,James B. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance [J]. The Journal of Machine Learning Research,2011,11(10):2837-2854.

[9] Ying Pan. Detecting Community Structure in Complex Networks via Node Similarity [J]. Physica A,2010,389:2849-2857.

[10] 胡　琼. 社会网络结构划分算法研究 [D]. 上海: 上海交通大学,2014.

[11] Lusseau D. The Emergent Properties of a Dolphin Social Network [J]. Proceedings of the Royal Society of London, Series B,2003,270(Suppl.):186-188.

[12] Sun P G, Gao L, Han Shanshan. Identification of Overlapping and Non-overlapping Community Structure by Fuzzy Clustering in Complex Networks [J]. Information Sciences, 2011,181(6):1060-1071.

[13] Andrea L, Santo F, Filippo R. Benchmark Graphs for Testing Community Detection Algorithms [J]. Physical Review E,2008,78(4).

编辑　金胡考