

# 基于并串结合机制的数据竞争消除系统设计与实现

沙 晗<sup>1</sup>, 应 欢<sup>2,3</sup>, 唐博文<sup>1</sup>, 乔 扬<sup>2</sup>

(1 首都师范大学, 北京 100048; 2 中国科学院大学, 北京 100049; 3 中国科学院 声学研究所, 北京 100190)

**摘 要:** 数据竞争是多线程程序最为常见的问题之一, 数据竞争的出现会导致程序的原子性遭到破坏, 从而使得推理程序的行为和改进程序的性能更加困难. 为解决数据竞争带来的问题, 提出了一种运行时的数据竞争消除系统 DRFS(Data-Race-Free-System). DRFS 将多线程程序的执行分为并行执行和串行执行两种模式, 采用合理有效的模式切换机制将两种模式结合执行, 为并行程序提供一个无数据竞争的运行环境. 实验结果表明, DRFS 在保证并行程序不会出现数据竞争的同时, 并未引入很大的性能开销.

**关键词:** 数据竞争; 并行程序; 串并结合; 共享内存

**中图分类号:** TP311

**文献标识码:** A

**文章编号:** 1000—7180(2016)07—0089—04

## Design and Implementation on the Data Race Free System Based on Combining Parallel and Serial Mechanism

SHA Han<sup>1</sup>, YING Huan<sup>2,3</sup>, TANG Bo-wen<sup>1</sup>, QIAO Yang<sup>2</sup>

(1 Capital Normal University, Beijing 100048, China

2 University of Chinese Academy of Sciences, Beijing 100049, China;

3 Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China;)

**Abstract:** Data races is one of the most common multithreaded program problems. Data races makes atomic program is destroyed, thus making the reasoning process behavior and improved program performance more challenging. to solve the problem introduced by data races of parallel programs, DRFS is proposed. This paper present DRFS(Data-Race-Free-System), a novel multi-threading runtime that divides the execution of multithread programs into two parallel mode and serial mode, DRFS adopts existing hardware support page protection mechanism, to eliminate multithreaded shared-memory data races, which provides a data-race-free environment for parallel programs. The Experimental results demonstrate that DRFS guarantes the free of data races while incurring acceptable performance and memory overhead.

**Key words:** data race; parallel program; serial parallel combination; shared memory

### 1 引言

随着芯片技术的发展, 单核处理器已经发展到了多核处理器, 开发编程架构由传统的串行编程模式逐渐转向并行编程模式, 多线程编程模型成为提高片上多核处理器性能的主要编程方式之一. 然而, 编写正确的多线程程序并不容易, 这是因为多线程程序的不确定性. 并行程序的不确定性因素有串行

程序的不确定的输入源(系统调用, 信号)以外, 还包括并行程序本身引入的线程调度的不确定性, 其表现为数据竞争. 当两个或多个线程同时访问共享内存, 且其中至少有一个线程试图写入内存, 这时就发生了数据竞争<sup>[1]</sup>. 这种情况通常被认为是一个故障, 在大多数情况下, 数据竞争会导致程序不可预知的结果或者不正确的执行. 本文设计并实现了针对 POSIX 多线程程序的数据竞争消除系统 DRFS

(Data-Race-Free-System).

2 数据竞争消除背景

内存共享有显式和隐式两种方式. 前者发生在多个进程通过使用内存映射的方法实现内存共享, 后者则是整个地址空间的共享<sup>[2-3]</sup>.

在软件层面, 监测和控制内存访问的主要工具是页面保护机制. 根据程序的时间与空间局部性, 一个线程在一定时间内可能访问一页中的多个位置. 如果能保证在同样的时间片内, 没有其他线程访问该页, 那么这一页在这个时间内对该线程来说就是私有的, 因此就不会出现共享内存的数据竞争. 这种机制就要求需要一种协议来控制页面拥有权的转移. 页面权限管理机制要求更新程序的页表项 (Page Table Entry, PTE). 在页式存储管理机制中, 操作系统会为每个进程配置一张页表, 属于该进程的所有线程共享该页表, 其中一个线程对页表项的更改将对其他所有线程都可见. 这就意味着在线程共享页表的机制下很难做到监测线程对共享内存的访问.

UPLAY 是中国科学院计算技术研究所体系结构国家重点实验室面向多线程执行实现的一种记录与回放系统. 该系统记录多线程执行过程的不确定性因素 (例如系统调用, 共享内存访问等), 按照记录的信息正确回放多线程的执行. UPLAY 通过重载一些必要的库函数将多线程改为多进程, 利用多进程的私有页表, 结合硬件支持的页保护机制, 在进程下模拟线程的状态. 利用访存异常处理函数, 采用并行读, 串行写的 CREW (Concurrent Read, Exclusive Write) 协议控制"线程"对共享数据的访问. CREW 协议是由 P. J. Courtois 等人提出针对内存访问顺序的控制协议. 协议规定某个页面 (以页为单位) 的数据可以同时被多个进程读取, 但是必须只能被一个进程改写. 一个页面可能处于的状态有三种: 无人读写 (public) 状态、共享读 (share-read) 状态、独占写 (owner-write) 状态. 如图 1 给出的 CREW 协议的状态转换规则. 该协议可以串行化应用程序中的共享内存的访问操作, 保证了多线程程序对共享内存的访问是安全的, 不会发生数据竞争.

3 DRFS 设计

DRFS 将多线程程序的执行分为并行执行和串行执行两种模式. 并行模式下, DRFS 基于 UPLAY 已有的对共享内存访问控制机制, 消除并行程序的

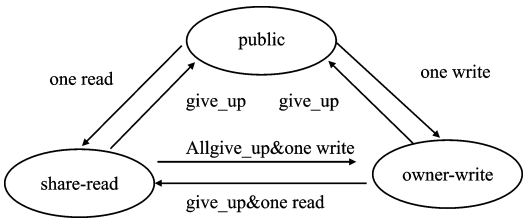


图 1 CREW 协议状态转换

数据竞争. 本节介绍 DRFS 框架设计. 首先介绍引入串行模式的原因, 最后介绍串行模式下 DRFS 设计方案以及 DRFS 串行并行模式的切换机制.

3.1 引入串行模式的原因

在并行程序中, 共享内存是实现线程间通讯的主要方式之一, 各个线程在执行过程中会频繁访问共享内存. 表 1 给出了 SPLASH-2 测试程序每纳秒对共享内存的访问次数, SPLASH-2 是斯坦福大学推出的并行程序测试集. 从表中可以看出各个程序对共享内存的访问较为频繁, 其中 raytrace\_splash 达到每纳秒 32 次的高频率访问次数.

表 1 SPLASH-2 程序集每纳秒对共享内存的访问次数

| Benchmark       | Number of shared access per ns | Benchmark     | Number of Shared access per ns |
|-----------------|--------------------------------|---------------|--------------------------------|
| barnce          | 5                              | water_nsquare | 12                             |
| fmm             | 4                              | water_spatial | 10                             |
| ocean_cp        | 18                             | radix         | 1                              |
| ocean_ncp       | 12                             | lu_cb         | 5                              |
| raytrace_splash | 32                             | fft           | 6                              |

并行模式机制下, DRFS 对所有共享页面初始状态设置页保护, 任何线程对共享页面的访问权限都是不可读不可写, 即 CREW 协议的 public 状态. 线程在访问共享内存之前, DRFS 首先获得该共享页面的状态, 然后 CREW 协议结合当前页面状态判断该线程是否有权限访问它所请求的共享内存. 如果某个线程试图使用不具有访问权限的共享内存 (这种情况下, 把这类线程称之为请求页面线程), 此时会触发 SIGSEGV 信号, 捕捉到信号后进入系统注册的信号处理函数. 在信号处理函数中请求页面线程将请求对共享内存的访问权限. 此时, 该线程处于挂起状态直到获得相关共享内存的访问权限. 因此, 并行模式下多线程程序的高频率访问共享内存会导致频繁触发 SIGSEGV 进入访存异常处理函数, 请求并等待获得相关页面的访问权限. 对于持有相关页面访问权限的线程 (这种情况下, 把这类线程称之为持有页面线程, 该类线程相对于同一共享页

面的请求页面线程)来说,释放自己已持有页面的同步点在库函数,如果持有页面线程长时间没有进入页面同步点释放自己所持有的页面,这样会使得请求该页面访问权限的线程在信号处理函数中等待获得访问权限的时间过长.如果多个线程都处于等待页面权限状态从而程序的并行程度将大大降低,以至于整体减速倍数超过线程数.

然而,将并行程序完全串行化执行虽然不会出现共享内存的数据竞争,但是对于一些规模较大的程序,串行程序的性能开销远远高于并行程序.因此,为了降低并行模式带来的开销,本文提出一种并行串行切换执行的策略.在程序执行的一段时间内,当部分线程长时间处于挂起等待状态,使得程序的并行性低于设定阈值时自动切换为串行模式.

3.2 串并模式切换机制

串行模式下保留线程改进程的机制,执行时的任意时刻有且只有一个线程在运行,不存在数据竞争的问题,因此关闭页面权限控制模块,各线程对共享内存的访问不再依照 CREW 协议并恢复页面的原始权限.串行模式下,线程间的切换点设计在各个线程的同步点或锁相关的 pthread 库函数的入口和出口处,一旦所有的线程都有了一次机会运行后将切换回并行模式执行.图 2 中,多线程程序的执行分为并行和串行两种运行模式, $T_3$ 检测到当前程序的并行度达到阈值(a),则切换运行模式为串行,并由  $T_3$ 通知其他线程做切换工作,直到  $T_2, T_4, T_1$ 所有线程在库函数内完成模式切换工作(b),同时四个线程按照完成切换工作的先后顺序加入等待队列.在串行模式下按队列顺序一一执行(c),当执行线程进入同步点或锁相关的库函数(其他线程占有该锁时)将唤醒队首线程.所有线程在串行模式下都执行一次后将切换回并行模式(d).

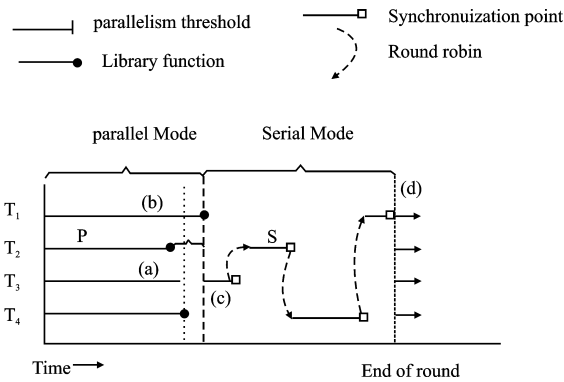


图 2 并串模式的切换及串行模式的执行

4 DRFS 实现

图 3 为 DRFS 系统结构图. DRFS 整体架构分为四大模块:库函数包装模块、模式切换模块、堆区数据管理模块、页面权限控制模块. DRFS 将最终编译成动态链接库,使用 Linux 环境变量 LD\_PRELOAD 优先加载来截获应用程序执行时的控制权,用户态下实现运行时消除共享内存多线程程序的数据竞争.本节主要介绍 DRFS 串行模式实现以及并串模式切换机制实现的关键问题.

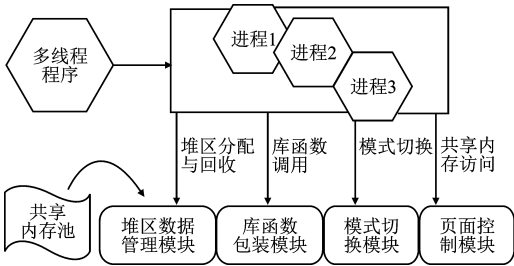


图 3 DRFS 系统结构图

4.1 串行模式的执行

“多线程”执行串行化即多线程程序在任意时刻只有一个线程处于活动状态. DRFS 为程序维护一个线程切换的等待队列,除当前活动的线程以外,其余线程均处于挂起状态.串行模式下,活动线程有两个时间点去完成线程切换工作.一是活动线程进入 pthread\_cond\_wait 等同步操作时,在真正调用同步函数之前完成唤醒队首线程的工作然后进入同步点,该线程调用同步函数返回后加入等待队列;二是活动线程进入锁相关的库函数,为了避免程序产生死锁,DRFS 要保证活动线程在调用加锁相关的库函数时一定能获取到锁资源,为此 DRFS 维护一个公共的锁变量公告板,公告板里记录已经被各个线程持有的锁变量.通过公告板来确定当前线程能否进行加锁操作.若公告板里没有当前的锁变量,活动线程将执行加锁操作;若相关锁变量已经被其他线程持有,活动线程则唤醒队首线程执行,然后加入等待队列.由于串行模式下,多线程程序在执行的任意时刻只有一个线程在运行,不存在多个线程并发访问共享内存的情况,所以在串行模式下不涉及页面权限的控制,各线程将不再依照 CREW 协议对共享内存进行访问.

4.2 运行模式切换

4.2.1 并行模式切换串行模式的时机

并行模式下,DRFS 基于 CREW 协议控制多线程对共享内存的访问,当一个线程使用该线程不具

有访问权限的共享内存时,触发段错误,进入信号处理函数请求该页面的访问权限.根据 CREW 协议若请求不成功,该线程将被挂起等待获得页面.在不失并行优势的前提下,为了降低频繁触发 sigsegrv 等待获取页面权限引入的性能开销,DRFS 添加串行运行模式,采用并行串行切换执行的机制.根据程序在执行过程中的并行性,DRFS 判断模式切换的时机.线程从正常执行开始直到请求共享页面访问权限这个过程所使用的时间,称之为工作时间( $T_{p-work}$ ),线程从请求共享页面访问权限到获得权限这个过程所使用的时间,称之为等待时间( $T_{p-wait}$ ).程序的并行性指的是工作时间所占着两者总时间的百分比,式(1)为并行性的计算公式.当一个或多个程序在一段时间内因请求共享内存的访问权限不成功而长时间处于挂起等待状态,使得程序的并行性低于 50% 的情况下,DRFS 将由并行模式切换为串行模式.

$$\text{Parallelism} = \left( \sum_{i=0}^N T_{iwork} \right) / \left( \sum_{i=0}^N T_{iwait} + \sum_{i=0}^N T_{iwork} \right) \quad (1)$$

#### 4.2.2 串行模式切换并行模式的时机

阿姆达尔定律(Amdahl's law)[5]是针对并行处理的可扩展性给出的一个模型,它指出使用并行处理的提速由问题的可并行部分所决定,即并程序的加速比受限于程序串行化部分.保持和改进系统的可扩展性要求最大限度地减少串行模式占总体时间的百分比.一次完整的模式切换称之为为一轮(如图 2),一轮所用的时间为  $P+S$ ,  $P$  为并行模式所占用的时间,  $S$  为串行模式所占用的时间,系统的可扩展性则受限于串行占总体时间的百分比,即  $S/(P+S)$ .这要求 DRFS 尽量减少串行模式的执行时间,提高并行程序的并行度.因此,DRFS 在串行模式下,一旦所有的线程都有一次机会运行后将切换回并行模式.

## 5 实验数据

### 5.1 测试环境

实验平台拥有 Intel Xeon CPU E7-4 8071, 87 GHz L2 的处理器, 256 kB 的二级缓存和 1 GB 的主存;操作系统为 Debian GNU/Linux 6.0, 内核版本是 2.6.32;编译器是 GCC-4.4.5. 测试集选择多线程的 SPLASH-2 和 PARSEC.

### 5.2 性能开销

#### 5.2.1 DRFS 并串结合的性能开销.

实验结果表明,在并行串行模式切换的机制下,

DRFS 在整体性能上达到最佳状态,平均性能开销控制在 4 倍,同时最大程度提升程序并行度保留了并程序优势,消除并程序的竞争.

#### 5.2.2 各个测试程序串行执行时间占总执行时间的百分比

图 4 给出测试程序串行模式执行时间占总执行时间的百分比.平均情况下,DRFS 串行模式执行时间占总体时间百分比可以控制在 40% 以内,其中 blackscholes 串行时间占总执行时间的百分比最少,只有 3.7%;而 raytrace\_splash 串行时间占总执行时间的 45%,这是由于相对于其他测试程序来说,raytrace\_splash 访问共享内存频率较高,线程处于等待状态时间过长致使频繁触发模式切换条件.

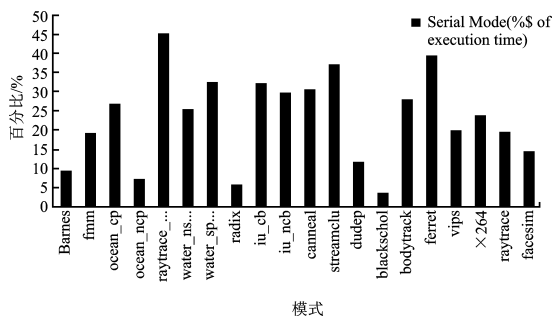


图 4 串行执行时间占总执行时间的百分比

## 6 结束语

本文提出了一种用户态、运行时的数据竞争消除系统 DRFS, DRFS 将程序的执行分为并行执行和串行执行两种模式,采用合理有效的模式切换机制将两种模式结合执行,利用现有硬件支持的页保护机制,运行时消除共享内存的数据竞争. DRFS 为并行程序提供一个无数据竞争、内存安全的运行环境.今后工作可以在 DRFS 现有机制下添加记录回放功能,最终实现一种低开销的多线程确定性回放系统.

### 参考文献:

- [1] Savage S, Burrows M, Nelson G, et al. Eraser: a dynamic data race detector for multithreaded programs [J]. Journal of Frontiers of Computer Science & Technology, 2011, 5(12): 1085-1093.
- [2] WU Xueguang, WEN Yanjun, WANG Ji, et al. Data race and atomicity checking for cprograms with multiple interruptions [J]. Journal of Frontiers of Computer Science and Technology, 2011, 5(12): 1085-1093.

比前面两种算法将负载不均衡度降低了 0.5~1.2 左右,可有效降低成本的同时,同时也很好地实现了负载均衡.

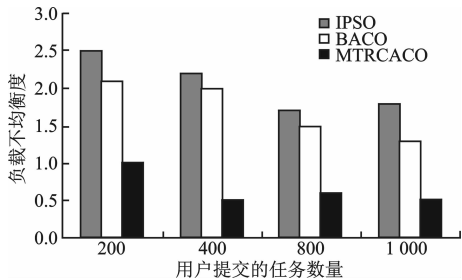


图 2 负载均衡率比较

## 7 结束语

本文提出了基于任务资源函数匹配度函数与成本函数的改进蚁群优化算法 MTRCACO,算法使得任务合理地分配到云计算系统资源,同时通过信息素因子改进系统负载不均的情况,实验结果表明,MTRCACO 算法性能明显优于 IPSO 和 ACO,有效降低了任务执行成本,提高云计算系统的负载均衡度.

## 参考文献:

[1] 魏赞,陈元元. 基于改进蚁群算法的云计算任务调度模型[J]. 计算机工程, 2015,41(1):12-16.

[2] 蔡琪,单东红,赵伟艇. 改进粒子群算法的云计算环境资源优化调度[J]. 辽宁工程技术大学学报(自然科学版),2016,35(1):93-96.

[3] 张焕青,张学平,王洲清,等. 基于负载均衡蚁群优化算法的云计算任务调度[J]. 微电子学与计算机,2015,32

(5):31-40.

[4] Ramesh D, Krishnan A. Optimal parameter identification in ant colony optimization for load balancing in grid computing [J]. European Journal of Scientific Research,2012,75 (3):370-376.

[5] Zheng Z N, Wang R. An approach for cloud resource scheduling based on parallel genetic algorithm [C] // In Proc Of 3 rd International Conference on Computer Research and Development. Shanghai: IEEE, 2011: 444-447.

[6] Brototi Mondal, KOUSIK Dasgupta, Paramartha Dutta, Load Balancing in Cloud Computing using Stochastic Hill Climbing-A Soft computing Approach [C] // International Conference on Computer Communication, Control and Information Tedsnology. [ s. l. ]: IEEE, 2012:783-789.

[7] Cristian Mateos, Elina Pacini, Carlos Garcia garion. An ACO-inspired algorithm for minimizing weighted flow time in cloud-based parameter sweep experiments [J]. Advances in Engineering Software , 2013, 56 (11):38-50

[8] Zhan S B, Huo H Y. Improved PSO-based task scheduling algorithm in cloud computing [J]. Information & Computational Science, 2013,13 (9): 3821-3829.

## 作者简介:

聂清彬 男,(1982-),硕士,讲师. 研究方向为云计算与物联网. E-mail:270104318@qq. com.

霍敏霞 女,(1983-),硕士,讲师. 研究方向为软件测试.

曹耀钦 男,(1962-),博士,教授,博士生导师. 研究方向为计算机网络.

(上接第 92 页)

[3] Savage S,Burrows Michael,Nelson Greg,et al. Eraser: a dynamic data race detector for multithreadel programs[J]. Acm Sigops Operation Systems Review, 1997,15(4):391-394.

[4] Prodromos Gerakios, Papaspyrou N,Sogonas K, et al. Race-free and Memory-safe Multithreading:Design and Implementation in Cyclone[R]. TLDP Int. 2010.

[5] Amdahl GM. Validity of the single-processor approach to achieving large scale computing capabilities[C] // Proceedings of the American Federationof Information Processing Societies. New York, N Y, USA: ACM,

1967;483-485.

## 作者简介:

沙 晗 女,(1991-),硕士研究生. 研究方向为并行程序、动态编译. E-mail:18813148088@163. com.

应 欢 女,(1988-),博士研究生. 研究方向为并行程序、动态编译.

唐博文 男,(1991-),硕士研究生. 研究方向为动态编译、程序分析、并行程序.

乔 扬 男,(1993-),硕士研究生. 研究方向为软件安全.