基于锁集合的动态数据竞争检测方法

章隆兵[1)] 张福新[1)] 吴少刚[1)] 陈意云[2)]

[1)](中国科学院计算技术研究所 北京 100080)

[2)](中国科学技术大学计算机科学技术系 合肥 230027)

摘 要 数据竞争使得共享存储程序难于调试.以前大部分针对共享存储程序的动态数据竞争检测工作都是通过维护发生序来实现.这种方法有一个重要缺点,即针对程序的一种输入,对程序的一次执行进行检测,不能检测出所有的可行数据竞争.文中利用存储一致性模型的框架模型,针对域一致性模型提出了增强发生序概念,并依此得出一种基于锁集合的动态数据竞争检测算法,克服了这个问题.在软件 DSM 系统 JIAJIA 上的实现获得了很好的性能,应用平均减速比为 3.14.利用该方法,在 TSP 程序中找到了大量的读写数据竞争的情况.

关键词 软件分布式共享存储系统;域一致性模型;数据竞争;增强发生序;JIAJIA

中图法分类号 TP302

A Lockset-Based Dynamic Data Race Detection Approach

ZHANG Long-Bing[1)] ZHANG Fu-Xin[1)] WU Shao-Gang[1)] CHEN Yi-Yun[2)]

[1)](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100080)

[2)](*Department of Computer Science & Technology, University of Science & Technology of China, Hefei* 230027)

Abstract Shared-Memory parallel program is difficult to debug because of data races. Most of the previous work in dynamic data race detection is based on Happen-Before Order. The tools based on Happen-before Order have a significant drawback that for an input set, a detection of the execution of a program can miss some feasible data races. This article describes a lockset-based dynamic data race detection technique to solve this problem. Our dynamic data race detection is targeted on a software distributed shared memory(DSM) system called JIAJIA which supports Scope Consistency. By using the frame model of memory consistency model, Enhanced-Happen-Before Order for Scope Consistency has been defined and in further step, a lockset-based dynamic data race algorithm is suggested. This article describes the implementation of lockset-based algorithm in JIAJIA and presents an experimental evaluation of lockset-based technique by looking for data races in five common shared-memory programs. In TSP program, some read-write data races have been correctly found. The slowdown of this lockset-based technique averages less that 3.14 for our applications, which means that our lockset-based technique has gotten good performance.

Keywords software distributed shared memory system; scope consistency; data race; enhanced-happen-before order; JIAJIA

收稿日期:2002-01-09;修改稿收到日期:2003-01-09.本课题得到国家自然科学基金项目(60073018,69896250)、中国科学院全国首届优秀博士学位论文作者专项基金及中国科学院计算技术研究所领域前沿青年基金(20016280-6)资助.**章隆兵**,男,1974 年生,博士,现为中国科学院计算技术研究所博士后,主要研究领域为计算机系统结构、机群计算、分布式共享存储系统. E-mail: lbzhang@ict.ac.cn. **张福新**,男,1976 年生,博士研究生,主要研究领域为微处理器设计和计算机系统结构. **吴少刚**,男,1973 年生,博士研究生,主要研究领域为计算机体系结构、共享虚拟存储系统、并行计算. **陈意云**,男,1946 年生,教授,博士生导师,主要研究领域为形式描述技术、程序设计语言的设计与实现、软件系统结构、软件安全.

1 引 言

近来,随着高速处理器和高速网络技术的发展,软件分布式共享存储 DSM 系统(Distributed Shared Memory System)的性能有了很大提高,获得了与消息传递系统 MPI/PVM 可比的性能.在这种情况下,制约软件 DSM 系统广泛应用的主要因素已经不再是性能了,系统的调试环境也是一个重要方面.

基于共享存储的并行程序可能会出现数据竞争情况.直观地说,数据竞争是指对一个共享变量的两个操作(其中至少有一个是写操作),没有被同步语句隔开.这种情况通常是由于错误的同步造成的,比如缺少必要的同步语句等.在共享存储程序中,这样的错误会使程序运行具有不确定性,因此很难调试.在软件 DSM 系统中实现动态数据竞争检测,可以在程序执行时动态检测出数据竞争,方便用户调试程序.

本文针对域一致性模型,提出了一种基于锁集合的动态数据竞争检测算法,并将该算法实现在软件 DSM 系统 JIAJIA 上.以前大部分动态数据竞争检测方面的工作[1~3]都是通过维护发生序(Happen-Before Order)来实现的.这种方法的主要缺点是:对于程序的一个输入,在一次执行中不能检测出所有数据竞争情况.本文工作与以前工作的显著不同点:首先是针对域一致性模型,其次在参考文献[4]中提出的存储一致性模型的框架模型基础上,通过定义增强发生序(Enhanced-Happen-Before Order),提出一种基于锁集合的数据竞争检测算法,克服了维护发生序方法的主要缺点.我们采用一些广泛使用的基准测试程序进行了测试,获得了很好的性能,平均减速比为 3.14.另外,我们使用基于锁集合的数据竞争检测技术,在 TSP 程序中找到许多读写数据竞争的情况.

本文第 2 节介绍域一致性模型;第 3 节介绍针对域一致性模型的基于锁集合的数据竞争检测算法;第 4 节描述了该算法在软件 DSM 系统 JIAJIA 上的实现;第 5 节中描述了性能测试结果;第 6 节描述了相关工作;第 7 节总结全文.

2 域一致性模型

为了提高共享存储系统的性能,通过逐渐放松访存事件次序,研究人员提出了许多弱存储一致性模型,包括释放一致性模型 RC(Release Consistency)[5]、懒惰释放一致性模型 LRC(Lazy Release Consistency)[6]、域一致性模型 ScC(Scope Consistency)[7]等.ScC 模型对访存事件次序的要求比 LRC 模型更宽松.在 LRC 中,当处理机 P 从处理机 Q 获得锁 l 时,Q 所看到(visible)的修改操作都被传给 P.但在 ScC 模型中,只有用锁 l 保护起来的区域中所做的修改才会传送给 P.本文采用文献[4]中提出的存储一致性模型的框架模型来描述 ScC 模型.

定义 1. 一个存储一致性模型 M 是一个二元组$\langle C_M, SYN_M\rangle$,其中 C_M是对访存操作的一个分类(或对同步操作的一种描述),SYN_M是确定处理机间访存操作执行次序的一种进程间的同步机制.

在共享存储程序中,如果两个访存操作访问的是同一个存储单元且其中至少有一个存数操作,则称这两个访存操作是冲突的,构成冲突访问对.共享存储系统中并行程序的执行结果由程序中冲突访问的执行次序决定,而冲突访问的执行次序又是由同步操作的次序决定的,因此可以把同步操作的执行次序定义为并行程序的一个执行.

定义 2. 程序 PRG 在存储一致性模型 M 中的一个执行,记为 $E_M(PRG)$,是该程序中同步操作的一个定序.

定义 3. 在存储一致性模型 M 中的一个执行 $E_M(PRG)$的同步序,记为 $SO_M(E_M(PRG))$,是在该执行中被 M 的同步机制 SYN_M所定序的普通访存操作对的集合.即 $SO_M(E_M(PRG))=\{(u,v)\mid u$ 在 $E_M(PRG)$中被 SYN_M定序在 v 之前执行$\}$.

存储一致性模型的本质是一种确定处理机间访存操作次序的同步机制,因此一个存储一致性模型就可以用定义 3 描述的同步序来描述.下面,我们用"$\xrightarrow{PO}$"来表示程序序(即指令在程序中出现的先后次序),将域一致性模型表示如下:

$$\begin{cases} C_{ScC}=\{r(x),w(x),acq(l),rel(l)\}, \\ SO_{ScC}(E(PRG))=\{(u_i,v_j)\mid acq_i(l)\xrightarrow{PO}u_i \\ \qquad\qquad \xrightarrow{PO}rel_i(l)\xrightarrow{E}acq_j(l)\xrightarrow{PO}v_j\}^+. \end{cases}$$

从这个描述可见,域一致性模型将访存操作分为读操作和写操作,同步操作分为获得锁操作(acq)和释放锁操作(rel);并且如果 u_i被"$rel_i(l)\xrightarrow{E}acq_j(l)$"定序在 v_j之前执行,则 u_i必须在锁 l 所保护的临界区内,否则 u_i和 v_j就不能被定序.直观地说,在处理机 j 从处理机 i 获得锁 l 时,只有在锁 l 的临界区内所

做的访存操作才对处理机 j 可见,即修改的值才传到处理机 j 上.

3　域一致性模型上的基于锁集合的算法

3.1　数据竞争定义

为了定义数据竞争,我们先来看一下并行执行发生序的定义.对于共享存储程序的一次并行执行,所有冲突访问对的执行次序就决定了并行执行的结果.在程序的一个并行执行中,进程内部的冲突访问对由程序序定序,进程间的冲突访问对由同步序来定序,因此把程序序和同步序的并集称为该并行执行的发生序(Happen-Before Order,HB),用来描述程序的一次并行执行.HB 序具体定义如下.

定义 4. 若 $E_{ScC}(PRG)$是程序 PRG 在 ScC 一致性模型中的一个执行,该执行的同步序与程序序的并集称为该执行的发生序(Happen-Before Order),记为 $HB_{ScC}(E_{ScC}(PRG))$,即 $HB_{ScC}(E_{ScC}(PRG))=PO(PRG)\cup SO_{ScC}(E_{ScC}(PRG))$,其中程序序 $PO(PRG)$是指令在程序中出现的先后次序.

参照文献[8]中的术语,我们给出可行数据竞争定义.并且在本文后面的部分,对术语可行数据竞争和数据竞争不作区分.

定义 5.　在程序的任一执行 $E_{ScC}(PRG)$中,如果冲突访问对(u,v)没有被 $HB_{ScC}(E_{ScC}(PRG))$定序,则称(u,v)构成可行数据竞争(feasible data race).

根据可行数据竞争的定义,一种直观地检测数据竞争的方法是维护 HB 序的方法,该方法通过在程序执行时,动态检查冲突访问对是否违反 HB 关系来实现数据竞争检测.下面,我们来看一个例子.

图 1 中的程序有两种可能的执行:$E^0(PRG)=rel_0(L)\xrightarrow{E}acq_1(L)$和 $E^1(PRG)=rel_1(L)\xrightarrow{E}acq_0(L)$.当输入 $flag=0$ 时,对于执行 $E^0(PRG)$,冲突访问对$(W_0(x),R_2(x))$,$(W_0(x),R_3(x))$被 $HB(E^0(PRG))$定序;对于 $E^1(PRG)$,冲突访问对$(W_0(x),R_2(x))$被 $HB(E^1(PRG))$定序,而$(W_0(x),R_3(x))$却不被 $HB(E^1(PRG))$定序.故可知$(W_0(x),R_3(x))$构成可行数据竞争.但是采用维护 HB 序方法,在对执行$E^0(PRG)$进行检测时,就不能检测出$(W_0(x),R_3(x))$构成可行数据竞争.当然对执行 $E^1(PRG)$进行检测时,该方法能检测出$(W_0(x),R_3(x))$构成可行数据竞争.

P_0	P_1
$acq_0(L)$	if($flag$) { $R_1(x)$ }
$W_0(x)$	$acq_1(L)$
$rel_0(L)$	$R_2(x)$
	$rel_1(L)$
	$R_3(x)$

图 1　一个共享存储的并行程序

从上面例子中可以看出维护 HB 序方法的缺点:对于一个程序输入,如果对一次执行进行检测没有找到数据竞争,并不能判定在该输入下就没有数据竞争.本文将通过定义增强发生序,提出基于锁集合的算法来解决这个问题.

另外,图 1 程序中$(W_0(x),R_1(x))$明显构成可行数据竞争.但是当输入 $flag=0$ 时,利用动态数据竞争检测方法根本无法检测出来.这实际上是动态检测方法本身的一个弊端,要想对程序的数据竞争做彻底的检查,只能采用静态方法在编译时进行,但这已被证明是 NP 完全问题.本文不做详细讨论.

3.2　增强发生序

参照存储一致性的框架模型中对同步关系的定义,我们为 ScC 模型定义一种增强同步序(Enhanced-Synchronization Order).定义如下.

定义 6. 在 ScC 模型中的一个执行 $E_{ScC}(PRG)$的增强同步序(Enhanced-Synchronization Order),记为 $ESO_{ScC}(E_{ScC}(PRG))$,表示为

$$ESO_{ScC}(E_{ScC}(PRG))=\{(u_i,v_j)\mid acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)\xrightarrow{E}acq_j(l)\xrightarrow{PO}v_j\xrightarrow{PO}rel_j(l)\}^+$$

$ESO_{ScC}(E_{ScC}(PRG))$定义的要点是,只有被相同锁保护的临界区内的访存操作,才能被 ESO_{ScC}定序.这一点不同于 $SO_{ScC}(E_{ScC}(PRG))$.在 $SO_{ScC}(E_{ScC}(PRG))$定义中,两个访存操作被 $SO_{ScC}(E_{ScC}(PRG))$定序,不一定要在相同锁保护的临界区内.利用增强同步序,我们可以进一步定义增强发生序.

定义 7.　若 $E_{ScC}(PRG)$是程序 PRG 在 ScC 模型中的一个执行,该执行的增强同步序与程序序的并集称为该执行的增强发生序(Enhanced-Happen-Before order),记为 $EHB_{ScC}(E_{ScC}(PRG))$,即 $EHB_{ScC}(E_{ScC}(PRG))=PO(PRG)\cup ESO_{ScC}(E_{ScC}(PRG))$.

根据增强发生序,我们可以得出下面定理.

定理 1.　程序 PRG 在 ScC 模型上的任一个执行 $E_{ScC}(PRG)$中,如果冲突访问对(u_i,v_j)不被 $EHB(E_{ScC}(PRG))$定序,则该冲突访问对构成可行数据竞争.

证明. 如果冲突访问对(u_i,v_j)由程序序 $PO(PRG)$定序,则根据 $EHB_{ScC}(E_{ScC}(PRG))$定义,(u_i,v_j)必

定被 $EHB_{ScC}(E_{ScC}(PRG))$ 定序，故 (u_i,v_j) 必定不由 $PO(PRG)$ 定序. 下面分情况讨论：

(1) 假设冲突访问对 (u_i,v_j) 被 $HB((E_{ScC}(PRG))$ 定序. 对于这种情况，则 (u_i,v_j) 必定被 $SO(E_{ScC}(PRG))$ 定序. 但又由于 (u_i,v_j) 不被 $EHB(E_{ScC}(PRG))$ 定序，则必定不被 $ESO(E_{ScC}(PRG))$ 定序. 故必定是下面两种情况中的一种：$acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_j(l)\xrightarrow{PO}rel_j(l)\xrightarrow{PO}v_j$，或者 $acq_j(l)\xrightarrow{PO}v_j\xrightarrow{PO}rel_j(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_i(l)\xrightarrow{PO}rel_i(l)\xrightarrow{PO}u_i$. 假设是前一种情况，根据获得锁的规律，则必存在一个执行 $E^1_{ScC}(PRG)$ 使得 $acq_j(l)\xrightarrow{PO}rel_j(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)$. 显然对于执行 $E^1_{ScC}(PRG)$，(u_i,v_j) 不被执行 $SO(E^1_{ScC}(PRG))$ 定序，也即 (u_i,v_j) 不被 $EHB(E^1_{ScC}(PRG))$ 定序. 故冲突访问对 (u_i,v_j) 构成可行数据竞争. 后一种情况同理可证.

(2) 假设冲突访问对 (u_i,v_j) 不被 $HB(E_{ScC}(PRG))$ 定序，则 (u_i,v_j) 显然构成可行数据竞争.

综上可知，定理成立. 证毕.

根据以上定理，针对程序任一个执行 E，如果找出冲突访问对 (u_i,v_j) 不被 $EHB(E)$ 定序，就可以认为 (u_i,v_j) 构成可行数据竞争.

推论 1. 程序 PRG 在 ScC 模型上的任一执行 $E_{ScC}(PRG)$ 中，如果冲突访问对 (u_i,v_j) 不被 $PO(PRG)$ 定序，并且不是由相同锁保护，当且仅当 (u_i,v_j) 构成可行数据竞争.

证明. 从两个方向证明：(1)在一次执行 $E_{ScC}(PRG)$ 中，如果 u_i,v_j 不是由相同锁保护，则必定不会出现以下情况：$acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_j(l)\xrightarrow{PO}v_j\xrightarrow{PO}rel_j(l)$ 或者 $acq_j(l)\xrightarrow{PO}v_j\xrightarrow{PO}rel_j(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)$，即 (u_i,v_j) 不被 $ESO(E_{ScC}(PRG))$ 定序；又知 (u_i,v_j) 不被 $PO(PRG)$ 定序，则必有 (u_i,v_j) 不被 $EHB(E_{ScC}(PRG))$ 定序，由定理 1，可知冲突访问对 (u_i,v_j) 必构成可行数据竞争. (2)当冲突访问对 (u_i,v_j) 构成可行数据竞争，则 (u_i,v_j) 必定不被 $PO(PRG)$ 定序，并且假设 (u_i,v_j) 被相同锁保护，则对于任一次程序执行，必是下面两种情况中的一种：$acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_j(l)\xrightarrow{PO}v_j\xrightarrow{PO}rel_j(l)$ 或者 $acq_j(l)\xrightarrow{PO}v_j\xrightarrow{PO}rel_j(l)\xrightarrow{E}\cdots\xrightarrow{E}acq_i(l)\xrightarrow{PO}u_i\xrightarrow{PO}rel_i(l)$. 由可行数据竞争定义，不管哪种情况，$(u_i,v_j)$ 都不构成可行数据竞争. 故可知 (u_i,v_j) 必定不被相同锁保护.

综上可知，推论成立. 证毕.

我们基于锁集合的数据竞争检测算法的主要思想就是检测在一次执行中，由不同进程发出的、没有被相同锁保护的冲突访问对. 根据推论 1，对于一个给定程序输入的一次执行，找到所有满足这种条件的冲突访问对就找到了程序中所有可行数据竞争. 在下一部分中，我们将在软件 DSM 系统环境下描述基于锁集合的算法.

3.3 基于锁集合的算法

在软件 DSM 系统中，通常将每个进程的执行划分成区间(Interval). 每个区间有一个序列号. 通常，进程维持一个当前序列号，当进程每次执行一个 release 或者 acquire 操作时，就会产生一个新的区间，并且将进程的当前序列号增 1，赋给新产生的区间. 图 2 即为区间的例子，描述了两个进程的执行，每个进程有两个区间.

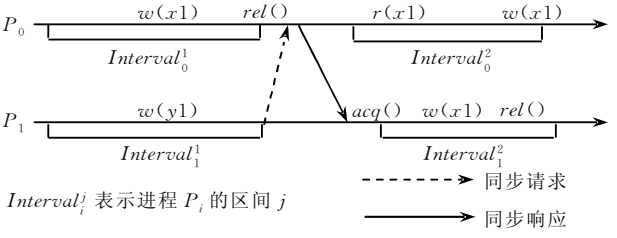

图 2 一个区间的例子

每个区间 $Interval$ 有 3 个集合：$Read(Interval)$，$Write(Interval)$ 和 $Lockset(Interval)$. $Read(Interval)$ 表示区间中所读变量的集合；$Write(Interval)$ 表示区间中所写变量的集合. $Lockset(Interval)$ 是程序执行时保护该区间的锁集合. 在本文中，我们将不被相同锁保护，且来自于不同进程的区间称为并发区间，因为这两个区间可能会同时执行. 由推论 1 可知只要找到在并发区间中发出的所有冲突访问对，就找到了所有可行数据竞争. 下面即为检查两个进程间是否存在数据竞争的基于锁集合算法的伪代码描述：

假设 $intervallist0$，$intervallist1$ 分别是 P_0，P_1 的区间列表；$get\_interval()$ 函数从区间列表中取出一个区间；$datarace\_warn()$ 表示发现数据竞争，发出报警信息.

```
interval0=get_interval(intervallist0);
while(interval0! =NULL) {
```

```
    interval1=get_interval(intervallist1);
    while(interval1! =NULL) {
      if (lockset(interval0)⋂lockset(interval1)
          =={}) {
        if ((Read(interval10)⋂Write(interval1)
            !={})||(Write(interval0)⋂
            Write(interval1)!={})||
          (Write(interval0)⋂Read(interval1)
              !={})) {
            datarace_warn();}}}
    interval0=get_interval(intervallist0);
}
```

我们还是以图 1 中的程序为例来说明基于锁集合的数据竞争检测算法. 假设程序输入中 $flag=1$,且程序执行是 $rel_0(L)\xrightarrow{E}acq_1(L)$,则进程 P_0 有一个区间,记为 $Interval_0^1$;P_1 有 3 个区间,记为 $Interval_1^1$,$Interval_1^2$,$Interval_1^3$. P_0 的区间信息为:$Lockset(Interval_0^1)=\{L\}$,$Read(Interval_0^1)=\{\}$,$Write(Interval_0^1)=\{x\}$. P_1 的区间信息为:$Lockset(Interval_1^1)=\{\}$,$Read(Interval_1^1)=\{x\}$,$Write(Interval_1^1)=\{\}$;$Lockset(Interval_1^2)=\{L\}$,$Read(Interval_1^2)=\{x\}$,$Write(Interval_1^2)=\{\}$;$Lockset(Interval_1^3)=\{\}$,$Read(Interval_1^3)=\{x\}$,$Write(Interval_1^3)=\{\}$.

执行算法如下:

1. $Lockset(Interval_0^1)\cap Lockset(Interval_1^1)=\{\}$,可见 $Interval_0^1$ 和 $Interval_1^1$ 是并发区间,故需要进行读写集合的比较:$Read(Interval_0^1)\cap Write(Interval_1^1)=\{\}$,$Write(Interval_0^1)\cap Read(Interval_1^1)=\{x\}$,$Write(Interval_0^1)\cap Write(Interval_1^1)=\{\}$. 可知找到对于共享变量 x 的冲突访问对 $(W_0(x),R_1(x))$构成数据竞争.

2. $Lockset(Interval_0^1)\cap Lockset(Interval_1^2)=\{L\}$,可见 $Interval_0^1$ 和 $Interval_1^2$ 不是并发区间,无需进行比较.

3. $Lockset(Interval_0^1)\cap Lockset(Interval_1^3)=\{\}$,可见 $Interval_0^1$ 和 $Interval_1^3$ 是并发区间,故需要进行读写集合的比较:$Read(Interval_0^1)\cap Write(Interval_1^3)=\{\}$,$Write(Interval_0^1)\cap Read(Interval_1^3)=\{x\}$,$Write(Interval_0^1)\cap Write(Interval_1^3)=\{\}$. 可知找到对于共享变量 x 的冲突访问对 $(W_0(x),R_3(x))$构成数据竞争.

同样的,对于输入中 $flag=1$ 且执行为 $rel_1(L)\xrightarrow{E}acq_0(L)$ 的情况,基于锁集合的算法也能找出 $(W_0(x),R_1(x))$,$(W_0(x),R_3(x))$构成可行数据竞争.

从上面例子可以看出,基于锁集合的方法克服了维护 HB 序方法的缺点:在输入中 $flag=1$ 的情况下,对程序的一次执行进行检测就能找到程序中所有可行数据竞争.

4　JIAJIA 系统上的实现

JIAJIA 系统是由中国科学院计算技术研究所胡伟武等人[9]开发的一个基于 Home 的软件 DSM 系统,它实现了一种基于锁的新型 Cache 一致性协议. 该协议支持域一致性模型和写无效的传播策略,并采用多写技术来避免假共享. JIAJIA 系统主要提供两种类型的同步:(1)锁(Lock):有两个关于锁的函数 $jia_lock(lockid)$和 $jia\_unlock(lockid)$. 进程调用 $jia_lock(lockid)$请求获得锁 $lockid$,调用 $jia_unlock(lockid)$释放锁 $lockid$. 一个锁变量一次只能被一个进程获得,其它的进程试图获得该锁时就需要等待,一直等到该锁被释放时才能获得. (2)路障(Barrier):实现路障的函数是 $jia\_barrier()$. 只有当所有进程到达路障后,进程才能继续往下执行,否则任何进程不能越过路障.

我们在 JIAJIA 系统上实现了基于锁集合的数据竞争检测方法,其基本思想是通过用户提供指导来获得每个区间读写共享变量的集合;通过扩展 JIAJIA 系统的同步协议实现基于锁集合的数据竞争检测算法. 下面部分,我们将会详细说明.

4.1　用户指导获得区间的读写集合

JIAJIA 系统是一个基于页的系统,其进行读写检测的单位是页,而我们进行数据竞争检测的单位是 4 个字节. 为了捕捉对共享变量的读写操作,我们为每个共享页分别维护一个读位图和写位图. 当对该共享页中的某个变量进行读写操作时,就将位图中相应位置为 1. 为了获得每个区间读写的变量集合,我们采用用户指导的方法,即由系统向用户提供一组接口,用户利用这些接口向系统提供信息. 为了简单起见,我们主要提供两个接口:(1)$direct\_read(addr,length)$:表明将要读$[addr,addr+length]$的共享变量;(2)$direct\_write(addr,length)$:表明将要写$[addr,addr+length]$的共享变量. 在实现 $direct_read(addr,length)$和 $direct\_write(addr,length)$ 时,根据参数 $addr$ 和 $length$,就可以计算出要存取的共享变量的所在的页地址,找到相应的位图,并将相应的位置为 1.

4.2　算法实现

系统中每个进程维护了一个区间列表(interval list). 每个区间的数据结构中记录在该区间内的所有读写的共享页集合及其相应页的读写位图,以及

保护该区间的锁集合.根据可行数据竞争的定义,只有在两个 Barrier 之间的区间中的冲突访问对,才有可能构成可行数据竞争.因此可以在执行 Barrier 时,由 Barrier 管理者收集系统中从上一个 Barrier 到当前 Barrier 之间的所有区间的信息,执行基于锁集合的数据竞争检测算法.当 Barrier 管理者执行检测算法之后,系统中所有区间信息都清除掉,准备记录 Barrier 执行完之后的新的区间信息.

为了实现以上的思想,我们扩展了 JIAJIA 系统的 *jia_lock*(),*jia_unlock*(),*jia_barrier*()函数.由于执行 *jia_lock*()和 *jia_unlock*()时,系统结束一个区间,同时也开始一个新区间,因此在执行 *jia_lock*()和 *jia_unlock*()时,需要将结束区间的相应信息保存到进程的区间列表中.而对于 *jia_barrier*(),我们加入了一个数据竞争检测阶段,以实现基于锁集合的算法.该阶段描述如下:(1)Barrier 管理进程接收系统中所有进程的区间列表;(2)将进程的区间列表进行两两比较,找出所有不被相同锁保护的并发区间对;(3)针对每个并发区间对,比较这两个区间的读写页集合,如果找到两个并发区间内同时都读写的页,则再比较相应页的读写位图.如果读写位图中存在对相同位的读写,则说明存在数据竞争,打印出相应的进程号、区间号及发生数据竞争的地址.

5 性能测试

我们的测试环境是以 100Mbps 交换以太网连接的四台机器,每个机器包括两个 P3 733MHz CPU,内存 1GB.采用的测试程序包括来自 SPLASH2 中的天体多体问题 Barnes、模拟海洋变化的 Ocean.美国 Rice 大学提供的旅行商问题 TSP 和逐次超松驰法程序 SOR 以及矩阵相乘程序 MM.在测试中,所有的库和应用程序都用 gcc-O2 编译.表 1 中描述了应用程序的输入和运行时的特征以及各种开销.其中,"共享内存"是指程序运行时实际分配的虚拟共享内存大小."Barrier 数"是程序执行的 Barrier 数目."Lock 数"就是程序运行时请求/获得锁的次数."正常执行时间"是指未加入数据竞争代码时,程序执行时间."执行时间"是指加入数据竞争代码后程序的执行时间."减速比"为程序执行时间与程序正常执行时间的比值,该比值反映了数据竞争方法的性能.

表 1 测试程序和测试结果

测试程序	规模	共享内存	Barrier 数	Lock 数	正常执行时间(s)	执行时间(s)	减速比
SOR	2048×2048	48MB	200	0	7.18	55.67	7.75
Ocean	1026×1026	226MB	380	0	349.99	789.24	2.26
MM	512×512	12MB	2	0	4.36	15.68	3.60
Barnes	16384 体	1544KB	12	32	34.90	37.06	1.06
TSP	19 个城市	792KB	3	679	7.09	7.32	1.03

从表 1 中可以看出,对于存取共享数据较多的程序,例如 SOR,MM,Ocean,由于调用了较多的获取共享变量读写信息的指导函数,使得性能较差.而对于共享数据存取很少的程序,例如 Barnes,TSP,由于调用数据存取的指导函数开销较少,因此数据竞争检测开销几乎可以忽略.总的来说,针对测试的五个程序,平均减速比达到了 3.14.这表明我们的方法获得了很好的性能,但是对于频繁访问共享变量的程序,性能需要进一步提高.

6 相关工作

在动态数据竞争检测方面做的工作很多[1~3,10],与本文工作最接近的是文献[1,2].文献[1,2]在实现 LRC 模型的软件 DSM 系统 CVM 上,实现了动态数据竞争检测,其基本思想是利用 Lamport 时间戳来维护 HB 序,对于一次执行中不满足 HB 序的冲突访问对就构成数据竞争.文献[3]也是通过维护 HB 序,采用与执行重放相结合的方法来实现对共享内存的多线程程序的动态数据竞争检测,但这种方法不支持多线程程序的分布式执行.与维护 HB 序的方法相比,本文方法有一个明显的优点.假定一个程序有 n 种可能的输入,对于输入 i ($1\leq i\leq n$),有 m_i 种可能的执行.那么对于维护 HB 序的方法,需要检查 $m_1+m_2+\cdots+m_n$ 个执行,而本文方法,只需要检查 n 个执行.文献[10]中实现对共享内存的多线程程序的数据竞争检测,同样不支持程序的分布式执行.文献[10]通过检查程序执行时是否违反一致的锁编程规律(locking discipline)来实现动态数据竞争检测.这种方法的好处与本文方法类似,即需要检测的执行数目少.但这种方法的开销很大,大部分应用的减速比在 10~30 之间.

7 结　论

本文介绍了在软件 DSM 系统 JIAJIA 上实现一种基于锁集合的动态数据竞争检测方法.我们的工作区别于以前工作的显著特点是针对域一致性模型,并且在存储一致性模型的框架模型基础上,通过定义增强发生序,得出一种基于锁集合的动态数据竞争检测算法,从理论上保证了数据竞争检测的正确性.利用本文中提出的方法,我们在 TSP 程序中找到许多读写数据竞争.但这是由于 TSP 中采用的提高程序并行性的编程技巧造成的,并不影响程序正确性.性能测试的结果表明,在进行数据竞争检测后,应用的平均减速比达到了 3.14.这表明对于大部分应用而言,我们的方法获得了很好的性能,基本上达到了实用水平.本文方法的不足之处是采用用户指导,增加了用户负担.我们未来计划采用代码插入技术来改进 JIAJIA 上的实现,使得无需用户指导,易于用户使用.

参 考 文 献

1 Perkovic D,Keleher P. On-line data race detection via coherency guarantees. In:Proceedings of OSDI'96,Seattle,Washington,1996.47～58

2 Perkovic D,Keleher P. A protocol-centric approach to on-the-fly race detection. IEEE Transactions on Parallel and Distributed Systems,2000,11(10):1058～1072

3 Ronsse M,Bosschere K. RecPlay:A fully integrated practical record/replay system. ACM Transactions on Computer Systems,1999,17(2):133～152

4 Hu W. Shared Memory Architecture. Beijing:Higher Education Press,2001(in Chinese)
(胡伟武. 共享存储系统结构. 北京:高等教育出版社,2001)

5 Gharachorloo K,Lenoski D,*et al*. Memory consistency and event ordering in scalable shared-memory multiprocessors. In:Proceedings of the 17th Annual International Symposium on Computer Architecture,Los Alamitos,California,1990.15～26

6 Keleher P,Cox A,Zwaenepoel W. Lazy release consistency for software distributed shared memory. In:Proceedings of ICSA'92,Gold Coast,Australia,1992.13～21

7 Iftode L,Singh J,Li K. Scope consistency:A bridge between release consistency and entry consistency. In:Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, Padua,Italy,1996.277～287

8 Netzer R,Miller B. What are race conditions?. ACM Letters on Programming Languages and Systems,1992,1(1):74～88

9 Hu W,Shi W,Tang Z. A software DSM system based on a new cache coherence protocol. Chinese Journal of Computers,1999,20(5):467～474(in Chinese)
(胡伟武,施巍松,唐志敏. 基于新型 Cache 一致性协议的共享虚拟存储系统. 计算机学报. 1999,20(5):467～474)

10 Savage S,Burrows M,Nelson G,Sobalvarro P,Anderson T. Eraser:A dynamic data race detector for multithreaded programs. ACM Transactions on Computer System,1997,15(4):391～411

ZHANG Long-Bing, born in 1974, Ph. D., he is now a Postdoctoral Fellow of Institute of Computing Technology, Chinese Academy of Sciences. His research interests include system architecture, cluster computing, software distributed shared memory system.

ZHANG Fu-Xin, born in 1976, Ph. D. candidate. His research interests include microprocessor design and system architecture.

WU Shao-Gang, born in 1973, Ph. D. candidate. His research interests include system architecture, software distributed shared memory system, parallel computing.

CHEN Yi-Yun, born in 1946, professor and Ph. D. supervisor. His research interests include applications of logic (including formal semantics and type theory), techniques for designing and implementing programming languages, formal specification, software security.