Journal of Computer Applications
计算机应用,2019,39(1):61 -65
ISSN 1001-9081
CODEN JYIIDU
2019-01-10
http://www.joca.cn

文章编号:1001-9081(2019)01-0061-05
DOI:10.11772/j.issn.1001-9081.2018071605

并发程序中数据竞争检测方法

张　杨,梁亚楠,张冬雯*,孙仕欣
(河北科技大学 信息科学与工程学院,石家庄 050000)
(* 通信作者电子邮箱 zdwwtx@163.com)

摘　要: 针对数据竞争检测过程中的误报和漏报问题,提出一种静态数据竞争检测方法。首先,使用控制流分析自动构造线程内和线程间函数调用图;然后,收集线程内变量访问事件信息,定义竞争产生条件并分析检测出所有可能的竞争;其次,为了提高检测的准确率,进行别名变量和别名锁的分析降低漏报和误报;最后,通过控制流分析来抽象访问事件之间的时序关系,并结合程序切片技术对访问事件的发生序关系进行判断,以此避免因忽略线程交互带来的误报。依据该方法,使用 Java 语言在 Soot 软件分析框架下实现了一个数据竞争检测工具。在实验中,对 JGF 和 IBM Contest 基准测试套件中的 raytracer 和 airline 等程序进行数据竞争检测,并与目前已有的数据竞争检测算法和工具(HB 算法和 RVPredict)进行对比。实验结果表明,与 HB 算法和 RVPredict 工具相比,该方法检测到的数据竞争总数分别增加了 81% 和 16%,数据竞争检测的准确率分别提升了约 14% 和 19%,有效地避免了数据竞争检测中的漏报和误报现象。

关键词: 并发程序;数据竞争;控制流分析;别名分析;程序切片
中图分类号: TP311.53　　**文献标志码:** A

Data race detection approach in concurrent programs

ZHANG Yang, LIANG Yanan, ZHANG Dongwen*, SUN Shixin
(*School of Information Science and Engineering, Hebei University of Science and Technology, Shijiazhuang Hebei* 050000, *China*)

Abstract: Aiming at the problems of false positive and false negatives in data race detection, a novel static data race detection approach was proposed. Firstly, intra-thread and inter-thread function call graphs were automatically constructed via control flow analysis. Secondly, the information of variable-access events within thread were collected, and possible races were detected based on the defined data race conditions. Then, in order to improve the detection accuracy, alias variables and alias locks were analyzed to reduce false negatives and false positives, respectively. Finally, the sequential relationship between access events was abstracted through control flow analysis, and program slicing was used to determine the happens-before relationship of access events, thereby reducing false positives caused by ignoring thread interactions. A data race detection tool was implemented by Java and Soot framework based on this approach. In the experimentation, several benchmarks from JGF and IBM Contest benchmark suites, such as raytracer and airline, were selected for evaluation, and the results were compared with existing data race detection algorithm and tool (HB (Happens-Before) and RVPredict). The experimental results show that, compared with algorithm HB and tool RVPredict, total number of data races detected by the proposed approach are increased by 81% and 16% respectively, the accuracy of this approach for data race detection are respectively increased by 14% and 19%, which effectively avoids false negatives and false positives.

Key words: concurrent program; data race; control flow analysis; alias analysis; program slicing

0　引言

随着多核处理器的普及和众核处理器的发展,越来越多的人开始使用并发编程来提高程序的性能。并发编程具有很多优势,它不仅可以减少程序的运行时间,而且可以提高程序的吞吐量和多核处理器的利用率。

虽然并发编程带了很多好处,但并发程序内部的并发性和不确定性仍然会导致一些难以避免的问题,包括死锁、数据竞争、原子性违背和顺序违背等,这些并发问题都有着难以检测、调试和修复的特点[1]。在这些并发问题中,数据竞争是指在多线程程序中,两个或多个线程在无时序限制情况下访问同一内存位置并且至少有一个线程执行写操作[2]。数据竞争常常是引起其他非死锁并发缺陷的根本原因,并且在所有并发缺陷中占有较大比例[3-4]。

依据检测的时机,数据竞争检测分为静态分析和动态分析两种。动态分析通过插桩来获取变量和别名的准确信息,但由于线程调度策略不同,程序的执行结果可能不同[5],这使得动态检测覆盖面不全,往往存在很多的漏报,并且检测开

收稿日期:2018-07-19;修回日期:2018-08-13;录用日期:2018-08-13。　基金项目:国家自然科学基金资助项目(61440012);河北省自然科学基金资助项目(F2016208007);河北省基础研究计划重点基础专项(18960106D)。

作者简介:张杨(1980—),男,河北秦皇岛人,副教授,博士,CCF 会员,主要研究方向:并发软件分析;　梁亚楠(1991—),女,河北邢台人,硕士研究生,主要研究方向:并发软件分析;　张冬雯(1964—),女,河北石家庄人,教授,博士,CCF 会员,主要研究方向:并发软件分析;孙仕欣(1994—),女,河北石家庄人,硕士研究生,主要研究方向:并发软件分析。

销很大。与动态分析相比,静态分析具有速度快、检测更加全面等优点,但由于静态分析的不可判定性,静态检测算法只是一种不完备的近似算法[6]。

很多国内外学者对数据竞争检测的问题进行了研究。其中动态检测主要分为三种:基于发生序关系的检测方法、基于锁集的检测方法、二者结合进行检测的方法。基于发生序关系方法中具有代表性的方法是 Djit +[7],它使用向量时钟进行数据竞争分析,FastTrack[8] 和 LOFT[9] 等都是在向量时钟基础上进行的改进。Savage 等[10] 提出基于锁集的检测工具 Eraser,通过共享变量持有的锁集情况判断竞争。ACCULOCK[11] 是第一个采用轻量级逻辑时钟平衡检测精度与覆盖率的二者混合方法。在静态检测方面,常用的检测工具包括 RacerX[12]、LOCKSMITH[13] 和 RELAY[14]。其中:RacerX 利用流敏感和过程间分析检测数据竞争和死锁;LOCKSMITH 首先使用标签流约束和抽象控制流图约束来进行锁集分析,然后展开共享变量分析,最后结合线性分析检测出数据竞争;RELAY 由于其扩展性堪称优良,能够应用在百万级别代码量的程序上,实际使用中获得了高度认可和广泛接受。虽然很多学者在数据竞争检测方面进行了相关研究,但针对竞争检测出现的误报和漏报现象有待于进一步研究与分析。

为了降低数据竞争检测的误报率和漏报率,本文提出一个面向并发程序的静态数据竞争检测方法。该方法在 Soot 软件分析框架[15] 下使用控制流分析、别名分析、时序分析、程序切片等分析技术对并发程序中的数据竞争进行检测,并开发了相应的检测工具。在实验中,将本文方法和 HB(Happens-Before)算法[16]、RVPredict[17] 工具进行了对比,实验结果表明,相对于其他两种方法,本文方法能够有效地发现并发程序中的数据竞争,可以改善检测过程中的误报和漏报问题。

1 相关工作

HB 的概念最初由 Lamport[16] 提出,Lamport 使用 HB 来定义分布式系统中事件之间的偏序关系,并且提出了一个分布式算法用于同步逻辑时钟系统,将偏序关系扩展为事件的某种全序关系。

FastTrack 是一个精确、有效的基于发生序关系的动态检测方法,它是在经典的 HB 方法 Djit + 上进行的改进。Djit + 使用向量时钟进行数据竞争分析,而 FastTrack 采用基于 epoch的轻量级逻辑时钟将 Djit + 的时间复杂度从 $O(n)$ 降到接近于 $O(1)$。

由于发生序关系对线程交错比较敏感,单纯使用这一方法会导致很多漏报。锁集算法通过判断访问操作发生时的锁集情况来进行数据竞争分析,它不敏感于线程交错,而单纯使用锁集算法会忽略其他的一些同步原语,导致很多误报,因此结合锁集算法和发生序关系的混合算法应运而生。ACCULOCK 采用二者混合的方法,使用 FastTrack 中提到的轻量级逻辑时钟来进行发生序关系分析,同时对解锁操作增加时间戳,根据锁集的时间戳去掉一些冗余的分析。由于该方法保留的是共享内存最后一次读和写相关的 epoch 和锁集,因此也存在一定的误报和漏报。

在静态检测方面,Choi 等[18] 实现了对并发程序作自动分析的静态工具。它以访问事件为中心,分别对特定路径和所有路径作别名分析,得到确定的竞争和可能竞争的对象对,但是由于它没有对线程间访问事件的发生序关系进行抽象分析(start/join 原语等),因此会导致很多误报。

RacerX 对 C 语言的竞争按模式匹配检测,它是一个静态工具,使用流敏感的过程间分析来检测竞争。RacerX 用于在大型复杂的多线程系统中,且分析速度很快,但是它不进行别名分析,只是根据经验对分析产生的竞争对按照可能性等级排列,所以准确度比较低。

吴萍等[19] 提出了一种精确、有效的对多线程程序静态检测的框架 JTool,它的分析算法将竞争问题分解为跨线程的控制流分析,应用了上下文敏感和流敏感的别名分析,并静态模拟了访问事件的时序关系以进行约束求解。

近几年,出现了很多数据竞争预测分析工具,RVPredict 采用因果预测分析方法,将抽象化的控制流信息添加到执行模型中,把竞争检测作为一个约束求解问题,利用 SMT(Satisfiability Modulo Theories)求解器来查找竞争。Liu 等[20] 利用指针分析对预测分析方法进行了改进,实现了一个竞争预测分析工具。它是第一个允许改变访问位置的预测分析工具,通过指针分析和预测分析的结合来解决访问变量依赖于访问位置的问题,并采用了混合编码方式以提高实用性。

2 竞争检测

2.1 检测框架

本文利用 Soot 分析工具对 Java 源代码进行中间转换,采用 Jimple 作为中间表示(Intermediate Representation, IR),通过控制流分析构造线程内和线程间的函数调用关系图,收集线程内的所有访问事件;通过定义数据竞争产生的条件,并依据条件收集所有可能产生竞争的访问事件对;为了提高检测的精确度,对所有的竞争访问事件进行别名分析和发生序关系分析,别名分析不仅考虑了别名变量的影响,还考虑到了别名锁带来的误报问题;在发生序关系分析中,本文采用控制流分析抽象线程内和线程间访问事件的时序关系,对访问事件进行切片分析并定义发生序关系产生的条件,完成两个事件的发生序关系判断,排除某些因线程交互造成的假竞争。竞争检测框架如图 1 所示。

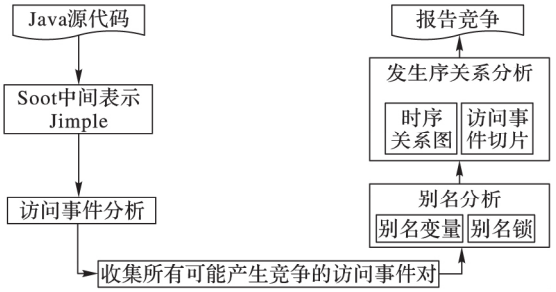

图1 数据竞争检测框架

Fig. 1 Framework of data race detection

2.2 Soot 分析

本文提出的框架使用 Soot 软件分析工具辅助完成。Soot 框架提供了一组用于分析和变换的中间表示 Jimple,它是一个紧凑、无栈、类型化的三地址代码中间表示法。在 Jimple 的基础上,Soot 不仅提供了一系列类和方法用于源程序的分析,还提供了以 Pack 为中心的扩展机制,一个 Pack 包括若干个变换,用户可以自行设计新的变换,将其加入到 Soot 的调度执行过程中以实现特定的功能。例如,本文利用 Soot 工具提供的类 ReachableMethods 和 TransitiveTargets 中的若干方法获取线程的所有直接和间接调用函数,并自定义方法排除调用

函数中无变量访问操作以及那些属于 Java 开发工具包(Java Development Kit, JDK) 的函数,最后将本文的函数调用图分析作为一个新的转换添加的 Soot 的调度过程中,完成线程内和线程间的函数调用关系图分析。

2.3　访问事件

由于竞争检测经常发生在一对事件之间,所以本文使用"访问事件对"来描述数据竞争检测情况。使用 Soot 扩展机制进行控制流分析,构建主线程和子线程关于线程内和线程间的函数调用关系图,每个线程对变量进行一次访问操作记作一个访问事件。从每个线程的函数调用图中收集该线程的所有访问事件。

将一个访问事件表示为:

$ACCESS_i$: 〈threadID,accessObject,iswrite,lockset〉

其中: threadID 为访问线程的 ID,accessObject 表示线程的访问变量,iswrite(布尔型) 表示访问操作是否为写操作,lockset 表示访问操作发生时所拥有的锁集。

下面给出一个可能存在数据竞争的示例程序,如图 2 所示(第 4 行与第 9 行代码存在竞争) 。该示例程序包含 3 个线程: main、t1 和 t2。主线程 main 创建了两个子线程 t1 和 t2; 线程 t1 在第 5 行和第 7 行分别获取锁和释放锁,因此第 4 行的访问操作不受锁保护; 线程 t2 在第 8 行获取锁之后该线程中的变量访问操作均受锁保护,直至第 11 行释放锁。第 1 行在没有锁保护的情况下,主线程 main 对变量 x 的域 g 进行了写操作,记作访问事件 $ACCESS_1$: 〈main,x. g,1,{ null} 〉; 类似地,线程 t1 在第 4 行和第 6 行的访问事件分别为 $ACCESS_4$: 〈t1,x. g,1,{ null} 〉和 $ACCESS_6$: 〈t1,x. f,1,{ lock(a) } 〉; 线程 t2 在第 9 行和第 10 行的访问事件分别为 $ACCESS_9$: 〈t2,y. g,1,{ lock(b) } 〉和 $ACCESS_{10}$: 〈t2,y. f,1,{ lock(b) } 〉。

假设a,b和x,y分别是一对别名

thread main	thread t1	thread t2
1 x.g=1;	4 x.g=2;	8 lock(b);
2 t1.start();	5 lock(a);	9 y.g=4;
3 t2.start();	6 x.f=3;	10 y.f=5;
	7 unlock(a);	11 unlock(b);

图 2　存在潜在数据竞争的示例程序

Fig. 2　Example with potential data race

2.4　数据竞争判定条件

两个访问事件 $ACCESS_i$ 和 $ACCESS_j$ 存在数据竞争当且仅当它们满足以下条件:

1) $threadID_i \neq threadID_j$;

2) $accessObject_i = accessObject_j$;

3) $write_i \vee write_j = 1$;

4) $lockset_i \wedge lockset_j = null$

依据数据竞争发生的条件,对访问事件进行判断,能够得到所有可能的数据竞争。

2.5　别名分析

别名现象是指两个互为别名的引用变量共同指向同一个对象时,其中一个引用对象改变,另一个引用变量的对象值也会跟着改变。

对两个访问事件的访问对象进行别名分析,目的是判断两个变量访问操作所指向的内存位置是否一致,避免因忽略变量的别名现象而带来的漏报。假设图 2 中 x,y 是一对别名,那么第 4 行和第 9 行的访问对象 x. g 和 y. g 将指向的是同一内存位置; 而假如没有考虑到 x,y 互为别名的现象,在竞争检测过程中,x. g 和 y. g 被作为两个不同的访问对象,那么会导致线程 t1,t2 之间实际的数据竞争 〈$ACCESS_4$,$ACCESS_9$〉被漏报。

别名现象还存在于锁集分析中,对锁集进行别名分析,能够在一定程度上降低误报。例如,图 2 中第 6 行和第 10 行的竞争访问对 〈$ACCESS_6$,$ACCESS_{10}$〉,两个访问操作所持有的锁集分别为{ lock(a) } 和{ lock(b) } 。假设 a,b 是一对别名,那么 $lockset_6 \wedge lockset_{10} \neq null$(其中,$lockset_i$ 表示第 i 条语句所在的锁集) ,根据锁的排他性,那么线程 t1 和 t2 将不可能同时访问域 f。如果不考虑别名现象,那么两个访问事件的锁集的交集为 null,该访问事件对很可能被报告为一个真实竞争,这会导致误报,因此,针对锁的别名分析能够避免某些误报情况的发生,提高数据竞争检测的精确度。

2.6　控制流分析

本文基于 Jimple 构造并发程序的控制流图(Control Flow Graph, CFG) 。CFG 为用在编译器中的一个抽象数据结构,它是一个有向图,可以用 $G = (N, E, nentry, nexit)$ 表示。其中,N 为节点集,$N = \{n1, n2, \cdots\}$,程序中每条语句对应图中的一个节点; E 为有向边集,$E = \{ \langle n1, n2\rangle \mid n1, n2 \in N\}$,且 $n1$ 执行后可能立即执行 $n2$; *nentry* 和 *nexit* 分别为程序的入口和出口节点。

对图 2 示例程序进行控制流图分析,可以得到:

节点集 $N = \{1,2,3,4,5,6,7,8,9,10,11\}$; 有向边集 $E = \{ \langle 1,2\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 3,8\rangle, \langle 4,5\rangle, \langle 5,6\rangle, \langle 6,7\rangle, \langle 8,9\rangle, \langle 9,10\rangle, \langle 10,11\rangle\}$。

基于所在线程,对 N 中所有节点进行分类; 通过收集每个节点的出度边,对 E 中所有边进行分类,得到如表 1 所示的基于线程的节点集和边集。

表 1　基于线程的节点集和有向边集

Tab. 1　Thread-based node sets and directed edge sets

节点集	边集
$N_{main} = \{1,2,3\}$	$E_{main} = \{ \langle 1,2\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 3,8\rangle\}$
$N_{t1} = \{4,5,6,7\}$	$E_{t1} = \{ \langle 4,5\rangle, \langle 5,6\rangle, \langle 6,7\rangle\}$
$N_{t2} = \{8,9,10,11\}$	$E_{t2} = \{ \langle 8,9\rangle, \langle 9,10\rangle, \langle 10,11\rangle\}$

2.7　发生序关系分析

2.7.1　时序关系图

通过相应的有向边将程序中的节点连接起来,能够得到各节点之间的时序关系图,图 3 中每一个节点代表程序中语句的一次执行,有向边表明了各节点之间的执行顺序。线程内的有向边用实线箭头表示,〈2,4〉和 〈3,8〉为跨线程有向边,用虚线箭头表示。

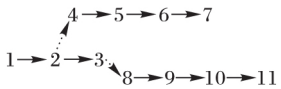

图 3　示例程序图 2 中各节点的时序关系

Fig. 3　Sequential relationship of nodes in Fig. 2

在一个时序关系图中,当两个访问事件的节点之间能够通过一个或多个有向边单向连接时,它们之间是存在发生序关系的。两个访问事件 $ACCESS_i$ 和 $ACCESS_j$ 存在发生序关系当且仅当它们满足以下条件:

1) $ACCESS_i$ 能够通过若干有向边到达 $ACCESS_j$(保证连接性) ;

2) $ACCESS_j$ 不能通过有向边到达 $ACCESS_i$(保证单向性) 。

例如,对于图 3 中节点 1 和节点 4(访问事件对

$\langle ACCESS_1, ACCESS_4 \rangle$)，虽然访问对象相同均为 x.g，但两者可以通过若干有向边单向连接，因此具有发生序关系，不会产生竞争；而节点4和节点9(不能通过若干有向边单向连接)之间不具有发生序关系。

2.7.2 访问事件切片分析

本文方法对时序关系图中的访问事件进行程序切片，然后定义产生发生序关系的条件，依据所得的切片是否满足特定条件判断是否具有发生序关系。程序 P 的切片 S 是一个可执行的程序，对某个程序点 s 处的变量 v 而言($\langle s, v \rangle$ 称为切片准则)，S 由程序 P 中可能影响 s 处变量 v 的值的所有语句构成[21]。例如，对于节点1和节点4(即针对访问事件对 $\langle ACCESS_1, ACCESS_4 \rangle$)，分别以切片准则 $\langle 1, g \rangle$ 和 $\langle 4, g \rangle$ 向后进行程序切片，收集所有通过有向边可达的可能影响变量 g 的节点，分别得到切片 S_1：$\langle 1,4,9 \rangle$ 和 S_4：$\langle 4 \rangle$。

针对任意两个访问事件的切片，定义产生发生序关系的条件。当两个访问事件分别以 $\langle s_i, v \rangle$ 和 $\langle s_j, v \rangle$ 作为切片准则得到程序切片 S_i 和 S_j 时，它们具有发生序关系需要满足的条件1)和2)可以抽象定义为：$(s_i \in S_j) \wedge (s_j \notin S_i)$。

例如，对于访问事件对 $\langle ACCESS_1, ACCESS_4 \rangle$，由于两个访问事件存在 start 原语产生的线程交互，因此两个访问事件不可能同时发生。对两者的切片进行发生序关系条件的判断，得到 $(4 \in S_1) \wedge (1 \notin S_4)$，所以节点1和节点4之间具有发生序关系 $\langle ACCESS_1, ACCESS_4 \rangle$ 为假竞争。同理，可以对节点4和节点9进行切片并进行条件判断，能发现 $\langle ACCESS_4, ACCESS_9 \rangle$ 不具有发生序关系。

由此可见，发生序关系分析能够对线程交互造成的访问事件之间的时序限制进行分析，排除部分假竞争，提高检测的准确度。

3 实验

3.1 实验环境与测试程序

在实验中，使用了 Dell Z820 工作站，该工作站中配备了两个 Intel Xeon E5-2650 处理器，主频为 2.60 GHz，每一个 CPU 有8个处理核，每个处理核均支持超线程，可支持32个线程同时运行，内存 128 GB。在软件方面，使用了 64 位 Windows 7 操作系统，JDK 版本是 1.8.0_31。

在测试程序的选择上，本文从 JGF(Java Grande Forum)基准测试套件[23]中选取了光线追踪程序 raytracer 和蒙特卡罗程序 montecarlo，这两个测试程是 JGF 测试程序中较大规模的测试程序，它们分别提供了 SizeA 和 SizeB 两种输入，在实验中选择使用较大数据集 SizeB 作为输入，这两个测试程序分别存在一个已知的数据竞争；此外，本文从 IBM Contest 基准测试套件[23]中选取了3个测试程序，分别是 bufwrite、mergesort 和 airline。表2对实验选取的测试程序的相关属性进行了说明。

表2 基准测试程序及其属性

Tab. 2 Benchmarks and their properties

测试程序	线程数	代码行数	字节数/KB
bufwrite	5	199	13.7
mergesort	5	298	20.8
airline	11	83	7.7
montecarlo	2	3605	47.1
raytracer	2	1906	45.7

3.2 实验结果与分析

为了验证本文提出的方法的有效性，将本文方法分别与 HB 算法和 RVPredict 方法进行了比较。在实验中，对检测的数据竞争数和检测所耗费的时间进行了测试，实验结果如表3所示。

表3 不同算法的实验结果与对比

Tab. 3 Experimental results and comparison of different algorithms

测试程序	HB		RVPredict		本文方法	
	竞争对象数	检测时间/s	竞争对象数	检测时间/s	竞争对象数	检测时间/s
bufwrite	2	0.8	2	1.0	2	0.9
mergesort	3	0.8	9	2.5	15	0.9
airline	8	0.8	9	0.9	10	0.6
montecarlo	0	12.0	0	13.5	1	8.6
raytracer	3	1.2	5	1.2	1	1.9
总计	16	15.6	25	19.1	29	12.9

从表3的实验结果可以发现：

1)本文方法能够有效地发现潜在的数据竞争，从而更大限度地避免漏报。

对于测试程序 bufwrite，三种方法检测的数据竞争数相同；而对于 mergesort、airline 和 montecarlo 三个测试程序，本文方法检测的数据竞争数均多于 HB 算法和 RVPredict。其中最为明显的是 mergesort 测试程序，本文方法检测到数据竞争15个，而 HB 算法和 RVPredict 检测结果均不足10个。为了保证检测结果的正确性，对相关代码进行了分析，经过确认，这些数据竞争均真实有效。

综上，能够看出本文方法具有较高的检测覆盖率，能够更大限度地避免漏报现象。这是因为：本文通过构造时序约束图来进行发生序关系分析，在时序约束图中并没有指定访问事件的执行路径；而 HB 算法对线程调度比较敏感，线程调度不同，访问事件的执行路径不同，因此存在很多漏报；RVPredict 将执行路径抽象为一系列访问事件序列，因此也可能会遗漏某些路径而导致漏报。此外，本文方法中别名变量的分析也在一定程度上降低了漏报的发生。

2)本文方法能够显著避免误报的发生。

对于测试程序 raytracer，已知实际有害竞争数为1，本文方法检测的数据竞争数为1，而 HB 算法和 RVPredict 检测的数据竞争数分别为3和5，这表明本文方法不仅能够准确检测实际竞争，而且还能够避免误报。从检测结果总数来看，本文检测到的竞争共29个，与 RVPredict 相比，本文方法检测到的数据竞争总数增加了16%；与 HB 算法相比，本文方法的检测竞争总数甚至增加了约81%。从检测的准确率来看，本文方法检测准确率为100%，而 HB 算法和 RVPredict 的准确率分别为87.5%和84%，与这两种方法相比，准确率分别提升了约14%和19%。这说明本文方法在提高数据竞争检测覆盖率的同时，能够保证检测的正确性，避免误报的发生。

本文方法中对于别名锁的分析能够降低某些数据竞争的误报现象，此外，通过时序约束图和程序切片的结合进行发生序关系分析能够避免那些因忽略线程交互带来的误报。

3)从检测时间上看，本文提出的方法没有明显的开销，而且大多数情况优于其他两种方法。

对于 bufwrite、mergesort 程序，本文方法比 HB 方法的检测时间略长，但是少于 RVPredict 的检测时间；对于 airline 和 montecarlo 程序，本文方法的检测时间明显少于其他两种检测方法；对于 raytracer 程序，本文方法的检测时间略长。总的来

说,尽管本文方法对于某些程序的检测时间多于其他两种方法,但是这些检测时间也都在可接受的时间范围之内,本文方法并没有产生明显的开销。

4 结语

本文提出了一种面向并发程序的静态数据竞争检测方法,该方法采用控制流分析、别名分析、时序分析以及程序切片技术对数据竞争进行检测,降低数据竞争检测过程中的误报和漏报。在实验中本文通过 5 个基准测试程序验证了该方法的有效性,并与 HB 算法和 RVPredict 工具进行实验对比,结果表明在没有增加检测开销的前提下,本文方法不仅能够有效地检测数据竞争,而且能够降低误报率和漏报率。进一步研究包括选取更多样化以及规模更大的基准程序对本文的方法进行测试。

参考文献 (References)

[1] YANG J, JIANG B, CHAN W K. HistLock +: precise memory access maintenance without lockset comparison for complete hybrid data race detection [J]. IEEE Transactions on Reliability, 2018, 68(3): 786 – 801.

[2] 禹振,杨振,苏小红,等. 多线程程序数据竞争检测和验证方法研究综述! [J]. 智能计算机与应用,2017,7(3): 123 – 126. (YU Z, YANG Z, SU X H, et al. A survey on methods of data race detection and verification on multithreaded program [J]. Intelligent Computer & Applications, 2017, 7(3): 123 – 126.)

[3] 苏小红,禹振,王甜甜,等. 并发缺陷暴露、检测与规避研究综述 [J]. 计算机学报,2015,38(11): 2215 – 2233. (SU X H, YU Z, WANG T T, et al. A survey on exposing, detecting and avoiding concurrency bugs [J]. Chinese Journal of Computers, 2015, 38(11): 2215 – 2233.)

[4] LU S, PARK S, SEO E, et al. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics [J]. ACM SIGARCH Computer Architecture News, 2008, 44(3): 11 – 21.

[5] 吴俞伯,郭俊霞,李征,等. 基于并发程序数据竞争故障的变异策略[J]. 计算机应用,2016,36(11): 3170 – 3177. (WU Y B, GUO J X, LI Z, et al. Mutation strategy based on concurrent program data racing fault [J]. Journal of Computer Applications, 2016, 36(11): 3170 – 3177.)

[6] 张昱,郝允允. Java 程序数据竞争的增量式检测[J]. 西安交通大学学报,2009,43(8): 22 – 27. (ZHANG Y, HAO Y Y. Incremental detection of data race for Java programs [J]. Journal of Xi'an Jiao-Tong University, 2009, 43(8): 22 – 27.)

[7] POZNIANSKY E, SCHUSTER A. MultiRace: efficient on-the-fly data race detection in multithreaded C ++ programs [J]. Concurrency & Computation Practice & Experience, 2007, 19(3): 327 – 340.

[8] FLANAGAN C, FREUND S N. FastTrack: efficient and precise dynamic race detection [C]// Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2009: 121 – 133.

[9] CAI Y, CHAN W K. LOFT: redundant synchronization event removal for data race detection [C]// Proceedings of the 2011 IEEE 22nd International Symposium on Software Reliability Engineering. Washington, DC: IEEE Computer Society, 2011: 160 – 169.

[10] SAVAGE S, BURROWS M, NELSON G, et al. Eraser: a dynamic data race detector for multi-threaded programs [J]. ACM Transactions on Computer Systems, 1997, 31(5): 27 – 37.

[11] XIE X, XUE J. ACCULOCK: accurate and efficient detection of data races [J]. Software Practice & Experience, 2013, 43(5): 543 – 576.

[12] ENGLER D, ASHCRAFT K. RacerX: effective, static detection of race conditions and deadlocks [C]// SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York: ACM, 2003: 237 – 252.

[13] PRATIKAKIS P, FOSTER J S, HICKS M. LOCKSMITH: context-sensitive correlation analysis for race detection [J]. ACM SIGPLAN Notices, 2006, 41(6): 320 – 331.

[14] VOUNG J W, JHALA R, LERNER S. RELAY: static race detection on millions of lines of code [C]// Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York: ACM, 2007: 205 – 214.

[15] LAM P, VERBRUGGE C, POMINVILLE P, et al. Soot (poster session): a Java bytecode optimization and annotation framework [C]// OOPSLA'00: Proceedings of the 2000 Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York: ACM, 2000: 113 – 114.

[16] LAMPORT L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 2008, 21(7): 558 – 565.

[17] HUANG J, MEREDITH P O, ROSU G. Maximal sound predictive race detection with control flow abstraction [J]. ACM SIGPLAN Notices, 2014, 49(6): 337 – 348.

[18] CHOI J D, LOGINOV A, SARKAR V. Static datarace analysis for multithreaded object-oriented programs [R]. Yorktown Heights, NY: IBM Research Division, 2001: 1 – 18.

[19] 吴萍,陈意云,张健. 多线程程序数据竞争的静态检测[J]. 计算机研究与发展,2006,43(2): 329 – 335. (WU P, CHEN Y Y, ZHANG J. Static data-race detection for multithread programs [J]. Journal of Computer Research and Development, 2006, 43(2): 329 – 335.)

[20] LIU P, TRIPP O, ZHANG X. IPA: improving predictive analysis with pointer analysis [C]// Proceedings of the 2016 International Symposium on Software Testing and Analysis. New York: ACM, 2016: 59 – 69.

[21] WEISER M. Program slicing [J]. IEEE Transactions on Software Engineering, 1984, SE-10(4): 352 – 357.

[22] SMITH L A, BULL J M, OBDRIZALEK J. A parallel Java grande benchmark suite [C]// Proceedings of the 2001 ACM/IEEE Conference of Supercomputing. Piscataway, NJ: IEEE, 2001: 8 – 8.

[23] FARCHI E, NIR Y, UR S. Concurrent bug patterns and how to test them [C]// Proceedings of the 2003 International Symposium on Parallel and Distributed Processing. Washington, DC: IEEE Computer Society, 2003: 286 – 296.

This work is partially supported by the National Natural Science Foundation of China (61440012), the Natural Science Foundation of Hebei Province (F2016208007), the Fundamental Research Program of Hebei Province (18960106D).

ZHANG Yang, born in 1980, Ph. D., associate professor. His research interests include concurrent software analysis.

LIANG Yanan, born in 1991, M. S. candidate. Her research interests include concurrent software analysis.

ZHANG Dongwen, born in 1964, Ph. D., professor. Her research interests include concurrent software analysis.

SUN Shixin, born in 1994, M. S. candidate. Her research interests include concurrent software analysis.