基于事件模型的可视化并行调试技术*

熊建新 王鼎兴 郑纬民 沈美明

(清华大学计算机科学与技术系 北京 100084)

摘要 并行程序存在资源竞争、时序紊乱、死锁等复杂问题,加上并行程序的不确定性,给调试带来了很大的困难.基于事件模型的可视化并行调试技术 EVTPD(event—based visualization techniques on parallel debugging)为解决这些困难提供了一个途径.EVTPD 的基本要素是事件监视与可视化重放.本文讨论了事件监视与可视化重放中的关键技术:事件描述、事件过滤、事件识别、时戳、记录重放、执行重放和视图定义,并且给出了一个基于工作站机群系统的并行调试器的结构.

关键词 事件,监视,可视化,并行,调试.

并行调试是一个值得引起人们兴趣的课题,这不仅因为它具有日益广泛的应用需求,而且更因为它从理论和技术上向我们提出了挑战.尽管这方面的研究从并行机出现之日起就已经开始进行,但至今仍有许多问题没有得到完美的解决.随着人们对高性能计算机日益强烈的需求,并行计算越来越深入到计算机应用的各个领域,并行程序的开发成了一个至关重要的问题,因此,并行调试技术的进一步研究刻不容缓.

传统的调试技术基于状态转换模型.在这种模型下,程序的执行就是程序状态的不断变化,导致这种变化的原因就是每一条机器指令的执行.调试的基本手段是设立断点,让程序在指定的地方暂停执行,然后通过获取内存单元或寄存器的内容,了解程序的状态,以期发现异常之处.高级语言一级的调试也是在此基础上加入符号与机器地址的映射以及数据对象的高级表示方法.基于状态转换模型的调试方法对串行程序的调试是比较有效的,但对于并行程序,问题要复杂得多.一方面,并行程序设计中会遇到资源竞争、时序紊乱、死锁这样一类与进程间相互作用有关的问题,这些问题用静态观察程序状态的方法难以发现,而且这种观察行为本身会对通讯的时序产生影响,导致超时、错序等新的问题.另一方面,并行程序中往往存在不确定性,程序的执行行为将因每次执行时各并行成分的进度而异.对于一个经过精心设计的表现良好的并行程序,这种不确定性也许是可以避免的,但对于包含错误的程序而言,这种不确定性的几率要大得多.不确定性的因素增加了调试的难度——在一次运行

---

* 作者**熊建新**,1969 年生,博士,主要研究领域为并行调试技术.**王鼎兴**,1937 年生,教授,博士导师,主要研究领域为并行计算机体系结构.**郑纬民**,1946 年生,教授,主要研究领域为并行处理技术.**沈美明**,1938 年生,教授,主要研究领域为并行软件工具与环境.

本文通讯联系人:熊建新,北京 100084,清华大学计算机科学与技术系

本文 1995—03—06 收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

时发生的错误,在另一次运行时以不同的形式表现出来,或者隐匿起来.对这些问题,传统的调试技术显得力不从心.

近年来,基于事件模型的调试技术的出现,为解决并行调试中出现的上述问题提供了一个可行的途径.[1,2]事件模型的基本思想是:将程序的执行过程看作是一个不断产生事件的过程,程序的一次执行产生的事件构成一张事件图.对程序的调试就是收集这张事件图,与预期的结果进行比较,看是否一致.基于事件模型的调试过程一般都分为 2 步,第 1 步是事件的监视;第 2 步是事件的重放.在重放阶段可以进行事件模式的检测,也可对程序状态进行查看.事件模型适合描述并行的执行过程,尤其是并行成分间的相互作用,两步式的方法又可解决不确定性问题,因而对并行程序的调试非常有利.

基于事件模型的调试技术常常与可视化技术结合使用.事件之间的关系纷繁复杂,文字性的描述容易使人感到困惑.利用可视化手段,不仅可以形象生动地表示事件之间的联系,还能直观地表示一些复杂的数据结构,如表、树等,方便调试的进行.

本文将讨论基于事件模型的可视化并行调试技术中的核心问题,并结合作者的实际工作给出一个并行调试器的结构.

1　基本概念

事件:每个可观察的独立的程序行为,都可称为一个事件.事件可以表达为一个元组形式〈事件类型,属性 1,属性 2,...,时戳,位置〉,属性随着事件类型的不同而不同,时戳和位置也是属性之一,但由于它们一般为所有的事件所共有,所以将它们单独列出.

原子事件:最基本的事件,是可观察的程序行为的最小单位.原子事件的定义并不是绝对的,它取决于用户的需求.一个利用消息传送机制编程的人可能把每个 SEND 和 RECV 调用都当作原子事件,而一个系统程序员则可能进一步将其划分成好几个原子事件.通常,人们习惯把操作系统提供的基本调用作为原子事件.

复合事件:也叫高层事件,由若干个原子事件或其它复合事件按一定规则组合而成.复合事件可当作一个整体进行观察,也可观察其各个组成部分.例如,在数据库管理中,读取一个记录是常见的操作.通常它包含打开文件、定位、读数据、关闭文件这样一个操作序列,我们可以将这样一个序列定义为复合事件"读取记录".这样,在观察程序行为时,我们看到的是"读取记录"这样简单明了的信息,而不是大量的 OPEN,LSEEK,READ,CLOSE 序列.由此可见,复合事件是一种事件抽象的手段.

事件模型:将程序的执行过程看成一个事件序列,对程序的观察、干预都以事件为单位.这个模型与传统的过程模型不同,过程模型将程序的执行看成是一条条指令、一行行代码的执行.并行程序由于存在异步性,用事件模型更能清楚地表达程序的状态.

可视化:广义地说,可视化是指用图形、图象或动画把要观察的对象形象、直观地表现出来.这个观察对象非常广泛,在本文的范围内,特指程序的执行过程和状态.可视化方法往往与事件模型相结合,事件的跟踪、过滤、记录与可视化重放,构成一个完整的处理过程.

视图:将若干有关系的事件进行可视化时,选择的一种表达方式.根据事件的特点,不同的事件可以用不同的视图表示,同一组事件也可以用不同的视图表示,从不同的侧面观察对象,以获取更全面的认识.

© 中国科学院软件研究所 http://www.jos.org.cn

2 事件监视

事件监视是事件模型中必不可少的部分.在一般的计算机系统中,若不采取必要的措施,一个旁观者程序是不可能知道另一个程序正在进行的操作的.为了收集到事件,被观察的进程应该主动发出信息,这一工作是由与目标程序相连的动态监视库完成的.动态监视库替换了标准库函数或系统调用的入口,使得在执行这些函数或系统调用时,能够产生相应的事件,其中包括给事件打上时戳.一个专门的监视进程能够收集这些事件,进行过滤、排序等处理后,写入一个记录文件中.

2.1 事件描述

事件描述语言(EDL, Event Description Language)可用于定义用户关心的事件,包括原子事件和复合事件.原子事件直接对应系统例程,复合事件则表述为原子事件和其它复合事件的组合.例如下面是2个原子事件的定义:

```
def primitive Send
    procedure=msgsnd
    parameter=src:int, dest:int, type:int
end
def primitive Recv
    procedure=msgrcv
    parameter=src:int, dest:int, type:int
end
```

而下面的语句则定义1个复合事件:

```
def event Sync
    combination=Send+Recv
    condition=(Send.src==Recv.dest)&&(Send.dest==Recv.src)
              &&(Send.type==MSG_SYNC)&&(Recv.type==MSG_SYNC)
    attribute=src:Send.src, dest:Send.dest
end
```

定义事件的组合关系时,"+"表示顺序相连,"*"表示无顺序的组合.

2.2 事件过滤

动态监视库产生的事件是最原始的事件,不仅数据量大,而且可读性不好.用户关心的是他自己定义的那些事件,其它的无关事件可以舍弃,这样一个工作称为过滤.完成过滤工作的部件叫过滤器.

动态监视库产生的原始事件包含2类:①与进程间相互作用有关的,如消息传送、同步等;②进程内部的,不影响进程间的关系,如读写文件.这2类事件必须区别对待.对于前者,由于它们正是导致程序不确定性的因素,我们必须忠实地记录下来,这样才可能正确地重放执行过程,因此不能将它们过滤掉;第2类事件则不同,它们可按用户的需求进行裁剪,是过滤器处理的对象.过滤器的工作机理是很简单的,仅仅是一个模式匹配的过程.只有用户在EDL描述中定义过的原子事件才能通过过滤器.

上面说的这个过滤器又叫全局过滤器.它作用于整个系统.只有通过了全局过滤器的事件才被记录下来.还有一些局部过滤器,用于复合事件的识别,不影响到事件的记录,我们在

© 中国科学院软件研究所 http://www.jos.org.cn

"事件识别"一节中将会谈到.

既然与进程通讯有关的原始事件总会被记录,那么要不要用 EDL 描述它们呢? 这要看具体情况,描述与不描述效果是不同的. 如果不用 EDL 描述,则这些记录下来的事件只是用来控制重放的顺序,用户是看不到的. 如果用户要观察它们,或者要利用它们构成复合事件,则必须在 EDL 描述中将它们定义成原子事件.

2.3 事件识别

过滤器把用户定义的原子事件从大量的原始事件中分离出来,而要进一步将原子事件构成复合事件,还需要一个事件识别过程,执行识别工作的部件叫识别器. 一个识别器由 3 部分组成:局部过滤器、有限状态自动机和操作序列. 如图 1 所示.

图1 识别器

识别器的主体是有限自动机,它可表示为 $FAM=(S,A,F,s0,T)$,其中 S 是状态集,A 是输入字母表,F 是状态转换函数,$s0$ 是起始状态,T 是终止状态集. A 是构成该高层事件的原子事件的集合,S,F 和 T 都与用户的定义有关. 一个专门的编译器对用户用 EDL 书写的事件描述进行处理,产生相应的自动机.

自动机的工作过程为:首先初始化为起始状态,当一个事件到来时,根据转换函数转向一个新的状态. 若这个新状态是终止状态,则说明成功匹配了一个用户定义的事件,执行规定的操作,并重置为起始状态. 若不能转向一个新的状态,也重置为起始状态.

局部过滤器的作用是滤除无关事件,保证自动机的正常工作,一个不属于 A 的事件将使自动机工作失败.

操作序列给用户在重放时以灵活的控制功能,可以打印一些信息,也可以设立断点等. 缺省的操作是产生一个复合事件. 值得注意的是,这个复合事件可能被另一个识别器所利用. 在监视阶段,操作总是缺省的.

在事件匹配过程中,存在多义性问题,主要表现在一个事件包含另一事件时应该怎样处理. 常用的方法是最大匹配,在一般情况下这与用户的意图是一致的. 如果用户有不同的需求,则只有修改事件描述.

2.4 时　戳

给事件加上时戳是必需的. 根据不同的情况,有不同的方法.

物理时钟最为直接了当,它的前提是要有一个全局时钟. 在许多紧耦合的并行机上,各个处理机用的是同一个时钟,用它来标识事件发生的时间,可以得到一个全序的事件图. 有的系统用的是多个时钟,但是有一个硬件的同步机构,也能得到一个全局时钟. 对于不存在全局时钟的系统,可以采用一种软件同步的方法,即在并行程序开始时,设一个同步点,每个处理机将该点的时间作为本地的起始时间,用事件发生时的本地时间与起始时间的差值作为时戳. 这种方法精确度非常有限,有时会出现时序倒错的问题(如消息的发送时间比接收时间晚).

逻辑时钟用一个逻辑值表示事件发生的先后,不能表示具体时间. 逻辑时钟不能定义事件的全序关系,只能得到一个偏序关系. 为了表示这种关系,我们给出下面定义:

定义. 设 E 是所有事件的集合,e_i,e_j 是 2 个事件,若 e_i 必然早于 e_j 发生,我们称 e_i,e_j 满

© 中国科学院软件研究所 http://www.jos.org.cn

足关系 r,记作 $r(e_i,e_j)$.若既不满足 $r(e_i,e_j)$,又不满足 $r(e_j,e_i)$,我们称 e_i 与 e_j 并行,记作 e_i $//e_j$,(E,r)构成一个偏序集.

最早提出逻辑时钟的是 Lamport[3],其思想是,在每个处理机上有一个整数计数器,每产生一个事件,该计数器加 1.发送消息时,将该计数器值一并发往对方.接收方收到消息后,根据消息中带的计数器值修改本地的计数器值(取二者较大的一个).若用 $LC(e)$表示事件 e 的逻辑时钟,则有:$r(e_i,e_j)\Rightarrow LC(e_i)<LC(j)$.注意因果不能倒置,也就是说 LC 的不同并不表示 2 个事件有必然的先后关系.

向量时钟由 Mattern 和 Fidge 提出.[4,5]假设有 N 个处理机,则用一个 N 维向量表示逻辑时间,向量的各个分量分别与各个处理机对应.每产生一个事件,将对应本处理机的分量加 1,发送消息时同样带上这个时间,接收方对本地逻辑时间进行修改(每个分量都取较大的一个).记向量时间为 $V(e)$,它有如下性质:若 e_i,e_j 分别属于处理机 i,j,则(1) $r(e_i,e_j)\Leftrightarrow V(e_i)_i\leqslant V(e_j)_i$;(2) $e_i//e_j\Leftrightarrow V(e_i)_i>V(e_j)_i$,且 $V(e_i)_j\leqslant V(e_j)_j$.可见向量时钟具有比较好的精确度,缺点是开销大.

还有一种逻辑时钟是区间时钟[6],它与 Lamport 时钟类似,不同的是它用前一事件与后一事件之间的区间表示时间.它的性质是:(1) $r(e_i,e_j)\Rightarrow I(e_i)$在 $I(e_j)$的左边;(2) $e_i//e_j$ $\Rightarrow I(e_i)$与 $I(e_j)$有重叠.$I(e_i)$是 e_i 的区间时间.

事件加上时戳后,可以进行排序,以保证正确重放.

3 可视化重放

事件监视是为调试收集必要的信息,这一阶段不能进行交互式调试,否则会严重干扰并行程序的执行.真正的调试活动都在重放阶段进行.重放时,由于有事件记录作保证,并行程序可以按照被监视时的同样的顺序执行,而且不受交互式调试行为的影响.

从本质上讲,有 2 种不同的重放方式:记录重放与执行重放.记录重放是将记录的事件按一定的方式展现给用户,程序并不重新执行;执行重放是在记录的控制下,重新执行程序,当然事件也可以象记录重放一样展现.记录重放适合观察程序的整体行为,特别是发现与通讯相关的错误,优点是速度快,任意可调;但是观察局限于被记录的事件,不能进一步了解程序的执行细节,如查看变量的内容、与源程序相对照等.执行重放时则可象传统调试器一样设断点、单步、查看变量.这对发现基本的程序错误是必须的.当然,这些好处是以执行程序为代价的.2 种重放方式的结合提供了灵活的并行调试功能.

可视化主要体现在将事件展现给用户的时候.用户可以定义许多不同的视图,从不同的角度观察程序的表现.

3.1 记录重放

记录重放的关键在于采用合适的视图[7,8],下面举几个常用的视图为例进行说明.事件——时间视图(图 2).可以全面反映各处理机上的事件流程,也即并行程序的执行过程.图 2 中,处理机 1 把数据分派到其它 3 个处理机上,计算完后,又负责合并各个处理机的结果.

负载——时间视图(图 3).用通讯、计算、空闲、阻塞 4 个状态描述各个处理机在不同时刻的负载情况,有助于分析导致程序性能不好的原因.还有其它各种视图,不再一一列举.用

© 中国科学院软件研究所 http://www.jos.org.cn

户可以定义自己的视图.

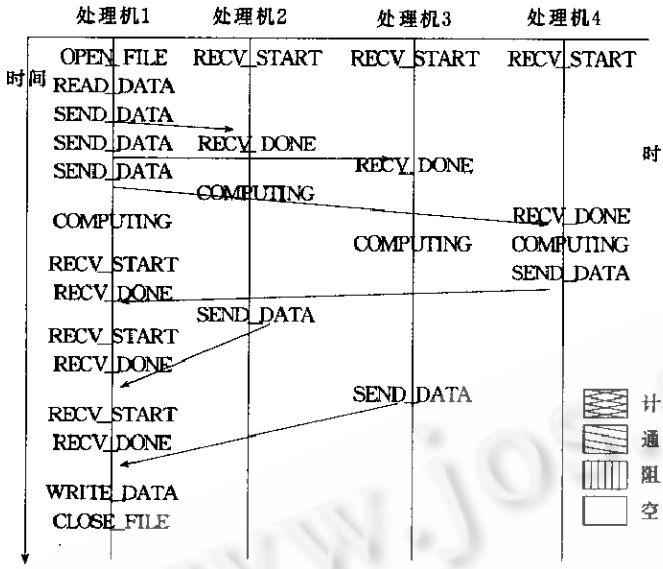

图2 事件——时间视图

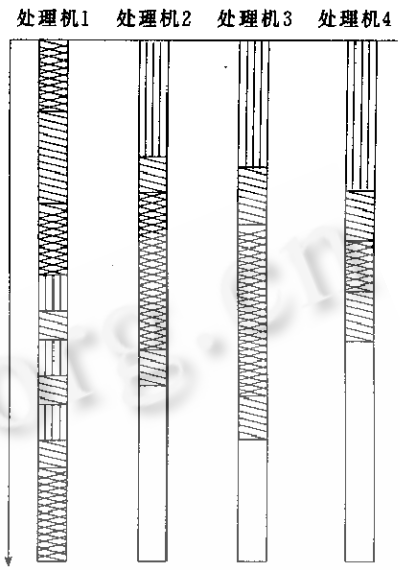

图3 负载——时间视图

3.2 执行重放

执行重放的主要目标是保证程序的执行顺序,完成这一功能还是依靠动态监视库.在动态监视库中,有一个软件开关,可以切换"监视"或"重放"状态.在监视状态时,对被监视的例程产生事件.在重放状态时,则用来控制例程的执行时间.控制过程是这样的:要执行某个被监视的例程时,先向一个称为"重放控制器"的进程发出请求,同时带上本地时戳;重放控制器根据事件记录文件的内容,确定该事件是否应该发生,若应该发生,则发一回答信号;否则将请求放入一个等待队列中,条件满足时再予以回答.例程收到响应信号后继续执行;否则阻塞.

执行重放时,可以象记录重放一样用不同的视图观察事件.而且更为重要的是,可以用交互式手段进行源级调试.当然,编译时需要指定特殊的开关以产生必要的源程序信息.我们可以为每个进程开一个源码窗口,可以设断点、查看变量的内容、单步执行等.也可以将某些进程结合成一个小组,进行同步控制,这自然包括同时控制所有进程.由于有重放控制机制保证各个进程的执行顺序,交互式调试活动不会对程序的行为产生严重的干涉.

3.3 视图定义

虽然系统可以提供大量的预定义视图以方便使用,但是给用户提供定义视图的手段还是非常必要的.定义视图主要是要确定 2 方面的内容:观察的内容和观察的方式.前者即感兴趣的事件集合,后者则是指如何将这些事件表达出来.常用的表达方式无非是时序图、直方图、线图、网络图、三维图及动画等.

视图的定义可以采用某种视图描述语言,但若从方便用户的角度来看,采用图形化视图定义界面比较好.一个典型的定义视图的步骤是:从事件列表中挑选感兴趣的事件和属性,并进行分类,为每类指定一个图形符号;从各种图表中选择一种合适的;将事件类与图表进行对应,比如指定横坐标为时间、纵坐标为处理机等.

© 中国科学院软件研究所 http://www.jos.org.cn

4 实 例

Amoeba 调试器[1]是比较典型的事件驱动的并行调试器例子. 它提供定义高层事件、过滤器和识别器的手段. 调试行为都是通过事件触发的. Amoeba 调试器采用 Instant Replay 的重放方法,这是一种执行重放策略. Amoeba 调试器是一个完整的调试工具,它缺乏的是可视化方面的支持.

ParaGraph[8]是一个对监视产生的数据进行可视化的工具. 它的优点是视图类型丰富. ParaGraph 使用的数据由 PICL(portable instrumented communication libarary)产生,可移植性好. ParaGraph 采用的重放策略属于记录重放,主要功能是分析通讯的性能以及寻找一些通讯中存在的问题. 由于它不具备调试器所需的基本功能,所以它只是一个调试辅助工具和性能分析工具.

XPVM[9]是基于 X－Window 的 PVM 集成环境. 它也提供了一个可视化监视工具. XPVM 的监视对象是所有的 PVM 系统调用,用户可以选择监视其中的一部分调用,但不可以定义自己的事件类型. 与 ParaGraph 一样,XPVM 中的可视化重放也是记录重放,也只能起辅助调试的作用. XPVM 本身并不提供调试器,它利用宿主机中现有的调试器进行调试工作. 每起动一个任务时,XPVM 为其运行一个相应的调试器,并将其 I/O 定向到一个窗口中. 由于事件记录并不用来控制程序的执行,所以不能保证程序会按确定的方式执行.

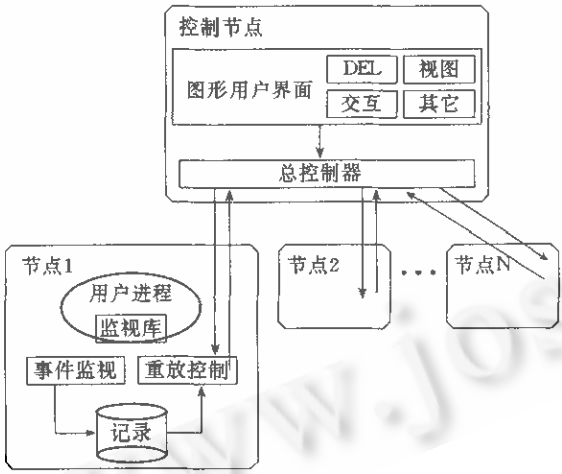

图4 并行调试器结构

工作站机群(Workstation Cluster),或者叫工作站网络 NOW (network of workstations),是当前颇受重视的一种并行系统. 我们的研究小组正在从事工作站机群系统上的并行程序设计环境与软件工具方面的研究,其中包括一个基于事件模型的可视化并行调试器,调试器的设计思想基本上遵循本文前面的论述. 该调试器是一个完整的工具,而不象 ParaGraph 那样只是一个辅助工具. 调试器支持事件重放和执行重放 2 种重放方式.

我们设计的并行调试器的结构图如图 4 所示. 这个示意图并不能精确表达调试器的结构,比如事件监视中包括过滤、识别、加时戳等功能,重放控制中有事件识别功能. EDL 描述到过滤器、识别器的转换等,都没有在图中表述.

5 总 结

并行调试技术是随着并行处理技术的发展而发展的,让并行调试器走向实用是研究人员的最终目标. 尽管还有不少的困难,但离目标已经越来越近. 基于事件模型的可视化并行调试技术具有很大的发展前途,值得我们进行理论和实践上的进一步研究.

© 中国科学院软件研究所 http://www.jos.org.cn

参考文献

1 Elshoff I J P. A distributed debugger for Amoeba. In: Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, published in ACM SIGPLAN Notices, 1989,24(1):1~10.

2 Bates P. Debugging heterogeneous distributed systems using event-based models of behavior. In: Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, published in ACM SIGPLAN Notices, 1989,24(1):11~22.

3 Lamport L. Time, clocks and the ordering of events in a distributed system. Communications of the ACM, 1978, 21(7):558~565.

4 Fidge C J. Partial orders for parallel debugging. In: Proceedings of the ACM SIGPLAN/ SIGOPS Workshop on Parallel and Distributed Debugging, published in ACM SIGPLAN Notices, 1989,24(1):183~194.

5 Mattern F. Virtual time and global states in distributed systems. In: Proceedings of the International Parallel Conference on Distributed Algorithm(1988), Gers, France, Amsterdam: North-Holland, 1989. 215~226.

6 Diehl C, Jard C. Interval approximations of message causality in distributed executions. In: Proceedings of the Symposium on Theoretical Aspects of Computer Sciences, Cardum, France, Germany: Springer-Verlag, 1992. 363~374.

7 Adam B, Jack D, Geist A *et al*. Visualization and debugging in a heterogeneous environment. Computer, 1993,26 (6):88~95.

8 Heath M, Etherideg J. Visualizing the performance of parallel programs. IEEE Software, 1991,8(5):29~39.

9 Geist A, Beguelin A *et al*. PVM: parallel virtual machine-a user's guide and tutorial for networked parallel computing. London: MIT Press, 1994. 135~139.

EVENT-BASED VISUALIZATION TECHNIQUES ON PARALLEL DEBUGGING

Xiong Jianxin Wang Dingxing Zheng Weimin Shen Meiming

(*Department of Computer Science and Technology Tsinghua University Beijing* 100084)

Abstract Debugging of parallel programs suffers from the existence of race condition, timing error, deadlock and nondeterminancy. EVTPD (event-based visualization techniques on parallel debugging) provides a way to solve these problems. Event monitoring and visualized replay are the two basis of EVTPD. In this paper, the following key techniques are discussed: event description, event filtering, event recognition, time stamping, trace replay, execution replay and view definition. As an example, a parallel debugger is designed for workstation cluster.

Key words Event, monitoring, visualization, parallel, debug.

© 中国科学院软件研究所 http://www.jos.org.cn