国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第11卷第4期 1989年12月 Vol.11,No.4

并行程序调试中的环境再现技术

昌 月 楼

（电子计算机系）

摘 要 为解决调试并行程序碰到的第一个困难：并行程序的不确定性，文中提出了环境再现技术。文中叙述了环境再现技术的思想和实现这一技术需要的数据结构，并用CCNPASCAL语言进行了描述，最后给出了一个典型的应用例子。

关键词 软件，调试，并行处理

分类号 TP31

1 调试并行程序的困难性

并行处理是当前计算技术领域中研究的热门课题之一，因为它是提高计算速度的有效手段。然而，调试并行程序却是一件很困难的工作，所以目前还未见到有效的实际的并行程序调试工具，对这一工具的开发很多尚处于实验阶段。在多处理机并行处理的环境下调试一个并行程序与在单处理机上调试一个顺序程序有很多不同之处，首先要解决的是并行程序的不确定性。

在单处理机情况下，指令的执行是串行的，只要输入数据不变，同一程序不管运行多少次，每次运行的轨迹总是一样的，所以说，顺序程序的运行状态是确定的。在多处理机的情况下，一个程序的各并发进程在各处理机上同时运行，既不能确保一个并行程序的各个进程所需的处理机总能得到满足，也不能确保各进程总是按一种不变的次序运行，也不知道哪个进程先执行完毕。能否获得必要的处理机，各进程的运行次序以及进程的完成情况都是由操作系统的调度策略按实际情况决定的，是随时改变的，这就是并行程序的不确定性[1]。

最简单的例子是两个处理机同时用do循环对数组求和。假定CPU_0对奇数下标的数组元素求和，CPU_1对偶数下标的数组元素求和。可能在某次运行时CPU_0先做完而等待CPU_1，而在另一次运行时则相反，CPU_1先做完而等待CPU_0。对于前者，CPU_1求和完了以后将其结果传给CPU_0，最后由CPU_0求总和；对于后者则相反，由CPU_1求总和。

除上述困难以外，调试并行程序还有一些其他困难，例如，调试工具对被调试程序的干预也给调试工作带来困难，等等。

研究报告 1988年8月5日收稿

2 目前的研究现状

基于并行程序的特点，目前对并行程序调试方法的研究有两个方向。

一个方向是沿用调试顺序程序的传统方法，用设断点、步进和跟踪等手段来观察程序的运行状态。用观察程序运行状态的方法只有在程序本身具有确定的状态才是可行的，只有这样，才能在反复运行同一程序时不断地孤立错误，最后发现错误。用这一方法调试并行程序时，首先要解决的问题是程序的不确定性，要保证并行程序的运行跟顺序程序一样具有确定的状态。这就是本文下面提出的环境再现技术要解决的问题。

另一个研究方向是采用一种全新的概念，从程序运行的行为（结果）而不是从其状态（过程）来调试程序。这就是基于行为的调试技术[2]。在这种技术中，程序员首先描述其程序的期望行为，然后让它与程序运行时的实际行为相比较，如果二者完全一致，则程序正确；否则，从相异点去查找错误。此法拟另作叙述。

3 环境再现技术

该技术的基本思想是当并行程序首次运行时用一个历史文件记下该次运行时的一些关键信息，这些信息反映了并行程序的运行状态。再次运行该程序时这些信息可用来控制各并行进程的运行，从而保证程序的运行状态不变，达到了环境再现的目的。

3.1 关键信息

关键信息是各进程之间相互作用时的信息。如果并行程序的各进程之间没有相互通信，则可以对每个进程单独地用传统的顺序程序调试技术进行调试。只有当各进程之间通过通信或同步原语相互作用时，才有可能出现不可重复的现象。所以，为了减少历史文件的大小和不过多地降低程序运行速度，没有必要把程序运行时的点点滴滴都记录下来，只要记录各进程之间的相互作用就够了，即只要记下各进程之间的输入输出关系。如果能使程序在每次运行时保持其各进程之间的输入输出信息不变，这就达到了我们所需要的环境再现的目的。比如在上面的简单例子中，如果能保证 CPU_0 上的进程每次运行时都是部份和接受者，而 CPU_1 上的进程每次运行时都是部份和发送者，这就克服了不确定性而成为确定性。

所以，应该把注意力集中在进程对共享资源的操作。共享资源可以是如下类型：共享变量（含同步及互斥原语），共享缓冲区，共享通道，共享设备（盘、带）。为简单计且不失一般性，以下的描述中假定共享资源是共享变量。各并发进程对共享资源的操作无非是读写操作，这就需要一种协议。为了尽可能地开发并行性同时又要保证数据的一致性，本文采用CREW协议，即“并发读互斥写”协议。本协议要求，只要有进程读共享变量，就不允许其他进程写该变量，而多个进程对该变量的并发读是可以的。只有全部读进程的读操作完成以后，才允许写操作进行；否则写进程必须等待。而一旦写进程获准写操作，这就是独占的操作。由此可见，对某一共享变量来说，各读写进程之间的关系是一种偏序关系，其

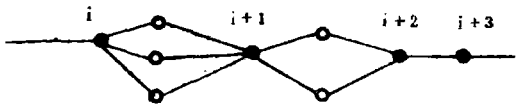

图 1 CREW协议中的读写关系

●：写 ○：读

示意图如图 1 所示。

由此可见，如果并行程序的每次运行中各进程对共享变量的读写都遵循一种不变的偏序关系，则就实现了所希望的环境再现。

3.2 记录偏序关系

共享变量的演变情况可以用版号来表示。变量的初始状态记为 0 号，以后每被修改一次，其版号依次递增。这些信息由进程在首次运行时记录在历史文件中。这样，在并行程序往后的运行中，各进程对共享变量的存取必须严格按版号进行。假定 P_0 和 P_1 首次运行时对共享变量V的存取关系为 $P_0/W\cdot V(1)\longrightarrow P_1/W\cdot V(2)$，那么在往后的运行中也要求它们保持这一关系，其中 P 为进程名，W为写操作，V的下标为版号。

这样做还不够，还必须在历史文件中记录各版号下读进程的总数，这是为了保持读写进程之间的先后关系：只有当某版号下所有读进程的读操作执行完毕以后，对该进程的写操作才能开始进行。例如在图 1，各●上面的 i、$i+1$、$i+2$、$i+3$ 都是该变量的版号，在 $(i+1)$ 版的情况下，只有当这两个读操作完成以后，才能修改此变量。于是，该变量成为第 $(i+2)$ 版。

4 实现方法

为了进一步阐明上述思想，下面用类 CCNPASCAL（并发 PASCAL）语言作一简单描述。

4.1 数据结构

每个进程有一个历史文件，用来记录该进程对共享变量的读写情况，对此文件的读写操作由进程本身进行。历史文件的结构可描述如下：

```
type
  HF=file of record
                OBJNAM: Char; /*共享变量名*/
                OBJVSN, TTLRD: integer;
                        /*版号和读进程数*/
                ACCTPE: ('READ', 'WRITE')
                        /*存取类型*/
              end;
Var
  HISFILE: HF
```

4.2 并发控制

多进程并行操作时，一方面它们要完成各自的计算任务，另一方面它们还要读写历史文件。下面的两个过程描述了并发进程在读/写共享变量前后怎样存取历史文件，其中使用了上小节中描述的历史文件结构。由于读共享变量是并发的，所以例 1 用的是类程（class）；反之，写共享变量是独占的，故例 2 是管程（monitor）。变量 OBJECT·TTLRD用来累计在某版号下读进程个数。此数将写入历史文件。变量OBJECT·ACTRD对读进程计数，只有当它为零时写操作才被允许。对这两个变量的修改也必须是互斥

例 1 读操作控制

```
Var
  READ, Class;
  type
    MD=('INITIAL', 'REPEAT'); /*并行程序运行方式*/
  Var
    OBJECT·VSN, OBJECT·ACTRD, OBJECT·TTLRD:integer;
    MODE:MD;
  procedure export BGN(PROCESS, OBJECT);
    begin
      if MODE='INITIAL' then /*首次运行时写历史文件*/
        begin
          HISFILE·OBJNAM:=OBJECT;    /*以下三句写历史文件*/
          HISFILE·OBJVSN:=OBJECT·VSN;
          HISFILE·ACCTPE:='READ';
          MDFY·INC(OBJECT·ACTRD)/*累计读进程个数*/
        end
      else  /*重复运行时进行核对，必要时等待*/
        begin
          if (HISFILE·OBJNAM¦=OBJECT) or
            (HISFILE·ACCTPE¦='READ')then ERROR;
          while HISFILE·OBJVSN¦=OBJECT·VSN do WAIT
        end
      end
    end BGN
  procedure export END(OBJECT);
    begin
      MDFY·INC(OBJECT·TTLRD); /*累计读进程数*/
      if MODE='INITIAL' then MDFY·DEC(OBJECT·ACTRD);
    end
  end END
end READ
```

例 2 写操作控制

```
Var
  WRITE:monitor;
    import
      Var:OBJECT·VSN, OBJECT·ACTRD, OBJECT·TTLRD, MODE
    end;
```

```
procedure export BGN(PROCESS, OBJECT);
  begin
    if MODE='INITIAL' then /*首次运行时写历史文件，必要时等待*/
      begin
        while OBJECT·ACTRD!=0 do WAIT /*等读进程完毕*/
        HISFILE·OBJNAM:=OBJECT; /*以下4句写历史文件*/
        HISFILE·OBJVSN:=OBJECT·VSN;
        HISFILE·ACCTPE:='WRITE';
        HISFILE·TTLRD:=OBJECT·TTLRD
      end
    else  /*重复运行时核对，必要时等待*/
      begin
        if(HISFILE·OBJNAM!=OBJECT) or
          (HISFILE·ACCTPE!='WRITE') then ERPOR;
        while HISFILE·VSN!=OBJECT·VSN do WAIT;
          /*版号不对时，等待*/
        while OBJECT·TTLRD<HISFILE·TTLRD do WAIT
              /*读进程总数与历史记载不对时，等待*/
      end
  end
end BGN
procedure export END(OBJECT);
  begin
    OBJECT·TTLRD:=0; /*将当前版号下读进程总数清零*/
    OBJECT·VSN:=OBJECT·VSN+1; /*更新版号*/
  end
end END
end WRITE
```

的，所以由管程MDFY来修改。

5　应用实例

例3描述了进行广播式通信的进程怎样存取历史文件。这里，共享资源为通道。显然，写通道是互斥的，读通道是并发的，所以它们分别用管程理和类程描述。

例 3　并发控制简例

```
Var
  SENDER:monitor;
    procedure export PUT (PORT, PUTTER) ;
    begin
```

```
        Find(PORT); /*寻找通道*/
        WRITE·BGN(PUTTER, PORT); /*处理历史文件*/
        Send message on PORT; /*发送信息*/
        WRITE·END(PORT)      /*处理历史文件*/
      end
    end PUT
  end SENDER

  Var
    RECEIVER: class;
    procedure export GET(PORT, GETTER);
      begin
        Find(PORT); /*寻找通道*/
        READ·BGN(GETTER, PORT); /*处理历史文件*/
        recieve message from PORT; /*接收信息*/
        READ·END(PORT)   /*处理历史文件*/
      end
    end GET
  end RECEIVER
```

6 结 语

本文没有企图描述调试并行程序的全过程，它只是为用传统手段调试并行程序提供了一个基础。只有在此基础上将不确定的并行程序变成确定性以后，才能用传统的方法去调试它。传统的调试方法无非是显示，设断点，步进，跟踪等等，不必赘述。

用这种方法调试并行程序时，对被调程序是有干预性(intrusiveness)的，因为它要求在被调程序中插入一些过程调用，而调试完以后又要将这些过程调用去掉，即它不是在保持程序运行的真实环境下调试的。而干预性对被调程序产生的副作用对并行程序来讲是不可忽视的，因为并行程序的状态是具有时序性的。这是本方法的缺点。

参 考 文 献

[1] CRAY COMPUTER SYSTEMS, Multitasking Programmer's Manual. CRAY RESEACH INC. 1986:10～50

[2] Bates P C and Wileden J C. The Journal of Systems and Software. 1983, 4 (3): 255～264

[3] Harter P K Jr., Heimbingner D M and King R. Proc. of the 5th International Conf. on Distributed Computing Systems. 1985: 498～506

[4] Bruegge B and Hibbard P. Proc. of the SIGSOFT/SIGPLAN Symp. on High-Level Debugging. 1983: 34～44

[5] 王善英，昌月楼. 数据库管理系统. 上海交通大学讲义，1983：43～102

国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第11卷第4期 1989年12月 Vol.11 No.4

YH-GKS的设计与实现

万 良 君

(电子计算机系)

摘 要 图形核心系统GKS是二维图形软件的唯一的国际标准，且为我国采纳为国家标准。YH-GKS是基于GKS，为银河机设计的图形软件系统。文中介绍了YH-GKS的设计特色与实现方法。

关键词 计算机图形学，并行计算机，图形核心系统，图段

分类号 TP31

YH图形核心系统，即YH-GKS，是为银河巨型机设计的图形软件系统。它基于目前唯一的被ISO正式批准为二维图形软件国际标准的图形核心系统GKS。此标准已被我国标准化组织采纳为中国国家标准。因为YH机是批处理系统的向量机，所以，在YH-GKS的设计与实现中，必须充分发挥向量计算机的优势，研究计算机图形学的并行算法。本文的主要目的在于，描述作为并行计算机图形系统的YH-GKS的特色，例如：YH图段存储器的向量双层表结构，YH图形文件的批处理，等等。

技术报告 1989年3月2日收稿

Environment Reappearing Technique in Debugging a Parallel Program

Chang Yuelou

(Department of Computer)

Abstract

The first problem of debugging a parallel program is the nondeterminancy of the program. The environment reappearing technique presented in this paper can solve the problem quite well. The paper discribes the idea of the technique, the data structure needed for it and gives a pidgin CCNPASCAL languge description of it. An example of the technique is briefly given in it.

Key words: software, debugging, parallel processing