

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/45 (2006.01)



[12] 发明专利说明书

专利号 ZL 200410068484.X

[45] 授权公告日 2008 年 10 月 1 日

[11] 授权公告号 CN 100422933C

[22] 申请日 2004.7.23

[21] 申请号 200410068484.X

[30] 优先权

[32] 2003.8.26 [33] US [31] 10/648,461

[73] 专利权人 微软公司

地址 美国华盛顿州

[72] 发明人 D·J·麦克拉迪 P·F·林塞斯
B·米赫塔

[56] 参考文献

US6502238B1 2002.12.31

US6327699B1 2001.12.4

CN1319203A 2001.10.24

US6230198B1 2001.5.8

审查员 苏 斐

[74] 专利代理机构 上海专利商标事务所有限公司
代理人 陈 斌

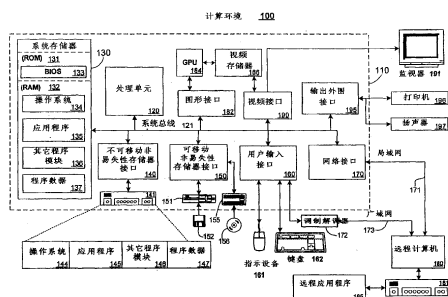
权利要求书 9 页 说明书 29 页 附图 15 页

[54] 发明名称

事务型处理过程的数据流分析的方法和系统

[57] 摘要

提供一种 XLANG/s 编译方法，它用一个程序信号流图的数据流分析来确定一个数据对象的使用周期。根据抽象计算机指令创建一个信号流图。对基本程序块指定一个深度优先次序并确定基本程序块两两之间的支配关系。确定信号流图内是否存在循环，如果存在循环，标识该循环。还确定关于数据对象的一个创建点，撤消点和锁定点。在计算机代码中插入指令以在创建点创建数据对象，在撤消点撤销数据对象和在锁定点锁定数据对象。



1. 一种计算机实现的编译计算机代码的方法，该方法包括：

根据抽象计算机指令创建一信号流图，其中该信号流图具有多个基本程序块和至少一个数据对象，其中将抽象指令转换成由计算机代码形成的一个语义分析树；

对所述多个基本程序块指定一个深度优先次序；

确定所述多个基本程序块两两之间的支配关系；

确定所述信号流图内是否存在循环，如果存在循环，标识该循环；

确定所述至少一个数据对象的应用；

根据该应用、标识的循环、所述多个基本程序块的支配关系和深度优先次序来为所述至少一个数据对象确定一创建点、撤消点和锁定点；以及

在所述计算机代码中插入指令以在所述创建点创建所述至少一个数据对象，在所述撤消点撤消所述至少一个数据对象并在所述锁定点锁定所述至少一个数据对象，其中：

确定一创建点进一步包括：

标识所述至少一个数据对象的第一次使用并从所述多个基本程序块中标识所述第一次使用发生在其中的第一基本程序块，其中包括所述第一次使用的所述数据对象的所述第一基本程序块是初始创建点；

标识所述至少一个数据对象在所述第一次使用后的一次使用并从所述多个基本程序块中标识所述第一次使用后的所述一次使用发生在其中的第二基本程序块；

计算所述第一基本程序块的预支配顶点与所述第二基本程序块的预支配顶点的交集；

确定所述交集是否包含所述第一基本程序块；以及

如果所述交集不包含所述第一基本程序块，则从所述交集中选择一新创建点；

确定一撤消点包括：

标识所述至少一个数据对象的最后一次使用并从所述多个基本程序块中标识第一基本程序块，所述最后一次使用发生在该第一基本程

序块中, 其中包括所述最后一次使用的所述数据对象的所述第一基本程序块是初始撤销点;

标识所述至少一个数据对象在最后一次使用前的一次使用并从所述多个基本程序块中标识最后一次使用前的所述一次使用发生在其中的第二基本程序块;

计算所述第一基本程序块的在后支配顶点与所述第二基本程序块的在后支配顶点的交集;

确定所述交集是否包含所述第一基本程序块; 以及

如果所述交集不包含所述第一基本程序块, 则从所述交集中选择一新撤销点; 以及

确定一锁定点进一步包括:

标识所述至少一个数据对象的第一次使用并从所述多个基本程序块中标识所述第一次使用发生在其中的第一基本程序块, 其中包括所述第一次使用的所述数据对象的所述第一基本程序块是初始锁定点;

标识所述至少一个数据对象在所述第一次使用后的一次使用并从所述多个基本程序块中标识所述第一次使用后的所述一次使用发生在其中的第二基本程序块;

计算所述第一基本程序块的预支配顶点与所述第二基本程序块的预支配顶点的交集;

确定所述交集是否包含所述第一基本程序块; 以及

如果所述交集不包含所述第一基本程序块, 则从所述交集中选择一新锁定点。

2. 根据权利要求1所述的方法, 其中从所述交集中选择一新创建点包括确定是否该新创建点与初始创建点在不同的循环内, 如果是这样, 那么选择一新创建点, 它绕过在初始创建点和新创建点之间的所有内部循环。

3. 根据权利要求1所述的方法, 其中从所述交集中选择一新的撤销点包括确定是否该新撤销点与初始撤销点在不同的循环内, 如果是这样, 那么选择一新撤销点, 它绕过在初始撤销点和新撤销点之间的所有内部循环。

4. 根据权利要求1所述的方法, 其中确定一锁定点包括:

确定一组至少一个数据对象;

从所述多个向所述至少一个数据对象进行写操作的基本程序块中创建第一组基本程序块;

从所述多个对所述至少一个数据对象进行读操作的基本程序块中创建第二组基本程序块;

从所述第一和第二组中消除不包含在一同步范围内的所述多个基本程序块中的任何一个; 以及

确定要为所述至少一个数据对象中的各个数据对象设置的锁定的类型。

5. 根据权利要求 4 所述的方法, 其中确定对所述至少一个数据对象中的各个数据对象的锁定的类型包括在所述第一组为空时, 选择对所述至少一个数据对象的一读锁定, 否则选择一写锁定。

6. 根据权利要求 1 所述的方法, 其中从所述交集中选择一新的锁定点包括确定是否所述新锁定点与所述初始锁定点在不同的循环内, 如果是这样, 那么选择一新锁定点, 它绕过在初始锁定点和新锁定点之间的所有内部循环。

7. 根据权利要求 1 所述的方法, 其中该计算机代码是 XLANG/s。

8. 根据权利要求 1 所述的方法, 其中该至少一个数据对象是变量。

9. 根据权利要求 1 所述的方法, 其中该至少一个数据对象是标记。

10. 根据权利要求 1 所述的方法, 其中该至少一个数据对象是一个消息。

11. 根据权利要求 1 所述的方法, 其中该信号流图相当于一个长期运行的事务处理。

12. 根据权利要求 1 所述的方法, 其中该信号流图相当于一个原子事务处理。

13. 根据权利要求 1 所述的方法, 其中该信号流图相当于带有一个异常处理程序的长期运行的事务处理。

14. 根据权利要求 1 所述的方法, 其中该信号流图相当于带有补偿的长期运行的事务处理。

15. 根据权利要求 1 所述的方法, 其中该信号流图相当于带有补偿的原子事务处理。

16. 根据权利要求 1 所述的方法, 其中该信号流图相当于带有一个异常处理程序和补偿的长期运行的事务处理。

17. 一种用于编译计算机代码的计算机系统，该计算机系统包括包括：
- 用于根据抽象计算机指令创建一信号流图的装置，其中该信号流图具有多个基本程序块和至少一个数据对象，其中将抽象指令转换成由计算机代码形成的一语义分析树；
 - 用于对所述多个基本程序块指定一深度优先次序的装置；
 - 用于确定所述多个基本程序块两两之间的支配关系的装置；
 - 用于确定所述信号流图内是否存在循环，如果存在循环，标识该循环的装置；
 - 用于确定所述至少一个数据对象的应用的装置；
 - 用于根据该应用、标识的循环、所述多个基本程序块的支配关系和深度优先次序来为所述至少一个数据对象确定一创建点，撤消点和锁定点的装置；以及
 - 用于在计算机代码中插入指令以在所述创建点创建所述至少一个数据对象，在所述撤消点撤消所述至少一个数据对象并在所述锁定点锁定所述至少一个数据对象的装置，其中：
 - 确定一创建点进一步包括：
 - 标识所述至少一个数据对象的第一次使用并从所述多个基本程序块中标识所述第一次使用发生在其中的第一基本程序块，其中包括所述第一次使用的所述数据对象的所述第一基本程序块是初始创建点；
 - 标识所述至少一个数据对象在所述第一次使用后的一次使用并从所述多个基本程序块中标识所述第一次使用后的所述一次使用发生在其中的第二基本程序块；
 - 计算所述第一基本程序块的预支配顶点与所述第二基本程序块的预支配顶点的交集；
 - 确定所述交集是否包含所述第一基本程序块；以及
 - 如果所述交集不包含所述第一基本程序块，则从所述交集中选择一新创建点；
 - 确定一撤消点包括：
 - 标识所述至少一个数据对象的最后一次使用并从所述多个基本程序块中标识所述最后一次使用发生在其中的第一基本程序块，其中包

括所述最后一次使用的所述数据对象的所述第一基本程序块是初始撤销点；

标识所述至少一个数据对象在最后一次使用前的一次使用并从所述多个基本程序块中标识最后一次使用前的所述一次使用发生在其中的第二基本程序块；

计算所述第一基本程序块的在后支配顶点与所述第二基本程序块的在后支配顶点的交集；

确定所述交集是否包含所述第一基本程序块；以及

如果所述交集不包含所述第一基本程序块，则从所述交集中选择一新撤销点；以及

确定一锁定点进一步包括：

标识所述至少一个数据对象的第一次使用并从多个所述基本程序块中标识所述第一次使用发生在其中的第一基本程序块，其中包括所述第一次使用的所述数据对象的所述第一基本程序块是初始锁定点；

标识所述至少一个数据对象在所述第一次使用后的一次使用并从所述多个基本程序块中标识所述第一次使用后的所述一次使用发生在其中的第二基本程序块；

计算所述第一基本程序块的预支配顶点与所述第二基本程序块的预支配顶点的交集；

确定所述交集是否包含所述第一基本程序块；以及

如果所述交集不包含所述第一基本程序块，则从所述交集中选择一新锁定点。

18. 根据权利要求 17 所述的计算机系统，其中从所述交集中选择一新创建点包括确定是否所述新创建点与所述初始创建点在不同的循环内，如果是这样，那么选择一新创建点，它绕过在初始创建点和新创建点之间的所有内部循环。

19. 根据权利要求 17 所述的计算机系统，其中从所述交集中选择一新的撤销点包括确定是否新撤销点与初始撤销点在不同的循环内，如果是这样，那么选择一新撤销点，它绕过在初始撤销点和新撤销点之间的所有内部循环。

20. 根据权利要求 17 所述的计算机所述，其中确定一个锁定点包括：

用于确定一组至少一个数据对象的装置;

用于从所述多个向所述至少一个数据对象进行写操作的基本程序块中创建第一组基本程序块的装置;

用于从所述多个对所述至少一个数据对象进行读操作的基本程序块中创建第二组基本程序块的装置;

用于从所述第一和第二组中消除不包含在一同步范围内的多个基本程序块中的任何一个的装置; 以及

用于确定要为所述至少一个数据对象中的各个数据对象设置的锁定的类型的装置。

21. 根据权利要求 17 所述的计算机系统, 其中确定对所述至少一个数据对象中的各个数据对象的锁定的类型包括在所述第一组为空时, 选择对至少一个数据对象的一读锁定, 否则选择一写锁定。

22. 根据权利要求 17 所述的计算机系统, 其中从所述交集中选择一新锁定点包括确定是否所述新锁定点与初始锁定点在不同的循环内, 如果是这样, 那么选择一新锁定点, 它绕过在初始锁定点和新锁定点之间的所有内部循环。

23. 根据权利要求 17 所述的计算机系统, 其中该计算机代码是 XLANG/s。

24. 根据权利要求 17 所述的计算机系统, 其中该至少一个数据对象是变量。

25. 根据权利要求 17 所述的计算机系统, 其中该至少一个数据对象是标记。

26. 根据权利要求 17 所述的计算机系统, 其中该至少一个数据对象是一个消息。

27. 根据权利要求 17 所述的计算机系统, 其中该信号流图相当于一个长期运行的事务处理。

28. 根据权利要求 17 所述的计算机系统, 其中该信号流图相当于一个原子事务处理。

29. 根据权利要求 17 所述的计算机系统, 其中该信号流图相当于带有一个异常处理程序的长期运行的事务处理。

30. 根据权利要求 17 所述的计算机系统, 其中该信号流图相当于带有补

偿的长期运行的事务处理。

31. 根据权利要求 17 所述的计算机系统, 其中该信号流图相当于带有补偿的原子事务处理。

32. 根据权利要求 17 所述的计算机系统, 其中该信号流图相当于带有一个异常处理程序和补偿的长期运行的事务处理。

33. 一种 XLANG/s 代码编译方法, 包括:

根据抽象计算机指令创建一信号流图, 该信号流图具有多个基本程序块和至少一个数据对象;

对所述多个基本程序块指定一深度优先次序;

确定所述多个基本程序块两两之间的支配关系;

标识所述多个基本程序块形成的循环;

根据抽象指令确定所述至少一个数据对象的应用;

根据该应用、标识的循环、所述多个基本程序块的支配关系和深度优先次序来为所述至少一个数据对象确定一创建点、撤消点和锁定点; 以及

在计算机代码中插入指令以在所述创建点创建所述至少一个数据对象, 在所述撤消点撤消所述至少一个数据对象并在所述锁定点锁定所述至少一个数据对象, 其中:

确定一创建点进一步包括:

标识所述至少一个数据对象的第一次使用并从所述多个基本程序块中标识所述第一次使用发生在其中的第一基本程序块, 其中包括所述第一次使用的所述数据对象的所述第一基本程序块是初始创建点;

标识在第二基本程序块中所述至少一个数据对象的第二次使用, 其中所述第二基本程序块是比所述第一基本程序块更高次序的基本程序块;

计算所述第一基本程序块的预支配顶点与所述第二基本程序块的预支配顶点的交集;

确定所述交集是否包含所述第一基本程序块; 以及

如果所述交集不包含所述第一基本程序块, 则从所述交集中选择一新创建点;

确定一撤消点包括:

标识所述至少一个数据对象的最后一次使用并从所述多个基本程序块中标识所述最后一次使用发生在其中的第一基本程序块，其中包括所述最后一次使用的所述数据对象的所述第一基本程序块是初始撤销点；

标识在第二基本程序块中所述至少一个数据对象的一次使用，其中所述第二基本程序块是比所述第一基本程序块更低次序的基本程序块；

计算所述第一基本程序块的在后支配顶点与所述第二基本程序块的在后支配顶点的交集；

确定所述交集是否包含所述第一基本程序块；以及

如果所述交集不包含所述第一基本程序块，则从所述交集中选择一新撤销点；以及

确定一锁定点进一步包括：

确定一组至少一个数据对象；

创建向所述至少一个数据对象进行写操作的第一组基本程序块；

创建对所述至少一个数据对象进行读操作的第二组基本程序块；

从所述第一和第二组中消除不包含在一个同步范围内的多个基本程序块中的任何一个；以及

确定要为所述至少一个数据对象中的各个数据对象设置的锁定的类型；

标识所述至少一个数据对象的第一次使用并从多个所述基本程序块中标识所述第一次使用发生在其中的第一基本程序块，其中包括所述第一次使用的所述数据对象的所述第一基本程序块是初始锁定点；

标识在第二基本程序块中所述至少一个数据对象的一次使用，其中所述第二基本程序块是比所述第一基本程序块更高次序的基本程序块；

计算所述第一基本程序块的预支配顶点与所述第二基本程序块的预支配顶点的交集；

确定所述交集是否包含所述第一基本程序块；以及

如果所述交集不包含所述第一基本程序块，则从所述交集中选择一新锁定点。

34. 根据权利要求33所述的方法, 其中从所述交集中选择一新创建点包括确定是否新创建点与初始创建点在不同的循环内, 如果是这样, 那么选择一新创建点, 它绕过在初始创建点和新创建点之间的所有内部循环。

35. 根据权利要求33所述的方法, 其中从所述交集中选择一新的撤消点包括确定是否新撤消点与初始撤消点在不同的循环内, 如果是这样, 那么选择一新撤消点, 它绕过在初始撤消点和新撤消点之间的所有内部循环。

36. 根据权利要求33所述的方法, 其中确定对所述至少一个数据对象中的各个数据对象的锁定的类型包括在所述第一组为空时, 选择对至少一个数据对象的一读锁定, 否则选择一写锁定。

37. 根据权利要求33所述的方法, 其中从所述交集中选择一新锁定点包括确定是否新锁定点与初始锁定点在不同的循环内, 如果是这样, 那么选择一新锁定点, 它绕过在初始锁定点和新锁定点之间的所有内部循环。

事务型处理过程的数据流分析的方法和系统

发明领域

本发明总地涉及计算机软件编译程序。具体地，本发明涉及在编译时执行程序代码的静态数据流分析的一种XLANG/s编译方法。更具体地，本发明涉及这样一种方法，其借助于静态数据流分析而在编译时自动确定数据对象创建，锁定和撤消点。

发明背景

商行可以经由一个诸如国际互联网或万维网那样的分布式环境用网络服务来与其他实体相互作用。为了确保成功地完成这样一个交互，必须设置一个或多个协议用于传送信息给参加者或传送来自参加者的信息，而且还必须在每一个参加者结束处安装特殊的商务应用程序。这种交互是信息驱动地。例如，购买者发送一个订货单给销售者。然后销售者核查他的存货清单来确定它是否能够提供所列项目。如果可以提供，销售者发送一个带有价格的回执返回给购买者。最后，购买者接受或拒绝销售者的报价（和 / 或可能发出另一个订单）。就如该实例中显然的，每一个参加者商务应用程序对信息的收到作出反应。

网络服务使用基于软件代码的计算机程序来执行诸如以上所描述的那些基于信息的业务往来。用一种程序设计语言交替地执行这种软件代码。程序设计语言一般允许程序员声明一个或多个保持程序状态的数据对象，诸如一个变量。

在一种能够执行上述业务往来的面向消息的语言中，数据对象可以是来自一些外部信号源并作为输入的一个信息。这样一个信息可以很大，因而存储多个数据对象可能损耗大量的存储资源。因此，一旦软件的正常运行不再需要一个数据对象的时候就应该删除或“撤消”该数据对象。通常，由程序员手工地往软件代码中插入撤消点，该撤消点是用来撤消一个或多个数据对象的代码指令。程序员通常会选择将要撤消的特定数据对象以及代码内将要发生这个删除的位置。通常，程序员会试图在最后使用该数据对象之后立即设置撤消点。照这种方式，将数据对象在存储器中保持尽可能简短的持续时间。

作为手工执行的结果，这种传统的撤消数据对象的方法因而在设置撤消点

和选择将要删除的数据对象中存在潜在的人为误差。在正执行一个基于信息的事务处理的程序的上下文中存在一个额外的缺陷。例如，在这样一个程序中，可以中断信息或者发生其他调用特殊“补偿”软件部分的误差，这些软件部分可以再发送一个信息或者尝试其他的处理来校正或补偿该误差。补偿部分对软件代码增加了复杂性，从而使程序员选择代码内专用位置的任务变得更为困难，其中应该设置一个撤消点。另外，用于基于信息的事务处理的程序代码也许是特别复杂的，带有许多并行处理的操作，具有不能完成某些操作或者可以在运行时间调用一个补偿部分的可能性。因此，在这样一种情况下选择一个有关一个撤消点的位置是困难的，如同必须考虑在处理过程中所采用的每一个可能的逻辑路径。

上述传统设置撤销点的缺陷同样适用于“创建”和“锁定”点以及其他任何类型数据对象处理操作。创建点是软件代码内的一个位置，其中创建一个数据对象。理论上，应该在第一次使用该数据对象之前立即设置一个创建点，再一次减少存储器资源的损耗。锁定点是软件代码内的一个位置，其中使不涉及当前事务处理的程序部分得不到诸如变量那样的一组共享对象。因此在事务处理期间，一个外部元件不能改变分配给数据对象的值，因而在处理过程期间避免误差。就像创建点那样，应该尽可能接近于使用数据对象而设置锁定点来避免不必要的变量锁定。然而，在执行基于信息的事务处理的软件程序中，不适当地设置一个锁定点可能会导致程序带有错误或甚至停止操作。在复杂的软件中，保持所有可能的逻辑路径的轨迹和伴随的补偿部分来在最适合的位置设置锁定点对程序员而言变得越来越困难，该最合适的位置是尽可能接近于使用数据对象而不在如果误差发生时会导致错误的位置。

本领域技术人员应该知道，编译程序将程序员所使用的高级代码转换成在运行时间能够快速执行的机器可读的语言。编译时，编译程序制订程序映射出不同的可能的逻辑路径来创建这种逻辑路径的一个信号流图。该信号流图用于程序正确性的语义分析和产生最有效的代码。

传统编译程序缺少分析程序，和 / 或产生信号流图的功能，以这样一种方式能够确定创建点，锁定点和 / 或撤消点最合适的设置。另外，传统编译程序缺少在程序中插入这些点的能力，所以将会在运行时在适当的点创建，锁定和 / 创建相应的数据对象。

因此,需要诸如XLANG/s那样的一种用于分析软件的方法,它能够以这样一种关于能够确定有关数据对象操作的适合点的方式执行基于信息的事务处理。特别地,需要一种创建程序信号流图的方法,该程序能够支持对数据对象的程序处理的分析。特别地,需要一种编译计算机代码的方法,此代码涉及产生一个数据流图并在该数据流图上执行静态数据流分析来确定对于程序中数据对象而言合适的创建点,锁定点和撤消点。更特别地,需要一个执行与这样一个程序相关联的信号流图分析并确定在运行时是否发生数据对象误差的编译程序。

发明概述

考虑到上述的局限性和缺点,提供一种XLANG/s编译方法,它用一个程序信号流图的数据流分析来确定其诸如变量那样的数据对象的使用期。根据该方法,根据抽象的计算机指令来创建一个信号流图,其中该信号流图具有多个基本程序块和一个数据对象,而且其中从计算机代码所形成的一个语义分析树中翻译出该抽象指令。对基本程序块指定一个深度优先次序并确定基本程序块两两之间的支配关系。确定关于信号流图内是否存在有循环,如果存在有循环,标识该循环。还确定关于数据对象的应用,并根据该应用,标识的循环,支配关系和多个基本程序块的深度优先次序,确定关于数据对象的一个创建点,撤消点和锁定点。在计算机代码中插入指令以在创建点创建数据对象,在撤消点撤销数据对象和在锁定点锁定数据对象。

附图的简要说明

连同附图来阅读将更好地理解上述摘要和以下优选实施例的详细描述。为了举例说明本发明,在附图中显示了本发明了示范性实施例;然而,本发明并不局限于所公开的特定方法和工具。在附图中:

图1是一个显示可以实现本发明的特征的示范性计算环境的框图;

图2是一个显示可以实现本发明的特征的示范性网络计算环境的框图;

图3是一个显示根据本发明一个实施例的一个示范性分解的框图;

图4是一个举例说明根据本发明一个实施例的一种示范性XLANG/s代码编译方法的流程图;

图5是一个显示根据本发明一个实施例的抽象指令的示范性信号流图的框图;

图6A—F是显示根据本发明一个实施例的基本程序块的示范性信号流图的

框图;

图7是一个显示根据本发明一个实施例的带有分配给每一个基本程序块的深度优先编号的示范性信号流图的框图;

图8是一个显示根据本发明一个实施例的带有赋值循环的一个示范性信号流图的框图;

图9是一个举例说明根据本发明一个实施例的一种为一个数据对象计算一个或多个创建点的示范性方法的流程图;

图10是一个举例说明根据本发明一个实施例的一种为一个数据对象计算一个或多个撤消点的示范性方法的流程图; 以及

图11A—B是一个举例说明根据本发明一个实施例的一种为一个数据对象设置锁定点的示范性方法的流程图。

示意性实施例的详细说明

概述

提供一种把XLANG/s代码分解成一个或多个表示代码的逻辑操作的信号流图的XLANG/s编译方法。例如, 该方法在信号流图中标识诸如变量那样的一个数据对象, 并跟踪在该数据对象上所执行的操作。该方法还尽可能迟至在对象的第一次使用之前才在XLANG/s代码内的一个最佳位置为这样一个对象设置一个创建点。另外, 该方法尽可能早在这种对象的最后使用之后为这种数据对象设置一个撤消点。该方法进一步在一个最佳位置上为一个共享数据对象设置一个锁定点: 在避免死锁代码的位置上并尽可能接近于这种对象的第一次使用之前。

示范性计算环境

图1举例说明一种其中可以实现本发明的适当计算系统环境100的一个实例。计算系统环境100仅仅是一种适当的计算环境的一个实例, 并不打算提出关于本发明使用范围或功能的任何限定。不应将计算环境100解释为具有任何涉及示范性运行环境100中举例说明的任何一个元件或元件组合的依赖性 or 必要性。

用其他大量的通用或专用计算系统环境或配置来运行本发明。可以适合于本发明所使用的众所周知的计算系统, 环境和 / 或配置的实例包括但并不局限于: 个人计算机, 服务器计算机, 手提式或膝上型装置, 多处理机系统, 基于微处理机的体系, 机顶盒, 可编程消费品电子技术, 网络个人计算机, 微型计

算机, 主机, 包括以上任何一个系统或装置的分布式计算环境等。

可以在计算机可执行指令的通用上下文中描述本发明, 例如由计算机执行的程序模块。通常, 程序模块包括执行特殊任务或执行特定抽象数据类型的例程, 程序, 对象, 部件, 数据结构等。还可以在其中由远程处理装置来执行任务的分布式计算环境中实践本发明, 这些远程处理装置通过一个通信网络或其他数据传输媒体进行链接。在一个分布式计算环境中, 可以在包含存储器装置的本地和远程计算机存储媒体中设置程序模块和其他数据。

参看图1, 一个用于执行本发明的示范性系统包括一个计算机110形式的通用计算装置。计算机110的部件可以包括但并不局限于: 处理单元120, 系统存储器130, 以及把包含系统存储器的多个系统部件连接到处理单元120的一个系统总线121。系统总线121可以是任何一种类型的总线结构, 这些总线结构包括一个存储器总线或存储器控制器, 一个外围总线, 以及使用任何一种总线体系结构的一个本地总线。举例来说但并不局限于此, 这种体系结构包括工业标准结构 (ISA) 总线, 微通道结构 (MCA) 总线, 扩展工业标准结构 (EISA) 总线, 视频电子标准协会 (VESA) 本地总线, 以及周边元件扩展接口 (PCI) 总线 (亦称板载总线)。

计算机110一般包括多种计算机可读介质。计算机可读介质可以是任何现有的可由计算机110存取的介质, 而且它包含易失性和非易失性介质, 可移动和不可移动介质。举例来说但并不局限于此: 计算机可读介质可以包含计算机存储介质和通信媒体。计算机存储介质包括以任何信息存储方法或技术来执行的易失性和非易失性, 可移动和不可移动介质, 这些存储的信息诸如计算机可读指令, 数据结构, 程序模块或其他数据。计算机存储介质包括单并不局限于: RAM, ROM, EEPROM, 闪速存储器, 或其他存储器技术, CD-ROM, 数字化视频光盘 (DVD) 或其他光盘存储器, 磁带盒, 磁带, 磁盘存储器或其他磁存储器装置, 或者其他可以用于存储所需信息并能够由计算机110存取的任何介质。通信媒体一般包含计算机可读指令, 数据结构, 程序模块或者在诸如载波或其他传送机制那样的一个调制数据信号中的其他数据, 而且通信媒体包括任何信息传送媒体。术语“调制数据信号”表示着这样一种信号, 它具有一个或多个自己的特征集合, 或者以这种方式进行变化来对信号中的信息进行编码。举例来说但并不局限于此: 通信媒体包括诸如有线网或直接连接的等的有线媒

体, 以及诸如声波, RF, 红外线和其他无线媒体那样的无线媒体。上述的任何组合应该包括在计算机可读媒体的范围之内。

系统存储器130包括以诸如ROM131和RAM132那样的易失性和 / 或非易失性存储器形式的计算机存储介质。在ROM131中一般存储了基本输入 / 输出系统133 (BIOS), 基本输入 / 输出系统133 (BIOS) 包含有助于在计算机110内的元件之间传送信息的基本线程, 例如在启动时。RAM132一般包含可由处理单元120立即存取的和 / 或立刻运行的数据和 / 或程序模块。举例来说但并不局限于此: 图1举例说明操作系统134, 应用程序135, 其他程序模块136, 和程序数据137。

计算机110还可以包括其他可移动 / 不可移动的, 易失性 / 非易失性的计算机存储介质。仅举例来说, 图1举例说明一个从不可移动、非易失性磁介质读数据并往不可移动、非易失性磁介质中写数据的硬盘驱动器140, 以及从可移动、非易失性磁盘152读数据并往可移动、非易失性磁盘152中写数据的磁盘驱动器151, 以及从诸如CD-ROM或其他光介质那样的可移动、非易失性光盘156读数据并往可移动、非易失性光盘156中写数据的光盘驱动器155。其他可以用在示范性操作环境中的可移动 / 不可移动的, 易失性 / 非易失性的计算机存储介质包括但并不局限于: 磁带盒, 闪速存储器卡, 数字化视频光盘, 数字视频磁带, 静态RAM, 静态ROM等。硬盘驱动器141一般通过诸如接口140那样的一个不可移动存储器接口连接到系统总线121, 而且磁盘驱动器151和光盘驱动器155一般通过诸如接口150那样的一个可移动存储器接口连接到系统总线121。

以上讨论的并在图1中举例说明的驱动器以及它们相关的计算机存储介质提供存储计算机可读指令, 数据结构, 程序模块和用于计算机110的其他数据。例如, 在图1中举例说明硬盘驱动器141存储了操作系统144, 应用程序145, 其他程序模块146和程序数据147。注意, 这些部件可以与操作系统134, 应用程序135, 其他程序模块136和程序数据137相同, 或者与它们不同。这里对操作系统144, 应用程序145, 其他程序模块146和程序数据147提供不同的编号来示意他们是不同的拷贝。用户可以通过诸如键盘162和指示设备161那样的输入设备在计算机110中输入命令和信息, 通常认为该指示设备是一个鼠标, 轨迹球或触摸垫。其他输入设备 (未显示) 可以包括一个麦克风, 游戏杆, 游戏垫, 卫星天线, 扫描仪等。这些输入装置和其他输入装置经常通过连接到系统总线上的一个

个用户输入接口160连接到处理单元120，但是可以通过其他接口和总线结构连接，诸如一个并行端口，游戏端口或一个通用串行总线（USB）。还可以经由诸如视频接口190那样的一个接口将监视器191或其他类型的显示装置连接到系统总线121。除了监视器以外，计算机还可以包括其他诸如扬声器197和打印机196那样的外围输出装置，并可以通过一个输出外围接口190连接它们。

计算机110可以运行于使用逻辑连接到诸如远程计算机180那样的一个或多个远程计算机的网络环境中。远程计算机180可以是个人计算机，服务器，路由器，网络PC，对等设备或其他通用网络结点，远程计算机180一般包括上述多个或全部与计算机110有关的元件，尽管在图1中仅仅举例说明了一个存储器装置181。所描述的逻辑连接包括一个局域网（LAN）171和一个广域网（WAN）173，但是还可以包括其他网络。这种联网环境在办公室，企业广域计算机网络，内部网和国际互联网中是常见的。

在LAN联网环境中使用的时候，计算机110通过一个网络接口或适配器170连接到LAN171。在WAN联网环境中使用的时候，计算机110一般包括调制解调器172或其他用于在诸如国际互联网那样的WAN173上建立通信的装置。可以经由用户输入接口160或其他适当的机制将调制解调器172连接到系统总线121，该调制解调器可以是内置调制解调器或外置调制解调器。在联网环境中，可以在远程存储器装置中存储所描述的与计算机110或其部分有关的程序模块。举例来说但并不局限于此：图1举例说明远程应用程序185常驻在存储器装置181上。将会理解到所示网络连接是示意性的，而且可以使用其他在计算机之间建立通信链路的方法。

示意性分布式计算框架或体系结构

多种分布的计算体制已经或正在按照个人计算和国际互联网的趋同现象进行发展。为应用程序和计算装置提供给个人和商业用户一个整压地互操作性的和启动网络的接口，逐渐地激活网络浏览器或面向网络的计算。

例如，微软的.NET平台包括服务器，诸如基于网络的数据存储器那样的积木式服务，以及可下载的装置软件。一般来说，.NET平台提供（1）使全程计算装置合作的能力以及具有自动更新用户信息并在所有这些信息上进行同步的能力，（2）增强的网络站点交互能力，通过大量使用XML而不是HTML来激活，（3）联机服务，其特征在于为了管理多个诸如电子邮件那样的应用程序或诸如

Office.NET平台那样的软件而从一个中心起始点存取并投送产品和服务到用户，

(4) 集中式数据存储，它不但将会提高效率并解除对信息的存取，而且还同步用户和装置中的信息，(5) 整合多个诸如电子邮件，传真和电话那样的通信媒体的能力，(6) 对于开发者而言，创建可再用模块的能力，从而提高生产率并减少程序设计错误的数量，而且(7) 以及其他许多跨平台的集成特征。

当这里所描述的示范实施例与保存在一个计算装置中的软件有关时，还可以经由一个操作系统，API或在一个协同处理器和请求对象之间的一个“中间人”对象来执行本发明的一个或多个部分，因此可以经由所有.NET语言和服务来执行，支持或存取服务，同时在其他分布式计算机制中。

示范性实施例

网络服务介绍

商业活动更频繁地经由分布式环境进行交互作用，诸如国际互联网或万维网。例如，消费者也许想知道对于一次预定旅行的租车价格。消费者可以在国际互联网上通过诸如旅行网站这样的中间商询问价格。中间商刚一收到消费者的询问就向多个租车商行发送报价询问。在接收到来自租车商行的答复之后，中间商向消费者发送该答复。然后消费者可以经由中间商预定一辆车并为该预定付费。使用信息来执行上述商务处理。例如，把消费者对中间商的询问当作一个给中间商的电子信息来执行，该电子信号包含诸如租用日期和时间，汽车类型，所需附加保险，消费者姓名与地址，信用卡信息和 / 或等所计划的汽车租赁的详细情况。中间商的报价询问是另一个信息，就如同来自租车商行的答复——从租车商行到中间商并从中间商到消费者——而且最终由消费者发送预定。

用于自动操作商务处理和用来执行该处理的信息是XLANG / s，以下更加详细地进行描述。可以理解，使用XLANG / s来执行的处理过程进行无误差的适当执行是重要的。另外，这种处理过程应该是加强到足以补偿外部误差的，诸如通信链路故障等。本发明一个实施例的操作在于提供一种程序设计和编译XLANG / s代码的方法以一种加强且无误差的方式执行这些处理过程。

XLANG / s介绍

XLANG/s是这样一种语言，它描述商务处理的逻辑序列以及通过使用多个技术部件或服务来执行商务处理。在版本0.55， Copyright©Microsoft1999-2000

上一个题为XLANG/s Language Specification的文献和Satish Thatte在Copyright© Microsoft Corporation 2001 上一个题为XLANG/s Web Services For Business Process Design的文献中所公开的XLANG/s比在此所公开的XLANG/s要更为详细，在此引入上述两篇文献的全部内容以供参考。把XLANG/s语言表示为可拓展标记语言。如上所述，可以用软件代码来执行商务处理。XLANG/s语言是一种用来描述商务处理和协议的现代的，特定领域的专用语言。XLANG/s还是一种说明性语言，这意味着它定义一个清楚的指令集，该指令集描述并执行商务处理中的步骤，步骤之间的关系，以及它们的语义和交互。XLANG/s代码不仅仅是说明性的；它还设计成是可执行的。由于XLANG/s的说明性的特性及其特定语义，所以产生的可执行码是确定性的；也就是说，由XLANG/s指令的语义集合对商务处理运行状态进行严格定义。因此，通过检验XLANG/s代码，可以确定由这种代码所执行的商务处理。定义这样一个可执行形式的商务处理为“组织”。

XLANG/s与许多因特网标准兼容。XLANG/s被设计为使用XML, XSLT (<http://www.w3.org/tr/xslt>), XPATH (<http://www.w3.org/TR/xpath>), XSD (XML方案定义) 和WSDL (网络服务描述语言) 作为提供的标准，而且它包含了对基于对象和信息的.NET进行合作的支持。由微软和IBM研究，(Copyright©2000 Ariba, 国际商业机器公司)，微软在2001年1月万维网附注上的一篇题为Web Services Description Language(WSDL)1.1的文献对WSDL进行描述，在此引用其全部内容以供参考。XLANG/s语言在语句构成上类似于C#，因此可以引用一个C#规范作为理解正确文法的一个帮助。XLANG/s内所包含的语义是微软，IBM和BEA为了定义商务处理语义而在日期2003年3月31日在版本1.1上公开的一篇题为Business Process Execution Language for Web Services中所定义的语义的反映。在这里也引用其全部内容以供参考。一般引用关于网络服务规范的商务处理执行语言作为“BPEL4WS”规范。因此可以理解，在把XLANG/s应用到一个商务处理的时候，该XLANG/s的使用是最有优势的。

XLANG/s定义了丰富的组用于定义一个商务处理的高级结构。XLANG/s指令一般属于两个类别中的一个：例如诸如接收或发送那样的对它们自己起作用的简单指令，以及包含或分组简单指令的复杂指令和/或其他复杂指令。XLANG/s还支持诸如字符串和Int32（整数）那样的低级数据类型。例如，还把

高级数据类型定义为诸如信息，端口（发送和接收信息的地方），相互关系和服务链接。这些数据类型还用来严格定义与商务处理相关联的语义，而且由诸如“时间”，“范围”那样的过程控制指令来对这些数据类型进行反码操作。

如上所述，XLANG/s服务器通过发送和/或接收信息来与外界进行通讯。信息类型是这种信息的结构定义。通过操作（例如接收，响应）来对信息起作用，而且操作也许是一个单个异步信息或一对请求-响应信息。操作不是输入就是输出。例如，销售者通过经由一个输入信息来接受一个订货单（来自潜在的顾客），可以开始一个交互——对一种服务/产品进行报价。然后，如果可以实现该订单，销售者可以对购买者返回一个回执。销售者可以发送附加信息给购买者（例如装运通知单，发货单）。一般地，这些输入和输出操作按照引用为一个服务过程的一个预定序列发生。销售者的服务器分别记录每一个订货单的交互状态和其他类似的交互。这在购买者可以同时同销售者进行多个订货过程的情况下是特别便利的。而且，可以在后台执行操作服务过程的每一个实例（例如更新存货清单，更新账户余额），而不用激活一个输入操作。

一种服务处理可以表示运用若干操作的一种交互。因而该交互具有一个严格定义的开始和结束。引用该交互作为服务的一个实例。可以以两种方式中任何一种方式来开始一个实例。可以用一些特殊执行功能来明确地例示一个服务，或者可以连同一个按照自己特性的操作来暗中例示一个服务，该操作意指一个例示操作。当定义其特性的处理过程终止时一个服务实例也终止。

根据拓展的交互历史来例示服务器运行。不仅仅把发送到这种服务器的信息投送到正确的目的端口，还把它们发送到定义该端口的服务器正确实例。一个端口是末端，其中由服务器发送和接收信息。充当服务器主机的基础结构支持该路由选择，因此避免负担每个根据执行一个用于实例路由选择的常规机制而执行的服务器。

现在转到图2，举例说明一种用于启动两个商务实体之间的通信的单一化计算机网络。例如，借助于通信链路222有效地将第一计算机220连接到网络210，第一计算机220是诸如以上连同图1所公开的计算机110那样的任何类型计算装置，如专用计算机等。网络210可以是任何类型用于使大量计算装置相互连接的网络，而且它可以是一个企业内部互联网，国际互联网等。通信链路222可以包含任何类型通信媒体，无论有线的，无线的，光学上的等等，第二计算机230（

如同第一计算机220)可以是任何类型的计算装置,而且可以借助于通信链路232将第二计算机有效地连接到网络210。通信链路232(如同通信链路222)可以是任何类型的通信媒体。在一个实施例中,通信链路222,232是相同类型的通信媒体,而在另一个实施例中,每一个通信链路222,232所采用的媒体是不同的。可选择地,可以有效地将附加计算机224,226连接到第一计算机220。可以理解,同样可以有效地将附加计算机有效地连接到第二计算机230(为了清楚而没有在图2中显示)。

例如,考虑一个通常的供给链路情况,其中购买者向销售者发送订货单。例如,购买者借助于网络210和通信链路222,233从第一计算机220发送信息到销售者的第二计算机230。假设购买者与销售者具有一种坚固的商业关系,而且静态设定为他们把涉及购置交互的文本发送到与相应端口相关联的URL。当销售者为此订单返回一个回执时,此回执被送往计算机220上购买者端的正确服务器实例,或选择性地前往另一个诸如附加计算机224,226那样的计算机。传送订单信息中所插入的一个标记(例如,cookie)是这样一种路由选择的执行方式,在回执之内复制该标记用于交互作用。该标记可以是标题中的信息包络或在商务文件(订货单)本身内的信息包络。可以在操作说明书中对每一个信息内标记的结构和位置进行说明性表示。该说明性信息允许XIANG / s适应的基础结构用标记自动提供实例路由选择。

在其生存期内,一个服务实例一般可以保持与其他服务实例的一个或多个对话,该其他服务实例表示交互中所包含的其他参加者。对话可以使用一种高级传送基础结构,该传送基础结构与一个对话中所包含的信息相关并把它们送往正确的服务实例。多数情况下,相关的对话可以包含参加者多于两个,或者可以使用带有直接插入到正在改变的商务文件中的相关标记的轻度传送基础结构。XIANG / s通过提供一个完全通用机制在一个服务实例之内执行操作的相关组来取址相关方案。可以将一组相关标记定义为一组为相关组中所有信息共享的属性。将这样一组属性称为相关集合。

本发明实施例的描述

在XIANG / s代码中执行诸如数据类型,标记,相关集合等那样的特征,使用变量来存储对执行这种特征而言所必需的值。通常就是这样连同其他高级软件语言,在运行时间之前由编译程序来编译XIANG / s。本领域技术人员应该

知道与编译处理相关的细节，因此在此不进行详细描述。执行本发明一个实施例作为部分XIANG / s代码编译过程，这是在完成编译过程的一些步骤之后。例如，在此假设程序员已经编写了一个商务处理的文本描述，其中该文本描述符合诸如XIANG / s那样的商务处理语言的语法和规范。在此还假设由一个分析器读取该文本程序，这意味着在文本表示中不存在文本错误或语法错误，其中该分析器确保输入是合式的，然后分析器将文本表示转换成标准的高级语义分析树表示。最后，在此假设解析该高级语义分析树以产生一系列低级抽象指令。当以固有顺序执行这些指令时，这些指令将执行文本表示中所表示的商务处理编写器的操作。

如上所述，编译过程中上述步骤的输出是一系列低级抽象指令。然后使用这些指令作为对于本发明一个实施例的输入。可以理解，附加的步骤或过程可以发生在初始化本发明一个实施例之前，而且同样地，可以合并以上一些步骤或所有步骤。任何这种情况同样与本发明一个实施例相适应。以下根据图3讨论先前所讨论的为一个实施例产生输入的处理过程的一个实例。可以理解，为了清楚而缩写在此所讨论的实例，而且该实例在大多数情况下包括大量的文本代码，以及因而的更庞大的语义分析树和更长的结果指令集合。

首先，提供一个商务处理的文本描述。如上所述，该描述尽可能与所使用的诸如XIANG / s那样的商务处理语言的语法和规范相一致。将会理解到它描述了最终将由按照本发明一个实施例的全部编译过程所产生的程序来执行的商务处理的特征。商务处理的一个文本描述的一个示范性片段可以如下所示：

```
receive(myPort,msgAmout);  
balance=myAccount.Withdraw(msgAmount);  
if(balance<0){  
    msgResult="overdrawn";  
}else{  
    msgResult="okay";  
}  
send(myPort,msgResult);
```

可以理解，以上片段仅仅是示范性的，因为商务处理的一个文本描述可以描述能够由商业软件执行的这种商务处理的一个特征。该特定片段描述一个商

务处理，该商务处理包含从一个帐号取走资金的请求。特别是，参考以上的文本描述，在端口myPort上接收一个信息。该信息包含从一个帐号取走一个总金额的指令，将此帐号标记为myAccount。如果提取后的余额小于零，对一个结果信息赋值“超支”以表示帐号资金不足。否则，对该结果信息赋值“OK”。最后，把该结果数据作为一个响应而发送返回到原始信息的发信人。

第二，如上所述，分析器读取以上公开的文本表示并创建一个语义分析树，该语义分析树捕获文本中所包含的形式和结构。现在返回到图3，提供一个基于上述文本描述的示范性语义分析树300。在图3中可以看到这一点，文本描述的逻辑结果因语义分析树300的布置图而变得明显。将会理解到并不需要以图形方式创建语义分析树300，实际上语义分析树300可以是作为在此所公开的全面编译过程的一部分而创建的一个短暂和内部的产品。

语义分析树300包含表示文本描述的逻辑结构的程序块。例如，程序块305表示receive(myPort,msgAmount)指令，然后通向程序块303，程序块303轮流表示balance=myAccount.Withdraw(msgAmount)指令。在图3中可以看到这一点，逻辑上比以上接收指令更为复杂的balance=myAccount.Withdraw(msgAmount)指令需要在程序块303内进一步逻辑表示。程序块305—313表示这样一种逻辑，该逻辑涉及在根据文本描述的指令撤销msgAmount之后，确定myAmount的余额。可以理解，用来表示一个文本描述的逻辑结构的抽象图形表示可以遵循任何规则，因为本发明一个实施例同样与任何这种条约相兼容。

程序块315表示if(balance<0) 指令的“如果”部分。程序块317连同程序块319—323，根据以上if(balance<0)指令来表示这样一种涉及确定msgAmount的余额是否小于零的逻辑。程序块325和333表示根据所确定的与程序块317有关的结果来产生一个信息的逻辑。例如，如果程序块317中所确定的余额小于零，根据程序块325以及内部程序块327—331产生一个信息。可以理解，程序块325—331表示以上文本描述的msgResult="overdrawn"指令。如果在程序块317中所确定的余额大于零，根据程序块333以及内部程序块335—339产生一个信息。可以理解，程序块333—339表示msgResult="okay"指令。最后，程序块341表示send(myPort,msgResult)指令，该指令描述发送根据程序块317—323的余额确定而产生于程序块325—331或程序块333—339中的一个信息。

然后将语义分析树转化成一系列抽象指令，在执行这些指令的时候执行商

务处理。

- 1) receiveInstruction[端口=myPort,信息=msgAmount]
- 2) ExpressionInstruction[余额=myAccount.Withdraw(msgAmount)]
- 3) JumpIfFalseInstruction[如果(balance<0)为失败,转到步骤6]
- 4) ExpressionInstruction[msgResult="overdrawn"]
- 5) JumpInstruction[转到7]
- 6) ExpressionInstruction[msgResult="okay"]
- 7) SendInstruction[端口=myPort,信息=msgResult]

可以理解, 以上所列指令仅代表一系列指令, 作为可以采用任何形式以及同样与本发明一个实施例相一致的任何形式的一个有效列指令。另外, 在与本发明一个实施例相一致的同时, 并不需要在用于本发明一个实施例中的抽象指令中出现以上的编号和括弧内的注释说明。如上所述, 然后本发明一个实施例将以任何形式出现的抽象指令用作输入。

现在参看图4, 流程图举例说明根据所示本发明一个实施例的一种编译 XIANG / s 代码的方法。如图4所示, 尽管方法400包含步骤405—440, 但是简单提一下, 并不是所有步骤405—440都需要出现在本发明的一个实施例中。在步骤405, 使用诸如以上连同图3所讨论的抽象指令那样的抽象指令来创建一个信号流图。以下根据图6A—F讨论与创建示范性信号流图有关的细节。在步骤410, 对步骤405中所创建的信号流图的基本程序块进行排序, 并且在程序块415中创建每一个基本程序块的支配顶点, 以下将根据图7对此进行详细的描述。在步骤420中创建循环, 以下将根据图8对此进行详细的描述。在步骤425, 分析每一个基础程序块内的数据对象应用。如上所述, 一个数据对象可以是一个变量, 标记等。在步骤430, 根据以下要详细讨论的图9中的方法确定一个或多和创建点。在步骤435, 根据以下图10中的方法确定一个或多个撤销点。最后在步骤440, 根据以下要讨论的图11A—B中的方法, 为每一个共享数据对象创建一个或多个锁定点。

创建信号流图

在本发明一个实施例中, 通过将指令列划分成基础程序块来构建一个信号流图, 然后将每一个基本程序块连接到它的直接后继块和直接先趋块。一个基本程序块是一系列连贯的程序设计语句或指令, 其中控制流在第一指令进入并

在最后指令离开，只有在程序块的末端才有停止和任何分支的可能性。在Aho,Sethi和Ullman的Compilers:Principles,Techniques,and Tools中可以发现关于基本程序块的附加参考，在此引用其全文以供参考。与本发明一个实施例相一致的一个示范性算法是在Aho,Sethi和Ullman的Compilers:Principles,Techniques,and Tools中的算法9.1。因此，举例来说，本发明的一个实施例把算法9.1应用于上述抽象指令1—7。因此，在上述实例中，基本程序块是{1, 2, 3}, {4, 5}, {6}和{7}。

查找后继和先趋

一旦已经把该类抽象指令或“指令流”划分为基本程序块，那么将该程序块双向连接到他们的后继和先趋。一个第一基础程序块的后继是任何简单的程序块，可以由第一基本程序块中的最后指令来延伸到这些基本程序块的第一指令。因此，在上述实例中，基本程序块{1, 2, 3}中的最后指令是JumpIfFalse指令。如果状态为错误的，则定义该指令来执行一个跳跃到另一个指令；否则，处理接下来相继的指令。因此，{1, 2, 3}的后继是{4, 5}和{6}。如同名称上所暗示的，先趋与后继正好相反。因此，在现有实例中，{4, 5}的先趋是{1, 2, 3}。

现在返回到图5，举例说明以上所列抽象指令的一个示范性信号流图。程序块501表示基本程序块{1, 2, 3}，程序块505表示基本程序块{4, 5}，程序块507表示基本程序块{6}以及程序块509表示基本程序块{7}。实线表示每一个基本程序块之间的后继关系，而虚线表示先趋关系。例如，用实线将程序块505和507分别连接到程序块509，从而表示基本程序块{7}是基本程序块{4, 5}和{6}的一个后继。同样地，用一个虚线将程序块509连接到程序块505和507，从而表示基本程序块{4, 5}和{6}是基本程序块{7}的先趋。

关于事务处理的信号流图

XIANG / s中的一个事务处理包含子处理，它显示事务处理特性。因此，包含在一个XIANG / s事务处理内的一个处理过程要么全面完成要么全面失败。可以理解，为一个特殊的XIANG / s事务处理的成功或失败作的打算允许程序员创建加强到足以补偿误差的商务应用程序。例如，一个事务处理可以是原子的或长期运行的。

一个原子的XIANG / s事务处理是这样一种软件事务处理：除非该事务处

理成功完成，否则在该事务处理中不发生反应而且不改变程序状态。在一定意义上它是原子的，成功完成和根本什么都不做，也就是说它没有“提交”。如同名称所暗示的，一个长期运行的XLANG / s事务处理是在对计算目的有意义的一个时期上发生的一个软件事务处理。在这样一种事务处理过程期间的任何点上可以出现一个异常。当一个异常出现在或施加于一个长期运行的事务处理中时，编译程序不能够是已经成功执行了的事务处理内的特定指令。

不管它的类型，每一个事务处理包含一个主体处理过程，零或多个异常处理程序过程和一个可选的补偿过程。可以理解，一个原子的事务处理不具有一个异常处理程序；然而，一个原子的事务处理可以是能够重发的。一个异常处理程序过程是一个子处理，该子处理运行于每当事务处理主体中的任何指令出现一个异常时，其中为此类异常设计了程序处理，或者运行于每当事务处理异常终止时。补偿过程是一个子处理过程，该子处理过程可以依赖于程序的配置，运行于一个事务处理为了使在事务处理主体中所执行的操作“失效”而确认之后。因此，可以理解，在一个实施例中事务处理是可以嵌套的，而且如果一个内部事务处理提交而一个外部事务处理异常终止的话，外部事务处理可以调用内部事务处理的补偿处理程序。

以下连同图6A-F所讨论的是对根据本发明一个实施例的事务处理信号流程图进一步举例说明的几个实际例子。与该讨论有关，用符号Px来表示一个子处理过程。在该符号中，下标x是子处理过程P的一个标识符。为了清楚，在此连同图6A-F假设所有子处理过程都包含一个单个基础程序块，尽管将会理解到可以执行本发明一个实施例来处理任意的复杂子处理过程。同样地，还将会理解到以下连同图6A-F所使用的代码实例仅仅是示范性的，所使用的这种与本发明一个实施例相关的代码可以是任何长度或复杂度的。

长期运行的事务处理

XLANG/s事务处理最简单的形式是一种无异常处理程序和补偿的长期运行的事务处理。例如，这样一种事务处理的文本描述由以下高级代码组成：

```
Pstart
Scope longrunning transaction LRT{
  body{
    Pbody
```

```

}
}

```

Pend

这交替地产生以下抽象指令流:

- 1) { Pstart }
- 2) BeginTransaction[主体@步骤3]
- 3) { Pbody }
- 4) CommitTransaction[再回到步骤5]
- 5) { Pend }

再一次如上所示, 该抽象指令流包括为了便于讨论而在此插入的编号和括弧的注释说明。因此, 和本发明一个实施例一致时是不需要这种编号和注释说明的。专用抽象指令BeginTransaction有事务处理主体处理过程Pbody作为其后继, 而且如果它们中任何一种出现在现有实例中的话, 则所有异常处理过程进行处理。专用抽象指令CommitTransaction有接下来连续的指令作为其后继, 而且如果其中某个指令出现在现有实例中的话, 事务处理的补偿则进行处理。

因为以上两种专用抽象指令通常导致一个跳跃或分支到代码的另一个区域中的一个指令而打断程序的直线处理, 所以这种抽象指令标记一个基本程序块的末端。因此, 以上片段具有三个基本程序块: {1, 2}, {3, 4}和{5}。现在返回到图6A, 举例说明上述基本程序块的一个信号流图。如同Pstart指令所表示的, 程序块601表示抽象指令BeginTransaction, 如同Pbody指令所表示的, 程序块603表示抽象指令CommitTransaction, 而且如同Pend所表示的, 程序块605表示事务处理的末端。箭头表示从程序块601经过程序块603到程序块605的程序流程。将会理解到在现有的单个实例中, 把信号流图分为三个程序块601-605对本发明一个实施例而言不是必需的, 因为按顺序处理这些指令不需要任何分支或补偿。

原子事务处理

现有实例举例说明一个实施例可以怎样处理一个原子事务处理, 该事务处理不同于长期运行的事务处理。可以理解, 以下的文本描述类似于以上连同图6A所讨论的长期运行的事务处理的文本描述。

Pstart

Scope atomic transaction AT{

```
Body{  
  Pbody  
}  
}  
Pend
```

这交替地产生以下抽象指令流，它也类似于以上连同图6A所讨论的长期运行的事务处理的抽象指令流：

- 1) { Pstart }
- 2) BeginTransaction[主体@步骤3]
- 3) { Pbody }
- 4) CommitTransaction[再回到步骤5]
- 5) { Pend }

在现有原子的事务处理实例中，CommitTransaction包含与以上连同图6A所描述的指令有略微不同的指令。如果事务处理提交，控制流进行到接下来连续的指令，如果事务处理异常终止，则控制流循环返回到事务处理的开始。因此可以理解，由于一个原子的事务处理能够在故障上进行重新操作，所以它形成一个隐式循环。现在返回到图6B，举例说明上述事务处理的信号流程图。如同图6A中的情况，如同Pstart指令所表示的，程序块601表示抽象指令BeginTransaction，如同Pbody指令所表示的，程序块603表示抽象指令CommitTransaction，而且如同Pend所表示的，程序块605表示事务处理的末端。没有标记的箭头表示从程序块601经过程序块603到程序块605的常态下无故障的程序流程。箭头A表示以上所述的由CommitTransaction指令形成的隐式循环。

可以理解，当确定与循环所影响的数据对象相关的撤消点等时，考虑虚箭头A所表示的隐式循环是非常重要的。例如，如果在事务处理Pbody中出现数据对象v最后的词法使用，那么在事务处理提交之前都不能撤消数据对象v。如果在Pbody中撤消了数据对象v，而且如果异常终止或重新操作该事务处理，那么在重新操作的过程中将丢失程序对象v的值。因此，如同以下将连同图10所讨论的，在确定撤消点的过程中考虑隐式循环的存在。在现有实例中，本发明的一个实施例将把v的撤消移至包含Pend的基本程序块之内。

带有异常处理程序的长期运行的事务处理

本发明举例说明在一个长期运行的事务处理中执行的XLANG/s异常处理程序的处理过程。如如上所述，原子的事务处理并不带有异常处理程序。例如，这样一个事务处理的一个文本描述可以由以下高级代码所组成：

```
Pstart
Scope longrunning transaction LRT{
  Body{
    Pbody
  }
  exceptions{
    catch{
      Pex
    }
  }
}
Pend
```

直接转到图6C，为了清楚而省略现有实例中的抽象指令流，举例说明上述信号流图。再一次如同图6A和6B中的情况，如同Pstart指令所表示的，程序块601表示抽象指令BeginTransaction，如同Pbody指令所表示的，程序块603表示抽象指令CommitTransaction，而且如同Pend所表示的，程序块605表示事务处理的末端。另外将简要提一下，程序块607表示与异常处理程序相关联的抽象指令Resume。没有标记的箭头再一次表示从程序块601经过程序块603到程序块605的常态下无故障的程序流程。

该信号流图假定一个异常能够在输入事务处理主体之后立即发生；因此，主体和异常处理程序都能够在逻辑上跟随BeginTransaction抽象指令。没有标记的实箭头跟随着表示正常程序流程的程序块601，与此同时，虚箭头A表示在一个异常事件中所采用的逻辑路径。程序块607包含专用抽象指令Resume，它设计为在事务处理之后跳跃到下一个指令。因此，当程序块607所表示的异常处理程序的处理过程完成时，采用虚箭头B所表示的逻辑路径进行到程序块605处的Pend。可以理解，程序块603的主体和程序块607的异常处理程序流向相同的后继，即程序块605的Pend。

带有补偿的长期运行的事务处理

现有实例举例说明在一个长期运行的事务处理中执行的一个XLANG/s补偿的处理过程。如上所述，补偿处理程序是XLANG/s代码模块，该代码模块试图使一个由于误差等而无法成功完成的事务处理失效或对其进行补偿。这样一种补偿处理程序可以执行程序员所需的任务，例如执行一个失败的事务处理相反的功能，执行一个使失败的事务处理的影响无效的功能等。例如，诸如这样一种事务处理的一个文本描述可以由以下高级代码组成：

```
Pstart
Scope longrunning transaction LRT{
body{
Pbody
}
compensateion{
Pcomp
}
}
Px
Pend
```

现在转到图6D，举例说明上述信号流图。再一次如同图6A—C中的情况，如同Pstart指令所表示的，程序块601表示抽象指令BeginTransaction，如同Pbody指令所表示的，程序块603表示抽象指令CommitTransaction，而且如同Pend所表示的，程序块605表示事务处理的末端。程序块609表示补偿处理程序过程和以上连同图6C所讨论的抽象指令Resume。如同Px所表示的，程序块611表示在Pbody之后任何额外的程序。没有标记的箭头再一次表示从程序块601经过程序块603和611到程序块605的常态下无故障的程序流程。

虚箭头A表示在一个需要补偿的程序状态的事件中所采用的逻辑路径。如同在图6D所能看到的，如同Pcomp指令所表示的，程序块609表示补偿处理程序。将会理解到，在图6D的信号流图中设置程序块609说明在事务处理成功确认后能够到达该补偿程序块的代码，在程序结束之前该补偿程序块的代码仍然是可达到的直到如程序块605所表示的程序的末端。在图6D的信号流图中，通过借助于

虚箭头A使程序块609成为程序块603的CommitTransaction指令的一个可能的后继,并通过借助于虚箭头B使程序的末端程序块605成为程序块609的后继,从而实现程序块609的有效性。可以理解,在一个实施例中,补偿程序块609能够流向程序中先于程序块605的点,这提供了在程序块609所流向的点处不再需要程序块609所表示的补偿功能。

带有补偿的原子事务处理

可以理解,可以将带有补偿的一个原子的事务处理的信号流图描述为以上连同图6B所讨论的原子事务处理和连同图6D所讨论的带有补偿的长期运行的事务处理的一个组合。现在转到图6E,举例说明这样一种信号流图。程序块601—611的描述由以上连同图6D所讨论的实线未标记箭头和虚箭头A和B组成。可以理解,虚箭头C与以上连同图6B所讨论的虚箭头A相等,虚箭头A表示能够在CommitTransaction抽象指令的失败上重新操作的隐式循环。

带有异常处理程序和补偿的长期运行的事务处理

可以理解,可以将带有异常处理程序和补偿的长期运行的事务处理的一个信号流图描述为以上连同图6C所讨论的带有补偿的长期运行的事务处理和以上连同图6D所讨论的带有补偿的长期运行的事务处理的一个组合。现在转到图6F,举例说明这样一种信号流图。程序块601—611的描述和实线未标记的箭头如上图6E所述。可以理解,虚箭头A和B与以上连同图6C所讨论的虚箭头A和B相等,它表示在一个异常事件中的控制流,借此借助于虚箭头A将控制流传到程序块607的异常处理程序并返回到程序块611处的程序。可以理解,虚箭头C和D与以上连同图6D所讨论的虚箭头A和B相等,它表示涉及借助于虚箭头C而发生的在程序块609的补偿处理程序事件中的控制流,并借此借助于虚箭头D将控制流返回到程序块605处程序的末端。

准备信号流图

可以理解,例如,一旦创建一个诸如以上连同图6A—F所描述的一个或多个信号流图那样的信号流图,本发明的一个实施例则可以注释说明每一个基本程序块上的信息。根据本发明一个实施例,一个深度优先的检索试图以信号流图的一个深度优先次序分配一个表示一个基本程序块位置的号码。另外,在一个实施例中确定两个支配集。以这种确定形成自上的一组支配顶点和自下的一组支配顶点。此外,在一个实施例中,如果任何程序块参与的一个确定,那么

也形成一个或多个循环。

还可以理解,本发明的一个实施例可以为任何级别的抽象指令复杂度而创建如以上连同图6A—F所讨论的信号流程图。例如,可以顺序和/或并行地组合图6A—B的信号流程图的任何数字的组合。用这种方式,本发明的一个实施例可以处理并编译任何复杂度的代码。

深度优先次序

根据一个实施例,在信号流程图上执行一个深度优先检索并对信号流程图内所包含的每一个程序块分配一个相应的深度优先编号。可以理解,任何用于查找信号流程图的一个深度优先次序的算法同样地与本发明的一个实施例相一致。例如,如Compilers:Principles,Techniques,and Tools.中所描述的,其中一个这种算法是算法10.14。

将会理解到一个深度优先检索的特性确保程序中与程序末端相关的最后基本程序块接收最低的编号,与此同时,程序中与程序开端相关的第一基本程序块接收最高编号。现在参看图7,举例说明一个具有分配给每一个基本程序块的深度优先编号的示范性信号流程图。信号流程图700包含程序块705—720,对其中每一个程序块分配一个深度优先编号。从信号流程图700底部程序块720处开始,可见已经给位于信号流程图700所表示的程序末端的程序块720分配了一个编号0。分别分配一个编号1和2给程序块710和715,它们是信号流程图700的末端接下来的两个程序块。可以理解,由于程序块710和715距离信号流程图700相同的长度,所以可以在保持与本发明一个实施例相一致时反转所分配的编号1和2。最后,对距离信号流程图700最远的程序块705分配一个编号3。还可以理解,深度优先次序的使用允许基本程序块705—720的有效标识,例如一个或多个位组。

查找支配顶点

例如,使用Compilers:Principles,Techniques,and Tools.中所描述的算法

10.16双向确定所有程序块705—720的支配关系。如上所述,任何用于查找基本程序块的支配点的算法同样地与本发明的一个实施例相一致。

确定是否一个程序块预支配另一个涉及开始于顶部表示信号流程图700的控制流的开端的程序块,并确定是否所有通向特殊程序块的逻辑路径必须已经首先通过了一个先前的程序块。例如,在信号流程图700,程序块720的预支配顶点是720和750,将会理解到一个程序块支配它自己。还将会理解到程序块710和715

并不预支配程序块720，这是因为控制流在到达程序块720之前并不需要经过程序块710或715。例如，控制流可以绕过程序块715而经过程序块705到程序块710然后到720。同样地，控制流可以绕过程序块710而经过程序块705到程序块715然后到720。

确定是否一个程序块在后支配另一个涉及开始于底部表示信号流图700的控制流的开端的程序块，并确定是否所有通向特殊程序块的逻辑路径必须已经最后地通过了一个先前的程序块。例如，在信号流图700，程序块705的在后支配顶点是程序块705和720。

查找循环

现在参看图8，例如，本发明的一个实施例通过使用在Compilers:Principles,Techniques,and Tools中所描述的算法10.1在信号流图800中定位并标识循环。如图8所示，信号流图800是一个双重嵌套循环。如上所述，本发明一个实施例同样与具有任何复杂度或循环数的信号流图800相一致。一个编号条约用于标识每一个循环，例如对属于相同循环的所有基本程序块提供相同的循环标识符。该循环标识符可以是任何类型的标识符，例如具有这种属性的一个整数：内部循环比外部循环具有更高编号。在图8中可以看到程序块805和835不是循环中的一部分，因此不对它们提供一个循环标识符。还可以看到在程序块820和825之间形成的循环嵌套在程序块810，815和830所形成的一个循环之内。因此，根据本发明的一个实施例，为了表示程序块820和825参与两个循环，对程序块820和825分配一个循环标识符2，为了表示程序块810，815和830参与一个循环，对程序块810，815和830分配一个循环标识符1。

分析数据对象应用

因此，在图4中步骤420完成时，如同以上连同图8所详细描述，根据本发明一个实施例的方法转到步骤430，该步骤分析每一个数据对象的应用。如上所述，例如，数据对象可以是变量，数据对象等。例如，一旦已经制订诸如图8的信号流图800等那样的信号流图，就形成一个经过该信号流图的通道。在该信号流图的每一个基本程序块之内，检验抽象指令来确定正使用哪一个程序对象以及和正在如何使用该对象。每一个程序对象S因而包含两个集合：从该对象读出的基本程序块的集合Sread，以及写入到该对象的基本程序块的集合Swrite。可以理解，仅仅根据 $Sread \cup Swrite$ 形成对一个对象读或写的基本程序块的集合Suse。

如同以上连同图4和图6A—F所讨论的,由XLANG/s编译程序使用本发明一个实施例所产生的信号流程图来执行几个任务。例如,计算以及插入创建点,撤消点和锁定点是可以由本发明一个实施例执行的其中一些任务,而且以下要连同图9—11B对其进行详细的描述。

为一个对象计算创建点

现在转到图9,举例说明一种为一个数据对象计算一个或多个创建点的方法900。在步骤905,集中不是从一个数据对象读出就是往一个数据对象写入的基本程序块的一个集合Suse。可以理解,集合Suse等效于从一个数据对象读出的基本程序块的一个集合Sread和往一个数据对象写入的基本程序块的集合Swrite的总和。将会理解到可以假设在集合Suse内数据对象的第一次词法使用时创建中间数据对象。因此,在此可以把包含数据对象第一次词法使用的基本程序块指定为Bcreate。即,例如涉及数据对象的最高编号基本程序块,这是因为如同以上连同图7论述的,深度优先次序程序开端附近的基本程序块提供更高的编号。在一个实施例中,Suse内数据对象的每一次词法上的连续应用是每一个连续的低编号基本程序块Bnext。另外,在步骤908以对象数据的第一次词法使用为基础计算一个初始集合Bcreate。

在步骤910,按照本发明一个实施例所述的一个编译程序计算Bcreate的预支配顶点和Bnext的预支配顶点的交集I。在步骤915,确定有关集合I是否包含Bcreate。如果包含,在步骤920,中间数据对象的创建点保持Bcreate。如果不包含,在步骤925,从集合I选出一个新创建点Bcreate,因此新创建点的深度优先编号是最小的编号,它大于当前创建点的深度优先编号。因此,用发生在这种数据对象之前的数据对象的创建点以避免调用该对象且该对象没有值的话会导致一个误差。同样地,尽可能接近于该对象的第一次使用来创建该创建点以避免不必要地负担存储器资源。可以理解,如果新创建点与上述创建点在不同的循环内,那么在绕过所有内循环之前,成功地从集合I选择编号越来越高的程序块。

在步骤930,确定集合Suse内的数据对象的另一种使用是否存在,如果存在,方法900为此接下来的使用而返回到步骤910。一旦考虑数据对象所有的使用,方法900进行到步骤935。在步骤935,如同在步骤905—930中所确定的那样,在任何内嵌程序块使用之前,在基本程序块Bcreate内插入一个执行创建数据对象

和初始化数据对象的指令。

为一个对象计算撤消点

现在转到图10, 举例说明一种为一个数据对象计算一个或多个撤消点的方法1000。在步骤1005, 如上所述图9的步骤905的情况, 集中不是从一个数据对象读出就是往一个数据对象写入的基本程序块的一个集合Suse。可以理解, 集合Suse等效于从一个数据对象读出的基本程序块的一个集合Sread和向一个数据对象写入的基本程序块的集合Swrite的总和。在一个实施例中, 假设在集合Suse内数据对象的最后一次词法使用时撤消中间数据对象。因此, 在此可以把包含数据对象最后一次词法使用的基本程序块指定为Bdestroy。例如涉及数据对象的最低编号基本程序块。在一个实施例中, Suse内数据对象每一次词法上的上述使用是每一个连续的高编号基本程序块Bprev。因此, 在步骤1008以对象数据的最后一次词法使用为基础计算一个初始集合Bdestroy。

在步骤1010, 按照本发明一个实施例所述的一个编译程序计算Bdestroy的在后支配顶点和Bprev的在后支配顶点的交集I。在步骤1015, 确定有关集合I是否包含Bdestroy。如果包含, 在步骤1020, 中间数据对象的撤消点保持Bdestroy。如果不包含, 在步骤1025, 从集合I选出一个新撤消点Bdestroy, 因此新撤消点的深度优先编号是最大的编号, 它小于当前撤消点的深度优先编号。因此, 用发生在最后使用这种数据对象之后的数据对象的撤消点以避免不必要地负担存储器资源。可以理解, 可以理解, 如果新撤消点与上述撤消点在不同的循环内, 那么在绕过所有内循环之前, 成功地从集合I选择编号越来越低的程序块。

在步骤1030, 确定集合Suse内的数据对象的另一种使用是否存在, 如果存在, 方法1000为此接下来的使用而返回到步骤1010。一旦考虑数据对象所有的使用, 方法1000进行到步骤1035。在步骤1035, 如同在步骤1005—1030中所确定的那样, 在任何内嵌程序块使用之后, 在基本程序块Bdestroy内插入一个执行撤消数据对象和清除数据对象的指令。

为共享对象计算锁定点

可以理解, 锁定点的计算类似于计算创建点, 这是因为在尽可能迟至一个共享数据对象的第一次使用之前才设置锁定是合乎需要的。然而, 不能逐一地设置用于共享对象的锁定。以下的示范性XLANG/s代码提供一个当计算锁定点时出现的计算实例。

```
Parallel{
  task{
    ...
    scope synchronized{
      a=5;
      b=6;
    }
    ...
  }
  task{
    ...
    scope synchronized{
      b=8;
      a=9;
    }
    ...
  }
}
```

在以上示范性代码中，在每一个并行任务中，以相反的次序存取对象**a**和**b**。
在以下代码实例的粗字体文本中提供一种设置**improper**锁定的实例。

```
Parallel{
  task{
    ...
    scope synchronized{
      LockWrite(a);
      a=5;
      LockWrite(b);
      b=6;
    }
    ...
  }
}
```

```
}  
task{  
...  
scope synchronized{  
LockWrite(b);  
b=8;  
LockWrite(a);  
a=9;  
}  
...  
}  
}
```

如果以上代码能够无中断或无误差地执行，那么可以接受以上锁定设置。然而，如果第一任务进行得足够远以获取a的锁定，然后插入该任务，则可以导致一个死锁程序。例如，如果在以上插入后，第二任务进行到获取b的锁定，那么它将会变成锁住的，这是因为它不能获取a的锁定。如果在该插入后恢复第一任务，它将不能够进行，这是因为它现在无法获取b的锁定。因此死锁处理过程。

通过考虑在一个异步范围内所有的共享数据对象的使用点，并为在任何对象的第一次使用之前的所有这种对象设置锁定，现有一个实施例围绕该问题运行。通过以一种穿越所有异步范围的相容次序获取锁定而避免死锁。可以理解，如果这种次序是相容的，可以使用任何锁定次序。

使用以下要讨论的图11A—B的方法，以下由粗体字文本所表示的锁定设置将源自于以上代码实例。

```
Parallel{  
task{  
...  
scope synchronized{  
LockWrite(a);  
LockWrite(b);  
a=5;
```

```
b=6;
}
...
}
task{
...
scope synchronized{
LockWrite(a);
LockWrite(b);
b=8;
a=9;
}
...
}
}
```

可以理解,如上所述,以上锁定设置消除死锁问题。以下将在图11A—B的讨论中看到这一点,一个实施例进一步识别共享数据对象的读和写。因此,如果在一个异步范围之内不曾写入一个共享数据,那么按照一个实施例所述的方法将仅获得一个写锁定。因此,激活单个记录器,多个阅读器异步处理。

现在转到图11A,提供按照本发明一个实施例所述的一种设置锁定点的示范性方法1100。在步骤1105,创建一个初始化一个空集合的基本程序块的集合V。在步骤1110,集合需要锁定的所有共享数据对象的一个集合L。在步骤1115,创建向数据对象执行写操作的基本程序块的一个集合Swrite,在步骤1120,从Swrite消除没有包含在同步范围之内内的所有基本程序块。在步骤1125,创建从数据对象中执行读操作的基本程序块的一个集合Sread,在步骤1130,从Sread消除没有包含在同步范围之内内的所有基本程序块。在步骤1135,确定关于所获取的锁定类型。例如,如果集合Swrite为空,则表示没有往数据对象中写入任何内容,然后数据对象仅获取一个读锁定;否则,它获取一个写锁定。

在步骤1140,把集合Swrite和Sread添加到V。在步骤1145,确定有关另一个数据对象是否在集合L之内,如果它在集合L之内,方法1100为此接下来的数据

对象而返回到步骤1115。否则，方法1100进行到以下图11B中的步骤1150。

现在转到图11B，举例说明按照本发明一个实施例所述的一种设置锁定点的方法1100的余下部分。将会理解到假设所有锁定发生在集合V内任何共享对象的第一次词法使用时。因此，在此把包含共享数据对象的第一次词法使用的基本程序块指定为Block。例如集合V中最高编号的基本程序块。在一个实施例中，V内数据对象的每一次词法上的连续应用是每一个连续的低编号基本程序块Bnext。另外，在步骤1150以数据对象的第一次词法使用为基础计算一个初始集合Block。

在步骤1155，按照本发明一个实施例所述的一个编译程序计算Bcreate的预支配顶点和Bnext的预支配顶点的交集I。在步骤1160，确定有关集合I是否包含Block。如果包含，在步骤1170，中间数据对象的锁定点保持Block。如果不包含，在步骤1165，从集合I选出一个新锁定点Bcreate，因此新锁定点的深度优先编号是最小的号码，它大于当前锁定建点的深度优先编号。因此，尽可能接近于该对象的第一次使用来创建锁定点以避免较早锁定资源，与此同时避免死锁事务处理。可以理解，如果新锁定点与上述锁定点在不同的循环内，那么在绕过所有内循环之前，成功地从集合I选择编号越来越高的程序块。

在步骤1175，确定集合V内的另一个词法上连续的程序块Bnext是否存在，如果存在，方法1100为此接下来的使用而返回到步骤1155。一旦考虑数据对象所有的使用，方法1100进行到步骤1180。在步骤1180，如同以上连同图11A所确定的，在任何内嵌程序块使用之前，以一种确定性次序设置一个锁定每一个共享对象的指令，该指令不是读程序块就是写程序块。

因此，提供了一种用于编译XLANG/s代码的方法和系统，该XLANG/s代码涉及静态数据流分析。连同多个附图的示范性实施例描述本发明时，需要理解到可以使用其他类似的实施例，或者在不偏离的情况下对执行本发明相同功能的所描述的实施例进行修改或添加。例如，本领域技术人员将认识到本申请中描述的本发明可以应用于任何类型应用环境中的任何配置的商务方法软件。因此，本发明不应该局限于任何单个的实施例，而应根据附加的权利要求在宽度和范围上进行构建。

计算环境 100

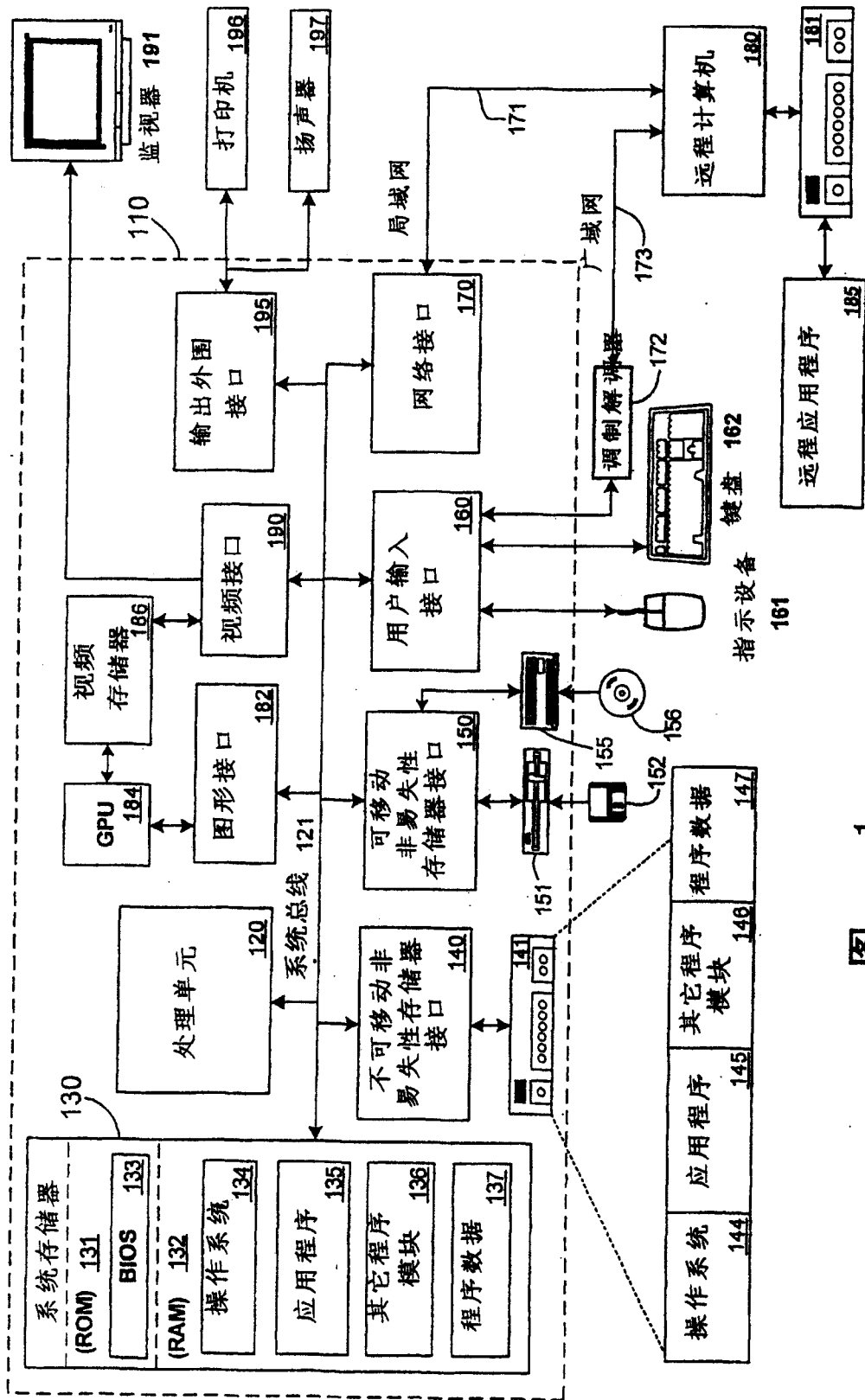


图 1

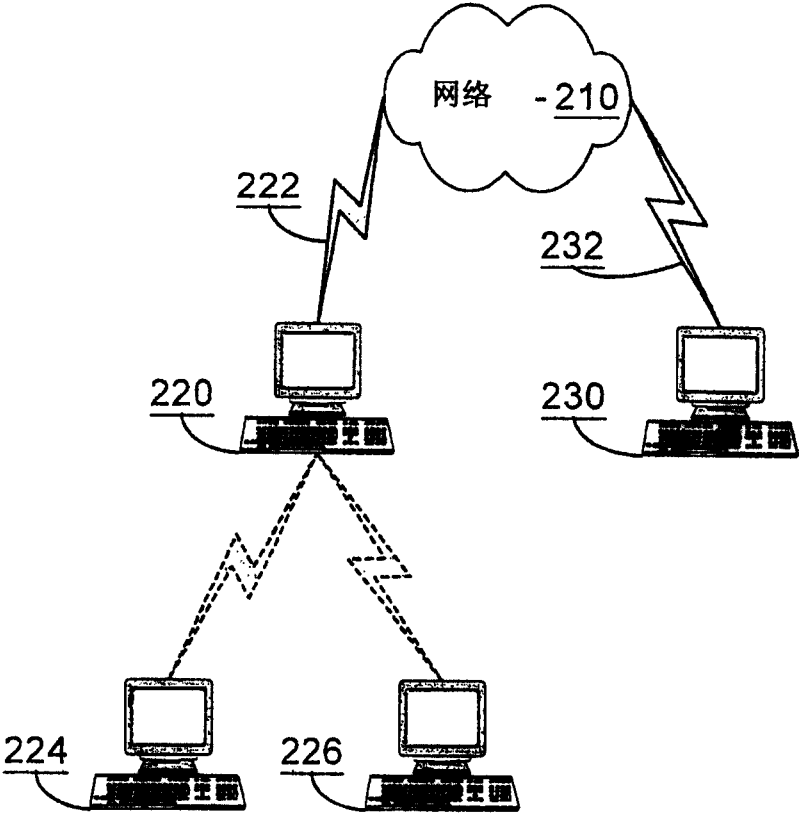


图 2

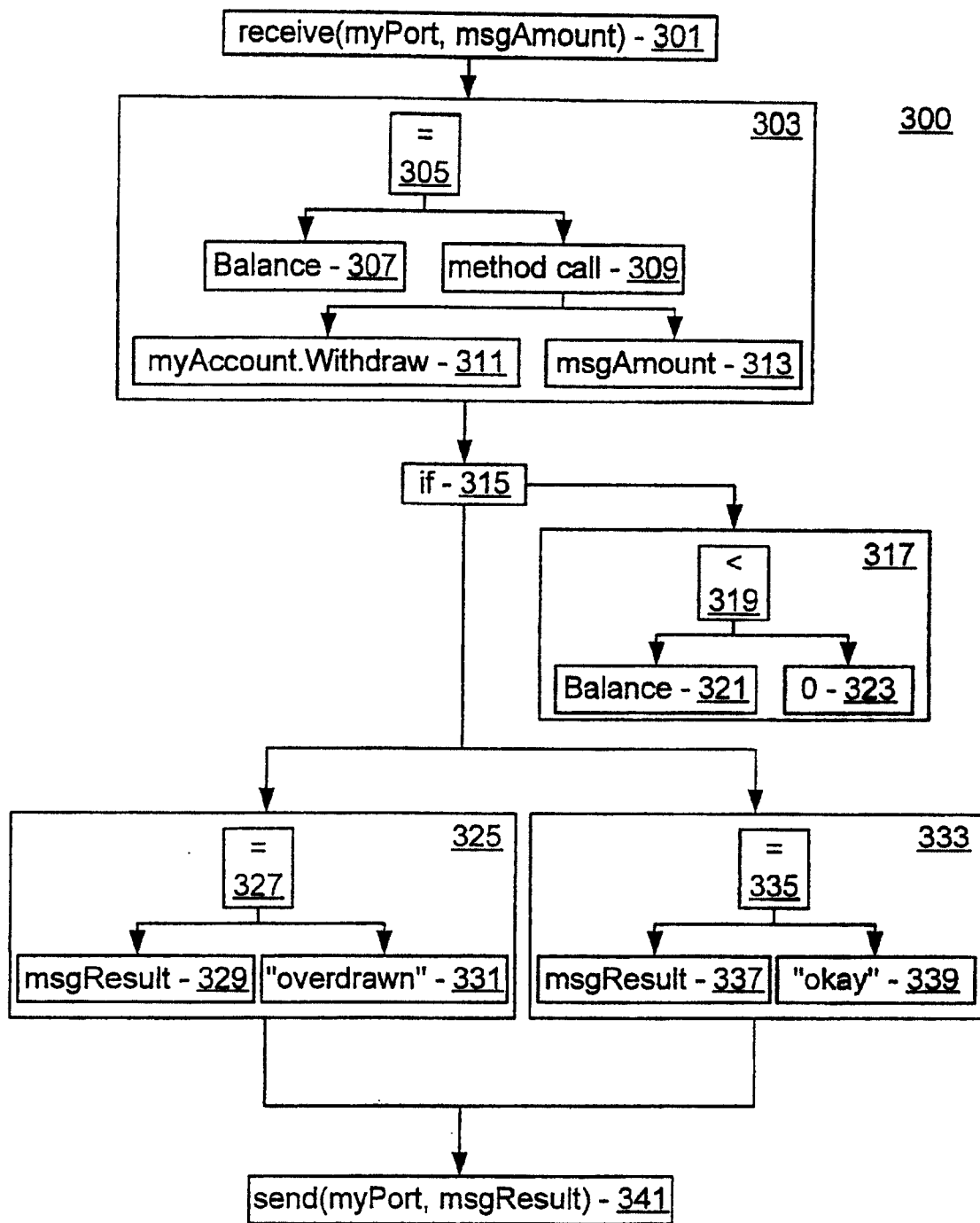


图 3

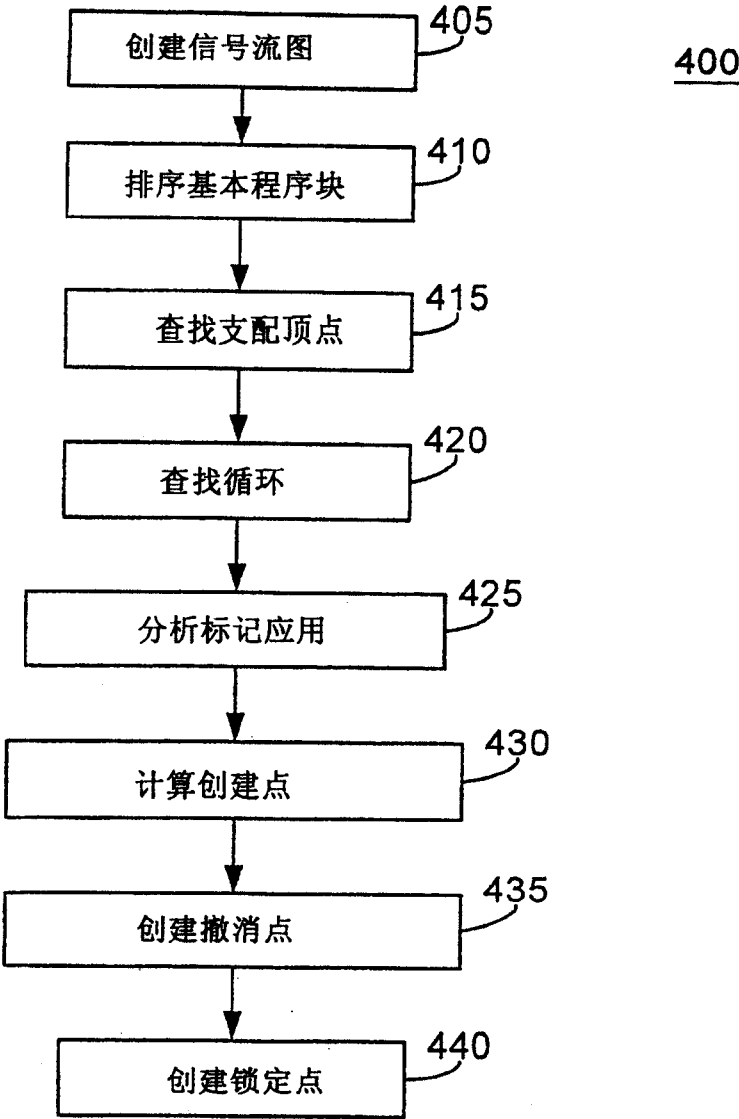


图 4

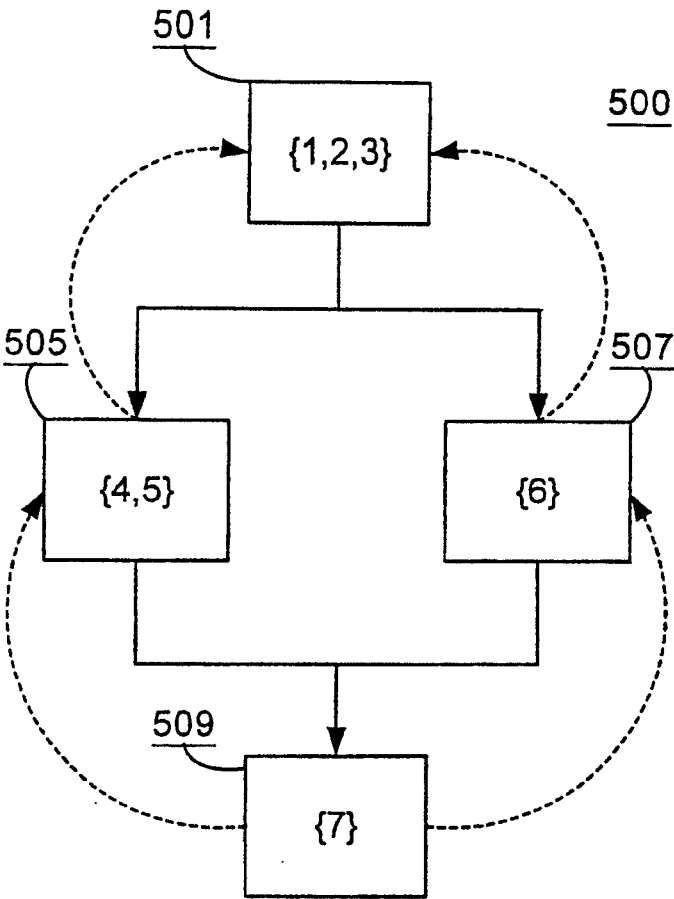


图 5

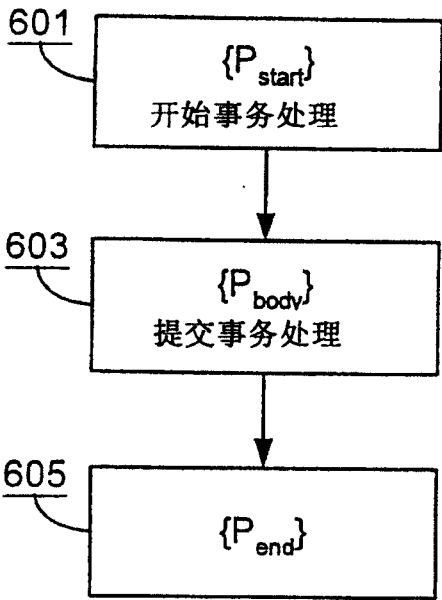


图 6A

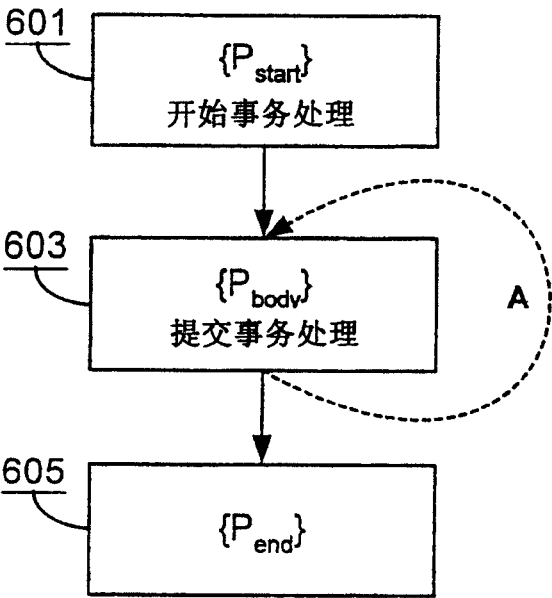


图 6B

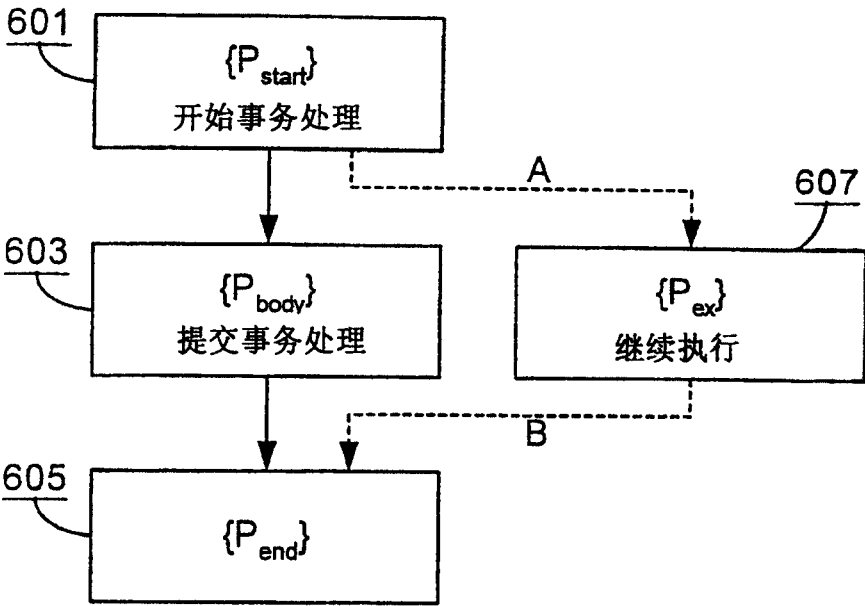


图 6C

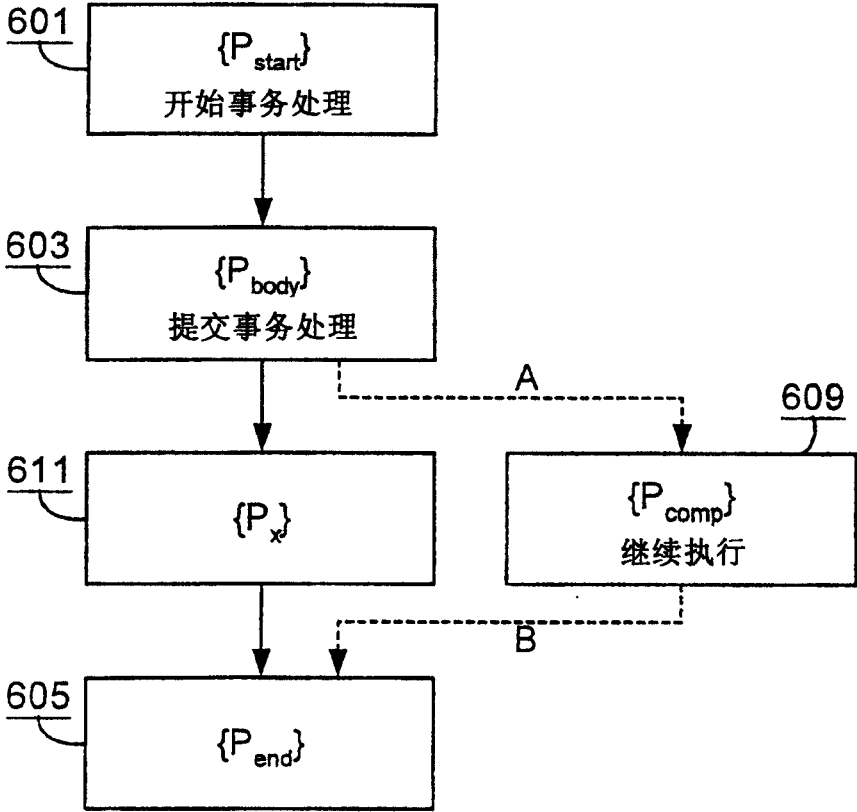


图 6D

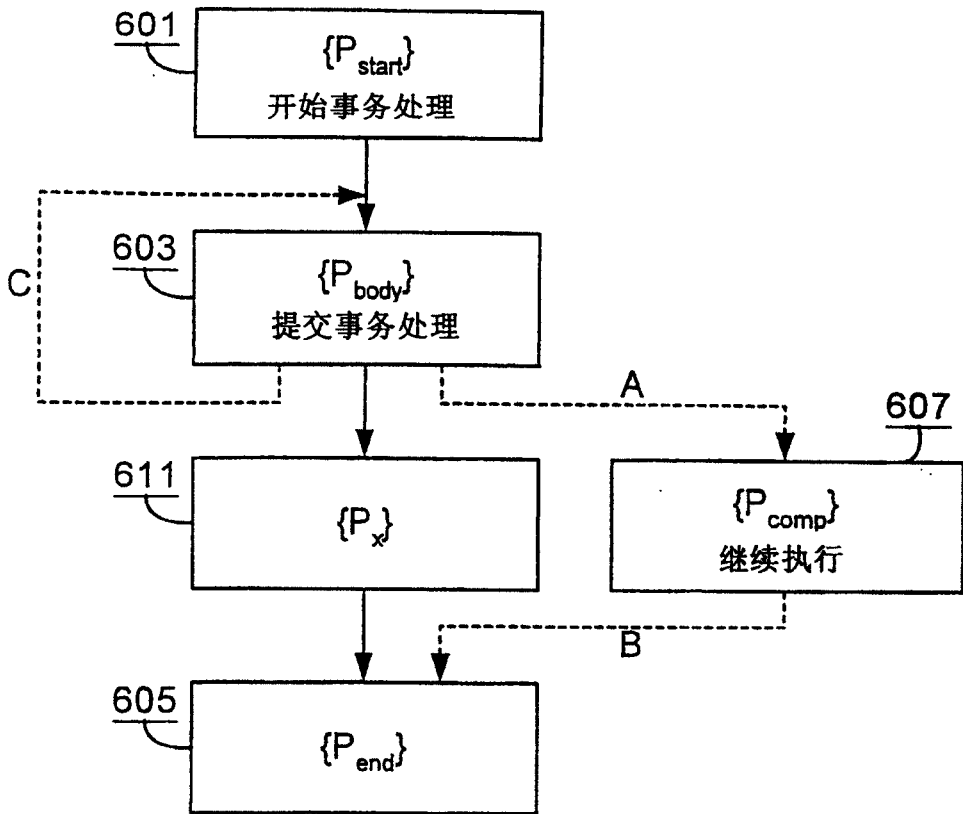


图 6E

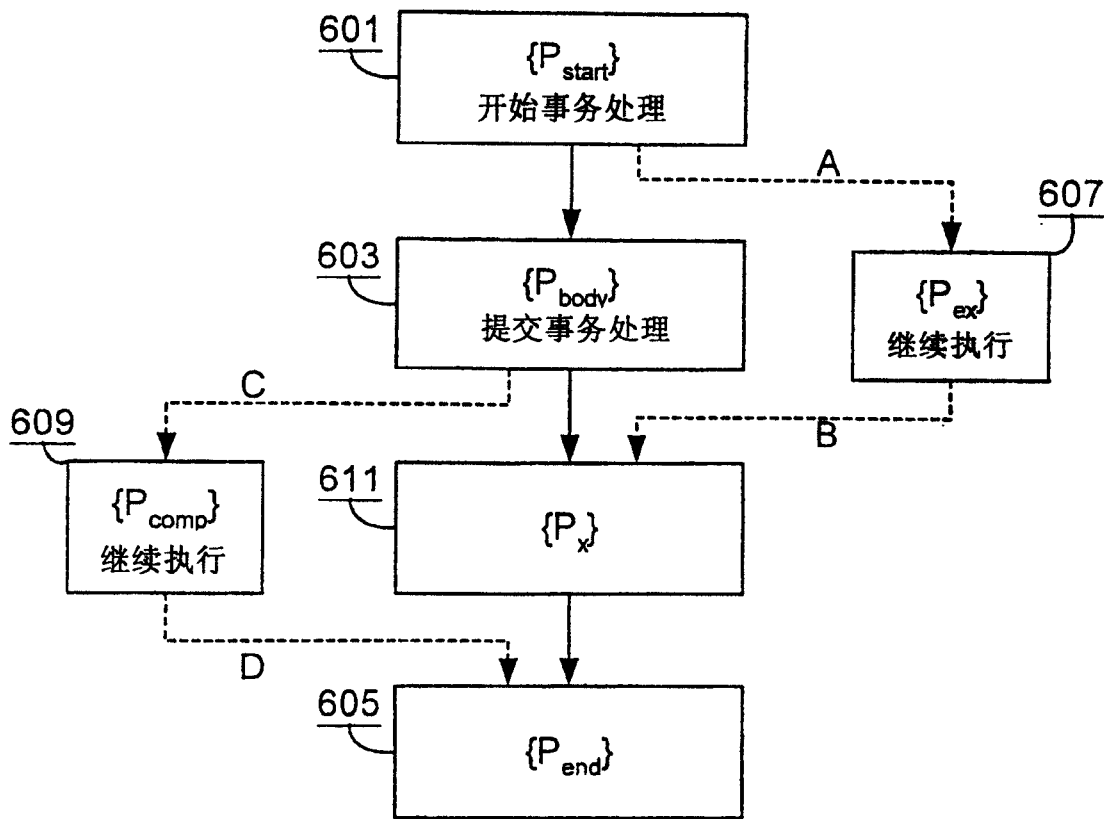


图 6F

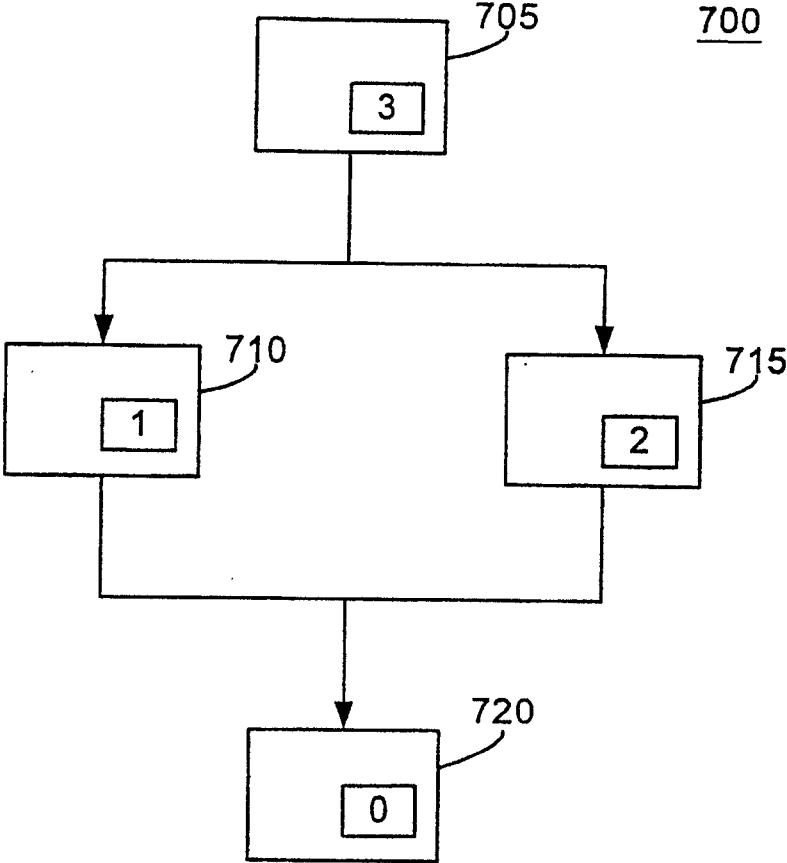


图 7

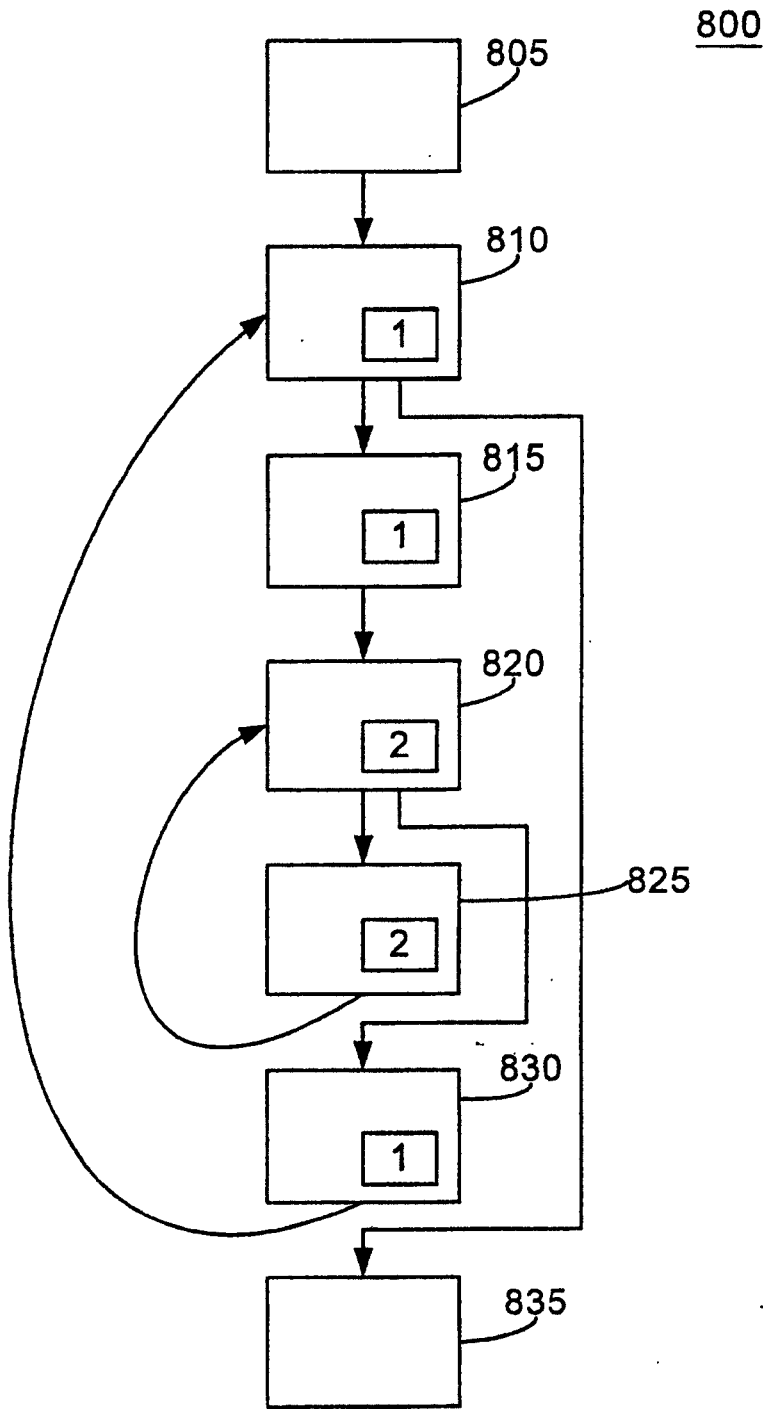


图 8

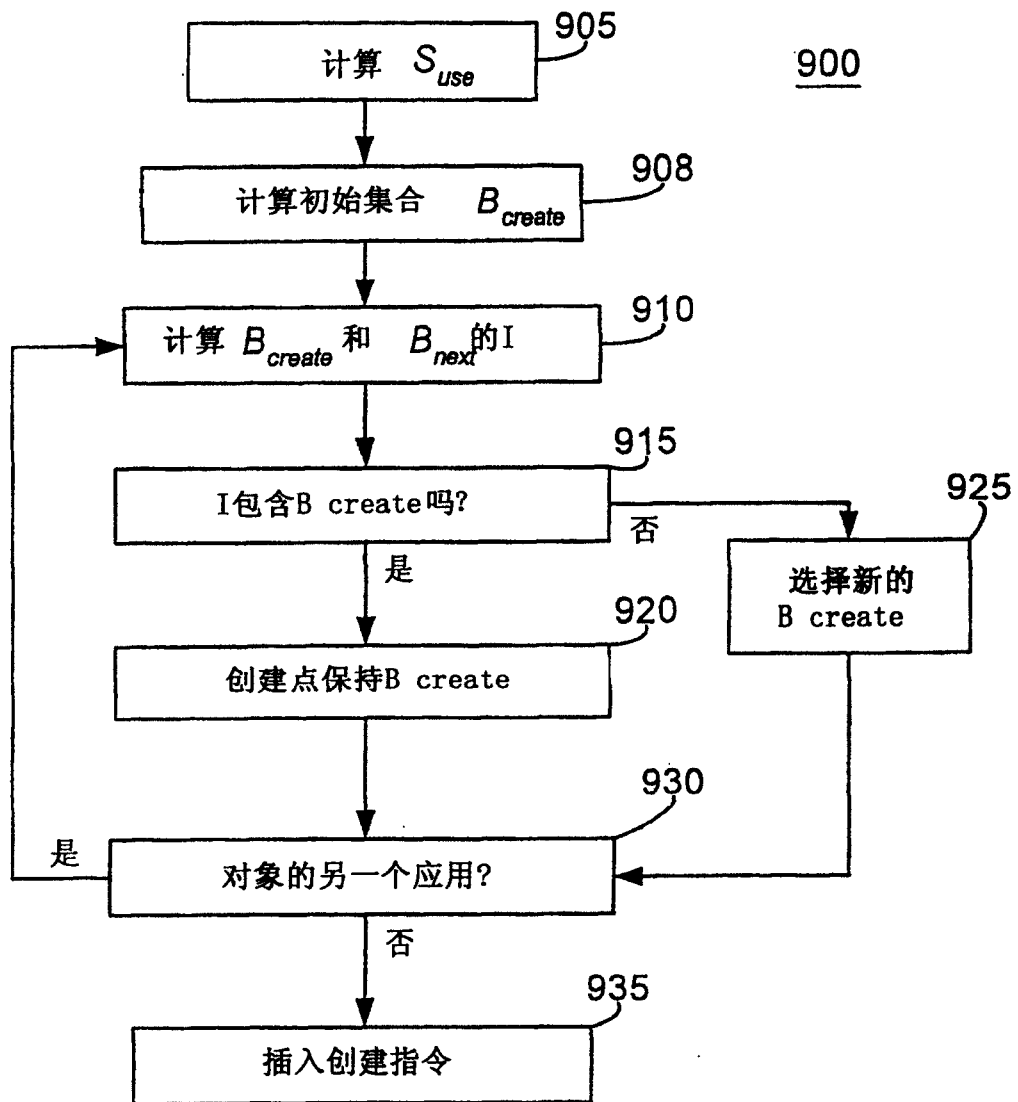


图 9

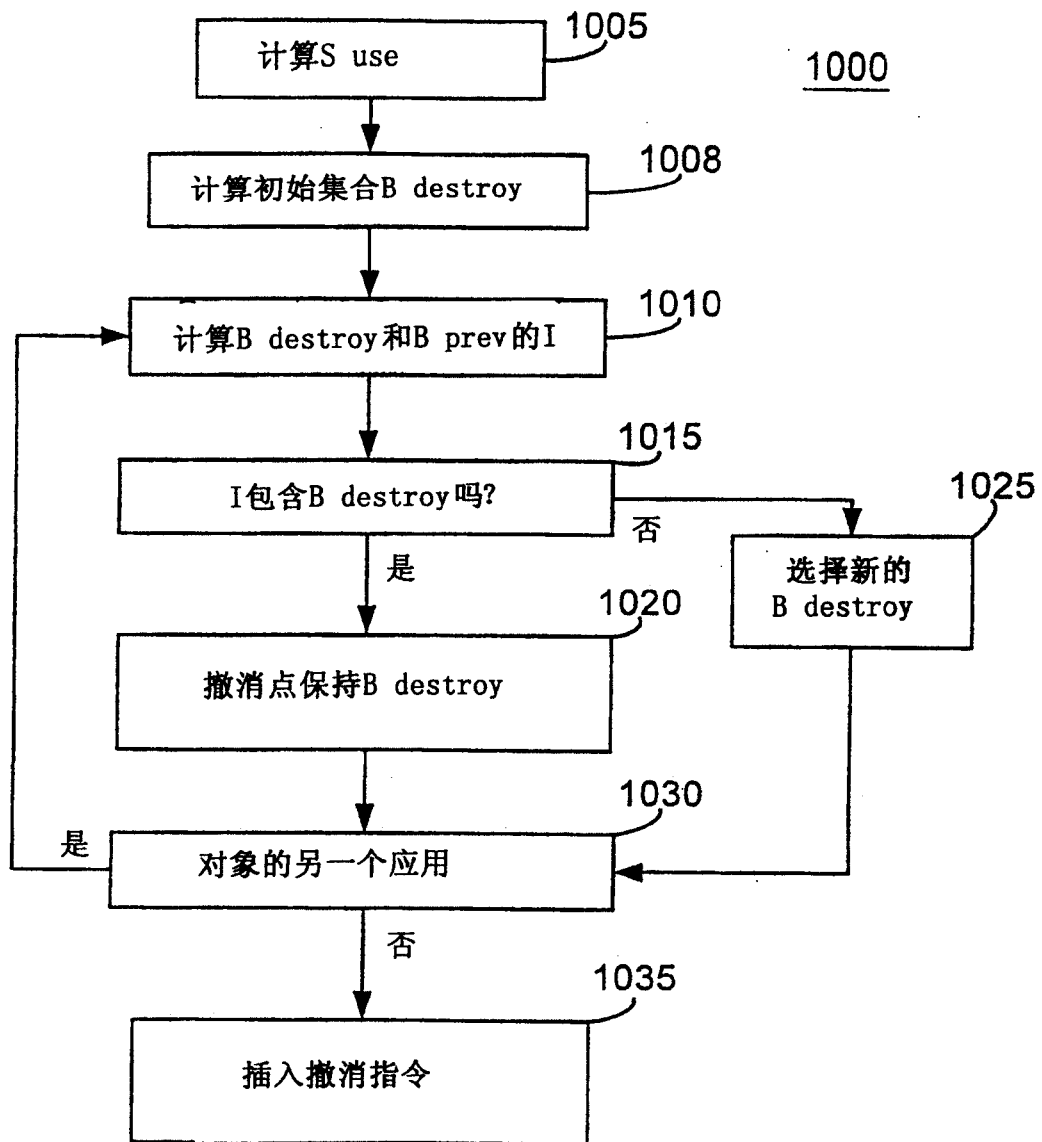


图 10

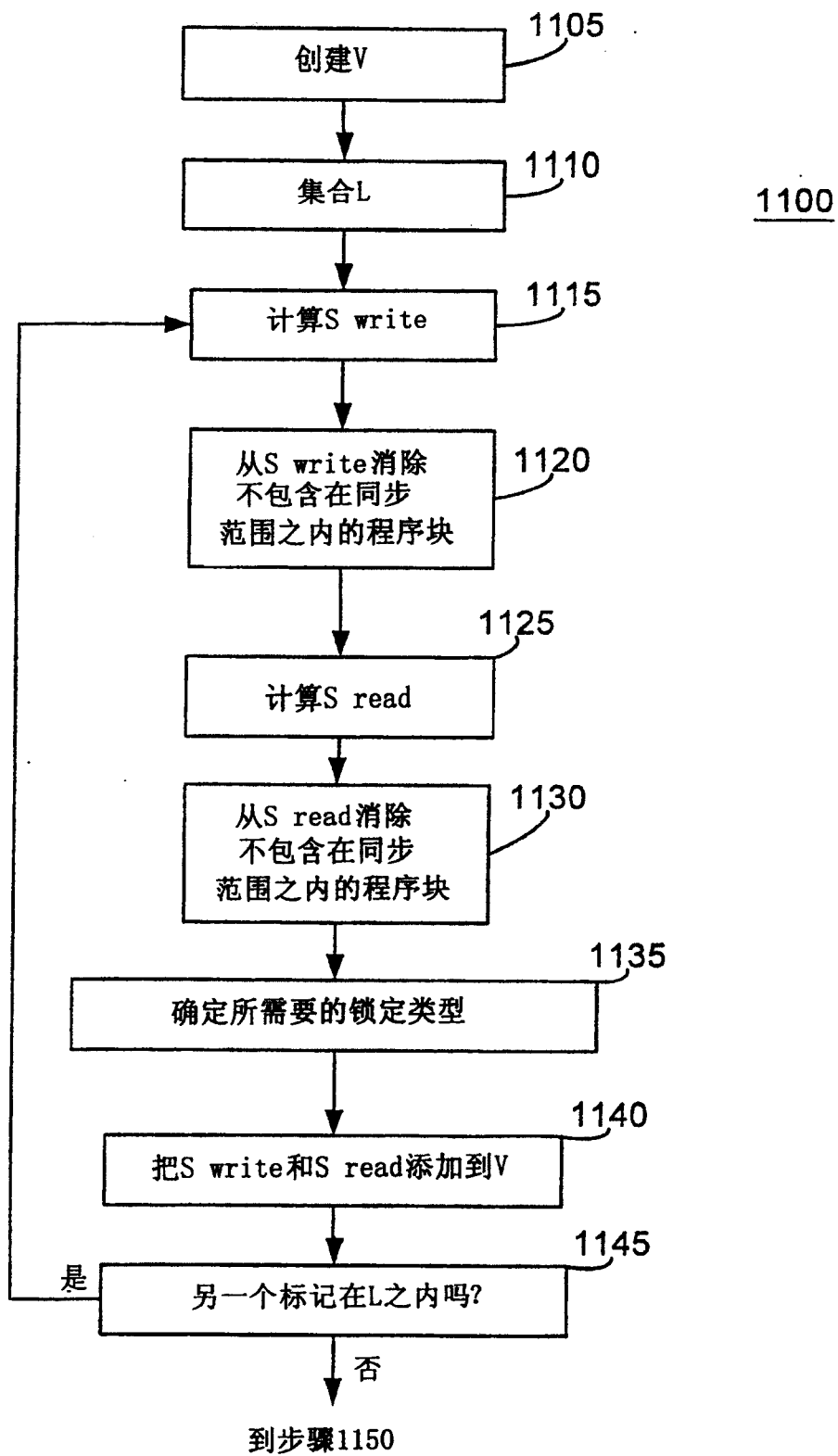


图 11A

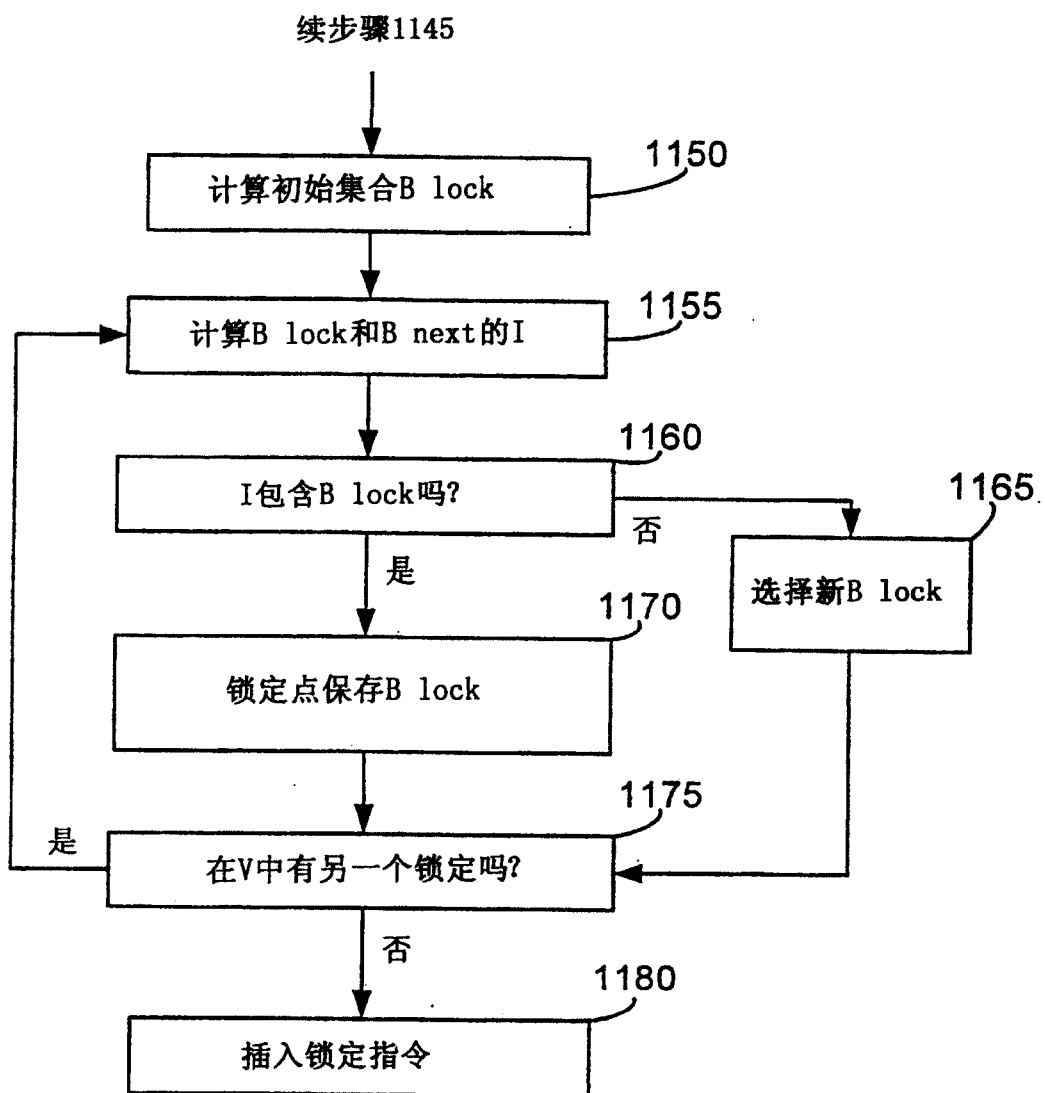
1100

图 11B