(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 1906578 B
(45)授权公告日 2010.11.17

(21)申请号 200480040477.7

(22)申请日 2004.11.05

(30)优先权数据
10/714,198 2003.11.14 US

(85)PCT申请进入国家阶段日
2006.07.14

(86)PCT申请的申请数据
PCT/US2004/037161 2004.11.05

(87)PCT申请的公布数据
WO2005/050445 EN 2005.06.02

(73)专利权人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 L·李 科顿·锡德 B·黄 威廉·哈里森三世 J·戴

(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 钱慰民

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

(56)对比文件
US 6044221 A,2000.03.28, 全文.
US 2002019910 A1,2002.02.14, 全文.

MICHAEL WOLFE.Optimizing Supercompilers for Supercomputers.MIT PRESS, LONDON,1989,78-79.

NOVILLO D ET AL.Concurrent SSA form in the presence of mutual exclusion.PARALLEL PROCESSING,1998 INTERNATIONALCONFERENCE ON MINNEAPOLIS,MN USA,1998,356-364.

MATTHEW ADILETTA, DONALD HOOPER AND MYLESWILDE.Packet over SONET: Achieving 10 Gigabit/sec PacketProcessing with an IXP2800.INTEL TECHNOLOGY JOURNAL Intel corp USA6 3.2002,6(3),29-39.

MICHAEL WOLFE.Multiprocessor Sychronization for Concurrent Loops.IEEE Software USA5 1.1988,5(1),34-42.

HWANG R-Y ET AL.Multiprocessor Sychronization for Concurrent Loops. IEEE PROCEEDINGS: COMPUTERS AND DIGITAL THCHNIQUES ,IEE,GB142 2.1995,142(2),107-109.

审查员 崔志鹏

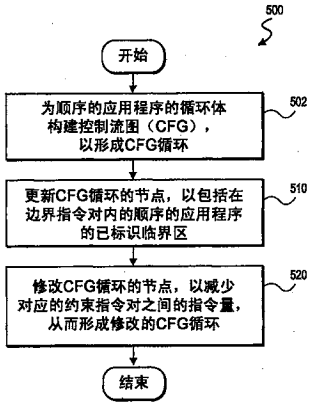
权利要求书 5 页 说明书 9 页 附图 13 页

(54)发明名称

用于自动线程划分编译器的装置和方法

(57)摘要

在一些实施方案中,描述了用于自动的线程划分编译器的方法和装置。在一些实施方案中,所述方法包括顺序应用程序到多个应用程序线程的变换。一旦被划分,所述多个应用程序线程作为多线程体系结构的各个线程并发地执行。因此,通过隐藏存储器访问延时来达到并行多线程体系结构的性能提高,隐藏存储器访问延时是通过或者借助于将存储器访问与计算或其他存储器访问重叠来进行的。描述了其他实施方案并主张对它们的权利要求。



CN 1906578 B

1. 一种用于自动线程划分编译器的方法，包括：

为顺序应用程序的循环体构建控制流图 (CFG)，以形成 CFG 循环；

更新所述 CFG 循环的节点，以包括在边界指令对中的顺序应用程序的已标识临界区；以及

从修改的 CFG 循环形成多个应用程序线程划分，其中所述 CFG 循环的节点被修改以减少在相应的边界指令对之间的指令量，从而形成所述修改的 CFG 循环；以及

同步并发执行应用程序线程的所述已标识临界区的执行，以确保所述已标识临界区按顺序的线程次序执行。

2. 如权利要求 1 所述的方法，其中更新所述 CFG 循环的操作包括：

选择所述顺序应用程序的已标识临界区；

在所述 CFG 循环的顶端节点内插入等待指令；

在所述 CFG 循环的底端节点内插入前进指令；以及

对所述顺序应用程序的每个已标识临界区重复所述选择、插入等待指令和插入前进指令操作。

3. 如权利要求 1 所述的方法，其中形成所述应用程序线程划分操作包括：

使用代码移动，用固定的等待边界指令在所述 CFG 循环的所述节点内将从已标识移动候选指令检测到的提升指令提升至相应的前驱基本块；

使用代码移动，用固定的前进指令在所述 CFG 循环的所述节点内将从已标识移动候选指令检测到的下降指令下降至相应的后继基本块；以及

用固定的等待指令和固定的前进指令在所述 CFG 循环的所述节点内重新排序已标识移动候选指令。

4. 如权利要求 3 所述的方法，其中用固定的等待指令提升检测到的提升指令的操作包括：

将所述 CFG 循环的基本块内除等待指令之外的每条指令标识为移动候选指令；

构建所述 CFG 循环的逆图；

用来自所述 CFG 循环的所述基本块初始化提升队列，所述基本块按照由所述逆图指示的拓扑次序排序；

将所述基本块的移动候选指令提升至相应的前驱基本块，直到不再从所述已标识移动候选指令检测到提升指令；以及

根据所述顺序应用程序的相关图提升从所述 CFG 循环的源基本块内的移动候选指令检测到的提升指令。

5. 如权利要求 4 所述的方法，其中提升从所述基本块的所述移动候选指令中检测到的提升指令的操作包括：

使基本块从所述提升队列中出对列，作为当前块；

基于所述顺序应用程序的相关图从所述移动候选指令计算提升指令；

将计算的提升指令提升到相应的基本块中；以及

当检测到改变时，使所述当前块的前驱块从所述 CFG 循环入队列到所述提升队列。

6. 如权利要求 4 所述的方法，其中用固定的前进指令下降检测到的下降指令的操作包括：

通过数据流分析，用固定的前进指令标识所述 CFG 循环的所述基本块内的移动候选指令；

用基于所述 CFG 循环中的拓扑次序排序的所述基本块初始化下降队列；

在所述基本块之间将检测到的下降指令下降至相应的后继基本块，直到不再从所述已标识移动候选指令检测到下降指令；以及

根据所述相关图，在包括前进指令的基本块内重新排序检测到的移动候选。

7. 如权利要求 6 所述的方法，其中在所述基本块之间下降检测到的下降指令的操作包括：

使基本块从所述下降队列出队列，作为当前块；

基于所述顺序应用程序的相关图从移动候选指令计算下降指令；

将计算的下降指令下降到相应的后继基本块；以及

如果检测到改变，使所述 CFG 循环中的当前块的后继块入队列到所述下降队列中。

8. 如权利要求 3 所述的方法，其中用所述固定的等待指令和前进指令执行指令提升的操作包括：

用基于所述 CFG 循环中的拓扑次序排序的所述基本块初始化提升队列；

通过数据流分析，用固定的前进指令和固定的等待指令标识所述 CFG 循环的所述基本块内的移动候选指令；

在所述基本块之间将检测到的提升指令提升至相应的前驱基本块，直到不再从所述已标识移动候选指令检测到提升指令；以及

基于所述顺序应用程序的相关图，在包括等待指令的基本块内重新排序所述已标识移动候选指令。

9. 如权利要求 8 所述的方法，其中被提升到最外层等待指令之外的移动候选指令不再被作为移动候选来对待；并且

其中在最外层前进指令之外的移动候选指令不再被作为移动候选来对待。

10. 一种用于自动线程划分编译器的装置，包括：

用于为顺序应用程序的循环体构建控制流图 (CFG)，以形成 CFG 循环的装置；

用于更新所述 CFG 循环的节点，以包括在边界指令对中的顺序应用程序的已标识临界区的装置；以及

用于从修改的 CFG 循环形成多个应用程序线程划分的装置，其中所述 CFG 循环的节点被修改以减少在相应的约束指令对之间的指令量，从而形成所述修改的 CFG 循环；以及

用于同步并发执行应用程序线程的所述已标识临界区的执行，以确保所述已标识临界区按顺序的线程次序执行的装置。

11. 如权利要求 10 所述的装置，其中所述用于更新所述 CFG 循环的装置包括：

用于选择所述顺序应用程序的已标识临界区的装置；

用于在所述 CFG 循环的顶端节点内插入等待指令的装置；

用于在所述 CFG 循环的底端节点内插入前进指令的装置；以及

用于为所述顺序应用程序的每个已标识临界区重复所述选择、插入和插入操作的装置。

12. 如权利要求 10 所述的装置，其中所述用于形成所述线程应用划分的装置包括：

用于使用代码移动，用固定的等待边界指令在所述 CFG 循环的所述节点内提升已标识移动候选指令的装置；

用于使用代码移动，用固定的前进指令在所述 CFG 循环的所述节点内下降已标识移动候选指令的装置；以及

用于用固定的等待指令和固定的前进指令在所述 CFG 循环的所述节点内提升已标识移动候选指令的装置。

13. 如权利要求 12 所述的装置，其中所述用于用固定的等待指令提升检测到的提升指令的装置包括：

用于将所述 CFG 循环的基本块内除等待指令之外的每条指令标识为移动候选指令的装置；

用于构建所述 CFG 循环的逆图的装置；

用于用来自所述 CFG 循环的所述基本块初始化提升队列，所述基本块按照由所述逆图指示的拓扑次序排序的装置；

用于提升所述基本块的移动候选指令，直到不再从所述移动候选指令检测到提升指令的装置；以及

用于根据所述顺序应用程序的相关图重新排序从所述 CFG 循环的源基本块中的移动候选指令检测到的提升指令的装置。

14. 如权利要求 13 所述的装置，其中所述用于重新排序从所述基本块的所述移动候选指令中检测到的提升指令的装置包括：

用于使基本块从所述提升队列中出对列，作为当前块的装置；

用于基于所述顺序应用程序的相关图从所述基本块的所述移动候选指令计算提升指令的装置；

用于将计算的提升指令提升到相应的基本块中的装置；以及

用于当检测到改变时，使所述当前块的前驱块从所述 CFG 循环入队列到所述提升队列的装置。

15. 如权利要求 13 所述的装置，其中所述用于用固定的前进指令下降检测到的下降指令的装置包括：

用于通过数据流分析，用固定的前进指令标识所述 CFG 循环的所述基本块内的移动候选指令的装置；

用于用基于所述 CFG 循环中的拓扑次序排序的所述基本块初始化下降队列的装置；

用于在所述基本块之间下降检测到的下降指令，直到不再检测到下降指令的装置；以及

用于根据所述相关图，在包括前进指令的基本块内下降检测到的移动候选的装置。

16. 如权利要求 15 所述的装置，其中所述用于在所述基本块之间下降检测到的下降指令的装置包括：

用于使基本块从所述下降队列出队列的装置，作为当前块；

用于基于所述顺序应用程序的相关图从移动候选指令计算下降指令的装置；

用于将计算的下降指令下降到相应的基本块的装置；以及

用于如果检测到改变，使所述 CFG 循环中的当前块的后继块入队列到所述下降队列中

的装置。

17. 如权利要求 12 所述的装置，其中所述用于用所述固定的等待指令和前进指令执行指令提升的装置包括：

用于用基于所述 CFG 循环中的拓扑次序排序的所述基本块初始化提升队列的装置；

用于通过数据流分析，用固定的前进指令和固定的等待指令标识所述 CFG 循环的所述基本块内的移动候选指令的装置；

用于在所述基本块之间提升检测到的提升指令，直到不再检测到提升指令的装置；以及

用于基于所述顺序应用程序的相关图，在包括等待指令的基本块内提升移动候选的装置。

18. 如权利要求 17 所述的装置，其中被提升到最外层等待指令之外的移动候选指令不再被作为移动候选来对待；并且

其中在最外层前进指令之外的移动候选指令不再被作为移动候选来对待。

19. 一种用于自动线程划分编译器的方法，包括：

根据多线程体系结构的线程计数将顺序应用程序划分为多个应用程序线程；以及

在多线程系统结构的各个线程中并发地执行所述多个应用程序线程；以及

在所述多个应用程序线程之间同步对顺序应用程序线程的已标识临界区的访问，以使所述顺序应用程序线程循环的所述已标识临界区按顺序的线程次序执行。

20. 如权利要求 19 所述的方法，其中生成所述应用程序线程的操作包括：

处理已标识临界区，以减少被包括在线程程序循环的临界区内的代码量。

21. 如权利要求 20 所述的方法，其中使用代码移动来减少所述线程程序循环的临界区内的所述代码量。

22. 一种用于自动线程划分编译器的装置，包括：

用于根据顺序应用程序内的已标识临界区将所述顺序应用程序划分为多个应用程序线程的装置；以及

用于在多线程系统结构的各个线程中并发地执行所述多个应用程序线程的装置；以及

用于在所述多个应用程序线程之间同步对顺序应用程序线程的已标识临界区的访问，以确保所述顺序应用程序线程循环的所述已标识临界区按顺序的线程次序执行的装置。

23. 如权利要求 22 所述的装置，其中所述用于生成所述应用程序线程的装置包括：

用于处理已标识临界区，以减少被包括在线程程序循环的临界区内的代码量的装置。

24. 如权利要求 23 所述的装置，其中使用代码移动来减少所述线程程序循环的临界区内的所述代码量。

25. 一种用于自动线程划分编译器的系统，包括：

处理器；

耦合到所述处理器的存储器控制器；以及

耦合到所述处理器的 DDR SRAM 存储器，所述存储器包括编译器，所述编译器包括：

用于根据顺序应用程序内的已标识临界区将顺序应用程序划分为多个应用程序线程，以使能在多线程系统结构的各个线程内所述多个应用程序线程的并发执行的装置；以及

用于在所述多个应用程序线程之间同步对顺序应用程序线程的已标识临界区的访问，

以确保所述顺序应用程序线程循环的所述已标识临界区按顺序的线程次序执行的装置。

26. 如权利要求 25 所述的系统，其中所述编译器还包括：

用于为顺序应用程序的循环体构建控制流图 (CFG)，以形成 CFG 循环的装置，用于更新所述 CFG 循环的节点，以包括在边界指令对中的顺序应用程序的已标识临界区的装置，以及用于修改所述 CFG 循环的节点，以减少在相应的约束指令对之间的指令量，从而形成修改的 CFG 循环的装置。

27. 如权利要求 26 所述的系统，其中所述编译器还包括：

用于使用代码移动用固定的等待边界指令在所述 CFG 循环的所述节点内提升已标识移动候选指令的装置；

用于使用代码移动用固定的前进指令在所述 CFG 循环的所述节点内下降已标识移动候选指令的装置；以及

用于用固定的等待指令和固定的前进指令在所述 CFG 循环的所述节点内提升移动候选指令的装置。

用于自动线程划分编译器的装置和方法

发明领域

[0001] 本发明的一个或更多个实施方案总地涉及多线程微体系结构。更具体地，本发明的一个或更多个实施方案涉及用于自动线程划分 (thread-partition) 编译器的装置和方法。

[0002] 发明背景

[0003] 硬件多线程化正成为现代处理器设计中的一项实用技术。在行业中已经公布了几种多线程处理器，或者它们在高性能计算、多媒体处理和网络分组 (packet) 处理领域中处于生产阶段。属于 Intel® Internet Exchange™ Architecture（因特网交换体系结构，IXA）网络处理器 (NP) 家族的因特网交换处理器 (IXP) 系列就是这样的多线程处理器的实施例。一般来说，为了满足分组处理的高性能要求，每个 IXP 都包括高度并行、多线程的体系结构。

[0004] 一般来说，NP 被特别地设计以执行分组处理。常规地，NP 可以用于作为高速通信路由器的核心部件 (core element) 来执行这样的分组处理。一般来说，用于执行分组处理的传统网络应用是使用顺序语义以常规的方式编码的。一般来说，这样的网络应用被编码为使用始终运行的分组处理单元（分组处理级 (PPS)）。因此，当新的分组到达时，PPS 执行一系列任务（例如分组的接收、路由表查找和分组的入队列）。因此，PPS 通常被表达为无限循环（或 PPS 循环），其中每次迭代处理不同的分组。

[0005] 因此，尽管高度并行，但是由现代 NP 提供的多线程体系结构不能在高度未占用处理器资源中利用这些并行化结果。毫无疑问，如果顺序应用程序运行在由 NP 提供的高级多线程体系结构之上，则可能会获得低的性能增益。为了达到高性能，编程者尝试了通过利用顺序应用的线程级并行性来完全利用 NP 提供的多线程体系结构。不幸的是，对于大多数编程者来说，手动的线程划分是一种挑战。

[0006] 附图简要说明

[0007] 在附图中，本发明的各种实施方案是以实施例的方式来说明的，而不是以限定的方式来说明的，其中：

[0008] 图 1 是根据本发明的一个实施方案，示出具有线程划分编译器的计算机系统的框图。

[0009] 图 2A-2C 根据本发明的一个实施方案，描绘了顺序分组处理级 (PPS) 到两个应用程序线程的变换。

[0010] 图 3A-3B 根据本发明的一个实施方案，示出顺序 PPS 循环 (loop) 序列的变换，所述序列包括被一个或更多个边界指令 (boundary instruction) 包围的临界区。

[0011] 图 4 是根据本发明的一个实施方案，示出包括多线程体系结构的处理器的框图。

[0012] 图 5 是根据本发明的一个实施方案，示出用于线程划分顺序应用程序的循环体的方法的框图。

[0013] 图 6A-6B 是根据本发明的一个实施方案，示出控制流图 (CFG 循环）的形成的框图。

[0014] 图 7 是根据本发明的一个实施方案，示出用于修改在边界指令对内包括已标识临界区的 CFG 循环的方法的流程图。

[0015] 图 8 是根据本发明的一个实施方案，示出用于减少相应的边界指令对之间的指令量的方法的流程图。

[0016] 图 9 是根据本发明的一个实施方案，示出用于提升 (hoisting) 移动候选指令的方法的流程图。

[0017] 图 10 是根据本发明的一个实施方案，示出流相关图 (flow dependence graph) 的框图。

[0018] 图 11 是根据本发明的一个实施方案，示出用于提升移动候选指令的方法的流程图。

[0019] 图 12 是根据本发明的一个实施方案，示出用于下降 (sinking) 移动候选指令的方法的流程图。

[0020] 图 13 是根据本发明的一个实施方案，示出用于下降移动候选指令的方法的流程图。

[0021] 图 14 是根据本发明的一个实施方案，示出用于提升移动候选指令的方法的流程图。

[0022] 图 15A-15C 是根据本发明的一个实施方案，示出移动候选的计算的框图。

[0023] 图 16 是根据本发明的一个实施方案，示出用于将顺序应用程序划分为多个应用程序线程以供并发执行所述程序线程的流程图。

[0024] 详细描述

[0025] 描述了用于自动线程划分编译器的方法和装置。在一个实施方案中，所述方法包括顺序应用程序到多个应用程序线程的变换。一旦被划分，所述多个应用程序线程作为多线程体系结构的各个线程并发地执行。因此，通过隐藏存储器访问延时 (latency) 来达到并行多线程体系结构的性能提高，隐藏存储器访问延时是通过或者借助于将存储器访问与计算或其他存储器访问重叠来进行的。

[0026] 在下面的描述中，使用了某些术语来描述本发明的特征。例如，术语“逻辑”代表被配置为执行一个或更多个功能的硬件和 / 或软件。例如，“硬件”的实施例包括，但不限制或不限定于，集成电路、有限状态机或甚至组合逻辑。集成电路可以采取诸如微处理器的处理器、专用集成电路、数字信号处理器、微控制器等等的形式。

[0027] “软件”的实施例包括应用、小应用程序 (applet)、例程 (routine) 或甚至指令串形式的可执行代码。软件可以被储存在任何计算机或机器可读介质中，例如可编程电子电路、包括易失性存储器（例如随机访问存储器等）和 / 或非易失性存储器（例如任何类型的只读存储器“ROM”，闪存存储器）半导体存储器设备、软盘、光盘（例如致密盘或数字视频盘“DVD”）、硬驱动器盘等等。

[0028] 在一个实施方案中，本发明可以被提供为可以包括具有存储在其上的指令的机器或计算机可读介质的制品，所述指令可以用于将计算机（或其他电子设备）编程为根据本发明的一个实施方案执行处理。计算机可读介质可以包括，但不限于，软盘、光盘、致密盘只读存储器 (CD-ROM) 和磁光盘、只读存储器 (ROM)、随机访问存储器 (RAM)、可擦除可编程只读存储器 (EPROM)、电可擦除可编程只读存储器 (EEPROM)、磁卡或光卡、闪存等等。

[0029] 图 1 是根据本发明的一个实施方案示出包括线程划分编译器 200 的计算机系统 100 的框图。如所示出的，计算机系统 100 包括 CPU 110、存储器 140 和耦合到存储器控制器中心 (MCH) 120 的图形控制器 130。如这里所描述的，MCH 120 可以被称为北桥，并且在一个实施方案中，MCH 120 可以被称为存储器控制器。此外，计算机系统 100 包括 I/O（输入 / 输出）控制器中心 (ICH) 160。如这里所描述的，ICH 160 可以被称为南桥或 I/O 控制器。南桥，或 ICH 160 耦合到本地 I/O 150 和硬盘驱动器设备 (HDD) 190。

[0030] 在示出的实施方案中，ICH 160 耦合到 I/O 总线 172，I/O 总线 172 耦合多个 I/O 设备，例如 PCI 或外设部件互连 (PCI) 设备 170，包括 PCI-express、PCI-X、第三代 I/O(3GIO) 或其他类似的互连协议。总起来说，MCH 120 和 ICH 160 被称为芯片组 180。如这里所描述的，术语“芯片组”是以对本领域技术人员来说众所周知的方式被用来在整体上描述耦合到 CPU 110 以执行期望的系统功能性的各种器件。在一个实施方案中，主存储器 140 是易失性存储器，包括但不限于，随机访问存储器 (RAM)、同步 RAM(SRAM)、动态 RAM(DRAM)、同步 DRAM(SDRAM)、双倍数据率 (DDR) SDRAM(DDR SDRAM)、Rambus DRAM(RDRAM)、直接 RDRAM(DRDRAM) 等等。

[0031] 系统

[0032] 与常规计算机系统相比，计算机系统 100 包括用于将顺序应用程序划分为多个应用程序线程（“线程划分”）的线程划分编译器 200。因此，编译器 200 可以桥接网络处理器的多线程体系结构和用于编码常规网络应用的顺序编程模型之间的差距 (gap)。解决该问题的一种方法是利用顺序应用的线程级并行性。不幸的是，对于大多数编程者来说，对顺序应用进行手动地线程划分是一种挑战。在一个实施方案中，提供线程划分编译器 200 来自动地将如图 2A 中所示的顺序网络应用线程划分为如图 2B-2C 中所示的多个程序线程。

[0033] 参照图 2A，示出顺序网络应用的顺序分组处理级 (PPS) 280。在一个实施方案中，PPS 280 变换为用于在例如图 4 的多线程网络处理器 400 中执行的第一程序线程 300-1（图 2B）和第二程序线程 300-2（图 2C）。在一个实施方案中，通过将如参照图 2A 示出的传统网络应用变换为如参照图 2B-2C 示出的多个程序线程，提高了多线程体系结构的性能。

[0034] 在一个实施方案中，顺序 PPS 无限循环 282 变换为多个程序线程 (300-1，300-2)，其中在程序线程之间具有优化的同步，以达到改进的并行执行。不幸的是，顺序 PPS 循环（例如 282）包括相关操作的顺序执行，所述相关操作包括例如循环承载变量 (loop carriedvariable)。如这里所描述的，循环承载变量是从循环的一次迭代到另一次迭代的数据相关关系。在一个实施方案中，循环承载变量包括两种属性 :(i) 循环承载变量在顺序 PPS 的 PPS 循环的回边有效 ;以及 (ii) 循环承载变量的值在 PPS 循环体中改变。代表性地，变量 i 286 代表 PPS 循环 282 的临界区 (critical section) 284 中的循环承载变量，如下面进一步详细描述的。

[0035] 在一个实施方案中，顺序 PPS 的临界区由包围的边界指令 (302 和 304) 标识，如图 2B 和图 2C 中所示。代表性地，顺序 PPS 280 被线程划分为两个程序线程 (300-1 和 300-2)。一旦被划分，程序线程的执行以线程 300-2（线程 1）的执行开始，其中通过例如输入程序初始化变量 i282。一旦被初始化，线程划分 300-1（线程 0）被告知开始执行。一旦临界区的执行完成，线程 300-1 通知线程 300-2。因此，程序线程 300-1 执行顺序 PPS 循环 282 的 0、2、4、…、N 次迭代，而程序线程 300-2 执行 PPS 282 的 1、3、5、…、M 次迭代。

[0036] 在一个实施方案中，通过减少被包括在临界区中的指令量来增加并行性水平。因此，通过最小化临界区代码，要求按严格顺序的线程次序执行的代码段（fragment）量被最小化。一旦检测到循环承载变量和相关操作，就通过边界指令区分临界区。因此，在一个实施方案中，对于执行线程划分的准备工作包括识别所有循环承载变量。在一个实施方案中，通过输入程序来执行对循环承载变量的识别，并且因此省略了关于检测循环承载变量的额外细节，以避免模糊本发明的实施方案的细节。

[0037] 在一个实施方案中，线程划分编译器 200（图 1）维护顺序应用程序的程序线程间的顺序语义。在一个实施方案中，线程划分编译器在线程划分之间引入同步，所述同步足以加强程序线程循环的迭代之间的相关性（dependency）。在一个实施方案中，提供 AWAIT（等待）操作和 ADVANCE（前进）操作来执行同步，并且确保程序线程划分循环按顺序的线程次序执行。因此，如参照图 3A-3B 所示出的，每个循环承载变量被指派到唯一的临界区中，以同步对循环承载变量的访问，从而形成程序线程 300-1（图 3A）和程序线程 300-2（图 3B）。

[0038] 在图 3A-3B 中示出的程序表示中，临界区被指示为 N，并且以特别的 AWAIT(N) 操作开始，以 ADVANCE(N) 操作结束。如这里所描述的，AWAIT 指令是指中止当前线程的执行，直到处理链上前一线程通知开始执行。如这里所描述的，术语 ADVANCE 操作是指告知处理链上的后一线程进入临界区并且开始执行临界区。在一个实施方案中，边界指令（即 AWAIT 和 ADVANCE 指令）同步对循环承载变量的访问，以按照严格顺序的线程次序完成执行。

[0039] 图 4 是示出多处理器的框图，所述多处理器例如被配置为提供多线程体系结构的网络处理器 (NP)400。在一个实施方案中，NP 400 执行图 3A 和 3B 的程序线程 300-1 和程序线程 300-2。在一个实施方案中，同步块 420 执行相应的 ADVANCE 和 AWAIT 指令之间的通信。代表性地，程序线程 300-1 和 300-2 由微体系结构线程 410(410-1，…，410-N) 并行执行。因此，NP 400 执行从顺序应用程序形成的线程划分 300-1 和 300-2。在一个实施方案中，可以用其他操作来并发地执行存储器访问，以通过在存储器访问期间完成线程执行来隐藏存储器访问延时。

[0040] 例如，在一个实施方案中，网络处理器 400 在图 1 的处理器 100 中实现，从而图 1 的处理器 100 实现多线程微体系结构。在一个实施方案中，可以在任何计算机体系结构上执行用于传统网络分组处理应用的自动线程划分的一般框架，所述自动线程划分的一般框架包括以下特征：(1) 多线程体系结构；(2) 由每个线程排他地进行的线程局部储存设备访问，以及由所有线程进行的全局储存访问；以及 (3) 线程间通信和同步机制信号。现在描述用于实现本发明的实施方案的过程方法。

[0041] 操作

[0042] 图 5 是根据本发明的一个实施方案，示出用于线程划分顺序应用程序的方法 500 的流程图。在处理框 502，为顺序应用程序的循环体构建控制流图 (control flow graph，CFG)，以形成 CFG 循环。如这里所描述的，CFG 是代表对程序的控制流的图，其中每个点 (vertex) 代表基本块 (basic block)，并且每条边代表基本块之间潜在的控制流。控制流图具有唯一的源节点（入口）。在一个实施方案中，CFG 循环的形成如参照图 6A 和 6B 示出的。

[0043] 如图 6A 中所示出的，用于顺序应用 600 的 CFG 600 包括节点 602、节点 604 和节点 606，以及回边 608。在一个实施方案中，线程划分主要着重于识别顺序应用程序的 PPS 循

环。代表性地，CFG 600 的 PPS 循环体由节点 604、节点 606 和回边 608 组成。在一个实施方案中，如图 6B 中所示，通过移除节点 602、边 603 和回边 608 来形成 CFG 循环 610。在一个实施方案中，CFG 循环 610 用于使能 (enable) 顺序应用程序的 PPS 循环体的变换，以最小化临界区。

[0044] 在处理框 510，CFG 循环的节点被更新，以包括在边界指令对中的顺序应用程序的已标识临界区。在一个实施方案中，最开始将 AWAIT 和 ADVANCE 操作对插入图 6B 的 CFG 循环 610 的顶端 604 和底端 606。在一个实施方案中，AWAIT 和 ADVANCE 操作被视为临界区的边界。在处理框 520，CFG 循环的节点被修改，以减少在相应的边界指令对之间的指令量，从而形成修改的 CFG 循环。

[0045] 图 7 是根据本发明的一个实施方案，示出用于更新图 5 的处理框 510 的 CFG 循环的节点的方法 511 的流程图。在处理框 512，选择顺序应用程序的已标识临界区。在一个实施方案中，由输入程序标识顺序应用程序的临界区。如这里所描述的，每个已标识临界区与一个循环承载变量相对应。在一个实施方案中，最开始每个临界区包括 PPS 循环中的所有指令。因此，初始临界区的范围就是整个 PPS 循环体。代表性地，在处理框 514，将 AWAIT 指令插入 CFG 循环的顶端节点。类似地，在处理框 516，将 ADVANCE 指令插入 CFG 循环的底端节点。

[0046] 相应地，在处理框 518，针对顺序应用程序的每个已标识临界区重复处理框 514-516。例如，如参照图 6B 所示出的，在节点 604 内插入 AWAIT 操作，而在节点 606 中插入 ADVANCE 操作。然而，在临界区之间或者在临界区内的操作的顺序的线程次序执行要求减少了所执行的并行执行量，因为在临界区中的操作量增加了。在一个实施方案中，被包括在临界区中的代码量被最小化，以增加程序线程的并行执行。在一个实施方案中，使用数据流分析以及代码移动来最小化要求以严格顺序的线程次序进行执行的代码段。

[0047] 相应地，一旦为用于顺序应用程序的所有已标识临界区的 CFG 循环 600 插入 AWAIT 和 ADVANCE 操作，则在 CFG 循环上执行代码移动，以减少被包括在由 AWAIT 和 ADVANCE 操作标识的临界区内的操作量。然而，本领域技术人员意识到可以使用其他数据分析或图论技术来执行临界区内的代码最小化，但是仍然落入所描述的发明的实施方案中。

[0048] 如这里所描述的，数据流分析不限于仅对变量的定义和使用（数据流）的计算。数据流分析提供用于计算关于通过程序或进程的路径的实际情况的技术。数据流分析概念的前提是控制流图 (CFG) 或简单的流图，例如参照图 6A 所示出的。CFG 是俘获控制流或程序的一部分的有向图。例如，CFG 可以代表进程大小的程序段。如这里所描述的，CFG 节点是基本块（永远按次序执行的代码部分），并且边 (edge) 代表在基本块之间的可能控制流。例如，如参照图 6A 所示出的，控制流图 600 由节点 602-606，以及边 603、605 和回边 608 组成。

[0049] 如这里所描述的，代码移动是一种用于块间和块内指令重新排序（提升 / 下降）的技术。在一个实施方案中，为了最小化已标识临界区中的指令 / 操作量，代码移动将不相关的代码移动到已标识临界区之外。为了执行块间和块内的指令重新排序，代码移动最开始标识移动候选指令 (motion candidate instruction)。在一个实施方案中，使用数据流分析来标识移动候选指令。代表性地，解决一系列的数据流问题来执行对已标识移动候选指令的提升和下降。

[0050] 图 8 是根据本发明的一个实施方案，示出用于修改图 5 的处理框 520 的 CFG 循环的节点的方法 522 的流程图。在处理框 524，使用代码移动，用固定的 AWAIT 边界指令来在 CFG 循环的节点内提升移动候选指令。在处理框 552，使用代码移动，用固定的 ADVANCE 操作来在 CFG 循环的节点内下降移动候选指令。在处理框 580，使用固定的 AWAIT 操作和固定的 ADVANCE 操作来在 CFG 循环的节点内提升移动候选指令。在一个实施方案中，方法 522 代表根据本发明的一个实施方案用于限制操作或减少以 AWAIT 和 ADVANCE 操作为边界的操作量的三阶段代码移动。

[0051] 在一个实施方案中，使用三阶段代码移动来最小化线程划分循环的已标识临界区内的操作量。代表性地，代码移动的前两个阶段分别用固定的 AWAIT 操作和固定的 ADVANCE 操作来执行代码移动。结果，ADVANCE 操作被放置到最优基本块，并且 AWAIT 操作被放置到最优基本块。在该实施方案中，代码移动的最后阶段用固定的 AWAIT 操作和固定的 ADVANCE 操作执行代码移动。代表性地，代码移动的最后阶段将不相关的指令移动到临界区之外，同时将 AWAIT 和 ADVANCE 操作放置到最优位置。

[0052] 图 9 是根据本发明的一个实施方案，示出用图 8 的处理框 524 的固定 AWAIT 操作来提升指令的方法 526 的流程图。在处理框 528，CFG 循环的基本块内的除 AWAIT 操作之外的每条指令被标识为移动候选指令。因此，AWAIT 操作不被标识为移动候选指令。在处理框 530，构建 CFG 循环的逆图 (inverse graph)。在一个实施方案中，根据由逆图指示的拓扑次序排序基本块。

[0053] 在处理框 550，确定是否不再从已标识移动候选指令检测到提升指令。直到是这种情况，在处理框 534，在 CFG 循环的基本块内提升从已标识移动候选指令检测到的提升指令。在处理框 552，如图 15A-15C 所示，根据顺序应用程序的相关图使用已标识移动候选指令的块内提升来重新排序 CFG 循环的源基本块中的移动候选指令。如这里所描述的，构造相关图来图示 PPS 循环体的数据相关性 (data dependence)，以提供关于数据相关性的信息。因此，提升或下降任何移动候选指令不能违背原始程序的数据相关性。如这里所描述的，相关图示出节点间的数据相关性和节点间的控制相关性。

[0054] 如参照图 10 所示出的，流相关图 620 示出 AWAIT 操作 634、对临界区内 (例如循环承载变量 i) 的访问操作 640 之间的流相关性，以及在对循环承载变量 i 和 ADVANCE 操作 650 之间的流相关性关系。因此，流相关图 620 确保对循环承载变量 i 的访问被临界区 n 同步，以使能循环承载变量 i 的顺序的线程次序执行。

[0055] 图 11 是根据本发明的一个实施方案，示出用于在图 9 的处理框 534 的 CFG 循环的基本块内提升指令的方法 536 的流程图。在处理框 538，使基本块从提升队列 (hoist queue) 中出队列，作为当前块。在处理框 540，基于顺序应用程序的相关图从基本块的移动候选指令计算提升指令。

[0056] 在处理框 542，将计算的提升指令提升到相应的基本块，所述提升指令可以被放置在所述基本块中。在处理框 544，确定是否检测到其他的代码移动，例如通过提升计算的提升指令检测到的改变。当在处理框 544 检测到改变时，在处理框 546，使来自 CFG 循环的当前块的前驱块 (predecessor) 入队列到提升队列中。在处理框 548，重复处理框 538-546，直到提升队列为空。

[0057] 图 12 是根据本发明的实施方案，示出用于以图 11 的处理框 552 的固定 ADVANCE

指令下降检测到的下降指令的方法 554 的流程图。在处理框 556，通过数据流分析标识 CFG 循环的基本块中除 ADVANCE 操作以外的移动候选指令。如上面所提到的，数据流分析通过解答一系列数据流方程来标识提升指令以及下降指令。通常形成数据流方程来建立路径断定 (path predicate) 的逻辑真和逻辑假。

[0058] 路径断定是关于在程序执行期间沿特定的控制路径发生事情的语句 (statement)，所述控制路径在所有这样的路径上普遍地或存在性地被量化。如这里所描述的，控制流路径是 CFG 循环中的路径。例如，定义可到达性问题询问每个控制流节点 n 和每个变量定义 d，是否 d 可以到达 n，其中到达意味着定义将值赋给变量，并且该变量之后不会被再定义。例如，路径断定可以根据以下方程表达。

[0059] REACHDEF (节点 n，定义 d) ＝存在从开始到 n 的路径 p，从而

[0060] d 在 p 发生，并且在 d 之后不发生定义 (1)

[0061] 如这里所描述的，数据流方程将对路径断定的答案表示为描述在每个节点处的解的方程系统。即，对于 CFG 中的每个节点，我们能够对于感兴趣的定义是否到达节点说是或者否。例如，考虑如下形式的任何单个的三地址代码语句：

[0062] d_i:x: ＝ y op z (2)

[0063] 该程序语句定义变量 x。因此，如这里所描述的，如果在控制流图的节点中包括这样的语句，则包括该程序语句的控制流图的节点 (N) 被设置为生成定义 d_i，并且删除定义 x 的之前的程序语句中任何其他定义。当根据集合来分析时，建立以下关系式：

[0064] gen[N] ＝ (d_i)

[0065] kill[N] ＝ Dx-(d_i)

[0066] 其中 dx 指程序中 x 的其他定义 (3)

[0067] 相应地，考虑基本块 N，断定哪些定义到达基本块 N 要求分析基本块 N 的前驱块。例如，令符号≺代表 CFG 中两个节点上的前驱块关系，如果在控制流图中存在从 p->b 的边，则我们说 p 是 b 的前驱块。因此，基于前驱块关系，生成以下数据流方程。

[0068] $$in[B]=\bigcup_{P\prec B} out[P] \quad (4)$$

[0069] out[B] ＝ gen[B] ∪ (in[B]-kill[B]) (5)

[0070] 相应地，为了通过数据流分析计算移动候选 (motion candidate)，并且根据本发明的一个实施方案，对于每个指令 i，以下数据流方程可以如下描述：

[0071] GEN[i] ＝ {N|AWAIT(N) 如果 i 是 AWAIT} (6)

[0072] KILL[i] ＝ {N|AWAIT(N) 如果 i 是 ADVANCE} (7)

[0073] $$IN[i]=\bigcup_{P\in Pred(i)} OUT[P] \quad (8)$$

[0074] OUT[i] ＝ GEN[i] ∪ (IN[i]-KILL[i]) (9)

[0075] 相应地，在一个实施方案中，如下计算移动候选：(1)AWAIT 被标识为移动候选；(2) 仅当 IN[i] 不等于空集 (0) 时指令 i 为候选。换言之，对于每个指令 i，根据数据流方程生成位向量，以确定指令 i 是否要被标识为移动候选。在所描述的实施方案中，ADVANCE 操作不被标识为移动候选。

[0076] 再次参照图 12，一旦标识了移动候选，在处理框 558，用 CFG 循环的基本块初始化下降队列。在一个实施方案中，基于 CFG 循环中的拓扑次序在下降队列中对基本块进行排

序。在处理框 560，移动候选指令在基本块之间下降，直到在处理框 574 中不再从移动候选指令检测到下降指令。在处理框 576，根据顺序应用程序的相关图，如图 15A-15C 所示，使用已标识移动候选指令的块内下降来使移动候选指令在包括 ADVANCE 操作的基本块内重排序。

[0077] 换言之，对下降指令以及提升指令的检测和下降不应该违背由顺序应用程序的相关图所指示的任何数据相关性，或者例如控制相关性。换言之，与相关图的一致性确保从顺序应用程序生成程序线程。在一个实施方案中，程序线程维持原始程序的顺序语义，并且执行与顺序应用程序的 PPS 循环相对应的程序线程迭代之间的相关性。

[0078] 图 13 是根据本发明的一个实施方案，示出用于下降图 12 的处理框 560 的移动候选指令的方法 562 的流程图。在处理框 564，使基本块从下降队列 (sink queue) 中出队列，作为当前块。在处理框 566，基于顺序应用程序的相关图为在基本块中标识的移动候选指令计算下降指令，以维持顺序应用程序的顺序语义。在处理框 567，计算的下降指令下降到相应的基本块中。在处理框 570，如果在处理框 568 检测到作为计算的下降指令下降的结果的代码移动改变，则使 CFG 循环中当前块的后继块 (successor) 入队列到下降队列中。在处理框 572，重复处理框 564-570，直到下降队列为空。

[0079] 图 14 是根据本发明的一个实施方案，示出用图 8 的处理框 580 的固定的 AWAIT 指令和 ADVANCE 指令来提升移动候选指令的方法 582 的流程图。在处理框 584，使用数据流分析，用固定的 AWAIT 操作和 ADVANCE 操作来从基本块中检测移动候选指令。在一个实施方案中，根据上面描述的数据流分析计算移动候选。

[0080] 在一个实施方案中，用 CFG 循环的基本块初始化提升队列。在一个实施方案中，基于 CFG 循环中的拓扑次序排序基本块。在处理框 586，在基本块之间提升移动候选指令，直到不再检测到提升指令。在处理框 588，基于顺序应用程序的相关图，在包括 AWAIT 指令的基本块内提升检测到的提升指令，以保持原始程序次序。

[0081] 在一个实施方案中，处理框 588 描述块内提升。在这样的实施方案中，在基本块中提升除 AWAIT 操作和 ADVANCE 操作之外的移动候选，所述基本块包括尽可能高的 AWAIT 操作而不违背相关图。在一个实施方案中，被提升到最外层的临界区之外的指令不再被认为是移动候选。例如，如参照图 15A-15C 所示出的，被提升或下降到最外层 ADVANCE 操作 664 或最外层 AWAIT 操作 662 之外的指令 (667/668) 不再被视为提升候选指令或下降候选指令。一旦在 CFG 循环上执行了代码移动，就形成修改的 CFG，所述修改的 CFG 可以用来形成顺序应用程序的并行版本的程序线程。

[0082] 图 16 是根据本发明的一个实施方案，示出用于将顺序应用程序划分为多个应用程序划分线程的方法 590 的流程图。在处理框 592，使用处理框 520（图 5）的修改的控制流图来形成顺序应用程序的程序线程。一旦形成多个程序线程，在处理框 594，在多线程体系结构的各个线程中并发地执行所述多个程序线程。在一个实施方案中，参照图 4 示出程序线程的并发执行。

[0083] 相应地，在一个实施方案中，线程划分编译器使用三阶段代码移动来提供顺序应用程序的自动多线程变换，以达到提高的并行化性。在多线程体系结构中，在任何一个时刻只有一个线程有效。因此，通过将存储器访问与计算或者由其他线程执行的存储器访问重叠来隐藏存储器延时，可以大大提高网络分组处理级的行处理速率 (line rate)。

[0084] 可替换的实施方案

[0085] 已经描述了用于提供多个程序线程的线程划分编译器的一个实现的几个方面。然而，线程划分编译器的各种实现提供包括、补足、补充和/或替代上述特征的很多特征。在不同实施方案的实现中，特征可以被实现为编译器的一部分或硬件/软件翻译处理的一部分。此外，出于解释的目的，前面的描述使用具体的术语来提供对本发明的实施方案的全面理解。然而，本领域的技术人员将会清楚，不要求所述具体细节来实践本发明的实施方案。

[0086] 此外，尽管这里描述的实施方案指向线程划分编译器，但是本领域技术人员将会意识到本发明的实施方案可以应用于其他系统。事实上，如由所附权利要求书所定义的那样，用于在临界区中执行代码移动的数据分析或图论技术落入本发明的实施方案中。为了最佳地解释本发明的实施方案的原理和它的实践应用，选择并描述上述实施方案。这些实施方案被选择，由此使本领域的其他技术人员能够用适合于所预期的特定使用的各种修改来最佳地利用本发明和各种实施方案。

[0087] 应该理解，尽管在前面的描述中已经阐述本发明的各种实施方案的很多特性和优势以及本发明的各种实施方案的结构和功能的细节，但是此公开仅仅是说明性的。在一些情况下，仅使用一个这样的实施方案来详细描述某些子配件(subassembly)。然而，应该认识到并期望，可以在本发明的其他实施方案中使用这样的子配件。在本发明的实施方案的原理内，在由所附权利要求书被表达的语句的宽泛的通用意义所指定的程度上，可以在细节上，特别是在组成部分的结构和管理上作出改变。

[0088] 已经公开示例性实施方案和最佳模式，可以对已公开实施方案作出修改和变化，同时仍然落入所附权利要求书所定义的本发明的实施方案的范围内。

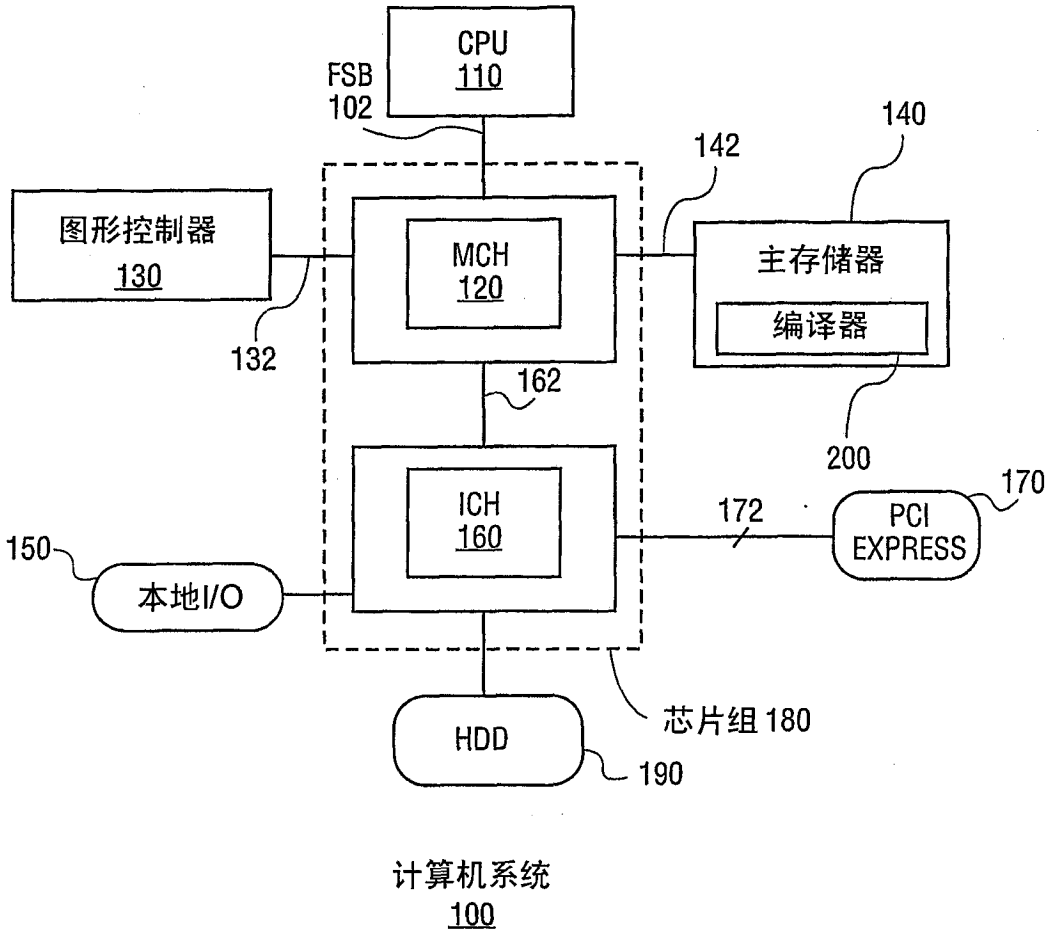

CPU
110
FSB
102
140
142
图形控制器
130
MCH
120
主存储器
编译器
132
162
200
ICH
160
172
PCI
EXPRESS
170
150
本地I/O
芯片组 180
HDD
190
计算机系统
100

图 1

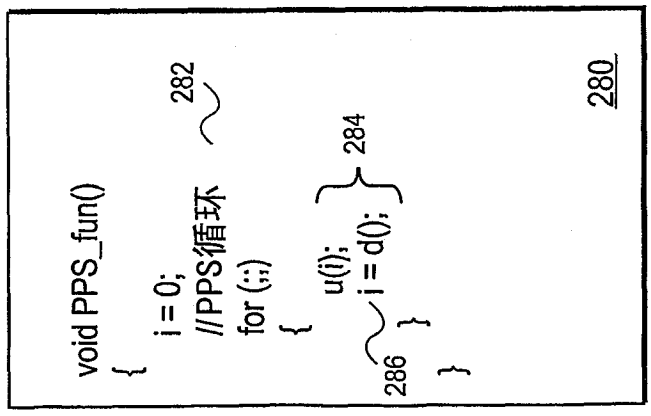

void PPS_fun()
{
i = 0;
//PPS循环
for (;;)
{
u(i);
i = d();
}
}
282
284
286
280

图 2A

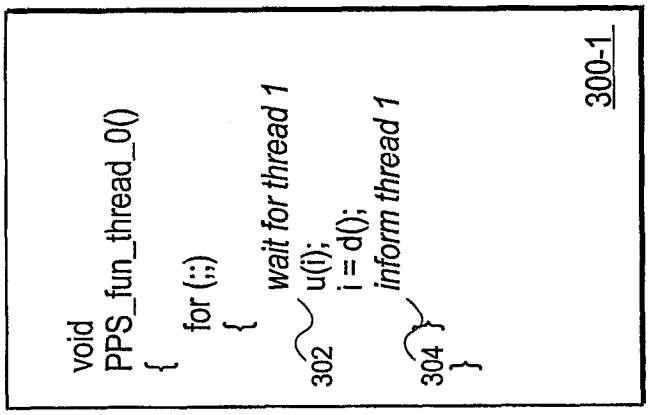

void
PPS_fun_thread_0()
{
for (;;)
{
wait for thread 1
u(i);
i = d();
inform thread 1
}
}
302
304
300-1

图 2B

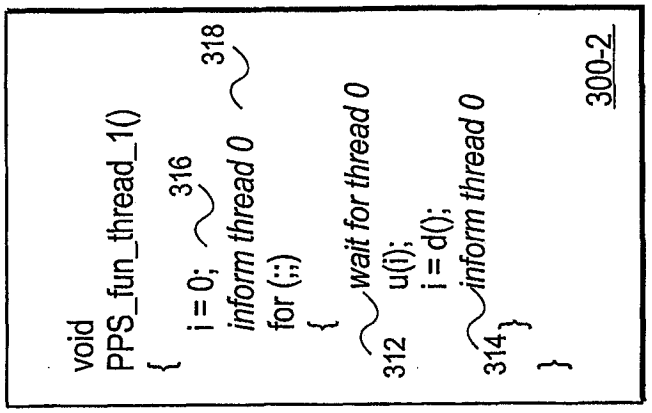

void
PPS_fun_thread_1()
{
i = 0;
inform thread 0
for (;;)
{
wait for thread 0
u(i);
i = d();
inform thread 0
}
}
316
318
312
314
300-2

图 2C

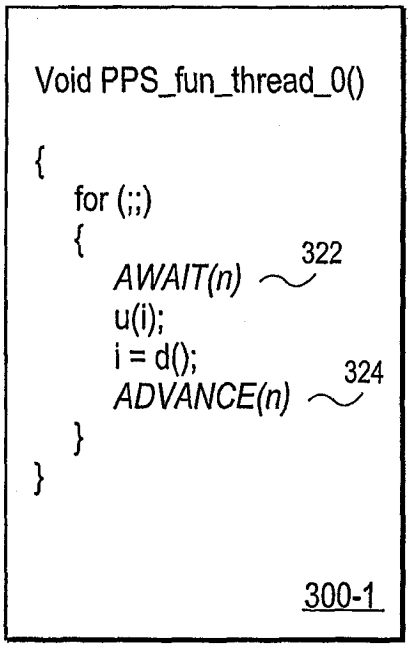

Void PPS_fun_thread_0()
{
for (;;)
{
AWAIT(n)
322
u(i);
i = d();
ADVANCE(n)
324
}
}
300-1

图 3A

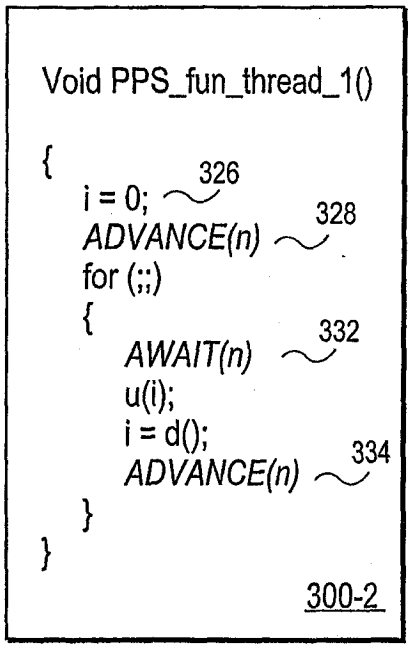

Void PPS_fun_thread_1()
{
i = 0;
326
ADVANCE(n)
328
for (;;)
{
AWAIT(n)
332
u(i);
i = d();
ADVANCE(n)
334
}
}
300-2

图 3B

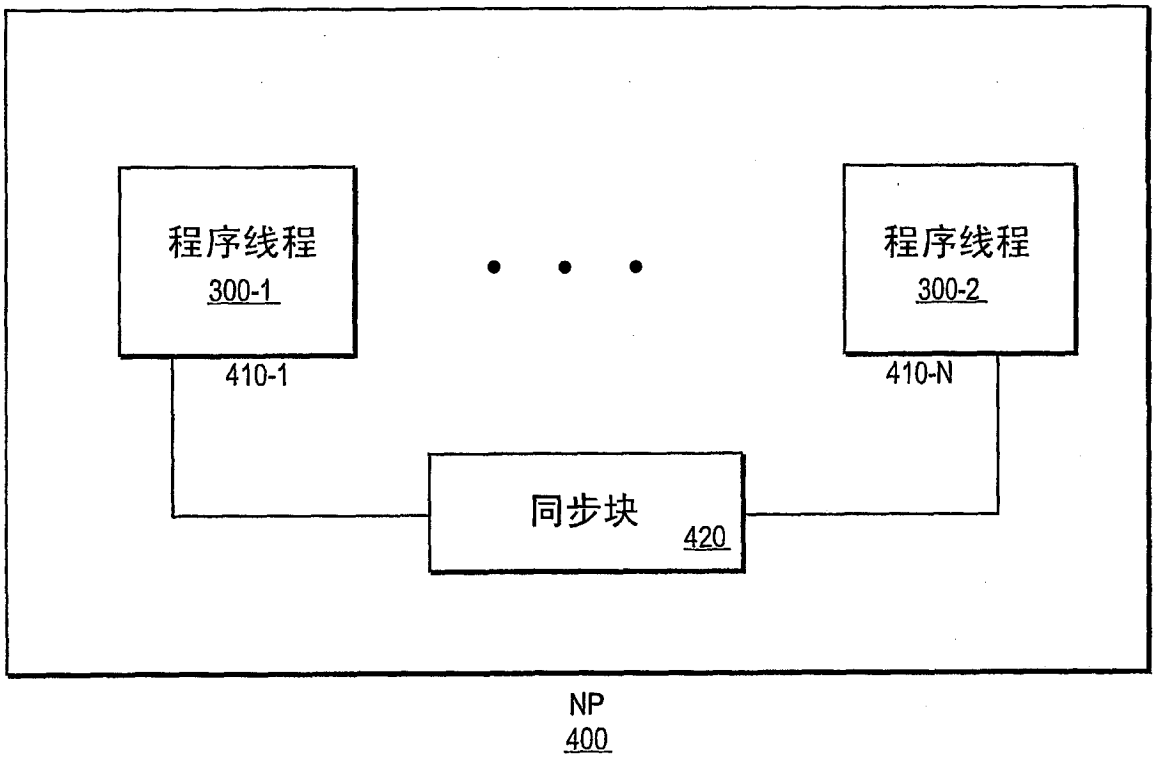

程序线程
300-1
410-1
程序线程
300-2
410-N
同步块
420
NP
400

图 4

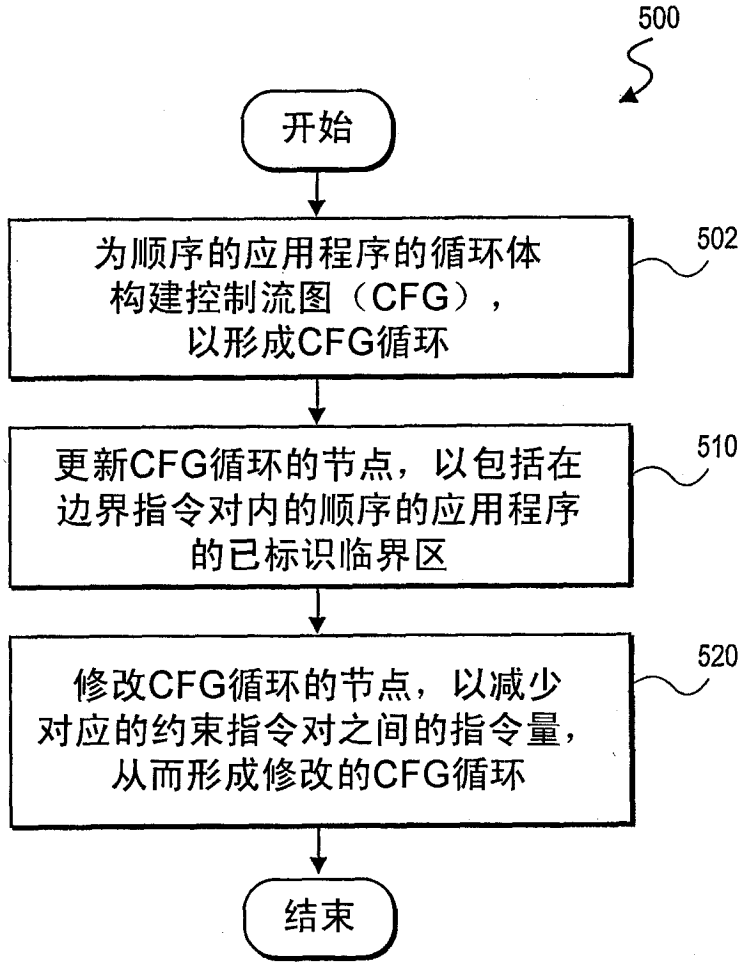

500
开始
为顺序的应用程序的循环体构建控制流图（CFG），以形成CFG循环
502
更新CFG循环的节点，以包括在边界指令对内的顺序的应用程序的已标识临界区
510
修改CFG循环的节点，以减少对应的约束指令对之间的指令量，从而形成修改的CFG循环
520
结束

图 5

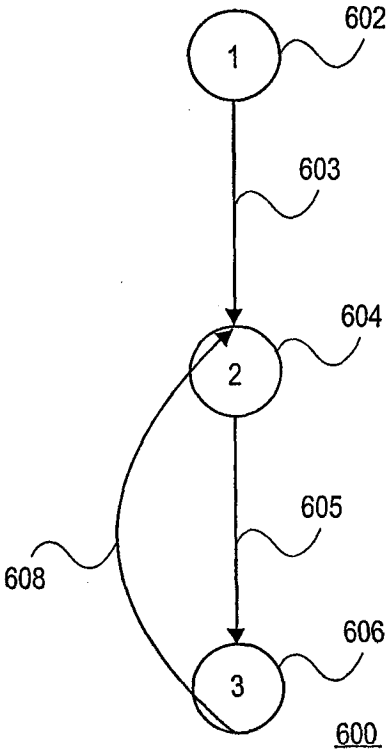


图 6A

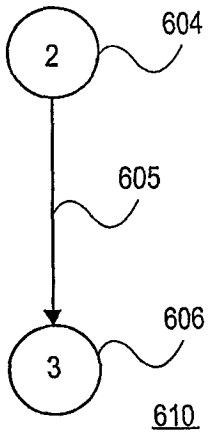


图 6B

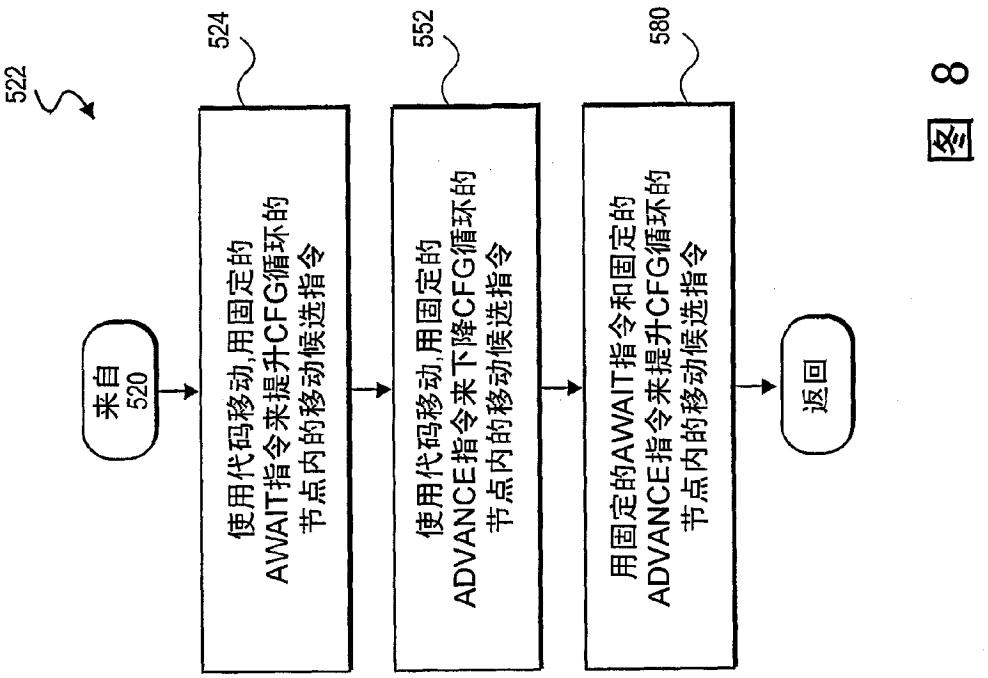

522
524
552
580
来自 520
使用代码移动,用固定的 AWAIT指令来提升CFG循环的 节点内的移动候选指令
使用代码移动,用固定的 ADVANCE指令来下降CFG循环的 节点内的移动候选指令
用固定的AWAIT指令和固定的 ADVANCE指令来提升CFG循环的 节点内的移动候选指令
返回

图 8

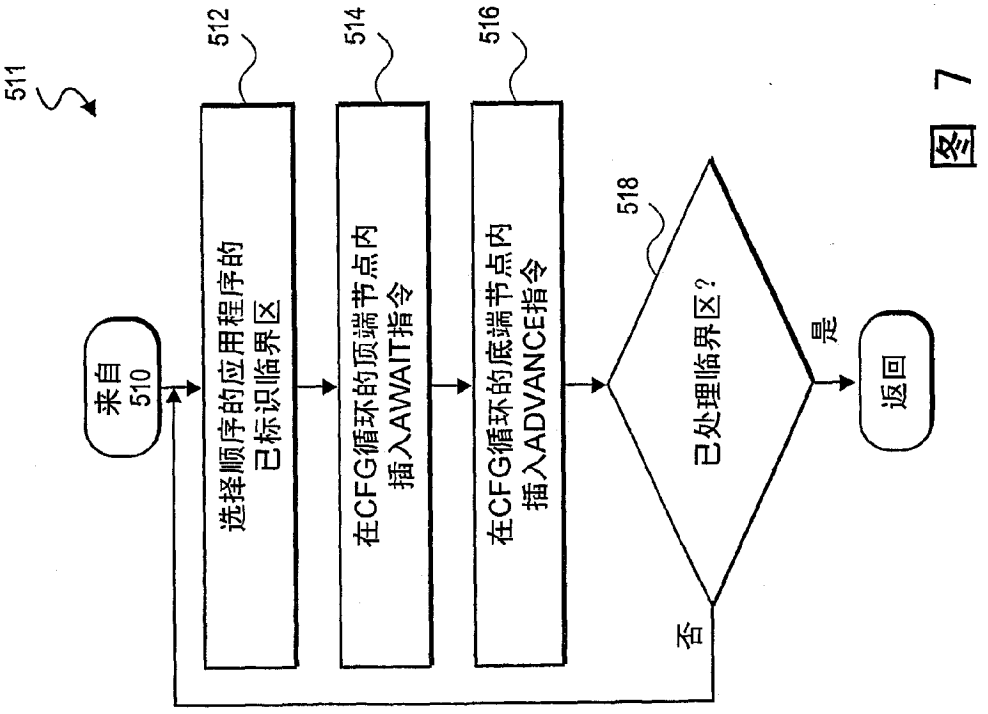

511
512
514
516
518
来自 510
选择顺序的应用程序的 已标识临界区
在CFG循环的顶端节点内 插入AWAIT指令
在CFG循环的底端节点内 插入ADVANCE指令
已处理临界区?
是
否
返回

图 7

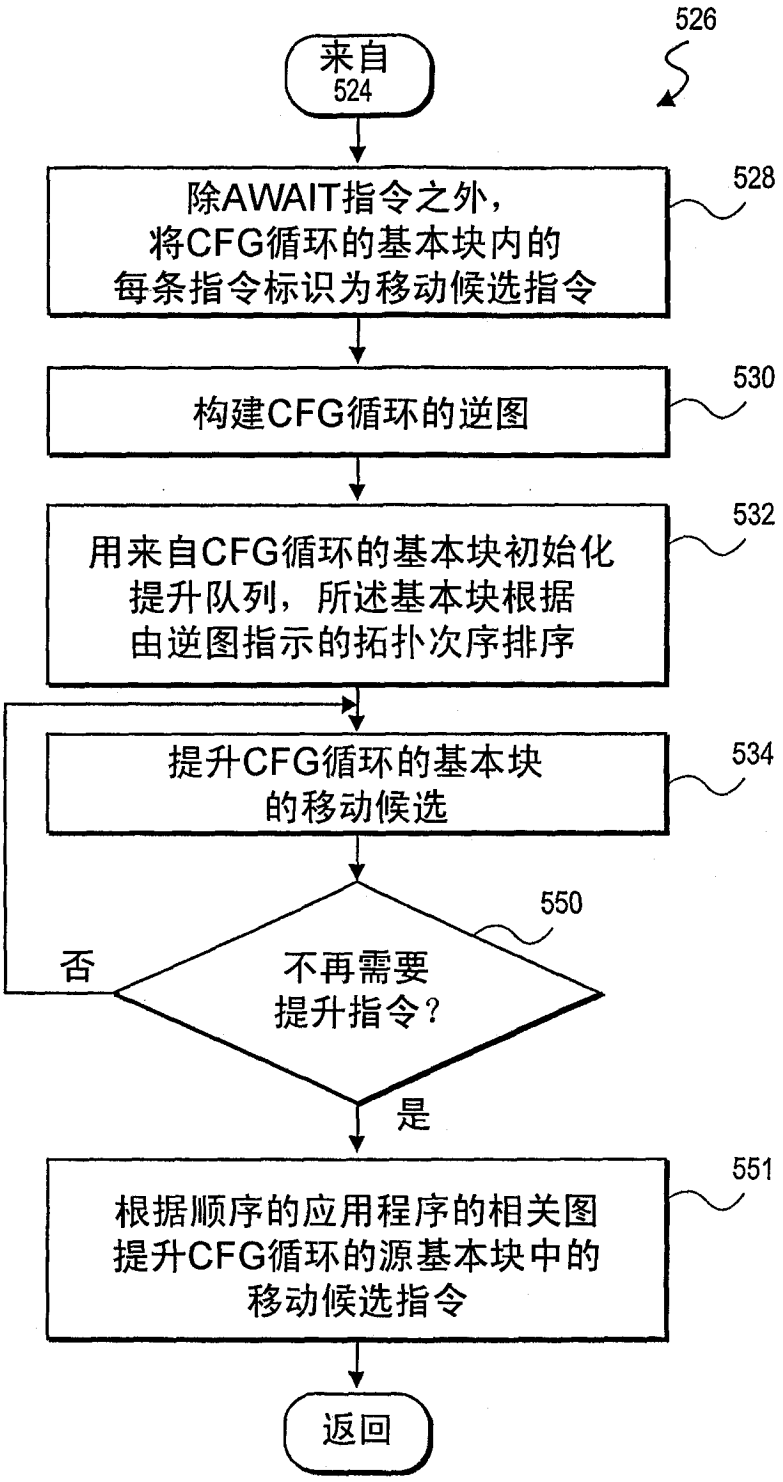

526
来自
524
除AWAIT指令之外，
将CFG循环的基本块内的
每条指令标识为移动候选指令
528
构建CFG循环的逆图
530
用来自CFG循环的基本块初始化
提升队列，所述基本块根据
由逆图指示的拓扑次序排序
532
提升CFG循环的基本块
的移动候选
534
不再需要
提升指令？
550
否
是
根据顺序的应用程序的相关图
提升CFG循环的源基本块中的
移动候选指令
551
返回

图 9

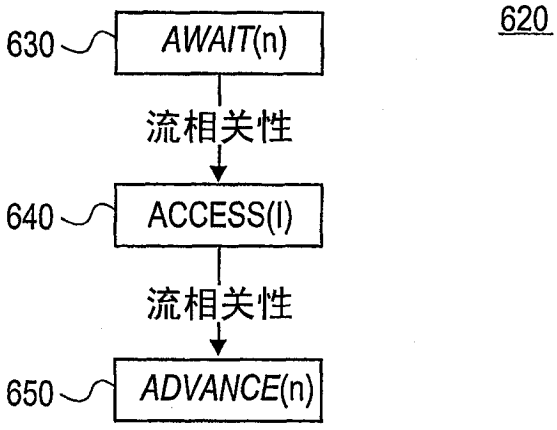


图 10

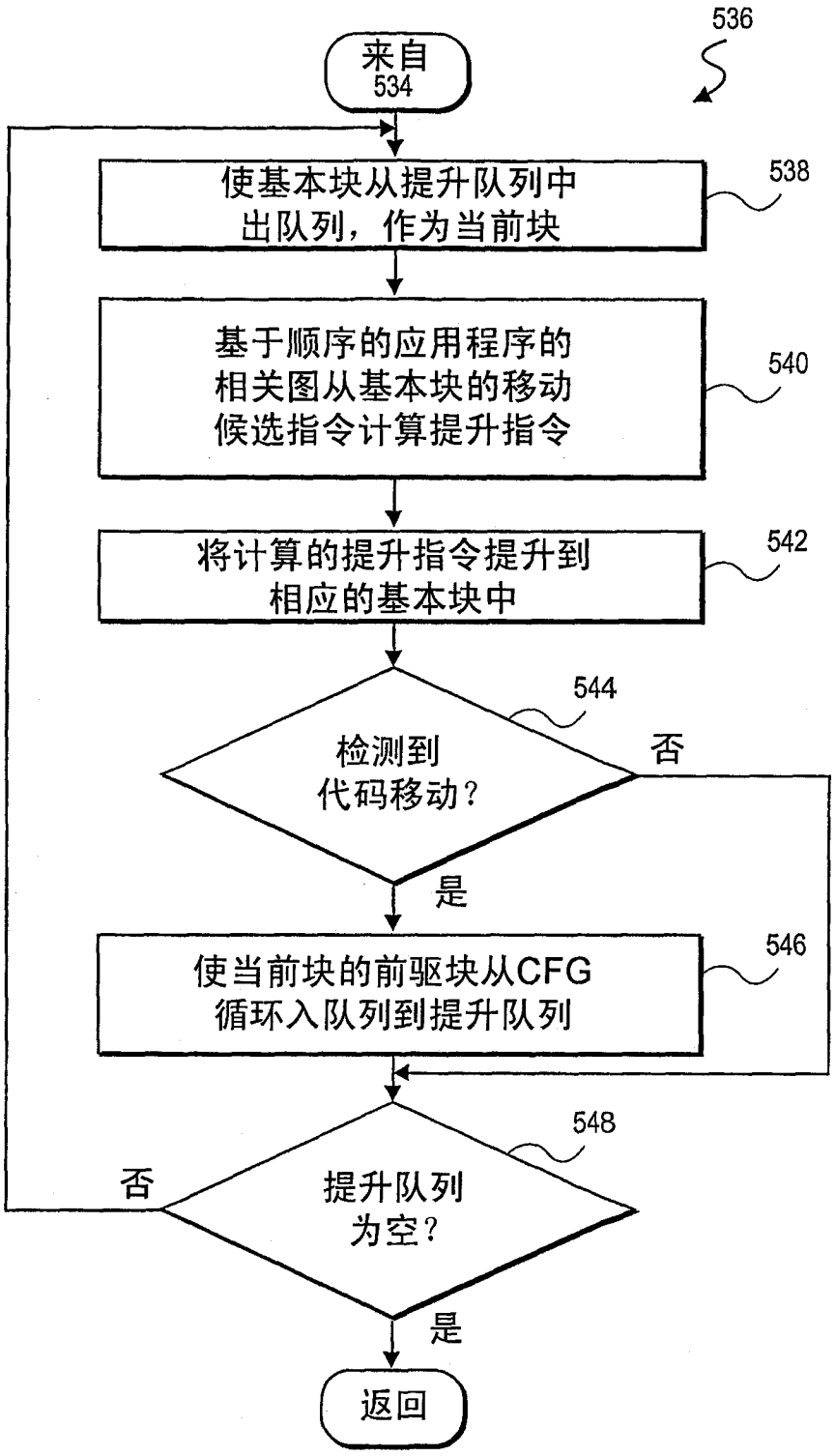

536
来自
534
使基本块从提升队列中
出队列，作为当前块
538
基于顺序的应用程序的
相关图从基本块的移动
候选指令计算提升指令
540
将计算的提升指令提升到
相应的基本块中
542
544
检测到
代码移动？
否
是
使当前块的前驱块从CFG
循环入队列到提升队列
546
548
否
提升队列
为空？
是
返回

图 11

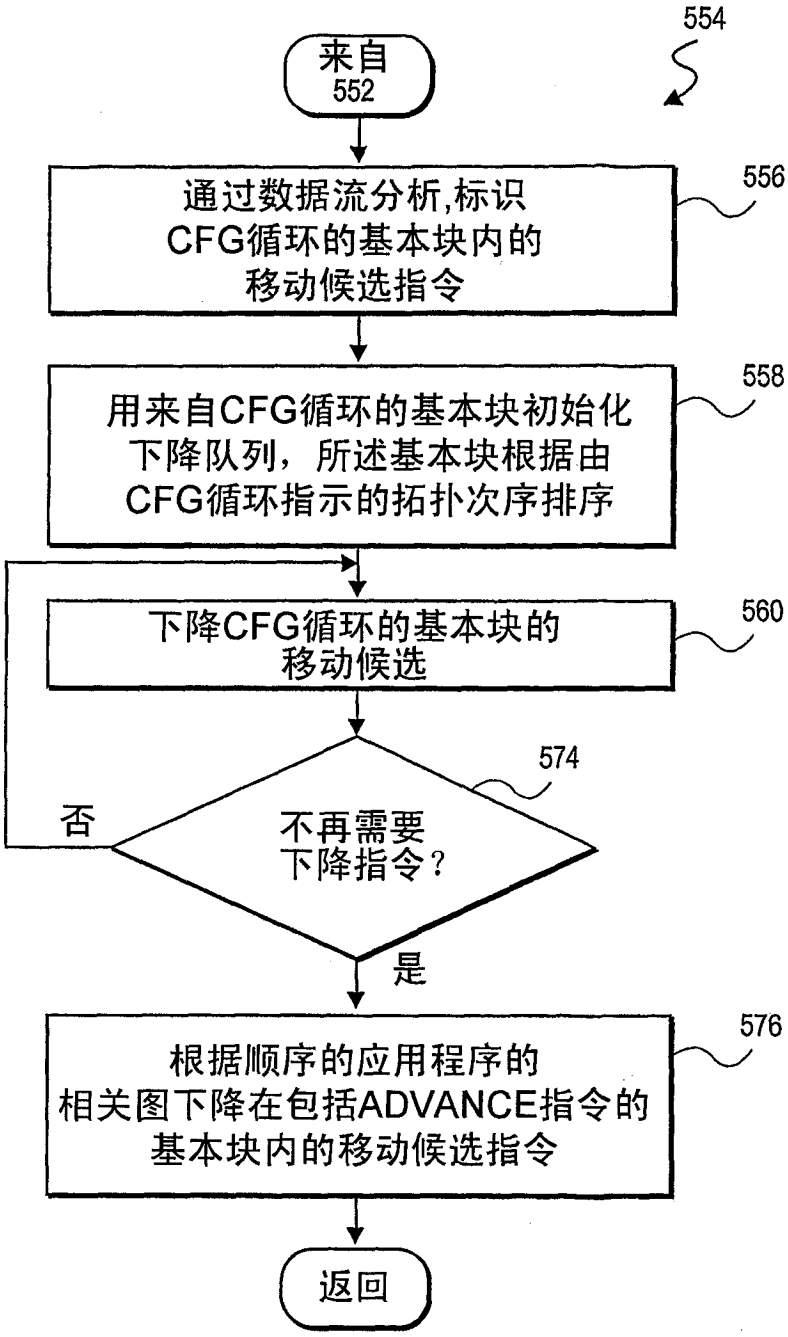

554
来自
552
通过数据流分析,标识
CFG循环的基本块内的
移动候选指令
556
用来自CFG循环的基本块初始化
下降队列，所述基本块根据由
CFG循环指示的拓扑次序排序
558
下降CFG循环的基本块的
移动候选
560
574
不再需要
下降指令？
否
是
根据顺序的应用程序的
相关图下降在包括ADVANCE指令的
基本块内的移动候选指令
576
返回

图 12

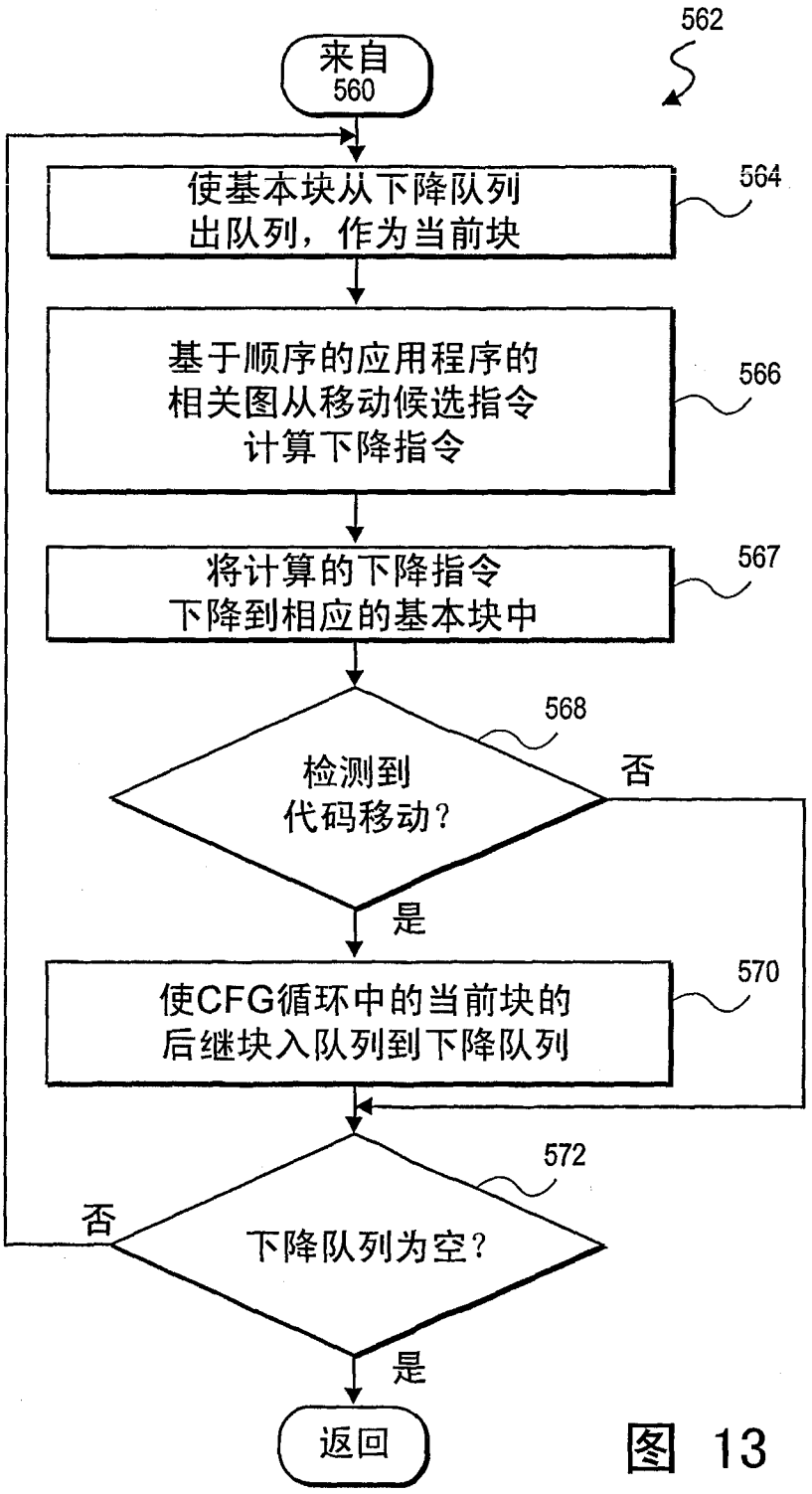

562
来自
560
使基本块从下降队列
出队列，作为当前块
564
基于顺序的应用程序的
相关图从移动候选指令
计算下降指令
566
将计算的下降指令
下降到相应的基本块中
567
检测到
代码移动？
568
否
是
使CFG循环中的当前块的
后继块入队列到下降队列
570
下降队列为空？
572
否
是
返回

图 13

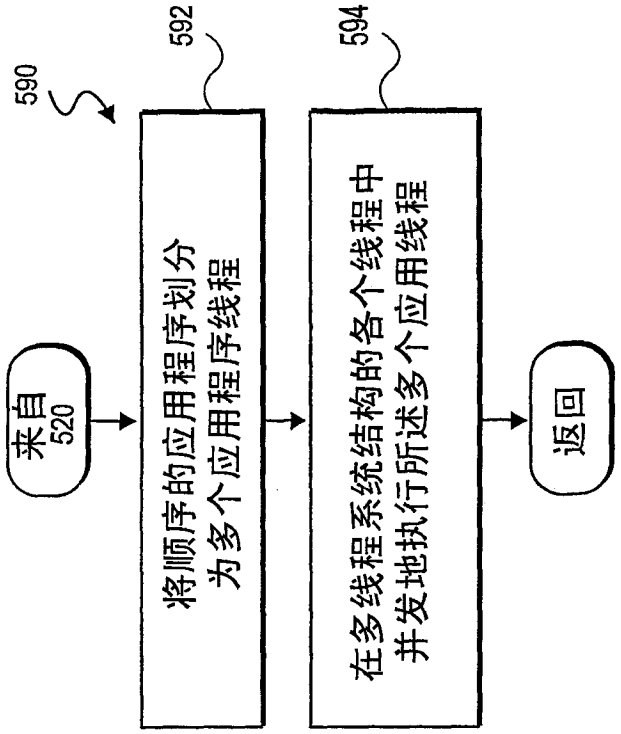

590
592
594
来自
520
将顺序的应用程序划分
为多个应用程序线程
在多线程系统结构的各个线程中
并发地执行所述多个应用线程
返回

图 16

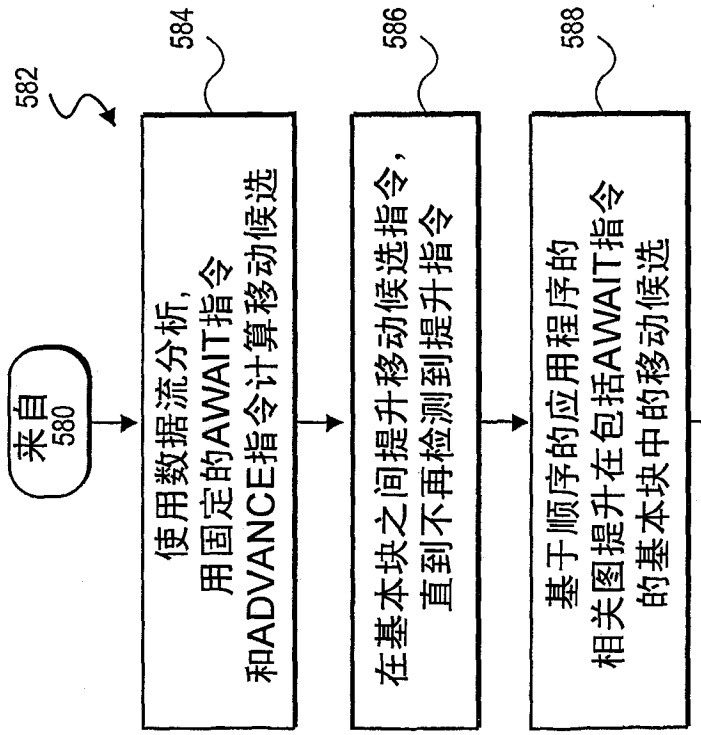

582
584
586
588
来自
580
使用数据流分析,
用固定的AWAIT指令
和ADVANCE指令计算移动候选
在基本块之间提升移动候选指令,
直到不再检测到提升指令
基于顺序的应用程序的
相关图提升在包括AWAIT指令
的基本块中的移动候选
返回

图 14

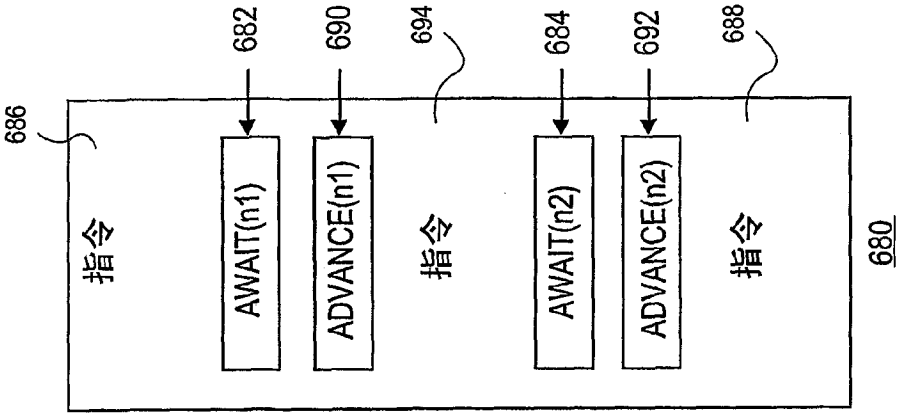


图 15A

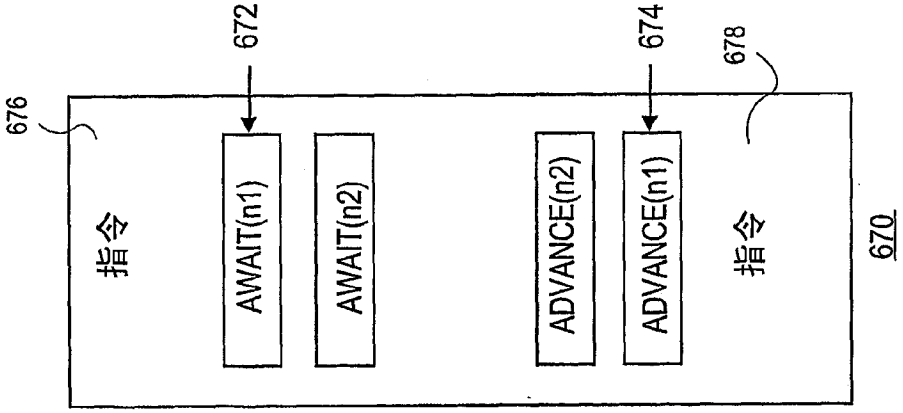


图 15B

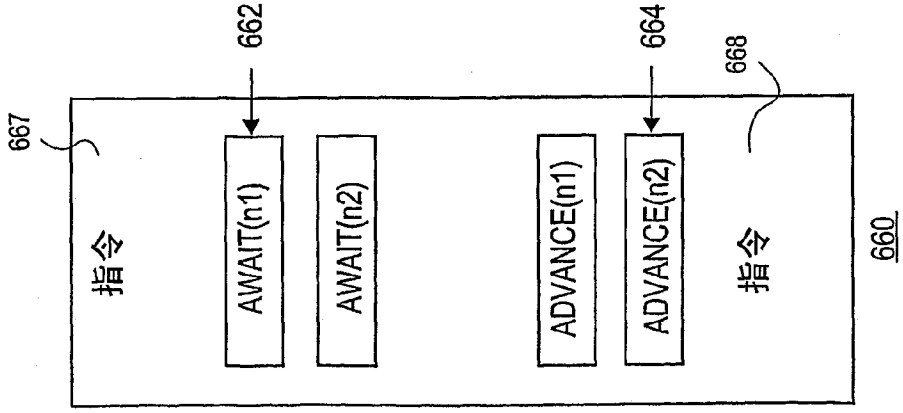


图 15A