[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/445

[12] 发明专利说明书

[21] ZL 专利号 98124672.9

[45] 授权公告日 2004 年 3 月 24 日

[11] 授权公告号 CN 1143210C

[22] 申请日 1998.10.5 [21] 申请号 98124672.9

[30] 优先权

[32] 1997.10.6 [33] US [31] 944330

[71] 专利权人 太阳微系统有限公司

地址 美国加利福尼亚州

[72] 发明人 U·赫尔茨勒 L·巴克

审查员 韩 燕

[74] 专利代理机构 中国专利代理(香港)有限公司

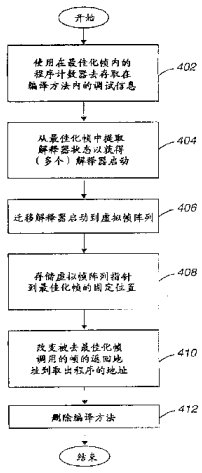
代理人 王 勇 王忠忠

权利要求书 4 页 说明书 14 页 附图 8 页

[54] 发明名称 动态去最佳化编译启动的方法和装置

[57] 摘要

依照本发明的一个方面，去最佳化一编译方法的计算机执行方法产生一数据结构，与控制堆栈相独立的数据结构被安排存储与该编译方法相关的信息。引用指示器，例如指针，被产生以将该数据结构和该帧相关。被编译在最佳化第一状态的该方法然后被去最佳化到最佳化的第二状态，在最佳化第一状态的方法可以被放弃，因此去最佳化该帧。当控制返回到该去最佳化帧时，迁移程序产生至少一新堆栈帧，使用在最佳化第二状态的方法可以继续执行。



ISSN1008-4274

1. 去最佳化一编译函数的计算机执行方法，该函数是在最佳化的第一状态，该函数进而是与在控制堆栈上放置的一帧相关，该计算机执行方法包括:

产生一数据结构，该数据结构独立于控制堆栈，其中，该数据结构包括与该函数相关的信息，该信息是从该函数和该帧的至少一个获得的;

产生引用指示器，该引用指示器与该帧相关，其中该引用指示器被安排引用该数据结构; 和

在该数据结构产生后，去最佳化该函数到最佳化的第二状态。

2. 权利要求 1 的计算机执行方法，其特征在于，所述数据结构是不依赖于机器的，和去最佳化该函数包括删除在最佳化的第一状态的该函数。

3. 前述权利要求任何一个的计算机执行方法，其特征在于，所述最佳化的第二状态是解释状态。

4. 权利要求 1 或 2 的计算机执行方法，其特征在于，所述帧是位于所述控制堆栈的中部。

5. 权利要求 1 的计算机执行方法，其特征在于，去编译该函数包括去最佳化包括在该函数内的最佳化启动。

6. 权利要求 5 的计算机执行方法，还包括在所述数据结构内存储所述去最佳化启动。

7. 权利要求 6 的计算机执行方法，进一步包括:

使用存储在所述数据结构内的去最佳化启动产生一新帧; 和

将新帧压入控制堆栈以代替该帧。

8. 前述权利要求 1、6 和 7 任何一个的计算机执行方法，其特征在于，所述数据结构是不依赖机器的，该计算机执行方法进而包括从第一计算机系统在向第二计算机系统传送所述数据结构。

9. 权利要求 8 的计算机执行方法，其特征在于，所述数据结构是在第一计算机系统中产生的，所述计算机执行方法进一步包括使用存储在所述数据结构内的去最佳化启动在第二计算机系统中产生一新帧。

10. 动态去最佳化在控制堆栈上第一帧的计算机执行方法，该第一帧具有一相关的编译函数，该编译函数包括编译启动，该计算机执行方法包括:

去最佳化与所述第一帧相关的编译函数，所述第一帧位于在控制堆栈上的第二帧的下面，其中，去最佳化该编译函数包括去最佳化编译的启动，去最佳化编译函数进而包括产生所述编译启动的解释器等价物;

从所述控制堆栈中移除所述第二帧;

从所述编译启动的解释器等价物中产生至少一个解释器帧; 和

将至少一解释器帧压入堆栈以取代所述第一帧。

11. 权利要求 10 的计算机执行方法，进而包括在数据结构内存储所述编译启动的解释器等价物，其中，该数据结构与所述第一帧相关。

12. 权利要求 11 的计算机执行方法，进而包括在所述至少一个解释器帧产生之后删除该数据结构。

13. 动态去最佳化一编译函数的计算机执行方法，该编译函数与从控制堆栈上大量帧中选出的一第一帧相关，该控制堆栈包括被安排以识别在控制堆栈上的当前帧的堆栈指针，该计算机执行方法包括:

产生一数据结构，该数据结构独立于所述控制堆栈，该数据结构包括与所述第一帧相关的解释器信息和与所述编译函数相关的函数信息; 其中，所述第一帧包括对该数据结构的引用;

删除所述编译的函数;

当堆栈指针识别所述第一帧为控制堆栈上的当前帧时，取出包括在所述数据结构中的解释器信息，其中，取

出解释器信息产生至少一个解释器帧；

将该至少一个解释器帧压入所述控制堆栈，其中，所述第一帧由该至少一个解释器帧所取代。

14. 权利要求 13 的计算机执行方法，进而包括：

存取与所述编译函数相关的存取信息；和

放置所述解释器信息到所述数据结构，其中，放置所述解释器信息到所述数据结构包括使用存取信息对所述解释器信息进行存取。

15. 权利要求 14 的计算机执行方法，其特征在于，放置所述解释器信息到所述数据结构包括：

从所述编译函数中提取函数信息；

从所述第一帧中提取所述解释器信息；

使用所述解释器信息和函数信息产生至少一个解释的启动；和

迁移该至少一个解释的启动到所述数据结构。

16. 权利要求 13-15 任何一个的计算机执行方法，其特征在于，在所述解释器信息被取出之前，所述编译函数被删除。

17. 权利要求 13-15 任何一个的计算机执行方法进而包括在堆栈上产生迁移帧，其中，所述迁移帧被安排以将所述至少一个解释器帧和从所述控制堆栈上的大量帧中选取的一第二帧相关。

18. 去最佳化一个函数的计算机系统，该函数是在最佳化的第一状态，该函数进一步与位于在控制堆栈上的一帧相关，该计算机系统包括：

一数据结构，该数据结构独立于所述控制堆栈，其中，数据结构包括与该函数相关的信息；

一引用指示器，该引用指示器与所述帧相关，其中，该引用指示器被安排引用所述数据结构；和

一去最佳化器，被安排去最佳化所述帧为最佳化第二状态，其中，该最佳化第二状态比所述最佳化第一状态有

较低的最佳化，该去最佳化器进而被安排去最佳化该函数为最佳化的第二状态。

19. 权利要求 18 的计算机系统，其特征在于，所述数据结构不依赖机器，所述去最佳化器进而被安排删除所述在最佳化第一状态的函数。

20. 权利要求 18 或 19 的计算机系统，其特征在于，所述最佳化第二状态是解释的状态。

21. 去最佳化一编译函数的计算机执行方法，该函数是在最佳化的第一状态，该函数进而是与控制堆栈上的一个位置相关，该计算机执行方法包括:

产生一数据结构，该数据结构独立于控制堆栈，其中，该数据结构包括与该函数相关的信息，该信息是从该函数和该控制堆栈上的所述位置获得的；和

产生一识别装置，该识别装置被安排使该数据结构与该控制堆栈关联。

22. 权利要求 21 的计算机执行方法，其特征在于，所述数据结构不依赖机器，所述计算机执行方法进一步包括在一第一计算机系统和一第二计算机系统之间传送所述数据结构。

23. 权利要求 21 的计算机执行方法，进一步包括:

在所述数据结构产生之后，去编译该函数为最佳化的第二状态。

24. 前述权利要求 21-23 任何一个的计算机执行方法，其特征在于，所述最佳化的第二状态是解释的状态。

25. 权利要求 24 的计算机执行方法，其特征在于，去编译该函数包括去最佳化在该函数中包含的最佳化启动。

动态去最佳化编译启动的方法和装置

本发明是关于在软件应用中去最佳化(deoptimizing)编译启动的方法和装置，更具体而言，本发明是关于在整个执行计算机程序时急切执行去最佳化编译启动的方法和装置。

计算机系统经常通过网络连接起来，例如计算机系统的局域网，因特网和内部网，这样它们可以共享资源．共享资源经常包括软件应用．一般而言，软件应用，或计算机程序可以不同格式分发到不同计算机系统，这是基于这样的事实，每个计算机系统所需的软件应用是以特定的格式。替换地，计算机程序可以不依赖机器的形式传送到计算机系统，例如以字节编码的形式，以使计算机程序的一种形式能被不同的计算机系统利用。

当计算机程序以不依赖机器的形式传送时，可以直接地解释程序，或者程序可以翻译成依赖机器的编码，即“机器码”，直接解释的程序比翻译成机器码的程序在计算机系统中占据较少的空间．然而，在多数情况下，直接解释的程序比翻译成机器码的程序具有较慢的执行速度．决定是否直接解释计算机程序，或是把计算机程序翻译成机器码，经常取决于与执行速度相关的空间的相对重要性。

一些计算机系统可以被安排支持解释的代码和编译的或机器码两者。在支持解释和编译代码两者的系统上执行程序的过程中，取消或删除编译的方法经常是有益的。取消包括编译启动的编译方法一般在计算机系统中节省空间。因此，当在计算机系统中需要附加的空间时，编译方法可以被删除，和由解释码等价物取代，因为解释的方法比它的编译等价物占据较小的空间。

进而，编译代码不得不被放弃，这是因为代码基于这些代码不再有效的假定。例如，编译代码可以被放弃，是由于新的类被装入或由于程序代码被改变。当该假定不再有效，即不再被保持时，如果计算机系统继续执行编译的代码，错误的结果就可能发生。因此，计算机系统必须一般停止执行编译码以防止出错，这在编译码当前正在一个或多个启动记录中执行也是如此．

在计算机程序或应用中的每一个方法典型地与计算机控制堆栈上的至少一帧相关。即，当编译的方法被取消时，和编译方法相关的任何帧即编译的帧必须实质转换为解释器的帧。一般而言，如上所述，当编译的帧是无效时，编译的帧可以转换为解释器的帧，转换这样无效编译的帧为解释器帧使实质上是转换无效的帧为有效的帧，这是本领域技术人员所欢迎的。

帧是存储在控制堆栈中的启动记录，这在所属领域是众所共知的。帧是关于一种方法，和被安排为存储执行该方法的信息。存储在帧中的信息包括控制状态变量，局部变量和表示堆栈。控制堆栈是在程序执行期间被利用按照它们的顺序的调用次序去存储方法或功能的帧的堆栈。当一方法被调用时，该方法的帧被压入控制堆栈。接着，当方法结束时，和该方法相关的帧被弹出堆栈和在控制堆栈顶部的新帧的函数重新开始执行，即重新获得控制。

编译的帧不能被转换为翻译码等价物，直至编译的帧到达控制堆栈的顶部，这是由于这样的事实，和编译的帧相比，解释器代码等价物经常需要控制堆栈上更多的空间。由于附加的空间不能在控制堆栈的中间被分配，因此，一方法不能被去编译或去最佳化直至相应该方法的的帧在控制堆栈的顶部。因此，当编译的帧位于控制堆栈的中间时，即编译的帧不是控制堆栈的最上面的帧时，编译帧不能由相应的解释帧所取代。

图 1 是包括编译帧的控制堆栈的图示表示。控制堆栈 104 包括上述的与方法相关的帧 108。例如，帧 108b 是包括与编译方法 116 相关的启动信息的记录。堆栈指针 112 识别堆栈 104 内最上面帧 108 的位置，即帧 108c，它是当前执行的帧。箭头 114 指示堆栈 104 增长方向，因此，如箭头 114 所示，帧 108c 是堆栈 104 内的最上面帧，由此，是控制帧。

编译方法 116，和帧 108b 包括能被用来产生编译方法 116 的解释器代码等价物的信息。编译方法 116 的解释器代码等价物一般能被存取和因此能在任何时间获得。然而，与解释器代码等价物相关的解释器的帧和帧 108b 相比，在堆栈 104 上占据不同的空间量。由于包括编译方法 116 的解释器代码等价物的帧，即解释器帧比帧 108b 在堆栈 104 上占据更多空间，解释器帧不能被插入堆栈 104，直到上述的帧

108b 被弹到堆栈的顶部。换言之，在方法 116 和帧 108b 被去最佳化之前，帧 108b 必须是堆栈 104 内的控制帧。

如上所述，当帧 108b 在堆栈 104 中间时，帧 108b 不被去最佳化。依此，当它确定，编译的方法 116 要被删除时，编译方法 116 被删除，和因此，帧 108b 的去最佳化必须被延迟。因此，编译的方法 116 可以标记删除，帧 108b 可以标记去最佳化，这样，当帧 108b 是堆栈 104 的最顶上的帧时，编译的方法 116 被删除，而帧 108b 去最佳化。

参看图 2，现描述图 1 的堆栈 104 内的帧 108 去最佳化。如图 1 所示，当帧 108c 从堆栈 104 中弹出，那末帧 108b 不再在堆栈 104 的中间位置。换言之，帧 108b 开始成为堆栈 104 的最顶上的帧。帧 108b 然后由解释器帧 108b′ 取代，该帧包括了编译方法 116 的解释器的启动。堆栈指针 112 可以指向解释器帧 108b′。一旦解释器帧 108b′ 被推到堆栈 104，正如删除的编译方法 116′ 所指示的，编译的方法 116 可被删除。先前由编译方法 116 占据的空间可以被重新分配以供其它使用。

恰在帧 108b 成为堆栈 104 最顶上的帧之前，帧 108b 也可以去最佳化。换言之，恰当帧 108c 将要返回和从堆栈 104 中弹出时，帧 108b 可以去最佳化。恰在帧 108b 成为最顶上的帧之前的帧 108b 去最佳化被熟知称为“懒”去最佳化(lazy deoptimization)。在懒去最佳化中，特定编译帧的去最佳化被延迟，直至在堆栈中仅比特定编译帧较高的那一帧将要返回到特定的编译帧。因此，在懒去最佳化中，恰当该帧将要重新开始执行时，发生了帧的去最佳化。

一般而言，当不足够的资源供其它目的使用时，至少分配给编译方法使用的一些计算机资源，例如存储器空间被预定再分配。当分配给编译方法的资源要被再分配或重新分配时，编译方法必须被删除，或被去最佳化，以腾出资源。因此，推迟编译方法的删除经常是不希望的，因为能立即被其它目的使用的资源将不能被获得，直至编译的方法被删除。即，由于在删除之前的延迟长度经常是相对比较长的，这取决于例如要被去最佳化的帧在堆栈中的位置等诸因素，去编译一个编译方法的过程和去最佳化一个相关的编译的帧可能是低效率的。因此，实质取消与去编译和去最佳化相关的机制是希望的。即，改进与计算机程序相关的编译方法的动态去编译和编译的帧的动态去最佳

化的效率的方法和装置是希望的。

这里公开了在执行计算机程序期间动态去最佳化控制堆栈的帧的方法和装置。所描述的方法特别适合在安排执行解释和编译字节编码的计算机系统中使用。依照本发明的第一方面，去最佳化编译方法的计算机执行方法包括建立一个数据结构。该数据结构从控制堆栈中分离出来，并被安排存储与要被去最佳化的编译的方法和控制堆栈的帧有关的信息。参考指示器，例如指针被产生与具有该帧的数据结构相关。该编译方法然后可以立即放弃。在一个实施例中，该数据结构是独立于机器，编译方法的去编译包括删除该编译的方法。

依照本发明的另一方面，在控制堆栈上动态去最佳化与编译方法相关并且包括编译启动的第一帧的方法包括去编译所编译的方法。当第一帧位于控制堆栈的第 2 帧之下时，编译方法可以被去编译。去编译所编译的方法包括去编译编译启动以产生编译启动的解释器等价物。第 2 帧最终从控制堆栈中弹出，解释器帧从编译启动的解释器等价物中产生。解释器帧然后压入堆栈以取代第一帧。在一个实施例中，编译启动的解释器等价物存储在与第 1 帧相关的临时数据结构内。在这样的实施例中，在解释器帧被产生之后，该数据结构可以被删除。

依照本发明的另一方面，动态去最佳化与从控制堆栈的大量帧中选取出的第一帧相关的编译方法的方法包括产生供第一帧参考的数据结构。从控制堆栈分离出来的该数据结构包括与第一帧相关的解释器信息和与编译方法相关的方法信息。当与控制堆栈相关的堆栈指针识别第一帧为当前帧时，编译方法被删除，解释器信息被取出。取出解释器信息包括产生解释器帧。解释器帧被压入控制堆栈以取代第一帧。

通过阅读下面的详细说明和研究附图，本发明的这些和其它优点将变得十分明显。

参照下面说明和附图发明将更好地被理解。

图 1 是包括编译帧的控制堆栈的示意表示。

图 2 是在解释帧已经取代编译帧之后的图 1 的控制堆栈 104 的示意表示。

图 3a 是依本发明一实施例的包括编译帧的控制堆栈 104 的示意表示

图 3b 是在编译帧已经被去最佳化和依本发明的实施例已经产生了虚拟帧阵列之后的图 3a 的控制堆栈 304 的示意表示。

图 3c 是在依本发明的实施例的去最佳化帧 308b′ 已经成为最顶上的帧之后的图 3b 的控制堆栈 304 的示意表示。

图 3d 是在依本发明的实施例的解释器帧取代了去最佳化帧之后的图 3c 的控制堆栈 304 的示意表示。

图 4 是过程流程图，它示出了依本发明实施例的与急切去最佳化编译帧的诸步骤。

图 5 是过程流程图，它示出了依本发明实施例的与打开虚拟帧阵列相关的诸步骤。

图 6 是控制堆栈的示意表示，它包括依本发明实施例的解释帧和迁移帧。

图 7 是示出了执行本发明的典型的一般用途的计算机系统。

图 8 是适合执行本发明的的虚拟机器的示意表示。

当在支持解释代码和编译代码两者的计算机系统运行计算机程序的执行时，如上所述，当在计算机系统内的资源要被更有效地分配时，典型希望地是删除编译方法。一般而言，去最佳化编译方法和进而去最佳化控制堆栈上相关的编译帧被延迟，直至相关编译的帧要么是堆栈最顶层帧，要么将要成为堆栈上的最顶层帧。由于去最佳化编译的帧可以包括产生比编译的帧占据堆栈更多空间的解释器帧，当该帧位于堆栈中部的什么位置时，由于空间的限制，该编译的帧并不被去最佳化。编译方法的删除的延迟也延迟了计算机系统内资源的可用性。

允许去最佳化编译方法和相关的编译帧发生，而不管编译帧的堆栈位置，允许实质上没有延迟地放弃编译的方法。为了允许去最佳化编译方法和相应的编译的帧发生而没有延迟直至控制要么返回或要么将要返回到编译的帧，可以产生一结构去临时保持与编译方法相关的解释器信息。这样去最佳化可以认为是“急切”去最佳化，因为它并不需要去最佳化被延迟直至一帧被去 最佳化或将要获得控制。

通过产生数据结构去保持离开堆栈的解释器信息，编译的帧可以由解释器信息产生的帧所取代，和当编译的帧在堆栈中部时，相应的方法能去最佳化。换言之，与编译帧相关的编译方法可以被删除或被去最佳化而不推迟删除直至编译的帧达到堆栈的顶部，由于从解释器

信息产生的解释器帧比最初的去最佳化的帧在堆栈上占据更多的空间，数据结构的使用使去最佳化急切发生而不遇到发布涉及在堆栈中部的解释器帧的大小的问题。即，当去最佳化帧最终达到堆栈的顶部时，数据结构临时保持着倾向被取出以产生解释器帧的信息。

图 3a 是控制堆栈的示意表示，它包括了依本发明实施例的与编译方法相关的帧。堆栈 304 包括大量最佳化的或编译的帧 308。堆栈指针 312 识别在堆栈内的当前帧，即帧“f4” 308d。当前帧 308d 是在包括堆栈 308 的整个计算机系统内具有控制的帧。即，当前帧 308d 是在堆栈 304 内最顶部的帧。在描述的实施例中，帧“f2” 308b（在这里和在下面称 308b），它与编译方法 316 相关。特别是，帧 308b 包括了包括在编译方法 316 内与编译启动相关的状态信息。

一般而言，被编译以给出最佳化水平的编译方法 316 可以被去最佳化以使得编译的方法 316 是在最佳化的较低水平。在一个实施例中，当编译方法 316 要被删除时，在编译方法 316 内的编译启动可以由字节代码等价物即解释器启动所代替。即，帧 308b，它被最佳化，但可由解释的帧或诸帧所代替。

由于各种不同的理由可以使编译方法 316 去最佳化。不同的理由包括但并不限制如下，由于程序代码变化使编译方法 316 无效，和附加的计算机存储空间的需要。通过去最佳化编译方法 316 或更具体而言帧 308b，当编译方法 316 无效时可以使帧 308b 有效。进而，当编译的代码一般比解释的代码占据更多的计算机存储器空间，去最佳化编译方法 316 典型释放计算机存储器空间。

与编译方法 316 的解释表示相关的信息可以包括在编译方法 316 和帧 308b 两者之内。一般而言，编译方法 316 包括静态信息而帧 308b 包括动态信息，这是本领域技术人员所熟知的。正如参照图 4 进行的如下描述，该信息被使用以产生数据结构，该结构实质上被安排保持解释器的启动，产生保持解释器启动的数据结构允许帧 308b 被去最佳化。图 3b 是在保持解释器启动的数据结构依本发明的实施例被产生之后的堆栈 304 的示意表示。被称为虚拟帧(vtrame)阵列的数据结构 324 保持着表示图 3a 编译方法 316 的解释器启动 328。

如图所示，编译方法 316 是由 3 个解释器启动 328 所表示，表示编译方法 316 的解释器启动 328 的数目一般变化较大，这取决于许多

不同因素．这样因素可以包括但并不限制于编译方法 316 的大小即与编译方法 316 相关的编译启动的数目和由计算机编译器执行的最佳化例如直接插入。

从去最佳化帧 308b′ 的指针识别虚拟帧阵列 324。换言之，指针 332 被安排指示，虚拟帧阵列 324 与去最佳化帧 308b′ 相关。一般而言，虚拟帧阵列 324 仅与单个去最佳化帧，例如去最佳化的帧 308b′ 相关。然而，在一些实施例中，大量去最佳化的帧，相邻的和不相邻的两者可以共享单个虚拟帧阵列。

虚拟帧阵列 324 的产生可以发生在任何时间。特别是，当去最佳化的帧 308b′ 是在堆栈 304 的中间时，虚拟帧阵列可以被产生。一旦虚拟帧阵列 324 被产生，编译方法 316 被删除，正如被删除的编译方法 316′ 所指示。编译方法 316 的删除允许系统资源，即与编译方法 316 相关的存储器空间被重新分配。特别是，在去最佳化帧 308b′ 到达堆栈 304 的顶部之前，即在去最佳化帧 308b′ 成为当前帧的任何时间可以删除编译方法 316。在任何时候只要需要，与编译方法 316 相关的系统资源因此可以重新分配，图 3a 的帧 308b 可以成为去最佳化帧 308b。

如图 3c 所示，当堆栈指针 312 指示，去最佳化帧 308b′ 是堆栈 304 上的当前帧时，那末虚拟帧阵列 324 被取出以产生堆栈 304 上相应的解释器帧，这如图 3d 所示。解释器启动 328a 被取出和实质上转化为放置在堆栈 304 上的解释器帧 328a′ 。类似的还有与解释器帧 328b′ 相关的解释器启动 328b，和相应解释器帧 328c′ 相关的解释器启动 328c。一旦编译器帧 328′ 被压入堆栈 304，就设置堆栈指针 312 以识别解释器帧 328c′ 为当前帧。在一个实施例中，系统可以附加地产生迁移帧 330，它被用来适当地连接所有解释器帧 328。与取出虚拟帧阵列 324 和因此在堆栈上产生解释器帧 328′ 的步骤将结合图 5 在下面描述。

如前所述，通过使用虚拟帧阵列急切去最佳化一个最佳化的帧允许删除与一个最佳化帧相关的编译方法实际发生在执行一程序的任何时间。通过不延迟去最佳化一帧直至该帧取得控制，计算机资源通过去最佳化能更有效地腾出。例如，在计算机系统内需要空间时，编译的方法能预定地被删除。允许相关的帧被去最佳化，甚至当该帧是在

堆栈中间也能使编译的方法实质上立即被删除。这样，当需要空间时能快速腾空在计算机系统内的空间。

参看图 4，与急切去最佳化一已最佳化的帧的诸步骤将结合本发明的实施例加以描述。一般而言，由于多种不同的理由，一已最佳化到一个给定水平的帧可以去最佳化到一个更低的最佳化水平。尽管理由可以变化很大，一些理由可以包括但并不局限于需要使无效的帧变得有效，需要节约计算机存储器的空间，和需要获得与机器无关的控制堆栈。

去最佳化一已最佳化的帧的过程由步 402 开始，其中与最佳化帧相关的程序计数器，或要被去最佳化的帧被获得。获得程序计数器以用来存取呈现在与最佳化帧相关的编译方法内的存取信息。存取信息一般包括但并不局限于与编译方法相关的源水平变量的值或位置，这是所属领域技术人员所理解的。

在步 404 中使用的存取信息从最佳化帧中取出解释器状态以用来获得解释器的启动。应该理解的是，可以有与最佳化帧相关的多个解释器启动。一旦获得与最佳化帧相关的解释器启动，解释器启动能被迁移，即放入步 406 中的虚拟帧阵列。作为结果，实质上在虚拟帧阵列内获得的所有信息，包括静态和动态的两者，都能被用来最终把解释器帧串压入堆栈，这将参照下面的图 5 加以讨论。

在步 408，在解释器启动被存储在虚拟帧阵列以后，指向虚拟帧阵列的指针被存储在对应的，即最佳化帧。一般而言，虚拟帧阵列指针可以存储在最佳化帧内的任何固定位置。例如，虚拟帧阵列指针可以存储在最佳化帧的第一局部变量内，因此能使虚拟帧阵列指针容易存取。从步 408，过程流移向步 410，那里可被去最佳化的帧调用的该帧的返回地址被设置为虚拟帧阵列的取出程序的地址。该取出程序它一般包括迁移程序，这将参照图 6 在下面更详细地加以描述。

在正被调用的帧即“被调用者”的返回地址变化以后，在步 412 可以删除从中获得存取信息的编译方法。一般而言，最初被设置为编译方法地址的被调用者的返回地址被变化以防止被调用者返回不存在的已被删除的编译方法。正如所属领域的技术人员所理解的。在一个实施例中，在该编译方法被允许删除之前，可以进行确定以弄清是否其它的帧引用该编译方法。如果其它帧引用该编译方法，在该编译方

法被删除之前，那末其它的帧典型地也要被去最佳化。应该理解的是，虽然在步 410 中改变返回地址之后，示出的一帧去最佳化过程已经结束，在去最佳化过程真实完成之前，相关的编译方法被删除。如上所述，编译方法的删除一般并不发生，直至在步 402 至 410 当中，与编译方法相关的所有的帧已经被处理。

如上结合图 3c 所述，一去最佳化的帧并不是在物理上由解释器帧所取代，直至去最佳化的帧成为当前帧。换言之，相应去最佳化帧的解释器帧并不被压入堆栈取代去最佳化帧，直到堆栈指针引用去最佳化帧。由解释器帧取代去最佳化帧一般涉及打开由解释器帧引用的虚拟帧阵列以获得解释器帧。图 5 是过程流程图，它示出了依本发明的一个实施例与取出虚拟帧阵列和放置从虚拟帧阵列取出的解释器的帧进入堆栈的步骤。在堆栈中用相应解释器帧取代去最佳化帧的过程始于步 502，其中获得与去最佳化帧相关的虚拟帧阵列，和计算相应解释器帧的大小。即，使用去最佳化帧的虚拟帧阵列，计算取代去最佳化帧的解释器帧的大小。计算解释器帧的大小使堆栈的空间能被分配以容纳解释器帧。每个解释器帧一般具有的尺寸取决于它的相应解释器启动的尺寸。

为了分配堆栈内解释器的空间，在步 504 对堆栈的堆栈指针进行修改。修改堆栈指针可以包括设置堆栈指针指向堆栈上从最佳化帧进一步向上的位置。典型地，设置堆栈指针以识别堆栈的位置，该位置将实际最终指向在堆栈内的与去最佳化帧相关的最高解释器帧的位置。

在步 504，堆栈指针改变之后，迁移程序的一堆栈帧被产生，在步 506，迁移程序被调用。由所属领域技术人员所理解的，迁移程序被安排在堆栈上放置解释器启动。在堆栈上放置解释器启动典型地包括了变换虚拟帧阵列信息为诸帧。在一个实施例中，变换解释器启动为解释器帧包括了产生与每一个解释器启动相关的返回地址字段和帧指针字段。调用迁移程序可以改变堆栈指针例如以识别与迁移程序相关的该堆栈帧。

在迁移程序内，解释器启动被迁移以产生堆栈上的解释器帧。特别是，每一个解释器启动“i”被迁移以产生堆栈上的一解释器帧，要产生的解释器帧的数目“N”依赖包括在虚拟帧阵列内的解释器启动的

数目。

当迁移程序被调用时，过程流从步 506 移动到步 508 和 510，这表示一个循环，其中，所有在虚拟帧阵列的解释器启动被各自迁移以产生堆栈上的相应的解释器帧。换言之，相应解释器启动的解释器帧被压入堆栈。该循环继续直至所有解释器启动已经被迁移，在此点完成取出虚拟帧阵列和在堆栈上放置解释器帧的过程。

一般而言，在堆栈上产生解释器帧涉及设置与解释器帧相关的各种地址和指针。被设置或更新的地址和指针可以广泛地变化。例如，与一解释器帧相关的的帧指针可以被重置为引用另一个解释器帧。一解释器帧的返回地址也可以被设置为适当地引用该解释器帧的调用者。图 6 是依本发明的一实施例的包括解释器帧的控制堆栈的示意表示。换言之，图 6 是在急切去最佳化过程已经被完成和解释器帧已经被压入堆栈后的控制堆栈的示意表示。控制堆栈 602 包括编译帧 606，它调用在去最佳化以后由翻译的帧 610 取代的(未示出)的另一编译的帧。

一帧，即编译的帧 606 一般包括含有局部变量和参数的程序数据的“数据部分”608。编译帧 606 同样也包括帧指针 614 和返回地址 618。帧指针 614 识别与编译帧 606 相关的连接。换言之，帧指针 614 识别由编译帧 606 参照的一帧即解释器帧 610a。返回地址 618 指定编译帧 606 要返回的堆栈 602 上的该帧的地址位置。返回地址 618 指定调用编译帧 606 的帧的地址，即返回地址 618 指定编译帧 606 的调用者的当前程序计数器。正如所属技术领域技术人员所理解的，帧指针 614 和返回地址 618 表示了至少部分依赖相关解释器和相关编译器配置的控制信息。在堆栈 602 内的诸帧也包括了各种其它指针。

当解释器帧 610 被压入堆栈 602 时，可以设置解释器帧 610 的帧指针 622 和返回地址 626。即，这样可以建立与解释器帧 610 指针相关的连接。如前所述，可以建立帧指针 622 以识别由解释器帧 610 调用的帧。例如，解释器帧 610a 的帧指针 622a 可被设置以识别解释器帧 610b.与帧指针 622 和返回地址 626 相关的信息一般依赖于堆栈 602 上每一解释器帧的特定位置，和在解释器功能的基础上方便加以计算。这样，此信息典型地不包括在相应的虚拟帧阵列内。包含在虚拟帧阵列的信息被取出到解释器帧 610 的数据部分 624。

设置最顶层解释器帧 610c 作为迁移帧 630 的调用者。如此所述，包括帧指针 634 和返回地址 638 的迁移帧 630 与被安排在堆栈 602 上设置解释器帧 610 的迁移程序有关。换言之，实质上安排迁移帧 630 以分配堆栈 602 上的帧 610，这时当迁移程序返回时，控制将在解释器帧 610 内。与迁移帧 630 相关的迁移程序也使用该领域的技术人员所熟知的方法设置帧指针 622 和返回地址 626 以引用它们的适当的被调用者和调用者。返回地址 638 设置给迁移帧 630 的调用者的计数器，而帧计数器 634 被设置以识别最顶层解释器帧 610c。

如图所示，迁移帧 630 由堆栈指针 612 识别和因此是在控制堆栈 602 内的控制帧。帧指针“fp”613 识别由堆栈指针 612 即在迁移帧 630 内帧指针 634 识别的该帧内的控制信息的位置。

如前所述，虚拟帧阵列实质上是不依赖机器，而解释器帧 610 不必依赖机器。因此，与迁移帧 630 相关的迁移程序实质上被用来将依赖机器的信息加到从虚拟帧阵列获得的信息。使用迁移程序可以执行去最佳化而不需要第 2 控制堆栈或去最佳化过程的线程。正如所属领域技术人员所理解的，迁移程序可以在不同的堆栈即第 2 控制堆栈上执行。

本发明可以在适当的计算机系统上执行。特别是，如上所述的去最佳化可以在任何适当的虚拟机上执行，例如参照下面图 8 描述的虚拟机。图 7 示出了执行本发明的典型的，一般用途的计算机系统。计算机系统 730 包括任意数量的处理器 732(也就是称为中央处理单元，或 CPUs)，它们连接到包括主存储装置 734(典型为只读存储器，或 ROM)和主存储装置 736(典型为随机存储器，或 RAM)的存储器。

计算机系统 730，或更具体的 CPU732 可以被安排支持虚拟机，这是所属领域的技术人员所理解的。参照下面图 8 描述在计算机系统 730 上的虚拟机的实例。如所属领域所共知的，ROM 单方向传送数据和指令到 CPU732，而 RAM 典型地被用来双方向传送数据和指令。CPU732 一般可以包括多个处理器。主存储装置 734，736 两者也可以包括任何适当的计算机可读介质，典型地是大规模存储装置的二级存储介质 738 也双向耦连到 CPU732 和提供附加的存储容量。大规模存储装置 738 是被用来存储包括计算机代码，数据或类似物的一种计算机可读介质。典型地，大规模存储装置 738 是一般比主存储装置 734，736

慢的存储介质例如硬盘或磁带。大规模存储装置 738 可以采取磁或纸带阅读器或一些其它公知装置的形式。应该理解的是，在大规模存储装置 738 内包含的信息可以在适当的情况下与标准方式合作以做为虚拟存储器而成为 RAM736 的一部分。特定的初级存储装置 734 例如 CD-ROM 可以单方向传送数据到 CPU732.

CPU 732 也耦连到一个或者多个输入/输出装置 740，它们可以包括但并限制于如此装置，例如视频监视器，跟踪球，鼠标，键盘，麦克风，触敏显示器，变换器卡阅读器，磁或纸带阅读器、输入板，输入笔，声音或手写识别器，或其它公知的输入装置，当然还有其它的计算机。最后，CPU 732 使用在 712 一般所示的网络连接可以任意地连接到计算机或通讯网络，即局域网，因特网和内部网。使用这样的网络连接，这就可以考虑，在执行上述方法和步骤中，CPU 732 可以从网络接收信息或可以输入信息到网络。经常可以表示为使用 CPU 732 执行的一系列指令的这样信息可以从网络接收或输出到网络，例如，可以以载波为载体的计算机信号的形式。上述的装置和材料在计算机硬件和软件技术中对所属技术人员而言是熟知的。

如前所述，虚拟机可以在计算机系统 730 上执行. 图 8 是由图 7 的计算机系统支持并适合执行本发明的虚拟机的图示表示。当计算机程序，即由 Palo Alto, California 的 Sun Microsystem 开发的 Java™ 编程语言内写的计算机程序被执行，源编码 810 提供给在编译时环境 805 内的编译器 820。编译器 820 翻译源编码 810 为字节编码 830。一般而言，在由软件开发者产生源编码 810 时，源编码 810 被译成字节编码 830。

字节编码 830 一般可以被再现，卸截或通过网络即图 7 的网络 712 分配，或存储在图 7 做为初级存储 732 的存储装置内。在所述的实施例中，字节编码 830 是平台独立的。即，字节编码 830 可以在运行适当虚拟机 840 的任何计算机系统上执行。在举例中，在 Java™ 环境中，字节编码 830 可以在运行 Java™ 虚拟机的计算机系统上执行。

字节编码 830 提供给包括虚拟机 840 的运行时环境 835. 使用处理器例如图 7 的处理器 CPU 732 可以一般地执行运行时环境 835. 虚拟机 840 包括了编译器 842，解释器 844，和运行时系统 846。一般提供字节编码 830 或到编译器 842 或到解释器 844。

如上所述，当字节编码830提供给编译器842时，在字节编码830中包含的方法被编译为机器指令。另一方面，当字节编码830提供给解释器844时，字节代码830以每次一个字节代码读入解释器。当每一个字节代码被读入解释器844时，解释器844然后执行由每一个字节定义的操作。一般而言，解释器844实质上连续地处理字节代码830和执行与字节代码830相关的操作。

当方法是从操作系统860被调用时，如果它确定，该方法是作为解释方法被调用，运行时系统846可以从解释器844获得方法。另一方面，如果它确定，该方法作为编译方法被调用，运行时系统846启动编译器842。编译器842然后从字节编码830产生机器指令，和执行机器语言指令。一般而言，当虚拟机结束时，机器语言指令被放弃。

虽然已经描述了本发明的几个实施例，但应理解，本发明可以许多其它形式实现而不脱离本发明的精神和范围。例如，用去最佳化给定帧涉及的步骤可被重新排序，移动，或添加。一般而言，本发明方法涉及的步骤可被重新排序，移动或添加而不脱离本发明的精神或范围。

实质上在堆栈内所有的帧可以被去最佳化，即虚拟帧阵列实质上可以为堆栈内所有的帧产生。另外，由于产生一个虚拟帧阵列来表示多个帧。由于虚拟帧阵列独立于机器，去最佳化整个堆栈和由此产生整个堆栈的虚拟帧阵列可使整个堆栈被输出到任何适当的计算机系统。换言之，独立于机器的虚拟帧阵列可以为不同计算机系统之间所共享。独立于机器的虚拟帧阵列可以被存储在介质例如磁片上以供以后重新开始执行。如上所述，替换地，独立于机器的帧也可以传送到其它计算机，例如不同结构的计算机，在虚拟帧阵列被取出进入翻译控制堆栈之后，可以继续执行。

在一个实施例中，从最佳化帧提取解释器状态和放置解释器状态到虚拟帧阵列可涉及使用静态和动态解释。即，使用与最佳化帧相关的程序计数器可以获得范围描述符，和动态信息实质上可以从最佳化帧直接获得。包括参数变量和动态信息并还包括范围描述符的参数变量的实际值的范围描述符可以然后被用来产生虚拟帧。该虚拟帧然后可以串行化成虚拟帧阵列。

尽管已经描述了使用单独的虚拟帧阵列用于要被去最佳化的每一

帧，应该理解的是，要被去最佳化的大量的帧也可以共用一个虚拟帧阵列。例如，当要被去最佳化的帧相邻地位于堆栈上时，与每一个相邻帧相关的解释器启动可以被迁移到一个共享虚拟帧阵列而不脱离本发明的精神和范围。

虽然以去最佳化编译帧为编译帧的解释表示对去最佳化已最佳化的帧进行描述，在一个实施例中，去最佳化已最佳化的帧另外可以应用为去最佳化具有特定最佳化水平的帧为具有较低最佳化水平的编译的帧。换言之，具有最佳化高水平的编译方法可以被“去编译”以反映最佳化的较低水平。减少最初编译方法的最佳化水平典型地释放与编译方法相关的存储空间。当与编译相关的最佳化水平被降低时，虚拟阵列空间可以包括最初编译方法较少最佳化的编译启动。因此，本实例并不被认为是示例和并不受限制，和本发明并不局限于在此给定的细节，可以和全范围的等价物在所属权利要求的范围内加以修改。

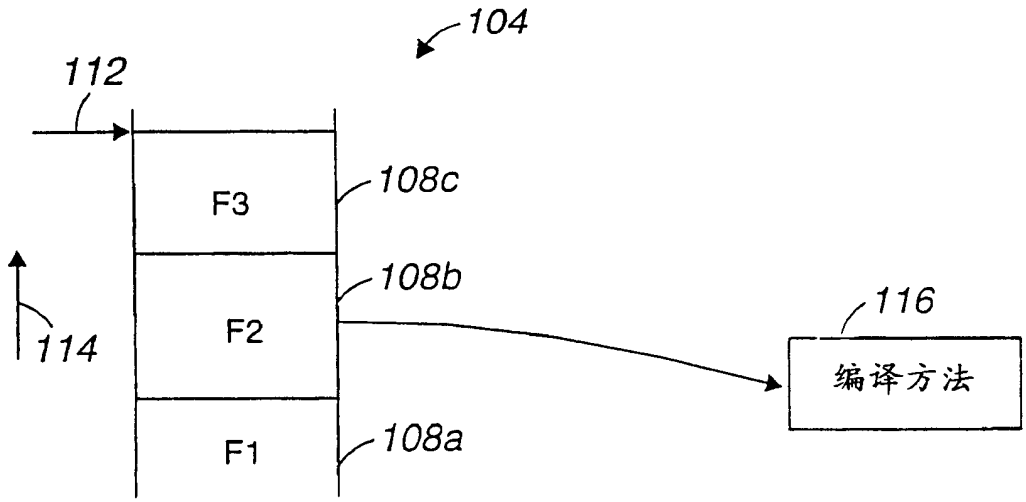

104
112
F3
108c
108b
F2
114
116
编译方法
F1
108a

图 1

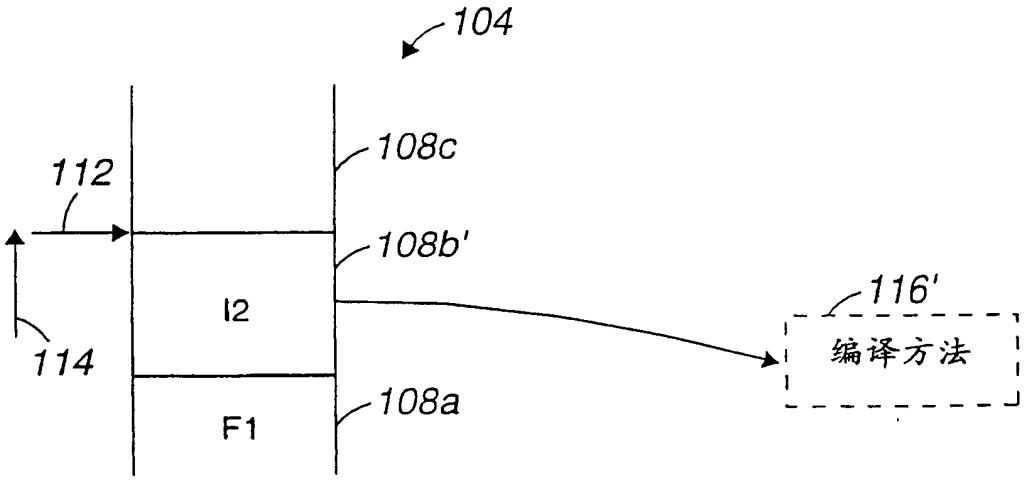

104
112
108c
108b'
I2
116'
编译方法
114
F1
108a

图 2

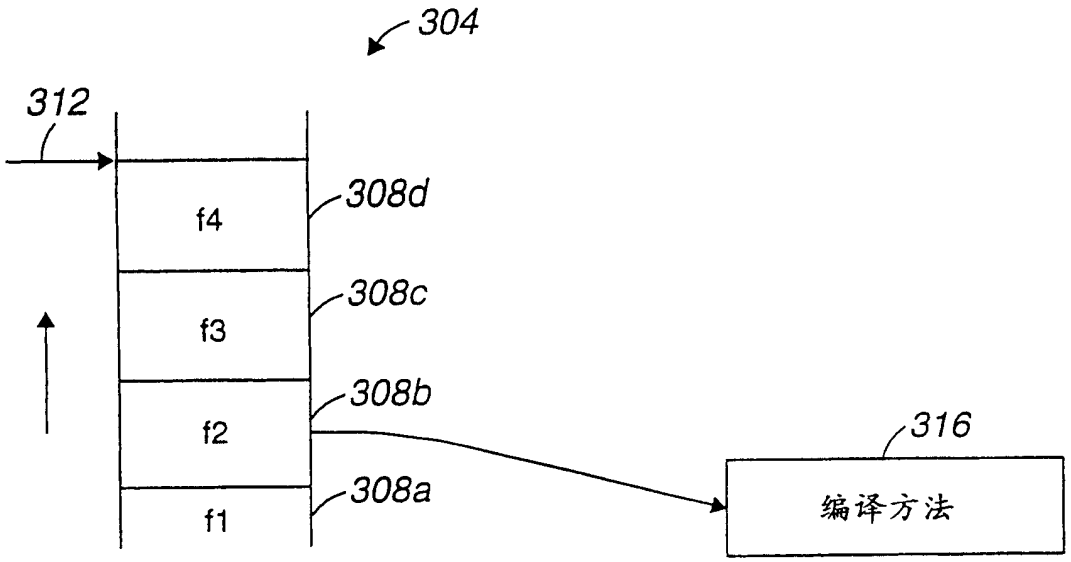

304
312
f4
f3
f2
f1
308d
308c
308b
308a
316
编译方法

图 3a

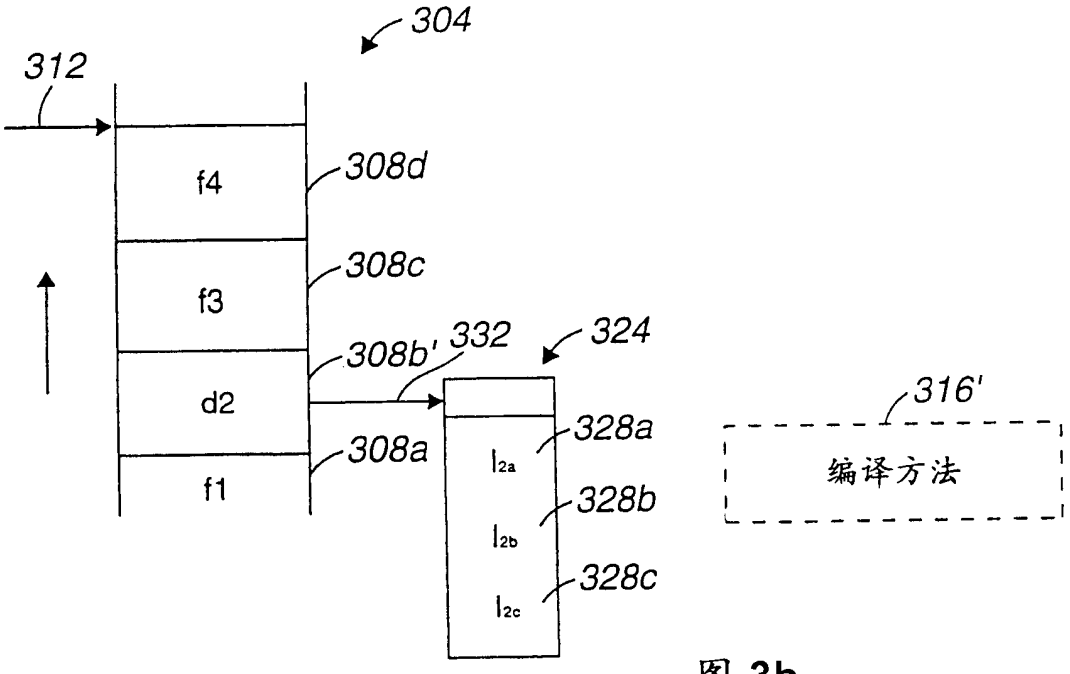

304
312
f4
f3
d2
f1
308d
308c
308b'
308a
332
324
328a
328b
328c
I_{2a}
I_{2b}
I_{2c}
316'
编译方法

图 3b

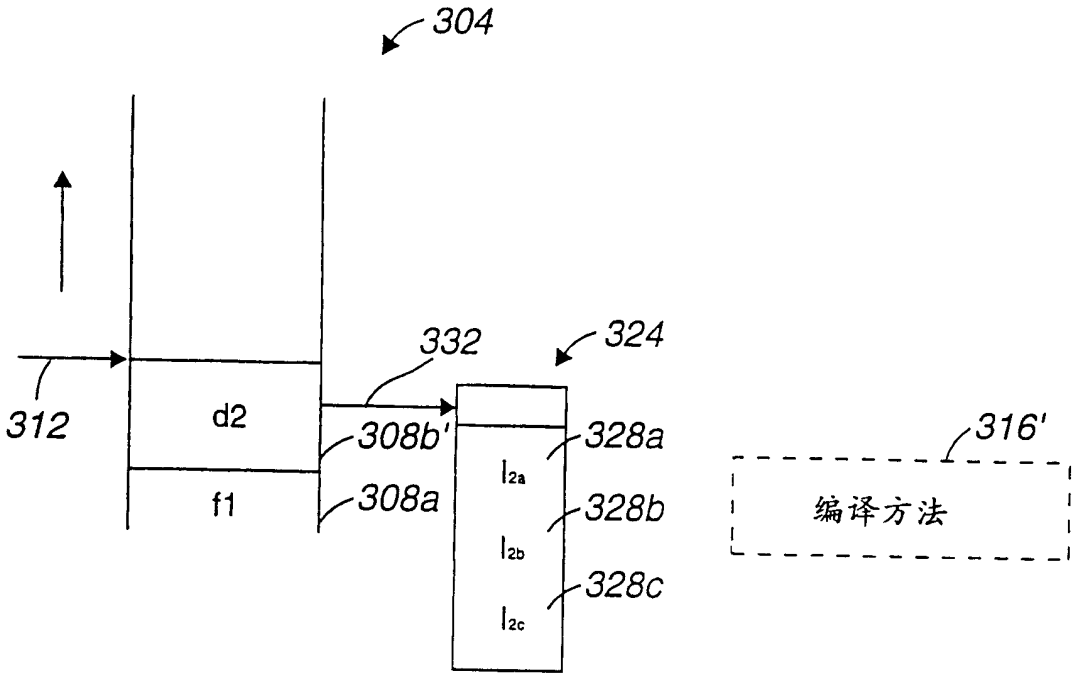

304
312
d2
f1
332
308b'
308a
324
328a
328b
328c
I2a
I2b
I2c
316'
编译方法

图 3c

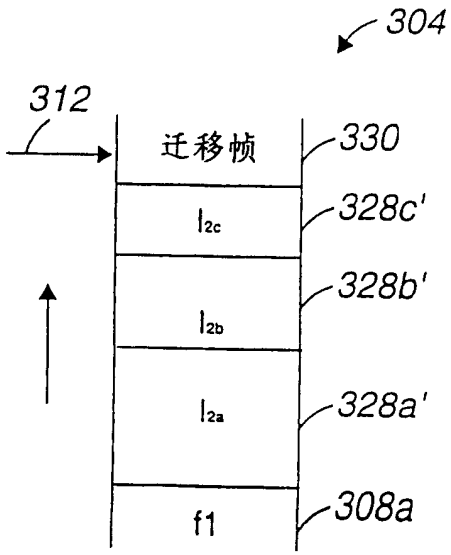

304
312
迁移帧
330
I2c
328c'
I2b
328b'
I2a
328a'
f1
308a

图 3d

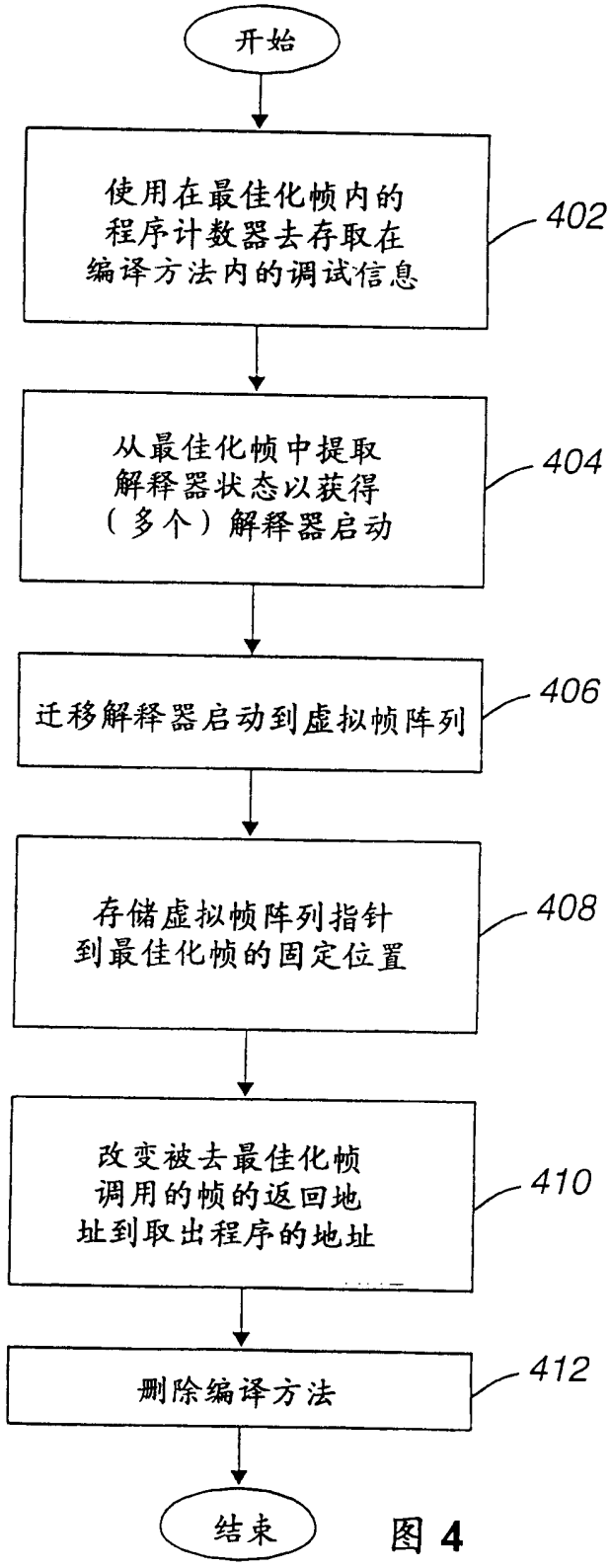

开始
使用在最佳化帧内的程序计数器去存取在编译方法内的调试信息
402
从最佳化帧中提取解释器状态以获得（多个）解释器启动
404
迁移解释器启动到虚拟帧阵列
406
存储虚拟帧阵列指针到最佳化帧的固定位置
408
改变被去最佳化帧调用的帧的返回地址到取出程序的地址
410
删除编译方法
412
结束

图 4

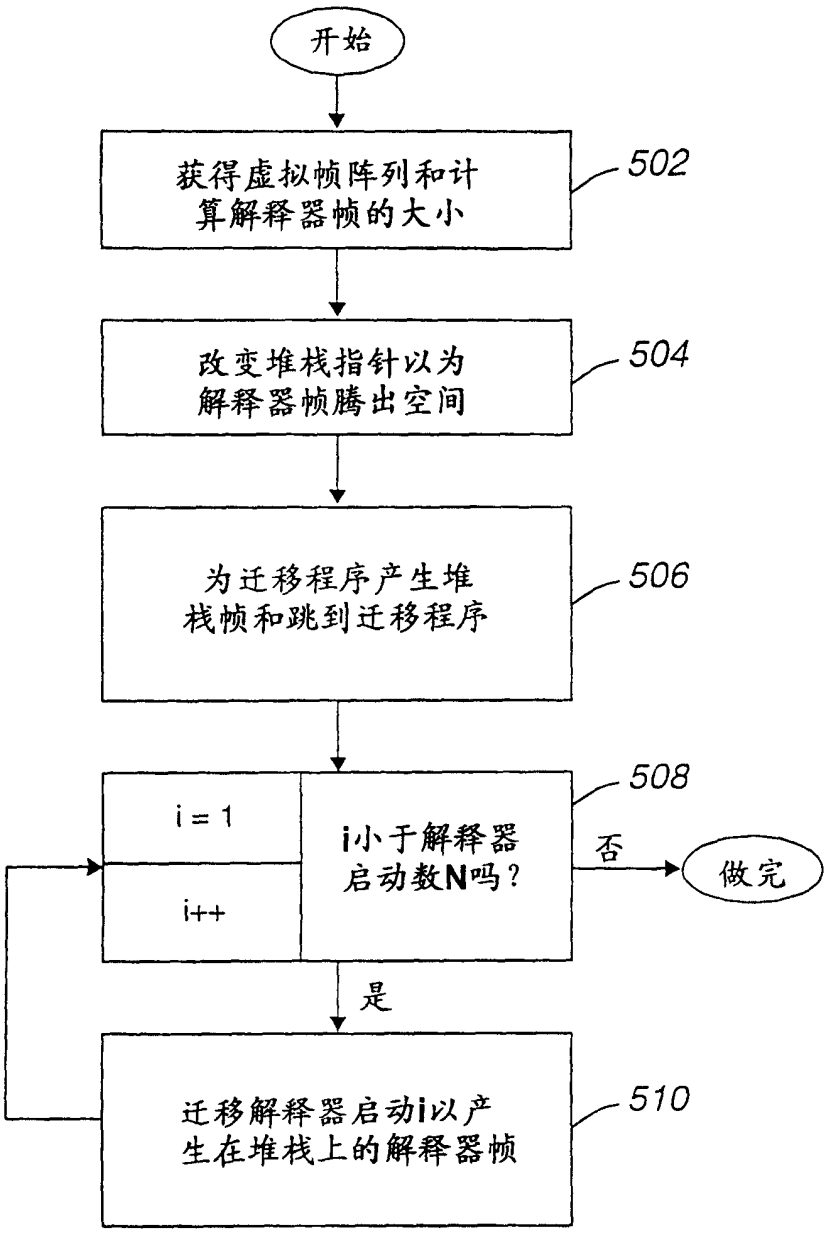

开始
获得虚拟帧阵列和计算解释器帧的大小
502
改变堆栈指针以为解释器帧腾出空间
504
为迁移程序产生堆栈帧和跳到迁移程序
506
i = 1
i++
i小于解释器启动数N吗？
508
否
做完
是
迁移解释器启动i以产生在堆栈上的解释器帧
510

图 5

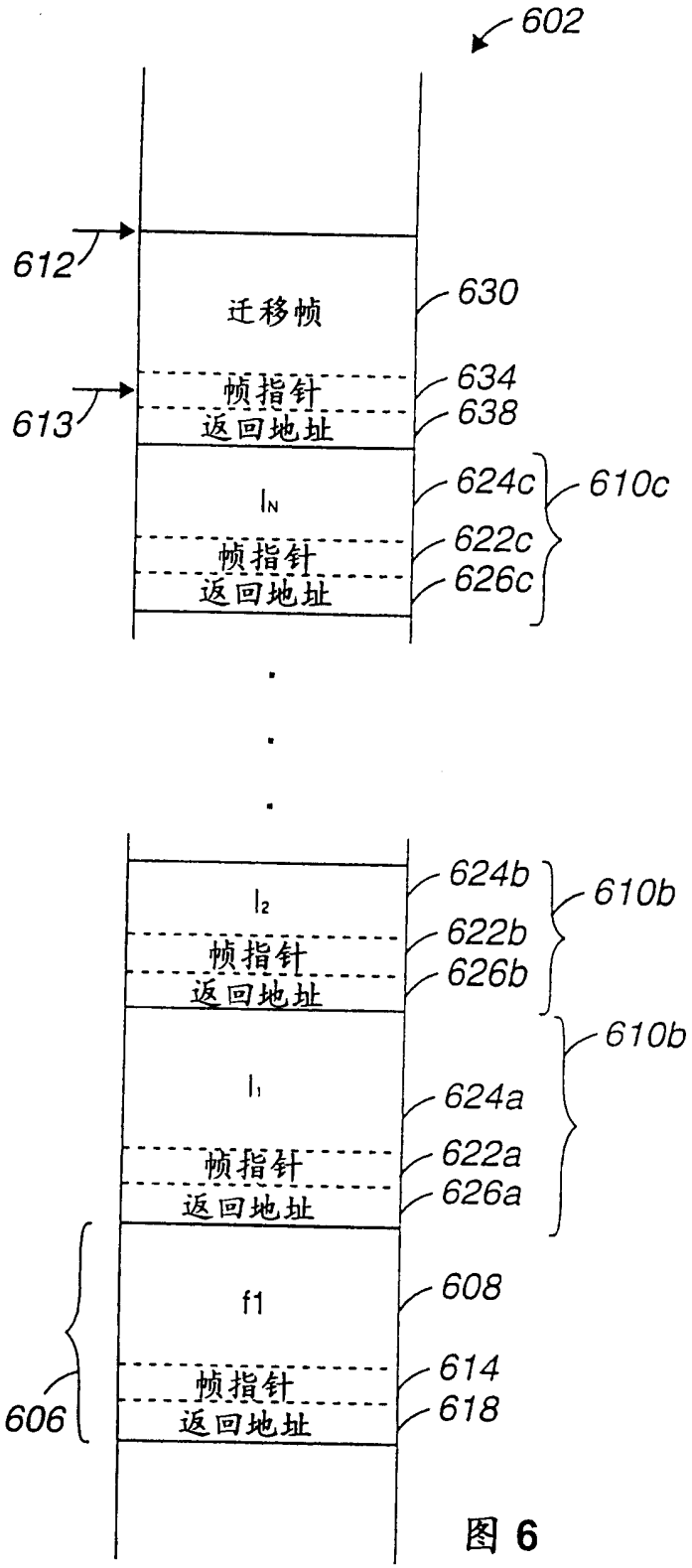

602
612
迁移帧
630
帧指针
634
613
返回地址
638
I_N
624c
610c
帧指针
622c
返回地址
626c
I_2
624b
610b
帧指针
622b
返回地址
626b
I_1
610b
624a
帧指针
622a
返回地址
626a
f1
608
606
帧指针
614
返回地址
618

图 6

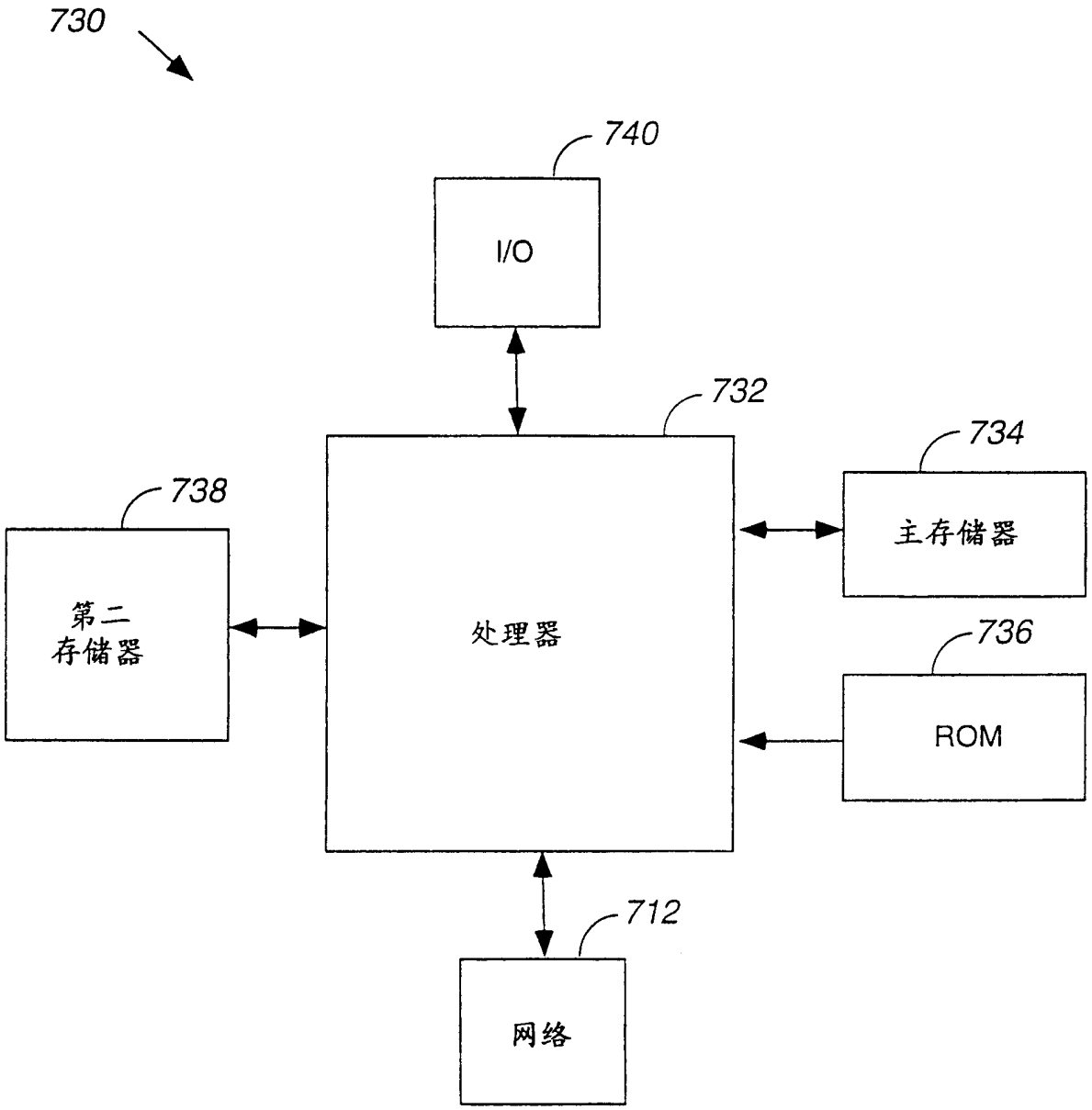


图 7

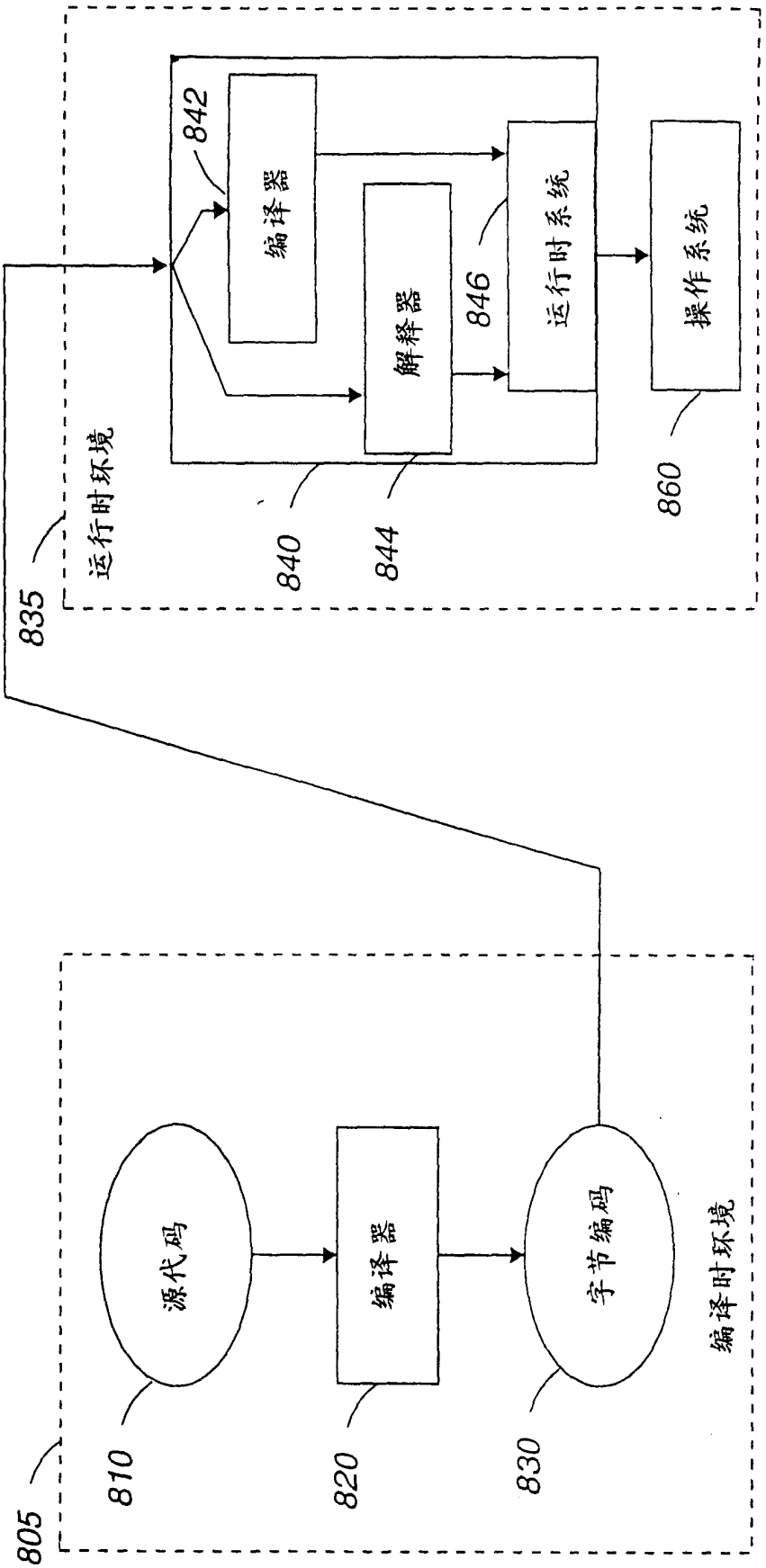

805
编译时环境
源代码
编译器
字节编码
810
820
830
835
运行时环境
编译器
解释器
运行时系统
操作系统
842
846
840
844
860

图8