

[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 200580005179.9

[45] 授权公告日 2009 年 4 月 22 日

[11] 授权公告号 CN 100481007C

[22] 申请日 2005.2.2

[21] 申请号 200580005179.9

[30] 优先权

[32] 2004.2.20 [33] US [31] 10/784,072

[86] 国际申请 PCT/US2005/003195 2005.2.2

[87] 国际公布 WO2005/083565 英 2005.9.9

[85] 进入国家阶段日期 2006.8.17

[73] 专利权人 英特尔公司

地址 美国加利福尼亚州

[72] 发明人 马库斯·梅茨格 凯姆·拉吉
奥恩德·卡派特 罗伯特·威斯纳

[56] 参考文献

EP1085411A2 2001.3.21

US5375242A 1994.12.20

US5848275A 1998.12.8

Automatic Recommendation of Compiler. E. GRANSTON, A, HOLLER. <http://www.cesr.ncsu.edu/fddo4/papers/granston.pdf>. 2001

Compiler optimization – space exploration. Triantafyllis, S., Vachharajani, M., Vachharajani, N., August, D. I. Proceedings of the International Symposium on Code Generation and Optimization. 2003

审查员 吴广平

[74] 专利代理机构 北京嘉和天工知识产权代理事务所

代理人 严慎

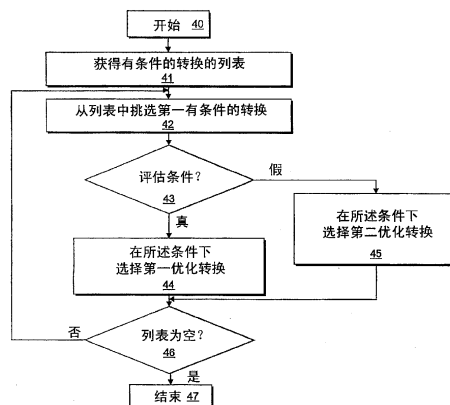
权利要求书 2 页 说明书 11 页 附图 6 页

[54] 发明名称

无需额外的代码分析来进行链接时代码优化的方法和系统

[57] 摘要

描述了用于进行代码优化的系统，所述系统包括编译器内的优化分析器，所述优化分析器为已编译代码产生第一优化转换和第二优化转换以及它们的满足条件。在链接器内放置优化转换模块，以确定当已编译代码和其他已编译代码链接时，应该选择第一和第二优化转换中的哪一个，以及在链接时执行第一和第二优化转换中被选择的那一个。还描述了进行代码优化的方法。



1. 一种计算机实现的编译系统，包括：

计算机系统；

在所述计算机系统中运行的软件编译器，所述软件编译器把源代码编译成已编译代码；

在所述计算机系统中运行的软件链接器，所述软件链接器把所述已编译代码与其他分别编译的代码链接成可执行程序；

所述编译器内的优化分析器，所述优化分析器运行在所述计算机系统中，以在编译时在不知晓与所述其他分别编译的代码有关的信息的情况下，为所述已编译代码产生第一优化转换和第二优化转换以及它们的满足条件；以及

所述链接器内的优化转换模块，所述优化转换模块运行在所述计算机系统中，以在链接时并且在知晓所述其他分别编译的代码的信息的情况下确定当所述已编译代码与所述其他分别编译的代码链接时，应该选择所述第一和第二优化转换中的哪一个，并且在链接时执行所述第一和第二优化转换中被选择的那一个。

2. 如权利要求 1 所述的编译系统，其中所述优化转换模块通过使用只在链接时可获得的信息来检查所述满足条件，以确定应该选择所述第一和第二优化转换中的哪一个。

3. 如权利要求 1 所述的编译系统，其中所述第一和第二优化转换以及它们的满足条件是有条件的重定位操作的形式。

4. 如权利要求 1 所述的编译系统，其中所述第一和第二优化转换选自包括地址基址绑定优化转换、函数克隆优化转换和数据分配优化转换的组。

5. 一种进行代码优化的方法，包括：

在编译时为已编译代码产生第一优化转换和第二优化转换以及它们的满足条件，所述已编译代码在链接时与其他分别编译的代码被链接成可执行程序，其中产生第一优化转换和第二优化转换以及它们的满足条件的操作是在编译时在不知晓与所述其他分别编译的代码有关的信息的情况下进行的；

在链接时并且在知晓所述其他分别编译的代码的信息的情况下，确定当所述已编译代码与所述其他分别编译的代码链接时，应该选择所述第一和第二优化转换中的哪一个；以及

在链接时执行所述第一和第二优化转换中被选择的那一个。

6. 如权利要求 5 所述的方法，其中所述第一和第二优化转换以及它们的满足条件以有条件的重定位的形式被产生。

7. 如权利要求 5 所述的方法，其中确定应该选择所述第一和第二优化转换中的哪一

个的操作还包括使用只在链接时可获得的信息来检查所述满足条件，以确定应该选择所述第一和第二优化转换中的哪一个。

8. 如权利要求 5 所述的方法，其中所述第一和第二优化转换是地址基址绑定优化转换。

9. 如权利要求 5 所述的方法，其中所述第一和第二优化转换是数据分配优化转换。

10. 如权利要求 5 所述的方法，其中所述第一和第二优化转换是函数克隆优化转换。

无需额外的代码分析来进行链接时代码优化的方法和系统

技术领域

本发明的实施方案涉及软件程序的编译和执行。更具体地，本发明的实施方案涉及在链接时（link-time）无需对程序代码的额外的代码分析来进行链接时代码优化的方法和系统。

背景技术

已经开发了面向对象编程语言（OOPL）来改善开发软件程序的生产能力。典型地，OOPL 包括 Java（由 Sun 微系统公司开发）和符合 CLI（公共语言架构）（由 Microsoft 公司开发）的其他编程语言。采用 OOPL，程序开发者可以创建被称为“对象”的小的可重用程序代码段（section）。所述对象一旦被创建，可以被快速简易地组合和重用以创建新的程序。

典型地，对以这样的语言编写的软件程序的编译涉及使处于源代码格式的软件程序的对象中的每一个被编译器单独地编译成处理器可执行的本机（native）或机器代码文件（即已编译对象文件）。随后已编译对象文件被链接器处理，所述链接器组合已编译对象文件以产生完整的可执行程序。随后，所述可执行程序适于在计算机系统上执行。

为了改善可执行程序的性能，在编译期间需要优化程序。优化可执行程序的一种现有解决方案是使编译器中的每一个对它编译的对象文件进行优化转换。已知的由编译器进行的优化操作典型地包括基址绑定（base binding）、函数克隆和部分求值（partial evaluation）。

然而，上面描述的现有解决方案的一个问题是在编译期间编译器不能具有整个程序的知识（即缺乏“全程序视角（whole program view）”），因为程序包括被分开编译的对象文件。这表示在编译时的许多优化转换（optimizing transformation）操作取决于只在程序的链接时可获得的信息。因此，即使编译器能够基于特定的信息块来进行优化转换操作，如果优化操作所需的信息在该时刻不可获得，那么编译器可能不能进行这样的操作。在另一方面，链接器并非足够强大到可以进行任何优化转换操作，所述优化转换操作典型地需要程序代码的大量分析和目标指令集结构的大量知识。在链接期间，链接器不需要代码分析和目标指令集结构的知识。

对上面描述的问题的一个现有的解决方案是采用在链接时优化整个可执行程序链接时优化器。然而，典型地，现有的链接时优化器是重量级的工具，它通常使得链接时优化比编译时优化昂贵得多。此外，为了优化转换，链接时优化器需要执行它自身的代码分

析（即不能够利用编译器进行的代码分析）。

因此，存在着对无需额外的代码分析来进行链接时代码优化的方法和系统的需求。

发明内容

根据本发明的一个方面，提供了一种计算机实现的编译环境系统，包括：计算机系统；在所述计算机系统上运行的软件编译器，所述软件编译器把源代码编译成已编译代码；在所述计算机系统上运行的软件链接器，所述软件链接器把所述已编译代码与其他分别编译的已编译代码链接成可执行程序；所述编译器内的优化分析器，所述优化分析器运行在所述计算机系统上，以在编译时在不知晓与所述其他分别编译的代码有关的信息的情况下，为所述已编译代码产生第一优化转换和第二优化转换以及它们的满足条件；以及所述链接器内的优化转换模块，所述优化转换模块运行在所述计算机系统上，以在链接时并且在知晓所述其他分别编译的代码的信息的情况下确定当所述已编译代码与所述其他已分别编译的代码链接时，所述优化转换模块确定应该选择所述第一和第二优化转换中的哪一个，并且在链接时执行所述第一和第二优化转换中被选择的那一个。

根据本发明的另一个方面，提供了一种进行代码优化的方法，包括：在编译时为已编译代码产生第一优化转换和第二优化转换以及它们的满足条件，所述已编译代码在链接时与其他分别编译的代码被链接成可执行程序，其中产生第一优化转换和第二优化转换以及它们的满足条件的操作是在编译时在不知晓与所述其他分别编译的代码有关的信息的情况下进行的；在链接时并且在知晓所述其他分别编译的代码的信息的情况下，确定当所述已编译代码与所述其他已分别编译的代码链接时，确定应该选择所述第一和第二优化转换中的哪一个；以及在链接时执行所述第一和第二优化转换中被选择的那一个。

附图说明

本发明的实施方案的特征和优点是以实施例的方式来示出的，并且不想把本发明的实施方案的范围限制在被示出的特定实施方案。

图 1 示出包括实现本发明的一个实施方案的优化分析器和优化转换模块的编译环境。

图 2 是一般地示出图 1 的优化分析器所进行的优化分析过程的流程图。

图 3 是一般地示出图 1 的优化转换模块所进行的优化转换过程的流程图。

图 4 是示出图 1 的优化分析器所进行的具体的基址绑定优化分析过程的流程图。

图 5 是示出图 1 的优化分析器所进行的具体的函数克隆优化分析过程的流程图。

图 6 是示出图 1 的优化转换模块所进行的对地址基址绑定和函数克隆的优化转换过程的流程图。

具体实施方式

图 1 根据本发明的一个实施方案，示出包括优化分析器 20 和优化转换模块 22 的编译环境 10，所述优化分析器 20 和优化转换模块 22 对已编译代码文件 13 进行链接时代码优化而无需额外的代码分析。换句话说，优化分析器 20 和优化转换模块 22 允许已编译代码文件 13 接收在已编译代码文件 13 的编译时（compile-time）期间被指定的链接时优化转换。

如下面将更详细描述，并且根据本发明的一个实施方案，优化分析器 20 在编译环境 10 的编译器 12 内。优化转换模块 22 在编译环境 10 的链接器 15 内。在编译时，由于信息的不可获得性，所述信息仅在链接时已编译代码文件 13 与其他已编译代码文件 19 被链接成可执行程序 16 时可获得，编译器 12 在编译时不能进行为已编译代码文件 13 指定的每个代码优化。因此，优化分析器 20 产生优化指示（directive）21，所述优化指示 21 为编译器 12 不能进行的每个优化指定至少第一优化转换和第二优化转换以及它们的满足条件。该信息可以被称为全程序视角数据或全局数据。

随后，优化指示 21 被发送到优化转换模块 22。当链接器 15 将已编译代码文件 13 与其他已编译代码文件 19 链接时，优化转换模块 22 针对在链接时可获得的全程序视角数据检查条件，以确定应该选择第一和第二优化转换中的哪一个。随后优化转换模块 22 在链接时为可执行程序 16 执行第一和第二优化转换中被选择的一个。下面将仍然结合图 1 到 6 更详细地描述优化分析器 20、优化转换模块 22 以及它们的功能。

如从图 1 可以看到的, 编译环境 10 用于将包含多个对象或模块 (包括源代码文件 11) 的软件程序转变成可执行程序 16。这表示源代码文件 11 是要被编译的软件程序的多个对象或模块中的一个。所述对象或模块中的每一个由某个用户以某种已知的高级编程语言编写。所述对象或模块中的每一个 (包括源代码文件 11) 可以被称为程序文件 (或代码文件)。在一个实施方案中, 所采用的编程语言是 C++ 编程语言。在另一个实施方案中, 编程语言是 JAVA (由 Sun 微系统公司开发)。

编译环境 10 的编译器 12 被用来将源代码文件 11 编译成已编译代码文件 13。编译器 12 还可以被用来将软件程序的其他代码文件编译成其他已编译代码文件 19。然而, 编译器 12 一次编译一个代码文件。编译环境 10 的链接器 15 用于将已编译代码文件 13 与其他已编译代码文件 19 链接, 以产生可执行程序 16。为了组合已编译对象文件 13 和 19 以产生可执行程序 16, 链接器 15 进行多个操作 (例如, 段分配、数据分配、符号处理和重定位)。例如, 链接器 15 将符号绑定到地址上并且进行重定位操作。重定位操作本质上是用给定符号的地址修补 (patch) 存储器字 (memory word) 的操作。编译器 12 确定在已编译代码文件 13 的某个地方需要何种类型的重定位操作, 而链接器 15 执行实际的修补操作。

已编译代码文件 13 和 19 可以是本机代码, 当所述本机代码被链接在一起形成可执行程序 16 时, 它可以直接在数据处理系统或计算机系统上被执行。术语本机代码指这样的机器代码, 即对具体体系结构或平台来说是特定的机器代码。可替换地, 已编译代码文件 13 和 19 (并且因此可执行程序 16) 还可以是中间语言代码 (例如, Java 字节代码), 所述中间代码可以随后被解释, 或者接着被即时 (just-in-time, JIT) 编译器在运行时系统 (或虚拟机) 中编译成可以被平台专用的目标计算机系统执行的本机或机器代码。

在被编译器 12 编译之前, 源代码文件 11 可以被储存在计算机系统的存储器 (两者均未示出) 中。例如, 已编译代码文件 13 和 19 可以被储存在将执行可执行程序 16 的计算机系统的存储器 (两者均未示出) 中。在一个实施方案中, 源代码文件 11、已编译代码文件 13 和 19, 以及可执行程序 16 被储存在同一计算机系统的存储器中。在可替换的实施方案中, 源代码文件 11、已编译代码文件 13 和 19, 以及可执行程序 16 被储存在不同的计算机系统中。

编译器 12 是以计算机系统为宿主 (或运行在计算机系统上) 的软件系统。链接器 15 也是以计算机系统为宿主 (或运行在计算机系统上) 的软件系统。例如, 计算机系统可以是个人计算机、个人数字助理、网络计算机、服务器计算机、笔记本式计算机、工作站、大型计算机或超级计算机。可替换地, 计算机系统可以是具有数据处理能力的任何其他的电子系统。

计算机系统包括操作系统 (未示出) 和系统专用硬件 (未示出)。操作系统可以是开放式标准 Linux 操作系统或其他类型的操作系统。计算机系统的系统专用硬件可以是包括

足够执行操作系统的所有必需模块的任何硬件。这包括作为计算机系统的 CPU 的处理器或微处理器。

编译器 12 还对它编译的代码文件进行代码优化。这表示编译器 12 对已编译代码文件 13 进行代码优化。编译器 12 进行代码优化来改善已编译代码文件 13 的性能。例如，代码优化包括地址基址绑定代码转换、函数克隆代码转换和数据分配代码转换。

然而，编译器 12 不能对它编译的代码文件进行所有的指定的代码优化。这是由于这样的事实：源代码文件 11 中的符号引用（reference）和地址常量可以引用被编译成其他已编译代码文件 19 的其他源代码文件中定义的数据和代码。当编译器 12 将源代码文件 11 编译成已编译代码文件 13 时，编译器 12 不能知道其他文件中的信息，因为它们被分开地编译。例如，如果已编译代码文件 13 中的符号引用和地址常量涉及在其他已编译代码文件 19 中定义的代码或数据，那么编译器 12 在编译期间不能获得该信息。因为在编译期间编译器 12 不具有已编译代码文件 13 所属的整个程序（即可执行程序 16）的知识（即没有全程序视角数据），所以编译器 12 不能定义和进行需要全程序视角数据的那些代码优化。

在这种情况下，并且根据本发明的一个实施方案，编译器 12 采用优化分析器 20 来为那些由于全程序视角数据（或全局数据）的不可获得性而不能在编译器 20 中进行的代码优化产生优化指示 21。优化指示 21 以有条件的格式定义那些优化。这表示优化指示 21 指定一些建议的优化转换，其中每一个具有合格条件（qualifying condition）。根据一个实施方案，优化指示 21 为编译器 12 不能进行的每个优化指定至少第一建议的优化转换和第二建议的优化转换，以及它们的满足条件。

例如，第一和第二优化转换以及它们的满足条件可以是以有条件的重定位操作的形式。第一和第二优化转换以及它们的满足条件还可以是以段的有条件的选择的形式。此外，优化分析器 20 可以为编译器 12 不能完成的每个优化产生比第一和第二优化转换多的优化转换。

优化指示 21 随后被发送到优化转换模块 22。当链接器 15 把已编译代码文件 13 和其他已编译代码文件 19 链接起来时，优化转换模块 22 针对在链接时可用的全程序视角数据检查优化指示 21 中的条件，以确定应该选择第一和第二优化转换中的哪一个。优化转换模块 22 随后在用于可执行程序 16 的链接时执行第一和第二优化转换中被选择的那一个。

例如，考虑以下的代码序列或指令集

```
LDR R0, = foo
.Lbar$Ref:
LDR R1, = bar
```

第一条是将符号“foo”的地址加载到寄存器 R0 的加载指令（load instruction）。第三条也是将符号“bar”的地址加载到寄存器 R1 的加载指令。第二条是指定下一条指令的地址的

符号定义。在这种情况下，如果符号“foo”或“bar”是全局符号，那么只在链接时才可以知道符号“foo”和“bar”被放置的位置。因此，编译器 12 不能对地址定义“.Lbar\$Ref:”进行任何优化转换。然而，如果给定符号“foo”和“bar”之间的距离，编译器 12 可以判断如何进行优化。在这种情况下，优化分析器 22 产生以下的优化指示 21：

```
.Lbar$Ref: =(((bar-foo)<2^8)
?(addi r1, r0, bar-foo)
: (.Lbar$Ref)
```

在这里，优化指示 21 将优化转换“.Lbar\$Ref:”指定为有条件的优化转换。这表示如果符号“foo”和“bar”的距离小于 2^8 ，那么原始的“.Lbar\$Ref:”转换被 ADD 操作（参见优化指示 21 中的第二条指令）取代。否则，原始的“.Lbar\$Ref:”被选择并进行。

当优化转换模块 22 接收上面列出的优化指示时，它评估所述条件（即是否符号“foo”和“bar”的距离小于 2^8 ）。如果是，优化转换模块 22 导致链接器 15 进行 ADD 指令（即第二条指令）。否则，优化转换模块 22 导致链接器 15 在“LDR R1, =bar”的“.Lbar\$Ref:”上进行原始优化转换。

在一个实施方案中，优化指示 21 是已编译代码文件 13 的一部分。在另一个实施方案中，优化指示 21 与已编译代码文件 13 是分开的。

例如，根据本发明的一个实施方案，该针对编译器的链接时优化转换可以被应用于地址基址绑定优化转换、函数克隆优化转换和数据分配优化转换。在地址基址绑定优化转换中，因为已编译文件 13 和 19 中的全局变量的最终分布（layout）由链接器 15 在链接时确定，所以编译器只可以为已编译代码文件 13 中的全局变量进行基址绑定（即从一个全局变量的地址计算另一个变量的地址）。在这种情况下，优化分析器 20 基于符号地址之间的相对距离（即用于那些转换的条件）指定不同的基址绑定优化转换。一旦在链接时已知地址，优化转换模块 22 检查条件以确定应该进行哪一种基址绑定优化转换。优化转换模块 22 随后执行所选择的基址绑定优化转换。

在函数克隆优化转换中，虽然函数克隆（即函数的专用版本）比函数本身高效得多，但仅仅适用于具有某些参数值的函数调用。然而，只可以在链接时使得该信息可获得。因此，优化分析器 20 指定条件（例如用于函数的某些参数值的某些调用点（call site））并且命令优化转换模块 22 基于条件为给定的调用点选择最适当的版本（即函数本身还是它的克隆）。

在一个实施方案中，优化分析器 20 和优化转换模块 22 中的每一个使用软件技术来实现。可替换地，优化分析器 20 和优化转换模块 22 中的每一个可以使用其他技术来实现。例如，优化分析器 20 和优化转换模块 22 中的每一个可以被实现为固件。图 2 一般地示出优化分析器 20 进行的优化分析过程，而图 3 一般地示出优化转换模块 22 进行的优化转换过程。

如从图2中可以看到,优化分析过程从已编译代码文件13(图1)的代码分析开始,以确定需要哪些优化以及为了优化需要哪些信息(即图2中的框31)。这里,优化所需的信息可以被称为优化信息。根据本发明的一个实施方案,优化分析器20(图1)进行该代码分析。可替换地,编译器12(图1)可以进行该代码分析。

在32处,确定优化信息对编译器12是可获得还是不可获得。根据本发明的一个实施方案,优化分析器20做出该确定。如果答案是“是”(即优化信息可获得),那么优化过程移动到框33。然而,如果确定优化信息对编译器12不可获得,那么优化分析移动到框34。

在33处,因为已经确定用于指定的优化操作的优化信息可获得,所以基于优化信息进行优化操作。根据本发明的一个实施方案,优化分析器20导致编译器12基于优化信息进行优化操作(或优化转换)。随后过程在框36结束。

在34处,因为优化信息不可获得,所以只指定了可能的优化转换。根据本发明的一个实施方案,优化分析器20指定可能的优化转换。在一个实施方案中,优化分析器指定第一和第二可能的优化转换。在另一个实施方案中,优化分析器20指定比第一和第二可能的优化转换更多的可能优化转换。

在35处,为这些可能的优化转换产生合格条件。根据本发明的一个实施方案,优化分析器20产生合格条件。过程随后在框36结束。

如上面描述的,图3示出优化转换过程。如从图3中可以看到,优化转换过程从获得包括所有的有条件的优化转换候选(candidate)的列表开始(即框41)。根据本发明的一个实施方案,优化转换模块22(图1)获得该列表。

在42处,从列表中挑选第一有条件的优化转换候选。根据本发明的一个实施方案,优化转换模块22挑选所述候选。如上面结合图1和图2描述的,每个优化转换候选指定第一和第二可能的优化转换以及它们的满足条件。

在43处,使用在链接时从链接器15(图1)可获得的信息来求值相关联的条件。根据本发明的一个实施方案,优化转换模块22做出该确定。在一个实施方案中,用于第一和第二优化转换的条件是相互排斥的。在这种情况下,如果第一优化转换的条件满足(即计算指示TURE(真)),那么过程移动到框44。然而,如果第二优化转换的条件满足(即计算指示FALSE(假)),那么过程移动到框45。

在44处,在所述条件下选择第一优化转换。根据本发明的一个实施方案,优化转换模块22做出选择。随后过程移动到框46。

在 45 处，在所述条件下选择第二优化转换。根据本发明的一个实施方案，优化转换模块 22 做出选择。随后过程移动到框 46。

在 46 处确定列表是否为空。根据本发明的一个实施方案，优化转换模块 22（图 1）做出该确定。如果列表为空，过程移动到框 47。如果列表不为空，那么过程返回框 42 以进行进一步处理。

图 4 到 6 以流程图形式示出由优化分析器 20（图 1）进行的优化分析过程以及由优化转换模块 22（图 1）进行的优化转换过程的一些具体的实施例。例如，图 4 示出地址基址绑定优化分析过程，而图 5 示出函数克隆优化分析过程。图 6 示出用于地址基址绑定和函数克隆两者的优化转换过程。

如从图 4 中可以看到，地址基址绑定的优化分析过程从查找两个存储器引用开始（即框 51）。根据本发明的一个实施方案，优化分析器 20（图 1）进行该操作。可替换地，该操作可以由编译器 12（图 1）来进行。

在 52 处，确定两个引用之间的距离。根据本发明的一个实施方案，优化分析器 20 做出该确定。可替换地，该确定可以由编译器 12 来做出。

在 53 处，确定距离是已知的还是未知的（即是否可以确定距离）。根据本发明的一个实施方案，优化分析器 20 做出该确定。如果距离是已知的（即 YES（是）），那么过程移动到框 54。如果距离是未知的（即 NO（否）），那么过程移动到框 55。

在 54 处，因为距离是已知的，可以进行基址绑定。根据本发明的一个实施方案，优化分析器 20 导致编译器 12 进行基址绑定优化。这里，编译器 12 用把距离值加到之前加载的符号地址的“ADD”指令来替换一个存储器引用——“LOAD-SYMBOL- ADDRESS（加载符号地址）”指令。过程随后移动到框 57。

在 55 处，产生描述基址绑定优化转换的重定位（relocation）。根据本发明的一个实施方案，优化分析器 20 产生重定位。在一个实施方案中，优化分析器 20 产生第一和第二重定位。可替换地，优化分析器 20 产生多于两个重定位。

在 56 处，为重定位中的每一个产生合格条件。这使得每个重定位操作成为有条件的重定位操作。根据本发明的一个实施方案，优化分析器 20 产生所述条件。每个条件假设距离在预先确定的值以内（例如，大于、小于或等于）。如上面描述的，因为在链接时将知道存储器引用或符号地址，所以在链接时将知道所述距离。随后过程移动到框 57。

在 57 处，确定是否有更多的未检查的存储器引用。根据本发明的一个实施方案，优

化分析器 20 做出该确定。如果确定有更多的未检查的存储器引用，那么过程移动回框 51。否则，过程在框 58 结束。

如从图 5 可以看到的，用于函数克隆的优化分析过程从查找用来克隆的候选函数开始（即框 61）。根据本发明的一个实施方案，优化分析器 20（图 1）进行该操作。

在 62 处，确定是否克隆所述函数是有益并且可能的。如上面描述的，虽然函数克隆（即函数的专用版本）比函数本身高效得多，但是只适用于具有某个参数值的函数调用。该信息在编译时可能不可获得。因此，可能编译器 12（图 1）在编译时进行函数克隆优化是不可能的。根据本发明的一个实施方案，优化分析器 20 确定是否克隆所述函数是有益并且可能的。如果克隆所述函数既是有益的又是可能的，那么过程移动到框 66。在另一方面，如果确定在编译时克隆所述函数是不可能的，那么过程移动到框 63。

在 63 处，通过产生函数的克隆的和专用的版本来优化函数。根据本发明的一个实施方案，优化分析器 20 做出该优化。根据另一个实施方案，优化分析器 20 通过产生函数的克隆版本来导致编译器 12 优化所述函数。

在 64 处，使用用于具体调用点的特定值来实例化函数的参数。这是为采用函数还是它的克隆版本规定合格条件。根据本发明的一个实施方案，分析器 20 实例化所述参数。

在 65 处，产生优化指示。在本发明的一个实施方案中，分析器 20 产生优化指示，从而如果参数规范（specification）匹配于指定的调用点，命令优化转换模块 22（图 1）重定位到克隆版本，并且如果参数规范不匹配于具体的调用点，命令优化转换模块 22（图 1）重定位到函数本身。换句话说，优化指示命令优化转换模块 22 基于条件为给定调用地点选择适当的版本（即函数本身或它的克隆版本）。过程随后在框 67 结束。

在 66 处，函数被克隆。根据本发明的一个实施方案，优化分析器 20 导致编译器 12 克隆函数（或对函数进行克隆）。过程随后在框 67 结束。

如上所述，图 6 示出用于地址基址绑定和函数克隆的优化转换过程。如从图 6 可以看到的，优化转换过程从获得包含用于重定位操作的所有候选的列表开始（即框 71）。根据本发明的一个实施方案，优化转换模块 22（图 1）进行该操作。这里，每个重定位候选或者是有条件的重定位（因为它包含指定的重定位操作及其合格条件），或者是常规的算术重定位。此外，并且在一个实施方案中，每个有条件的重定位包括具有其合格条件的第一重定位表达式和具有其合格条件的第二重定位表达式。在一个实施方案中，合格条件是互相排斥的条件。

在 72 处，确定在列表中是否剩余任何的重定位候选。根据本发明的一个实施方案，优化转换模块 22 做出该确定。如果在列表中没有剩余，过程在框 73 结束。然而，如果在

列表中剩余至少一个重定位候选，那么过程移动到框 74。

在 74 处，从列表中挑选并移除第一重定位候选。根据本发明的一个实施方案，优化转换模块 22 进行该功能。

在 75 处，确定重定位是有条件的重定位，或者仅仅是常规的算术重定位操作。根据本发明的一个实施方案，优化转换模块 22 做出该确定。如果重定位是有条件的重定位，那么过程移动到框 77。如果重定位仅仅是常规的算术重定位，那么过程移动到框 76。

在 76 处，应用指定的算术重定位。根据本发明的一个实施方案，优化转换模块 22 导致链接器 15（图 1）应用所述算术重定位。过程随后移动到框 72。

在 77 处，检查有条件的重定位的条件来看条件是否满足。根据本发明的一个实施方案，优化转换模块 22 做出

在 78 处，选择和执行有条件的重定位的第一重定位表达式。根据本发明的一个实施方案，优化转换模块 22 做出该选择，并且导致链接器 15 执行所选择的第一重定位表达式。过程随后移动回到框 72。

在 79 处，选择和执行有条件的重定位的第二重定位表达式。根据本发明的一个实施方案，优化转换模块 22 做出该选择，并且导致链接器 15 执行所选择的第二重定位表达式。过程随后移动到框 72。

根据本发明的实施方案，上面描述的针对编译器的链接时优化技术还可以被应用来增强其他编译器优化。例如，在全局数据的最终分布只可以在链接时确定的数据分配优化（即数据局部性（locality））中，可以使用上面根据一个实施方案描述的技术。设计数据分配优化被设计来改善已编译代码的缓存行为。它为每个函数确定哪些数据是被大量使用的数据（或热数据）以及哪些数据是没有被大量使用的数据（或冷数据）。随后在存储器中被大量使用的数据被分配为彼此临近，并且与没有被大量使用的数据分开。然而，因为全局数据的最终分布仅可以在链接时被确定，优化分析器 20（图 1）可以提议对正被编译器 12（图 1）编译的当前代码文件最有益的数据分布。该信息随后作为链接器指示被发送到优化转换模块 22（图 1）。优化转换模块 22 可以随后从所有要被链接在一起的已编译代码文件收集所有被提议的数据分布，以选择对整个程序最有益的数据分布。作为进一步的实施例，上面根据一个实施方案描述的技术可以用于段的有条件的选择。

根据本发明的实施方案，图 2-6 是示出优化分析过程和优化转换过程的流程图。在附图中示出的步骤（procedure）中的一些可以顺序地、并行地或以不同于所描述的顺序进行。应该意识到，并非所有描述的步骤是要求的，并且可以添加额外的步骤，还可以用其他步骤来代替示出的步骤中的一些。

在前面的说明书中，已经参照本发明的具体的示例性实施方案描述了本发明的实施方案。然而，明显可以对本发明做出各种修改和改动，而不背离本发明的实施方案的宽泛的精神和范围。因此，说明书和附图被认为是描述性的，而不是限制性的。

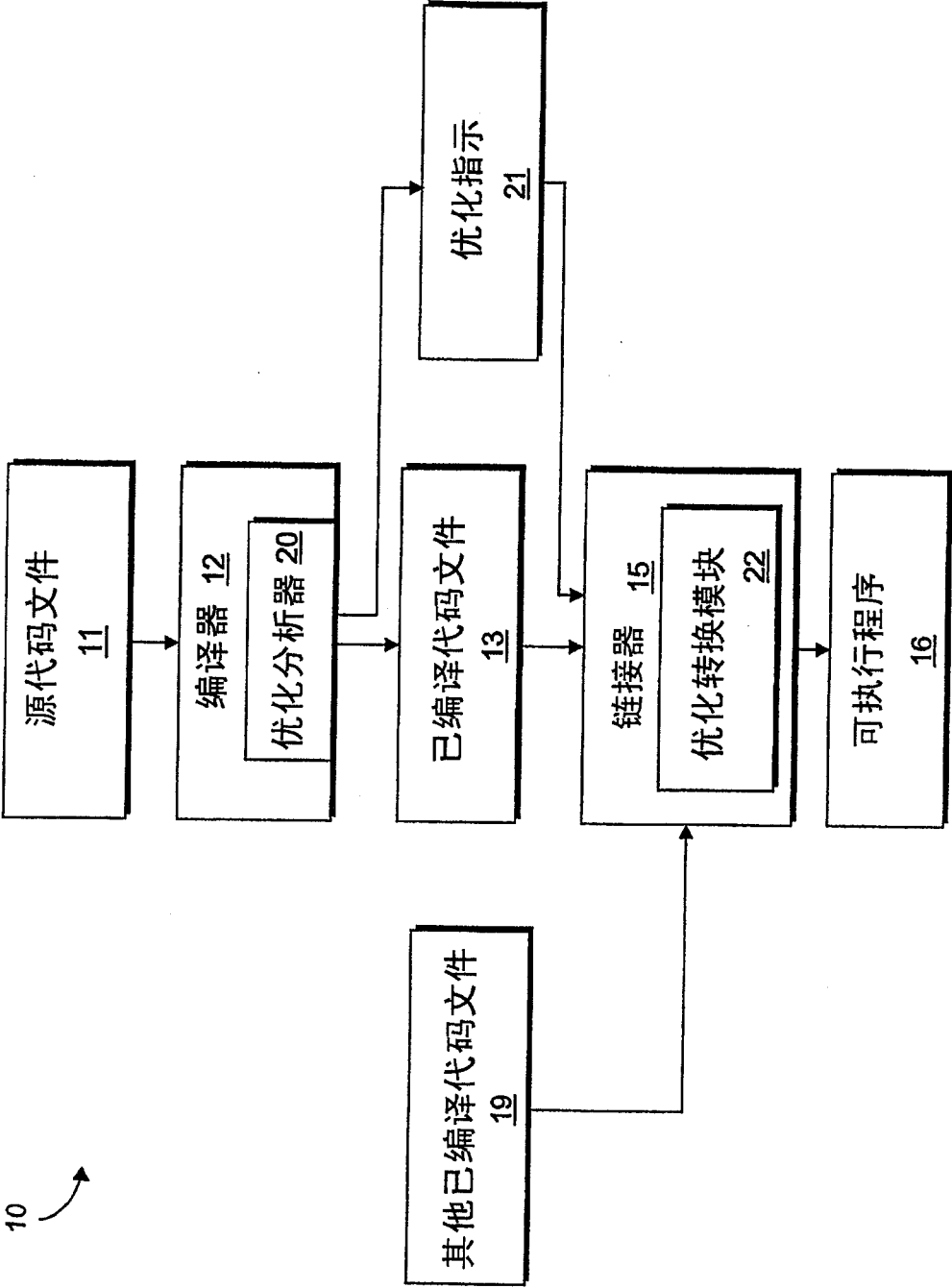


图 1

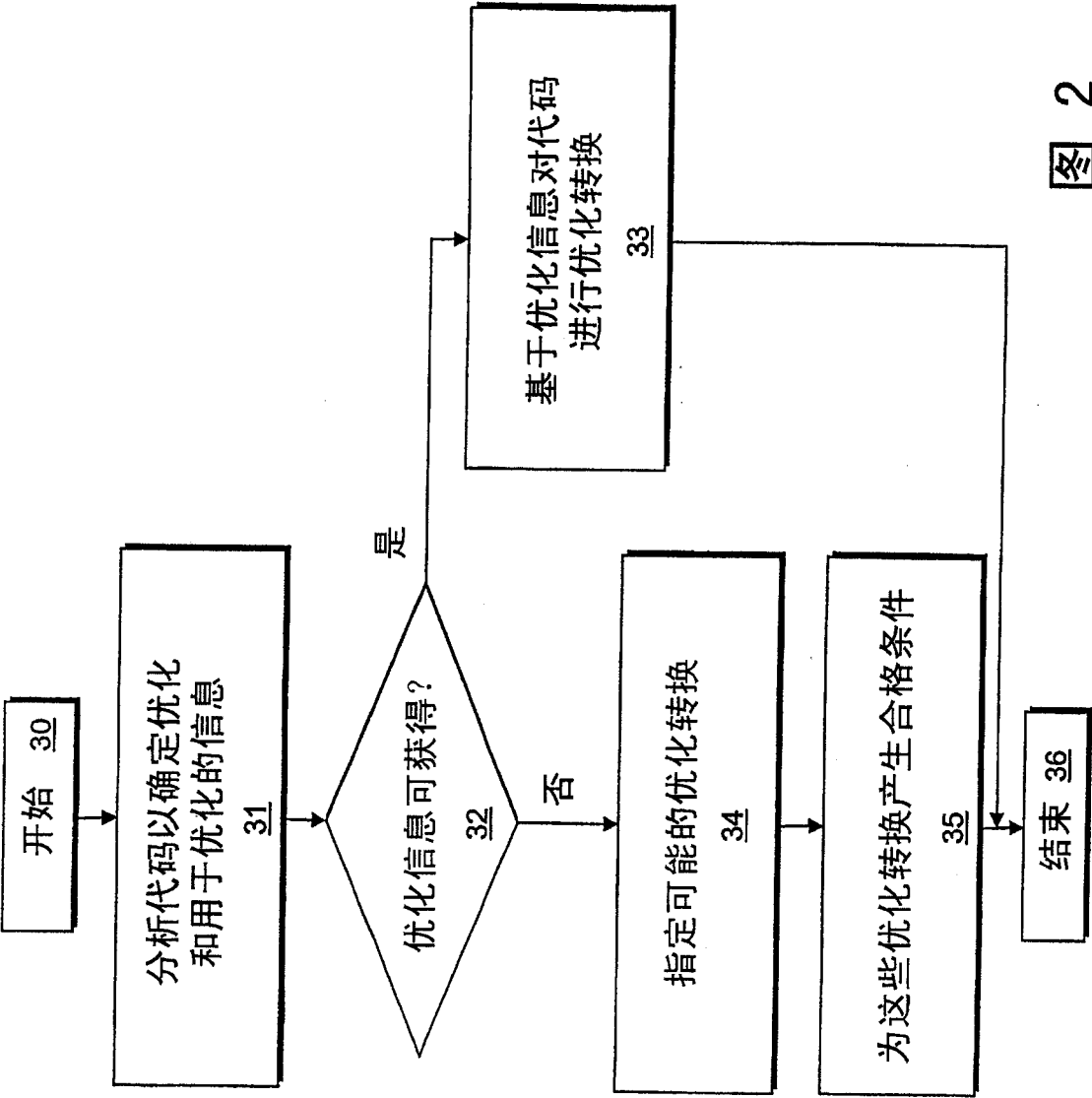


图 2

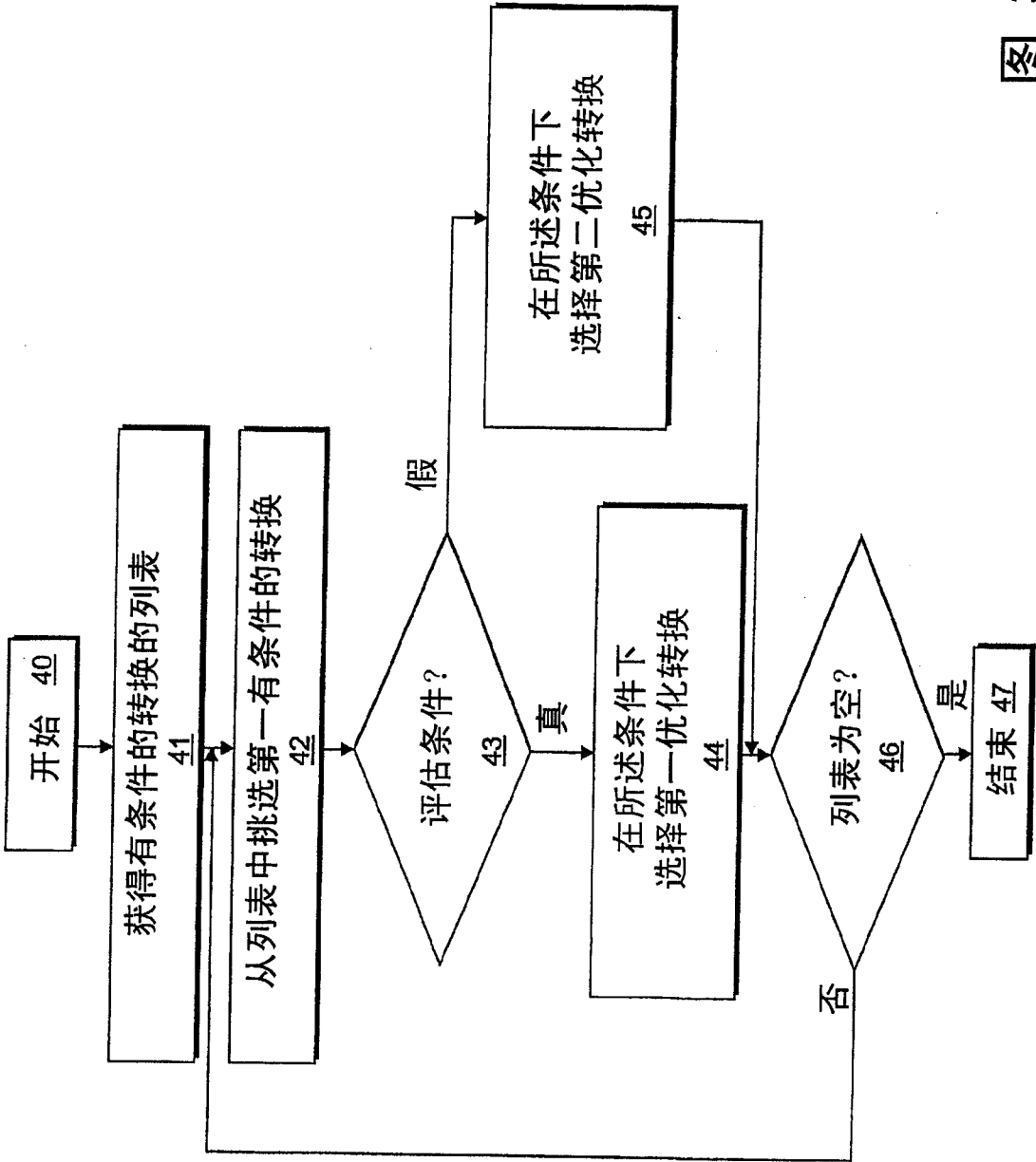


图 3

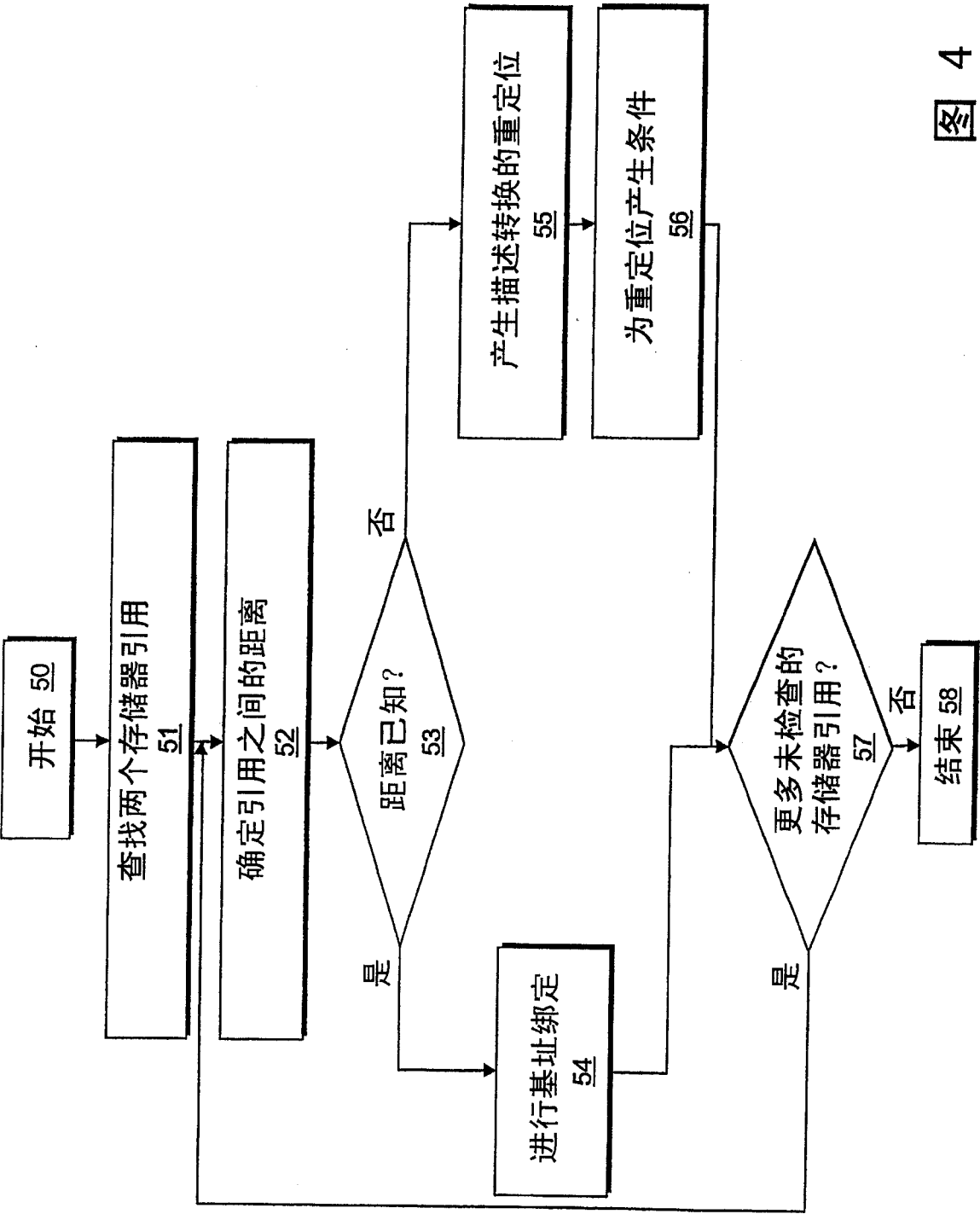


图 4

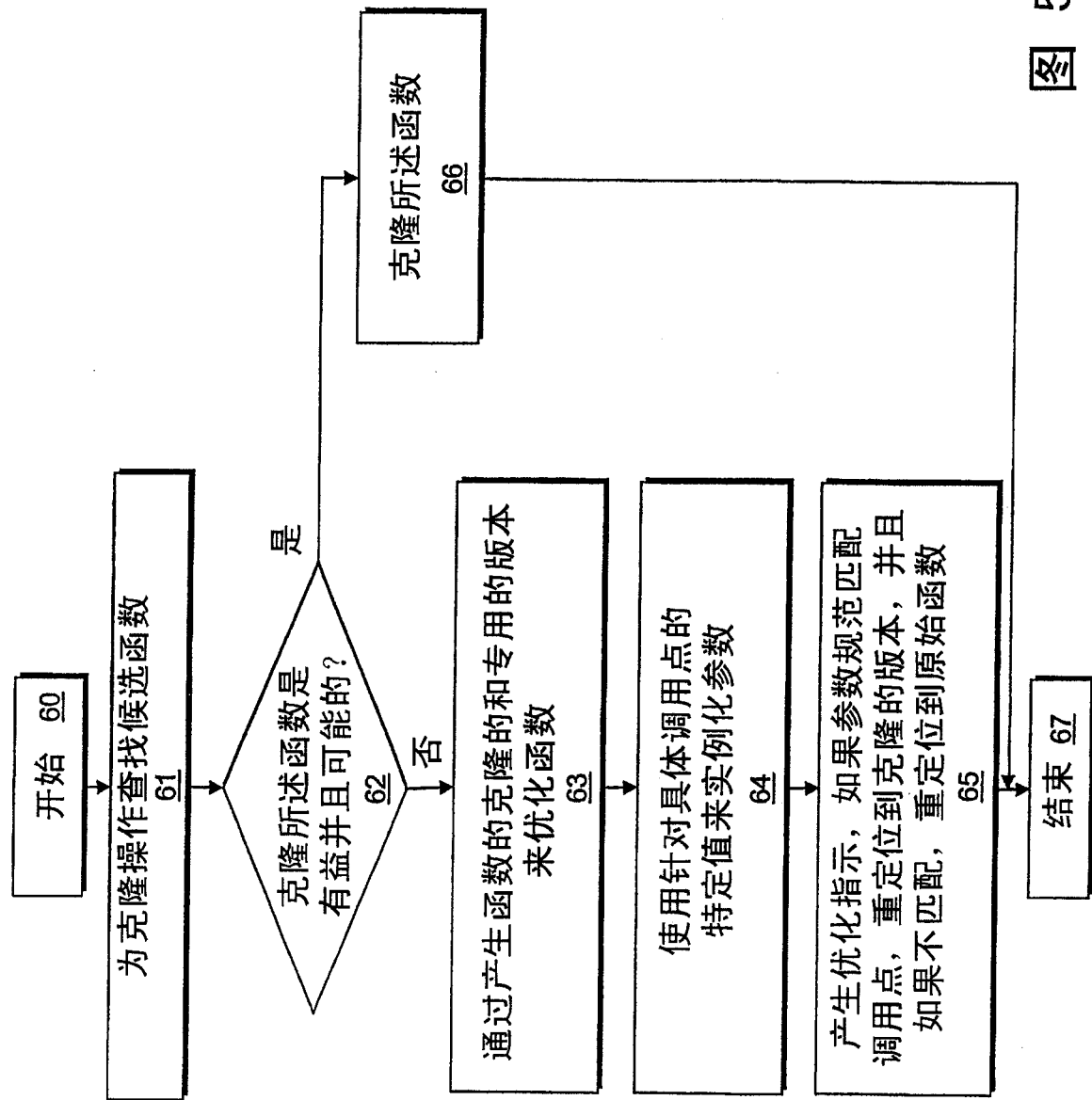


图 5

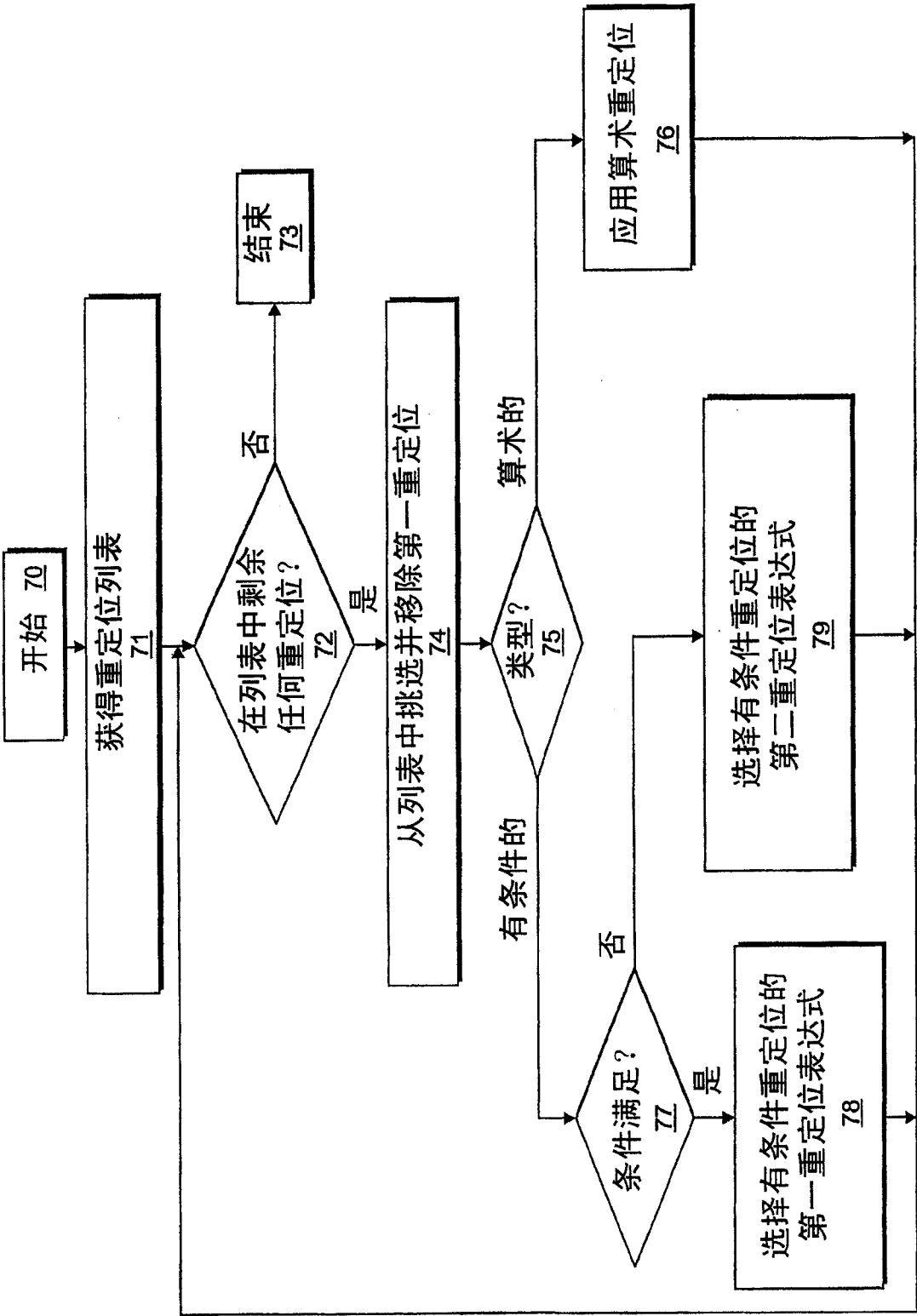


图 6