(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 101542460 B
(45)授权公告日 2012.03.21

(21)申请号 200780042823.9
(22)申请日 2007.11.19
(30)优先权数据
11/602,092 2006.11.20 US
(85)PCT申请进入国家阶段日
2009.05.19
(86)PCT申请的申请数据
PCT/US2007/085149 2007.11.19
(87)PCT申请的公布数据
WO2008/064185 EN 2008.05.29
(73)专利权人 微软公司
地址 美国华盛顿州
(72)发明人 A·L·小布朗
(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 蔡悦
(51)Int.Cl.
G06F 15/16(2006.01)
G06F 9/06(2006.01)
(56)对比文件
US 6336110 B1,2002.01.01, 全文.
US 2004/0267679 A1,2004.12.30, 全文.
US 6684359 B2,2004.01.27, 全文.
US 2002/0169587 A1,2002.11.14, 全文.
审查员 董刚

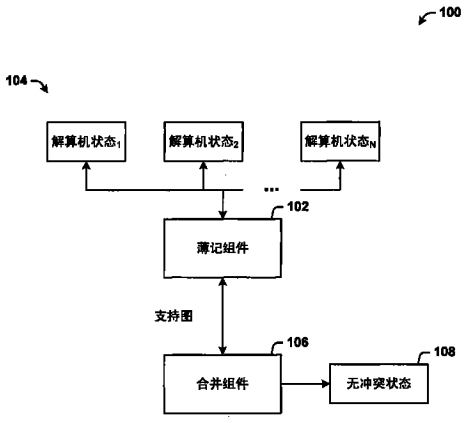
权利要求书 2 页 说明书 14 页 附图 16 页

(54)发明名称

并行化的约束满足问题解算机中的无锁状态合并

(57)摘要

并行约束满足问题(CSP)解算机中的解算机状态合并。并行CSP解算机的计算线程的处理期间的解算机状态被表示为一组支持图。这些支持图以成对的方式合并,从而产生一新的无冲突图。该合并过程是无循环的,冲突被移除并且线程处理是无锁的。该体系结构一般可应用于具有特定形式特性的任何CSP解算机(例如,布尔SAT解算机)。提供了一种便于解算机处理的系统,该系统包括用于将计算线程的输入解算机状态表示为一组图的薄记组件,以及用于将该组图中的至少两个输入图成对合并成表示该计算线程的最终状态的合并图的合并组件。



CN 101542460 B

1. 一种便于约束解算机处理的计算机实现的系统 (100)，包括 ：

用于将计算线程的输入解算机状态 (104) 表示为一组支持图的薄记组件 (102)，所述输入解算机状态接收自运行在所述计算线程上的并行解算机 ；

用于将所述一组支持图中的至少两个输入图成对地合并成表示所述计算线程的最终状态的合并图的合并组件 (106) ；以及

用于约束传播以生成所述合并图的完整性的传播组件，所述传播组件便于作为所述约束传播的一部分来进行不按时间顺序的回溯以便在不改变中间假设的情况下直接改变早先的假设，并便于在并行解算机的合并图的约束传播期间添加不止一个学习到的约束，所述并行解算机是并行约束满足问题解算机，其所述合并图是无冲突的。

2. 如权利要求 1 所述的系统，其特征在于，所述并行约束满足问题解算机是并行布尔可满足性解算机。

3. 如权利要求 1 所述的系统，其特征在于，所述并行约束满足问题解算机根据评估点阵 A 和一组值 D 来定义，其中 D 和 A 是相同的集合。

4. 如权利要求 1 所述的系统，其特征在于，所述合并组件以无锁的方式将所述输入解算机状态合并成所述合并图。

5. 如权利要求 1 所述的系统，其特征在于，还包括用于在约束传播过程期间基于猜测来推断变量赋值的学习和推理组件。

6. 如权利要求 1 所述的系统，其特征在于，所述输入解算机状态来自在所述计算线程上操作的两个并行布尔可满足性解算机，并且所述合并组件帮助将 n 个新的且不同的文字添加到 L 完整的、K 一致的演绎图的 n 个副本中的每一个，其中 n 是正整数。

7. 一种约束解算机处理的计算机实现的方法，包括 ：

将计算线程的输入解算机状态表示为一组支持图 ；

接收与并行解算机的解算机状态相关联的支持图，所述解算机状态与所述计算线程的处理相关联 (202) ；

削减来自所述并行解算机中的每一个的支持图以合并具有相同文字的节点 (204) ；

将经削减的支持图成对地合并成表示所述计算线程的最终状态的合并图 (206) ；

采用不按时间顺序的回溯作为约束传播的一部分来在不改变中间假设的情况下直接改变早先的假设，以通过传播约束来解决所述合并图的完整性 ；

在并行解算机的合并图的约束传播期间添加不止一个学习到的约束，所述并行解算机是并行约束满足问题解算机 ；以及

处理所述合并图中的冲突的文字以使得所述合并图无冲突。

8. 如权利要求 7 所述的方法，其特征在于，还包括同时操纵各个并行解算机的支持图。

9. 如权利要求 7 所述的方法，其特征在于，还包括从冲突的文字集中消除冲突的文字并将所述集合中的其余冲突的文字指定为新演绎图的假设。

10. 如权利要求 7 所述的方法，其特征在于，所述并行约束满足问题解算机是并行布尔可满足性解算机。

11. 一种计算机实现的解算机系统 (300)，包括 ：

用于将计算线程的输入解算机状态表示为一组支持图的计算机实现的装置 ；

用于从并行约束满足问题解算机接收所述一组支持图的计算机实现的装置 (102)，所

述一组支持图表示与计算线程的处理相关联的并行解算机状态；

用于削减来自所述并行约束满足问题解算机中的每一个的支持图的计算机实现的装置（106）；

用于将经削减的支持图成对地合并成合并图的计算机实现的装置（106）；

用于消除所述合并图中的冲突以输出表示所述计算线程的最终状态的无冲突的合并图的计算机实现的装置（312）；

用于采用不按时间顺序的回溯作为约束传播的一部分来在不改变中间假设的情况下直接改变早先的假设，以通过传播约束来解决所述合并图的完整性的计算机实现的装置；

用于在并行解算机的合并图的约束传播期间添加不止一个学习到的约束的计算机实现的装置，所述并行解算机是并行布尔可满足性解算机。

并行化的约束满足问题解算机中的无锁状态合并

[0001] 背景

[0002] 诸如处理器、存储器和存储等硬件方面的技术进步持续地担当用于创建更大且更复杂的、通过处理许多不同类型的媒体数据类型（例如，语音、文本和视频）来提供更丰富的用户体验的软件应用程序、开发程序等的催化剂。

[0003] 在单处理器系统中这些厂商所依靠的硬件支持可能是遥远的，因为与摩尔定律相关联的历史上的电路高速发展（speedup）不再看上去是可容易地获得的。摩尔定律的原理方面是通常由于设备制造方面的技术进步，芯片上的晶体管数量将约每十八个月翻倍一次。历史上，当这实现时，处理器时钟速度也可被提高。然而，现在与更紧密地压缩的晶体管相关联的热密度太高以致于提高时钟速度意味着无法高效且有效地散热。因此，更小的设备不再直接转换成更快且更凉快运行的机器。

[0004] 正在利用的一种替换方案是简单地采用更多的设备。换言之，例如，在处理器领域，设计并行或多处理器系统以适应软件需求。然而，并行处理系统需要用于处理算法或计算线程处理的复杂的协调技术。约束解决在测试这些协调技术时是有用的。然而，传统顺序算法众所周知地难以按有效地利用所有可用共享存储器的并行处理器的方式来重构。

[0005] 诸如布尔可满足性（SAT）解算机等约束满足问题（CSP）解算机对于先前的观察结果也绝不例外。通常，顺序 CSP 解算机具有包括部分解（对约束变量中的某一些的赋值）的当前状态，该解算机试图从该当前状态移至具有通过对一个或多个当前未赋值的变量赋值来创建的增广解的新状态。该新的赋值可通过约束的传播来引起其他赋值。约束的传播进而可导致检测到肯定是部分完成的、必须被改变为新赋值并重新传播的当前赋值中的冲突（为了解除冲突）。

[0006] 在并行处理系统中，该问题解决制度的并行实现必然具有以刚刚描述的方式传播约束的若干并行计算。问题是将若干无冲突解算机状态合并（后传播）成单个无冲突解算机状态。

[0007] 概述

[0008] 以下呈现了本发明的简化概述，以提供对所公开的本发明的某些方面的基本理解。该概述不是详尽的概览，它不旨在标识关键 / 重要的元素，也不旨在描绘其范围。其唯一的目的是以简化的形式来介绍一些概念，作为稍后提出的更为详细的描述的序言。

[0009] 所公开的体系结构提供了对通用约束满足问题（CSP）解算机中的并行处理的支持。解算机的计算线程的状态被表示为一组支持图。各组支持图是经常是通用 CSP 解算机的重要组件的真值维护系统（TMS）的高效实现中的已识别机制。如此处所描述的，支持图通过以成对的方式来合并这些图从而产生新的无冲突图来以新方式使用。这允许通过在多个新赋值上映射约束的并行传播并通过合并从多个传播所得的状态来归约到新问题解算机状态（具有更多赋值的变量）来构造 CSP 解算机。该体系结构一般可应用于具有特定形式特性的任何 CSP 解算机。例如，在一个实现中，该体系结构可特别地在布尔可满足性（SAT）解算机的上下文中应用。

[0010] 此处所公开和所要求保护的体系结构包括便于解算机处理的计算机实现的系统。

该系统包括用于将计算线程的输入解算机状态表示为一组图的薄记组件。合并组件执行该组图中的至少两个输入图到表示计算线程的最终状态的合并图的成对合并。

[0011] 为了实现前述及相关目的，在这里结合下列描述及附图来描述所公开的本发明的某些说明性方面。然而，这些方面仅指示了其中可利用此处公开的原理的各种方法中的少数几种，且旨在包括所有这些方面及其等效方面。结合附图阅读下面的详细描述，则其他优点和新颖特征将变得清楚。

[0012] 附图简述

[0013] 图 1 示出了根据并行实现的便于解算机处理的系统。

[0014] 图 2 示出了根据本发明的处理解算机状态的方法。

[0015] 图 3 示出了采用学习和推理来推断对答案赋值的猜测的替换并行解算机系统。

[0016] 图 4 示出了从示例性约束传播过程中导出的 (L，K) 演绎图。

[0017] 图 5 示出了用于处理从图 4 的支持图输出的冲突约束的新的图。

[0018] 图 6 示出了准备并合并两个支持图的方法。

[0019] 图 7 示出了用于成对合并过程的削减两个输入支持图的方法。

[0020] 图 8 示出了处理合并图的冲突的方法。

[0021] 图 9 示出了削减支持图并将其合并成顺序 SAT 解算机的最终合并的无冲突图的方法。

[0022] 图 10 示出了削减支持图并将其合并成并行 SAT 解算机的最终合并的无冲突图的方法。

[0023] 图 11 示出了将本发明的解算机状态处理应用于多核处理系统的系统的图。

[0024] 图 12 示出了将根据本发明的解算机状态处理应用于多核多处理器系统的系统的图。

[0025] 图 13 示出了将根据本发明的解算机状态处理应用于跨各单独计算系统的解算机状态处理的系统的图。

[0026] 图 14 示出了将应用程序的真值维护系统和推断引擎用作大型搜索空间中的问题解算机的 CSP 解算机系统的图。

[0027] 图 15 示出了可用于采用所公开的并行化的解算机状态体系结构的计算系统的框图。

[0028] 图 16 示出了可利用根据所公开的本发明的并行化的解算机处理的示例性计算环境的示意性框图。

[0029] 详细描述

[0030] 所公开的体系结构提供了对于常规上已众所周知地难以甚至作为顺序问题来解决的共享存储器的并行处理器系统的解决方案。本发明提供了并行约束满足问题 (CSP) 解算机中的无锁状态合并。解算机的计算线程的状态被表示为一组支持图。(为了简化起见，在不会随之发生混淆时将使用短语“支持图”而不是“一组支持图”。) 这些支持图通过以描述无锁的过程并产生新的无冲突支持图的成对方式合并这些图来以新的方式使用。该体系结构一般可应用于其中底层问题可归纳为布尔可满足性的任何 CSP 解算机，并且在一个具体实现中，该体系结构可特别地在布尔可满足性 (SAT) 解算机的上下文中应用。

[0031] 在该 SAT 解算机的具体实现中，SAT 解算机问题开始于以两种方式中的一种引起

该问题的一组布尔公式。期望知道该公式是否始终为真。换言之，期望确定该公式是否是定理。通过对任何公式取补（或反），一等价问题是确定该公式是否是可满足的。即询问，存在使该公式为真的变量赋值吗？出于机械化的目的，这可以是处理该问题的方式。它作为可满足性问题而不是定理证明问题来处理。

[0032] 在数字设计的空间中，期望知道布尔公式是否是定理。显示否定式不是可满足的，而不是证明将显示的公式是定理。布尔公式以诸如例如析取范式等正则形式来呈现。析取范式是其中公式组是有穷的表示，并且在任一个公式中只出现两个逻辑连接符，即，求反符（逻辑 NOT）和析取（或逻辑 OR）。因此，为了使原始问题（现在是正则形式）变为可满足的，存在确保这些析取公式为真的变量赋值。正则形式的每一个公式被称为子句。如此处所使用的，术语“子句”也可指约束。

[0033] 通常在诸如 CSP 解算机等这些复杂领域中，不存在可用于导出甚至部分解的分析方法。必须假设答案，这基本上是猜测。因此，初始假设一旦被程序认为是真就可随着时间改变并在稍后被发现为假。因此，该程序具有基于稍后被发现为假的假设来撤销其可能已做出的推断的问题。为了高效，该过程试图尽可能少地撤销。

[0034] 由于为了展示可满足性而猜测答案的部分赋值，SAT 面临刚刚描述的问题。该猜测使得其他布尔变量答案由于新猜测的逻辑结果而被赋值。这一猜测的结果是或者最终做出使得所有变量变为已赋值的一组成功的猜测，或者所做出的最后一次猜测导致此时放弃这些猜测中的一个的逻辑不一致性。

[0035] 常规上，可采用按时间顺序的回溯来向后移动，从而选择在某一时刻的一个变量的值。该回溯可继续直到变量没有剩下供赋值的合法值。

[0036] 所公开的本发明通过提供撤销最相关的假设（例如，只改变一个假设以及对中间假设不做改变）的能力来采用“不按时间顺序的”回溯。换言之，与其中备份改变当前位置和某一早先位置之间的所有节点的常规系统相反，该算法能够以任意路径返回，挑选一个假设并且只改变该假设。

[0037] 现在参照附图描述本发明，其中相同的附图标记用于指代全文中相同的元素。在以下描述中，为解释起见，描绘了众多具体细节以提供对本发明的全面理解。然而，显然，本发明可以在没有这些具体细节的情况下实现。在其它情况下，以框图形式示出了公知的结构和设备以便于描述它们。

[0038] 最初参考各附图，图 1 示出了根据并行实现的便于解算机处理的系统 100。系统 100 包括薄记组件 102，其用于将与计算线程并行解算机的处理相关联的解算机状态 104（表示为解算机状态$_1$、解算机状态$_2$，...，解算机状态$_N$，其中 N 是正整数）表示为一组支持图。解算机状态 104 能够以并行的方式从不同的系统接收以供薄记组件 102 处理。薄记组件 102 将解算机状态 104 处理成支持图以供后续合并。系统 100 还可包括合并组件 106，其用于将该组支持图中的至少两个输入支持图成对地合并成表示计算线程的最终状态的合并图。

[0039] 薄记和合并组件（102 和 106）对支持图的操纵可以是同时的。换言之，来自一个解算机的解算机状态在处理来自另一个解算机的解算机状态期间接收并处理。

[0040] 在一个实现中，薄记组件 102 从在计算线程上操作的并行 CSP 解算机接收输入解算机状态 104。如将在下文中更详细地描述的，CSP 解算机根据评估点阵 A 和一组值 D 来定

义，其中 D 和 A 是同一集合。在一更具体的替换实现中，薄记组件 102 从在计算线程上操作的并行布尔 SAT 解算机接收输入解算机状态 104。

[0041] 在输入解算机状态 104 是来自在计算线程上操作的两个并行 SAT 解算机的情况下，合并组件 102 帮助将 n 个新的且不同的文字添加到 L 完整的、K 一致的演绎图的 n 个副本中的每一个，其中 n 是正整数。这也将在以下进一步描述。

[0042] 合并组件 106 在没有循环的情况下以无锁的方式将输入解算机状态合并成合并图，并消除合并图中的冲突，由此输出无冲突的图 108。

[0043] 对于无锁处理，实现并行性的一种方式是具有处理并行线程处理的能力。换言之，存在最终需要结果最终化 (finalization) 的多个并行线程。

[0044] 执行并行线程处理的一种常规方式是第一线程简单地将其他线程锁定在某一共享数据结构之外直到该第一线程已达到一结果。该第一线程对获取与先前线程结果一致的结果的其他线程负有责任。

[0045] 本发明通过提前一步并行地处理并且然后组合结果来避免常规锁定。因此，该方法是无锁的，这表现在，因为存在 m 个独立代理，其中 m 是正整数，所以每一个代理都提前一步并且为了获得总答案，这些代理必须组合结果。在该过程的中心，一次组合两个结果。

[0046] 在合并支持图时，存在可能发生的若干不合乎需要的事情。第一，在两个图之间发生冲突。换言之，变量 x 已在第一图中被赋予一真值而另一图中的同一变量 x 具有赋值假。通过求解，合并这两个图并发现导致该冲突的一组假设。一旦发现，就撤回相关联的假设或假设中的一个。

[0047] 第二问题可能在一个图中的赋值是假设而另一个图中的相同的赋值是通过单项消解来导出的时候发生。如果现在试图合并图，则所得图将展示循环。为了解决，使用贪婪法以非常经济的方式来放弃假设。这将在下面将更详细地描述。

[0048] 图 2 示出了根据本发明的处理解算机状态的方法。尽管出于解释简明的目的，此处例如以流图或流程图形式示出的一个或多个方法被示出并描述为一系列动作，但是可以理解和明白，本发明不受动作的次序的限制，因为根据本发明，某些动作可以按与此处所示并描述的不同的次序和 / 或与其它动作同时发生。例如，本领域技术人员将会明白并理解，方法可被替换地表示为一系列相互关联的状态或事件，诸如以状态图的形式。而且，并非所有示出的动作都是实施根据本发明的方法所必需的。

[0049] 在 200，从并行解算机的解算机状态中生成支持图。在 202，接收解算机状态的支持图，该解算机状态与计算线程的处理相关联。在 204，同时削减来自每一个解算机的支持图以合并具有相同文字的节点。在 206，将支持图成对地合并成表示计算线程的最终状态的合并的支持图。在 208，启动约束传播以实现合并图的完整性。在 210，在约束传播期间利用不按时间顺序的回溯来改变先前的假设并解决冲突。在 212，消除冲突的文字以输出无冲突的合并图。

[0050] 现在参考图 3，示出了采用学习和推理来推断对答案赋值的猜测的替换并行解算机系统 300。系统 300 包括生成解算机状态的多个系统（例如，两个顺序解算机系统）302。例如，第一系统 304（表示为系统 $_{A}$），例如，第一顺序 CSP 解算机，生成解算机状态 306（表示为解算机状态 $_{A1}$，...，解算机状态 $_{AS}$，其中 S 是正整数），而第二系统 308（表示为系统 $_{B}$），例如，第二串行 CSP 解算机，生成解算机状态 308（表示为解算机状态 $_{B1}$，...，解算机状态 $_{BT}$，

其中 T 是正整数）。

[0051] 薄记组件 102 从相应的第一和第二系统 (304 和 308) 接收解算机状态 (306 和 310) 并创建关于并行线程处理并解决系统约束的解算机状态的图（例如，支持图）。这些图被传递给合并组件 106 以便合并成具有无冲突状态 108 的合并图。然而，该合并图可能不是完整的。因此，利用由传播组件 312 来促进的约束传播来确保合并图中的完整性。这将在下面将更详细地描述。传播组件 312 还帮助作为约束传播的一部分的不按时间顺序的回溯以便在不改变中间假设的情况下直接改变早先的假设。

[0052] 系统 300 还采用帮助在约束传播期间基于猜测来做出关于变量赋值的推断的学习和推理组件 314（此处也被称为推断引擎）。

[0053] 系统 300 可以在其中并非所有答案都是已知的应用中操作。因此，做出关于世界上正在发生什么的假设。例如，当服务器程序被展示给更多数据时，可能发生该程序所做出的早先的假设将被发现为假。该程序然后具有撤销可能已基于假前提做出的推断的问题。

[0054] 在可满足性问题中，可能存在类似的问题，因为为了展示可满足性，可能必须猜测答案的赋值。换言之，在顺序的情况下，当做出新猜测时，这导致其他布尔变量答案由于该新猜测的逻辑结果而被赋值。此外，两件事情中的一件最终发生。或者做出使得所有变量变为已赋值的一组成功的猜测，或者所做出的最后一个猜测导致逻辑不一致性，此时放弃这些猜测中的一个。

[0055] 在布尔可满足性的最经典版本中，改变最后一个猜测的符号，于是如果早先布尔变量被赋值为假，则将该赋值改为真并且该过程再次继续。逻辑不一致性在赋值可以是既不为真也不为假时发生是可能的，这意味着某一早先做出的赋值肯定是错误的。

[0056] 标识导致所观察到的冲突的一组实际假设，而不是在需要时重复备份并前进。一旦已观察到冲突，期望确切地诊断出是什么导致了可能是或不是最后一个假设所造成的冲突。可精确地做出这一诊断并且因此标识一组不兼容的假设。

[0057] 本主题体系结构（例如，结合选择）可采用各种基于学习和推理的方案来实现其各个方面。例如，用于确定要切换（例如，从真到假或从假到真）哪个赋值的过程可经由自动分类器系统和进程来促进。

[0058] 分类器是将输入属性向量 x = (x1, x2, x3, x4, xn) 映射到类标签 class(x) 的函数。分类器也可输出该输入属于一个类的置信度，即 f(x) = confidence(class(x))。这样的分类可采用概率和 / 或其它统计分析（例如，分解成分析效用和成本以最大化对一人或多人的期望价值）来预测或推断用户期望自动执行的动作。

[0059] 如此处所使用的，术语“推断”和“推论”通常是指从经由事件和 / 或数据捕捉的一组观测推理或推断系统、环境和 / 或用户的状态的过程。例如，推断可用于标识特定的上下文或动作，或可生成状态的概率分布。推断可以是概率性的，即，基于对数据和事件的考虑计算所关注状态的概率分布。推断也可以指用于从一组事件和 / 或数据合成更高级事件的技术。这类推断导致从一组观察到的事件和 / 或储存的事件数据中构造新的事件或动作，而无论事件是否在相邻时间上相关，也无论事件和数据是来自一个还是若干个事件和数据源。

[0060] 支持向量机 (SVM) 是可采用的分类器的一个示例。SVM 通过在可能的输入空间中查找以最佳方式将触发输入事件和非触发事件分离开来的超曲面进行操作。直观上，这使

得分类对于接近但不等同于训练数据的测试数据正确。可采用其它定向和非定向模型分类方法，包括，例如，各种形式的统计回归、朴素贝叶斯、贝叶斯网络、决策树、神经网络、模糊逻辑模型以及表示不同独立性模式的其他统计分类模型。如此处所使用的分类也包括用于分派排序和 / 或优先级的方法。

[0061] 如从本说明书中可以容易地理解的，本主题体系结构可以使用显式训练（例如，经由一般训练数据）以及隐式训练（例如，经由观察用户行为、接收外来信息）的分类器。例如，SVM 经由分类器构造器和特征选择模块内的学习或训练阶段来配置。因此，可采用分类器来根据预定准则自动学习和执行多个功能。

[0062] 换言之，学习和推理组件 314 可将学到的约束应用于冲突处理。一个示例，如下所述，可将一个学到的约束添加到顺序 SAT 解算机以便解决冲突。类似地，在并行 SAT 解算机中，可在约束传播处理期间添加若干学到的约束。

[0063] 在查看约束传播示例之前，提供预备信息以便更好地理解。约束满足问题 (CSP) 通常如下描述 :给定一有限变量集 V、（通常是有限的）一组值 D，（可能是无限的）评估点阵 A 和有限函数（约束）集 $D^{|V|} \rightarrow A$，在 $D^{|V|}$ 中找出一元组以使得在所有约束下该元组的图像之间的点阵满足在 A 中最大。以下是其中 D 和 A 两者都是布尔点阵的描述，从而在 SAT 解算机的上下文中具体描述了本发明。尽管有此限制，但本发明可推广到其中 D 和 A 是相同的集合的任何 CSP。

[0064] 变量被表示为 $x_1, x_2, \ldots, x_n$，其中 n 是正整数。文字是“有符号的 (signed)”变量 :$x_1, \neg x_1, x_2, \neg x_2, \ldots$。L 是 1 的范围在 L 上的文字集。子句是包括表示为□的空子句的有限文字集。子句$\{x_1, \neg x_2, x_3, \neg x_4\}$经常被改写为$x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$（其元素的逻辑 OR(∨)）。任何有限命题公式集都可被呈现为在逻辑上等价的有限子句集。

[0065] 变量可以是未赋值的，或被赋予值真（例如，x_1）或假（例如，$\neg x_1$）。对变量 x 的值为真的赋值与文字 x 合并。对变量 x 的值为假的赋值与文字$\neg x$合并。评估作为将赋值从变量提升到子句的结果归因于子句。满足具有赋值真的子句，而违反具有赋值假的子句。可满足性是是否存在满足一组子句的逻辑变量的赋值的问题。重言式是每一个赋值是否满足子句集的问题。可满足性和重言式是其中公式当且仅当其否定式是不可满足的时候才是重言式的对偶概念。

[0066] 让 K 成为 L 中的文字上的子句集。(L, K) 演绎图是有向图，其节点用与 L 的子集配对的单独文字来标记并且其边用 K 的成员来标记。当不存在关于哪个 L 和 K 是有意义的混淆的时候，该图简单地被称为演绎图。节点是仅在存在其中边用子句 k 来标记的从第一个节点到第二个节点的有向边的情况下的对于另一个节点的 k 前项。

[0067] (L, K) 演绎图仅在以下情况下才是正确标记的 (well-labeled) :节点上的标签是其前项节点的标签的并集，如果节点 1 不具有传入弧，则它用 {1} 来标记，对于文字 1 的节点的所有传入边都用 $1 \vee k$ 形式的子句来标记，来自文字 1 的节点的所有传出边都用$\neg l \vee k$形式的子句来标记。只要存在入射到 1 节点的被标记为 $1 \vee 1_1 \vee \ldots \vee 1_m$ 的边，就存在也入射到 1 节点的被标记为 $1 \vee 1_1 \vee \ldots \vee 1_m$ 的 m-1 个其他边。

[0068] (L,K) 演绎图中用由其本身组成的单元素集（例如，$\neg x_9 @ \{\neg x_9\}$）来标记的节点 1 被称为假设文字。正确标记的 (L,K) 演绎图在入射到节点的传入弧用正好一个子句来标记的情况下被唯一地证明。正确标记的 (L,K) 演绎图仅在它不包含节点 1 和¬1两者的情况下

是K一致的。正确标记的(L,K)演绎图仅在以下情况下是K完整的:对于$l \vee l_1 \vee \ldots \vee l_m$形式的K中的每一个k而言,只要$\neg l_1,\ldots,\neg l_m$在该图中,l就在输入入射边用k来标记的图中。无环正确标记的(L,K)演绎图G在不存在更大的(在节点方面)正确标记的图G'的情况下是L完整的。

[0069] 图4和图5示出了采用支持图来进行冲突处理的并行化的解算机示例。图4示出了从示例性约束传播过程中导出的(L,K)演绎图400。为了示出以上定义的应用,考虑可满足性问题和所尝试的部分解的示例。开始于初始约束集C,其包括:

[0070] $\omega_1 \equiv (\neg x_1 \vee x_2)$

[0071] $\omega_2 \equiv (\neg x_1 \vee x_3 \vee x_9)$

[0072] $\omega_3 \equiv (\neg x_2 \vee \neg x_3 \vee x_4)$

[0073] $\omega_4 \equiv (\neg x_4 \vee x_5 \vee x_{10})$

[0074] $\omega_5 \equiv (\neg x_4 \vee x_6 \vee x_{11})$

[0075] $\omega_6 \equiv (\neg x_5 \vee \neg x_6)$

[0076] $\omega_7 \equiv (x_1 \vee x_7 \vee \neg x_{12})$

[0077] $\omega_8 \equiv (x_1 \vee x_8)$

[0078] $\omega_9 \equiv (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$

[0079] 以及当前假设栈。在演绎图400中,假设被建模为不具有输入弧的假设节点。因此,图400的假设包括:

[0080] $x_1@\{x_1\}$,

[0081] $\neg x_9@\{\neg x_9\}$,

[0082] $\neg x_{10}@\{\neg x_{10}\}$,以及

[0083] $\neg x_{11}@\{\neg x_{11}\}$。

[0084] 图400包括四个假设节点:第一假设节点402,其中x_1已被赋值为真(例如,x_1);第二假设节点404,其中x_9已被赋值为假(例如,$\neg x_9$);第三假设节点406,其中x_{10}已被赋值为假(例如,$\neg x_{10}$);以及第四假设节点408,其中x_{11}已被赋值为假(例如,$\neg x_{11}$)。

[0085] 查看一瞬时快照,图400指示假设节点具有对x_1的赋值真(例如,x_1),而假设节点404具有对x_9的赋值假(例如,$\neg x_9$)。现在考虑被标记为ω_2边,即,如以上所列出的约束子句$\omega_2 \equiv (\neg x_1 \vee x_3 \vee x_9)$;换言之,它可读作"非$x_1$或$x_3$或$x_9$"。然而,注意,在约束$\omega_2$中,$x_1$作为$\neg x_1$来否定地出现,这意味着图400中为真的赋值使得析取为假,并且如被否定地赋值的x_9也使得x_9析取为假。但总约束ω_2必须为真,且能够使得约束子句ω_2为真的仅剩的方式是将x_3赋值为真(例如,x_3)。这正是图400在节点410处通过将x_1赋值为真所指示的。因此,基于输入节点402和404处的假设,以及相关联的约束ω_2,满足约束ω_2的唯一方式是将x_3赋值为真。这是单项消解的应用。

[0086] 继续查看其他节点,到节点412的边使用约束$\omega_1 \equiv (\neg x_1 \vee x_2)$。然而,在节点402处的$x_1$被赋值为真,这使得满足约束$\omega_1$的仅剩的可能性是将在节点412处的$x_2$赋值为真。查看被标记为$\omega_3$的边,约束子句$\omega_3 \equiv (\neg x_2 \vee \neg x_3 \vee x_4)$只可通过将在节点414处的$x_4$赋值为真来满足。换言之,414的输入节点是分别将$x_3$和x2赋值为真的410和412,从而使得满足$\omega_3$的仅剩的可能方式是将$x_4$赋值为真。

[0087] 查看被标记为 ω_4 的边，约束子句 $\omega_4 \equiv (\neg x_4 \vee x_5 \vee x_{10})$ 只可通过将在节点 416 处的 x_5 赋值为真来满足，因为输入 x_{10} 被赋值为假并且输入 x_4 被赋值为真。被标记为 ω_5 的边使用约束子句 $\omega_5 \equiv (\neg x_4 \vee x_6 \vee x_{11})$，该子句只可通过将在节点 418 处的 x_6 赋值为真来满足，因为输入 x_{11} 被赋值为假并且输入 x_4 被赋值为真。最后，被标记为 ω_6 的边使用约束子句 $\omega_6 \equiv (\neg x_5 \vee \neg x_6)$，其基于对在相应的节点 416 和 418 处的 x5 和 x6 的赋值真而约束失败。

[0088] 约束传播导致 (L，K) 演绎图 400 并且当到达最后一个节点 420 时，输出新导出的约束（以不同的形式来编写的）。

[0089] $\omega_C(\kappa(\omega_6)) \equiv \neg(x_1 \wedge \neg x_9 \wedge \neg x_{10} \wedge \neg x_{11})$，

[0090] 其中 κ（卡帕）表示“冲突”并且该约束是什么出错的报告。在此，输出 $\omega_C(\kappa(\omega_6))$ 指示 x_1 为真，x_9 为假，x_{10} 为假，并且 x_{11} 为假无法同时成立，必须放弃其中之一以便进一步处理。这等价于正确形式的非 x_1 或 x_9 或 x_{10} 或 x_{11}。图 5 示出了用于处理从图 4 的支持图输出的冲突约束的新的图 500。选择对以上各项中的三个或四个的假设（例如，$\neg x_9$, $\neg x_{10}$ 和 $\neg x_{11}$）产生了新的图 500 以及又一新导出的输出约束。

[0091] $\omega_C(\kappa(\omega_9)) \equiv \neg(\neg x_9 \wedge \neg x_{10} \wedge \neg x_{11} \wedge x_{12} \wedge x_{13})$。

[0092] 更具体而言，图 500 通过做出假设（丢弃早先假设中的一个），即，将假设节点 502、504 和 506 分别用作 $\neg x_9@\{\neg x_9\}$，$\neg x_{10}@\{\neg x_{10}\}$ 和 $\neg x_{11}@\{\neg x_{11}\}$ 来创建。开始于这些假设并将该新导出的约束 $\omega_C(\kappa(\omega_6))$ 应用于各边产生对在节点 508 处的 x_1 的赋值假以满足约束 $\omega_C(\kappa(\omega_6))$。

[0093] 查看被标记为 ω_7 的边，引入了另一假设节点 510（被赋值为 $x_{12}@\{x_{12}\}$），并且约束子句 $\omega_7 \equiv (x_1 \vee x_7 \vee \neg x_{12})$ 只可通过将在节点 512 处的 x_7 赋值为真来满足，因为输入 x_1 被赋值为假并且输入 x_{12} 被赋值为真。

[0094] 查看被标记为 ω_8 的边，约束子句 $\omega_8 \equiv (x_1 \vee x_8)$ 只可通过将在节点 514 处的 x_8 赋值为真来满足，因为输入 x_1 被赋值为假。现在考虑被标记为 ω_9 的边，约束子句 $\omega_9 \equiv (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$ 无法使用当前输入（节点 512、节点 514 和假设节点 516）来满足。因此，在节点 518 处输出的新导出的约束 $\omega_C(\kappa(\omega_9)) \equiv \neg(\neg x_9 \wedge \neg x_{10} \wedge \neg x_{11} \wedge x_{12} \wedge x_{13})$ 是演绎图 500 出了什么问题的报告。该约束导出过程继续直到不存在冲突状态。

[0095] 更一般而言，一旦获取了每一个并行解算机线程的解算机状态的支持图，就可开始合并。尽管描述集中于成对合并，但合并可使用不止两个图来实现也在考虑范围之内。状态合并是无锁的。换言之，实现并行性的一种常规方式涉及线程处理。例如，在多个并行线程正在处理数据的情况下，最终期望合并这些线程的结果。这样做的一种方式是通过第一线程简单地将其他线程锁定在某一共享数据结构之外直到该第一线程完成其过程。因此，其他被锁在外的线程对写入与已被写入的结果一致的结果负有责任。该常规体系结构至少是低效的。所公开的体系结构避免了这一显式锁定机制。

[0096] 如上所述，该算法提前一步并行地进行处理并且然后组合结果。换言之，为了到达总结果，每一个代理都提前一步进行处理并组合单独的中间结果，并且这继续直到到达总答案。另外，解决冲突处理并且该合并过程是无循环的。

[0097] 如上所述，在解冲突 (deconflict) 处理中，在这些图中的一个或另一个中可能放弃一个或多个假设。此外，期望对于关于放弃什么假设非常节约。通过定义供所公开的体系结构使用的所有信息集（例如，假设、演绎、图，...），归约算法（此处也被称为 redux）以放

弃正在合并的两个图之间的尽可能少的假设的方式来做出选择。回想在单个图的情况下，标识这些冲突的目的是标识如果被放弃将具有尽可能小的后果的假设，而不是必须备份若干先前的假设，这是对于一组支持图的目的。

[0098] 现在将相同的技术应用于一对图，以无循环的方式合并这些图以使得如果放弃一假设，则影响或丢失尽可能小。这可通过利用“贪婪”法或算法来解决。换言之，该方法找出通过做出看上去在特定时刻是最佳的选择来最小化被放弃的东西的容易的方式。可采用利用多得多的分析的更密集的算法；然而，在一个实现中，该贪婪法足够了。

[0099] 图 6 示出了准备并合并两个支持图的方法。在 600，接收供处理的两个支持图。在 602，使用贪婪法来削减（或减少）这些图以便合并对应于相同的文字的节点以减少或消除循环。在 604，将支持图合并成合并图。在 606，在需要时通过传播约束来处理合并图以实现完整性。在 608，在需要时处理合并图以解决冲突。在 610，输出表示无冲突的解算机状态的最终合并图。

[0100] 在准备关于如何能够合并两个 (L，K) 演绎图的更详细的描述时，呈现以下定义：如果 G、G’是非循环的、L 完整的 (L，K) 演绎图，则

[0101] $C_{G,G'}$ 是图 G 和 G’所共有的假设集。

[0102] A_G 是 G 的假设集。

[0103] D_G 是 G 的推论集。

[0104] $A_{G,G'}$ 是 G 中作为 G’的推论的假设集。

[0105] $B_{G,G'}$ 是 G 中未在 G’中提到的假设集

[0106] $A_G = C_{G,G'} \cup {}_{AG,G'} \cup B_{G,G'}$

[0107] $A_{G'} = C_{G,G'} \cup A_{G',G} \cup B_{G',G}$

[0108] $f_G: A_G \rightarrow 2^{L\text{-}A_G} - f_G$ 是产生 G 中的假设的依赖项的函数。

[0109] $h_G: L\text{-}A_G \rightarrow 2^{A_G} - h_G$ 是产生 G 中的推论的前项的函数。

[0110] 图 7 示出了用于成对合并过程的削减两个输入支持图的方法。在 700，在该特定实现中，根据被称为 redux 的函数来启动该过程。函数 redux(G，G′) 产生如下定义的一对图 (G_1，G_2)。在 702，提供了第一定义：令 $1 \in A_{G,G'}$ 为一节点以使得 $|f_{G'}(1)|$（依赖项集的大小）至少与 $A_{G,G'}$ 中的任何其他假设节点的选择一样大。在 704，提供了第二定义：令 $1' \in A_{G',G}$ 为一节点以使得 $|f_{G'}(1')|$ 至少与 $A_{G',G}$ 中的任何其他假设节点的选择一样大。在 706，如果 $|f_{G'}(1)|$ 比 $|f_{G'}(1')|$ 小，则颠倒 G 和 G′ 的角色。在 708，选择 $1'' \in h_{G'}(1)$ 以使得 $|f_{G'}(1'')|$ 与 1″ 的任何其他选择一样小。在 710，将 G″ 定义为缺少 1″ 和任一节点 1_2 的 G′ 的子图以使得 1_2 处于 $|f_{G'}(1'')|$ 中。在 712，如果 $A_{G,G'} = A_{G'',G} = \varnothing$，则退出 redux，并返回对 (G，G”)。在 714，如果非 $A_{G,G'} = A_{G'',G} = \varnothing$，则调用 redux(G，G”)。

[0111] 最终，redux 削减了这两个图以使得对应于相同的文字的节点能够在不害怕循环的情况下合并。如下所示，redux 在其试图放弃尽可能少的派生 (derived)（非假设文字）的意义上以“贪婪的”方式削减。期望以最终实现尽可能多的并行前向进展的方式来消减。然而，可以理解，可采用消减图以便在不害怕循环的情况下合并的其他算法。

[0112] 注意，合并图（再次称之为 G）不再是 L 完整的，这可通过传播约束以使其是 L 完整的来补救。合并图包含冲突的文字 l 和 $\neg l$ 也是可能的，在这种情况下，针对该图计算 deconflict(G)，其产生无冲突的图。图 8 示出了处理合并图的冲突的方法。在 800，在该特

定实现中，根据被称为 deconflict 的函数来启动移除冲突。在 802，选择 $1' \in h_G(1)$ 以使得 $|f_{G'}(1')|$ 与任何其他这样的选择一样小。在 804，选择 $l'' \in h_G(\neg l)$ 以使得 $|f_{G'}(1'')|$ 与任何其他这样的选择一样小。在 806，如果 $|f_{G'}(1')| \leqslant |f_{G'}(1'')|$，则将 1（否则是 1′）选为将连同其所有依赖项一起被删除的假设。在 808，重复该过程直到消除所有冲突的节点。

[0113] redux 和 deconflict 函数可通过安置 merge 函数来在 SAT 解算机中采用，该 merge 函数首先对一对图应用 redux，并且然后对结果应用 deconflict。图 9 示出了消减支持图并将其合并成顺序 SAT 解算机的最终合并的无冲突的图的方法。在 900，在顺序 SAT 解算机中启动图消减（例如，使用 redux）和冲突处理（例如，使用 deconflict）。在 902，顺序 SAT 解算机的内循环将一个新的文字添加到 L 完整的、K 一致的演绎图。在 904，削减支持图以合并节点并产生无循环处理。在 906，传播约束以使得合并图 L 完整。在 908，通过可任选地添加一个学到的约束来提供无冲突的合并图（使用例如，函数 deconflict）。在 910，在需要时重复该过程以输出解算机状态的合并的、无冲突的图。

[0114] 图 10 示出了削减支持图并将其合并成并行 SAT 解算机的最终合并的、无冲突的图的方法。在 1000，在并行 SAT 解算机中启动图削减（例如，使用 redux）和冲突处理（例如，使用 deconflict）。在 1002，并行 SAT 解算机的内循环将 n 个新的且不同的文字添加到 L 完整的、K 一致的支持图的 n 个副本中的每一个。在 1004，削减这 n 个图以合并节点并产生无循环处理。在 1006，在每一个演绎图中传播约束以使得合并图 L 完整。在 1008，通过可任选地添加学到的约束来输出无冲突的合并图（使用例如，函数 deconflict）。

[0115] 图 11 示出了将本发明的解算机状态处理应用于多核处理系统 1102 的系统 1100 的图。多核处理系统 1102 是共享处理器并行处理系统，其可由两者都在同一管芯上制造的第一处理核 1104（表示为核 1）和第二处理核 1106（表示为核 2）来促进。处理系统 1002 还可包括用于正在处理的计算线程的共享缓冲的板载存储器 1108。尽管被示为在同一管芯上，但存储器 1108 可位于该管芯的外部并用于相同的目的。

[0116] 为了支持每一个核（1104 和 1106）对共享线程的处理，提供了一对解算机。例如，提供第一解算机 1110（表示为解算机 $_1$）和第二解算机 1112（表示为解算机 $_2$）（两者都可以是例如 CSP 解算机）以便在处理器系统 1102 的多个核（1104 和 1106）之间执行线程期间执行约束处理。

[0117] 提供了用于根据所公开的本发明的成对的支持图处理的状态系统 1114。状态系统 1114 提供了对以支持图形式从相关联的并行化的解算机（1110 和 1112）接收到的解算机状态（表示为解算机 $_1$ 状态和解算机 $_2$ 状态）的成对处理。状态系统 1114 可包括上述用于实现支持图归约（或削减）、支持图合并、冲突处理和变量赋值的薄记组件 102、合并组件 106、传播组件 312 和推断组件 314。

[0118] 可以理解，状态系统 1114 可严格地用软件、严格地用硬件（例如，作为 ASIC 一专用集成电路设备或现场可编程门阵列 (FPGA)）或作为硬件和软件两者的组合来实现。或者，状态系统 1114 的组件（102、106、312 和 314）可单独地作为硬件和 / 或软件的组合来实现。

[0119] 图 12 示出了将根据本发明的解算机状态处理应用于多核多处理器系统的系统 1200 的图。多核多处理器系统 1200 包括各自是共享存储器并行处理系统的第一多核处

理器系统 1202 和第二多核处理器系统 1204。该第一处理器系统 1202 包括第一处理核 1206（表示为核 $_1$）和第二处理核 1208（表示为核 $_2$），这两者在同一管芯上制造并且共享用于正在处理的计算线程的共享缓冲的共享存储器 1208。

[0120] 该第二多核处理器系统 1204 包括第一处理核 1212（表示为核 $_1$）和第二处理核 1214（表示为核 $_2$），这两者被在同一管芯上制造并且共享用于正在处理的计算线程的共享缓冲的共享存储器 1216。

[0121] 为了支持每一个核（1206 和 1208）对共享线程的处理，提供了相应的解算机集。例如，提供第一解算机 1218（表示为解算机 $_1$）和第二解算机 1220（表示为解算机 $_2$）（两者都可以是例如 CSP 解算机）以便在处理器系统 1202 的多个核（1206 和 1208）之间执行线程期间执行约束处理。类似地，为了支持每一个核（1212 和 1214）对共享线程的处理，提供了相应的解算机集。例如，提供第三解算机 1222（表示为解算机 $_3$）和第四解算机 1224（表示为解算机 $_4$）（两者都可以是例如 CSP 解算机）以便在处理器系统 1204 的多个核（1212 和 1224）之间执行线程期间执行约束处理。

[0122] 每一个解算机（1218、1220、1222 和 1224）将解算机状态传递给状态处理系统 1226。例如，将解算机状态（表示为 S_1 状态和 S_2 状态）从该第一处理系统 1202 转发到状态系统 1226 并且将解算机状态（表示为 S_3 状态和 S_4 状态）从该第二处理器系统 1204 转发到状态系统 1226。

[0123] 提供了用于根据所公开的本发明的成对的支持图处理的状态系统 1226。状态系统 1226 提供了对以支持图的形式从并行化的解算机（1218、1220、1222 和 1224）接收到的解算机状态的成对处理。例如，在这些多核多处理器系统中，线程计算处理可跨核（1206、1208、1212 和 1214）的任何组合来发生。例如，处理可使用核 1208、1212 和 1214 来发生。因此，来自这三个核的状态应被传递给状态系统 1226 以便根据所公开的算法来进行成对处理。为了对其进行支持，状态系统 1226 可包括用于基于线程正在经历的状态处理来从每一个解算机（1218、1220、1222 和 1224）中选择相关状态的选择组件 1228。换言之，无关状态将不会被选择以供支持图处理。然而，可以理解，状态处理现在也可通过包括附加状态系统（未示出）以使得每一个处理器系统（1202 或 1204）都具有一个专用状态系统来并行地执行。

[0124] 如上所述，状态系统 1226 还可包括上述用于实现支持图归约（或削减）、支持图合并、冲突处理和变量赋值的薄记组件 102、合并组件 106、传播组件 312 和推断组件 314。

[0125] 另外，将会认识到，状态系统 1226 可严格地用软件、严格地用硬件（例如，ASIC、FPGA）或作为硬件和软件两者的组合来实现。

[0126] 状态系统（图 11 的 1114 和 / 或 1226）还可被实现为用于安装到用于处理解算机状态的计算系统中的独立软件和 / 或硬件可插入模块。例如，具有高速接口和存储器（例如，非易失性或闪存）的卡可用作处理器子系统的合适的接口以及用于解算机状态处理。或者，或与之组合，可安装根据所公开的本发明来处理解算机状态的软件模块。

[0127] 图 13 示出了将根据本发明的解算机状态处理应用于跨各单独计算系统的解算机状态处理的系统 1300 的图。在此，第一计算系统 1302 是具有用于执行程序和数据处理的系统处理器 1304（以及单个核 1306）的单处理器系统。第二计算系统 1308 是具有用于执行程序和数据处理的多处理器子系统 1310（以及两个处理器 1312 和 1314）的多处理器系

统。这些系统 (1302 和 1308) 被设置为在网络 1316(或合适的高速接口) 上进行通信并且进一步与图 12 的状态系统 1226 进行通信以便进行解算机状态处理。如果一线程跨这两个系统 (1302 和 1308) 处理,则类似于以上在图 12 中所提供的描述,解算机状态可跨这两个系统处理。

[0128] 图 14 示出了采用真值维护系统 1402 和推断引擎 1404(使用学习和推理) 以便作为大型搜索空间中的问题解算机来应用的 CSP 解算机系统 1400 的图。引擎 1404 探查各替换方案、做出选择并为了正确性分析选择。当发生冲突时,真值维护系统帮助消除冲突并相应地更新其知识库以供将来使用。

[0129] 如在本申请中所使用的,术语“组件”和“系统”旨在表示计算机相关的实体,其可以是硬件、硬件和软件的组合、软件、或者执行中的软件。例如,组件可以是但不限于,在处理器上运行的进程、处理器、硬盘驱动器、(光和 / 或磁存储介质的) 多个存储驱动器、对象、可执行代码、执行的线程、程序、和 / 或计算机。作为说明,运行在服务器上的应用程序和服务器都可以是组件。一个或多个组件可以驻留在进程和 / 或执行的线程内,且组件可以位于一台计算机内上 / 或分布在两台或更多的计算机之间。

[0130] 现在参考图 15,示出了可用于采用所公开的并行化的解算机状态体系结构的计算系统 1500 的框图。为了提供用于其各方面的附加上下文,图 15 及以下讨论旨在提供对其中可实现本发明的各方面的合适的计算系统 1500 的简要概括描述。尽管以上描述是在可在一个或多个计算机上运行的计算机可执行指令的一般上下文中进行的,但是本领域的技术人员将认识到,本发明也可结合其它程序模块和 / 或作为硬件和软件的组合来实现。

[0131] 一般而言,程序模块包括执行特定任务或实现特定抽象数据类型的例程、程序、组件、数据结构等等。此外,本领域的技术人员可以理解,本发明的方法可用其它计算机系统配置来实施,包括单处理器或多处理器计算机系统、小型机、大型计算机、以及个人计算机、手持式计算设备、基于微处理器的或可编程消费电子产品等,其每一个都可操作上耦合到一个或多个相关联的设备。

[0132] 所示的本发明的各方面也可在其中某些任务由通过通信网络链接的远程处理设备来执行的分布式计算环境中实施。在分布式计算环境中,程序模块可以位于本地和远程存储器存储设备中。

[0133] 计算机通常包括各种计算机可读介质。计算机可读介质可以是可由计算机访问的任何可用介质,且包括易失性和非易失性介质、可移动和不可移动介质。作为示例而非限制,计算机可读介质可以包括计算机存储介质和通信介质。计算机存储介质包括以用于存储诸如计算机可读指令、数据结构、程序模块或其它数据之类的信息的任意方法或技术实现的易失性和非易失性、可移动和不可移动介质。计算机存储介质包括但不限于 RAM、ROM、EEPROM、闪存或者其它存储器技术、CD-ROM、数字视频盘 (DVD) 或其它光盘存储、磁带盒、磁带、磁盘存储或其它磁存储设备、或可以用于存储所需信息并且可以由计算机访问的任何其它介质。

[0134] 再次参考图 15,用于实现各方面的示例性计算系统 1500 包括计算机 1502,计算机 1502 包括处理单元 1504、系统存储器 1506 和系统总线 1508。系统总线 1508 提供了用于包括,但不限于系统存储器 1506 的系统组件对处理单元 1504 的接口。处理单元 1504 可以是各种市场上可购买到的处理器中的任意一种。双微处理器和其它多处理器体系结构也可用

作处理单元 1504。

[0135] 系统总线 1508 可以是若干种总线结构中的任一种，这些总线结构还可互连到存储器总线（带有或没有存储器控制器）、外围总线、以及使用各类市场上可购买到的总线体系结构中的任一种的局部总线。系统存储器 1506 包括只读存储器 (ROM) 1510 和随机存取存储器 (RAM) 1512。基本输入 / 输出系统 (BIOS) 储存在诸如 ROM、EPROM、EEPROM 等非易失性存储器 1510 中，其中 BIOS 包含帮助诸如在启动期间在计算机 1502 内的元件之间传输信息的基本例程。RAM 1512 还可包括诸如静态 RAM 等高速 RAM 来用于高速缓存数据。

[0136] 计算机 1502 还包括内置硬盘驱动器 (HDD) 1514（例如，EIDE、SATA），该内置硬盘驱动器 1514 还可被配置成在合适的机壳（未示出）中外部使用；磁软盘驱动器 (FDD) 1516（例如，从可移动磁盘 1518 中读取或向其写入）；以及光盘驱动器 1520（例如，从 CD-ROM 盘 1522 中读取，或从诸如 DVD 等其它高容量光学介质中读取或向其写入）。硬盘驱动器 1514、磁盘驱动器 1516 和光盘驱动器 1520 可分别通过硬盘驱动器接口 1524、磁盘驱动器接口 1526 和光盘驱动器接口 1528 连接到系统总线 1508。用于外置驱动器实现的接口 1524 包括通用串行总线 (USB) 和 IEEE 1394 接口技术中的至少一种或两者。其它外部驱动器连接技术在本发明所考虑的范围之内。

[0137] 驱动器及其相关联的计算机可读介质提供了对数据、数据结构、计算机可执行指令等的非易失性存储。对于计算机 1502，驱动器和介质容纳适当的数字格式的任何数据的存储。尽管以上对计算机可读介质的描述涉及 HDD、可移动磁盘以及诸如 CD 或 DVD 等可移动光学介质，但是本领域的技术人员应当理解，示例性操作环境中也可使用可由计算机读取的任何其它类型的介质，诸如 zip 驱动器、磁带盒、闪存卡、盒式磁带等等，并且任何这样的介质可包含用于执行所公开的发明的方法的计算机可执行指令。

[0138] 多个程序模块可被存储在驱动器和 RAM 1512 中，包括操作系统 1530、一个或多个应用程序 1532（例如，上述无锁 CSP 解算机处理系统）、其它程序模块 1534 和程序数据 1536。所有或部分操作系统、应用程序、模块和 / 或数据也可被高速缓存在 RAM 1512 中。应该明白，本发明可以用各种市场上可购得的操作系统或操作系统的组合来实施。

[0139] 用户可以通过一个或多个有线 / 无线输入设备，例如键盘 1538 和诸如鼠标 1540 等定点设备将命令和信息输入到计算机 1502 中。其它输入设备（未示出）可包括话筒、IR 遥控器、操纵杆、游戏手柄、指示笔、触摸屏等等。这些和其它输入设备通常通过耦合到系统总线 1508 的输入设备接口 1542 连接到处理单元 1504，但也可通过其它接口连接，如并行端口、IEEE 1394 串行端口、游戏端口、USB 端口、IR 接口等等。

[0140] 监视器 1544 或其它类型的显示设备也经由接口，诸如视频适配器 1546 连接至系统总线 1508。除了监视器 1544 之外，计算机通常包括诸如扬声器和打印机等的其它外围输出设备（未示出）。

[0141] 计算机 1502 可使用经由有线和 / 或无线通信至一个或多个远程计算机，诸如远程计算机 1548 的逻辑连接在网络化环境中操作。远程计算机 1548 可以是工作站、服务器计算机、路由器、个人计算机、便携式计算机、基于微处理器的娱乐设备、对等设备或其它常见的网络节点，并且通常包括以上相对于计算机 1502 描述的许多或所有元件，尽管为简明起见仅示出了存储器 / 存储设备 1550。所描绘的逻辑连接包括到局域网 (LAN) 1552 和 / 或例如广域网 (WAN) 1554 等更大的网络的有线 / 无线连接。这一 LAN 和 WAN 联网环境常见于办

公室和公司，并且方便了诸如内联网等企业范围计算机网络，所有这些都可连接到例如因特网等全球通信网络。

[0142] 当在LAN网络环境中使用时，计算机1502通过有线和/或无线通信网络接口或适配器1556连接到局域网1552。适配器1556可以方便到LAN 1552的有线或无线通信，并且还可包括其上设置的用于与无线适配器1556通信的无线接入点。

[0143] 当在WAN网络环境中使用时，计算机1502可包括调制解调器1558，或连接到WAN 1554上的通信服务器，或具有用于通过WAN 1554，诸如通过因特网建立通信的其它装置。或为内置或为外置以及有线或无线设备的调制解调器1558经由串行端口接口1542连接到系统总线1508。在网络化环境中，相对于计算机1502所描述的程序模块或其部分可以存储在远程存储器/存储设备1550中。应该理解，所示网络连接是示例性的，并且可以使用在计算机之间建立通信链路的其它手段。

[0144] 计算机1502可用于与操作上设置在无线通信中的任何无线设备或实体通信，这些设备或实体例如有打印机、扫描仪、台式和/或便携式计算机、便携式数据助理、通信卫星、与无线可检测标签相关联的任何一个设备或位置（例如，公用电话亭、报亭、休息室）以及电话。这至少包括Wi-Fi和蓝牙™无线技术。由此，通信可以如对于常规网络那样是预定义结构，或者仅仅是至少两个设备之间的自组织(ad hoc)通信。

[0145] 现在参考图16，示出了可利用根据所公开的本发明的并行化的解算机处理的示例性计算环境1600的示意性框图。系统1600包括一个或多个客户机1602。客户机1602可以是硬件和/或软件（例如，线程、进程、计算设备）。客户机1602可例如使用通过本发明而容纳cookie和/或相关联的上下文信息。

[0146] 系统1600还包括一个或多个服务器1604。服务器1604也可以是硬件和/或软件（例如，线程、进程、计算设备）。服务器1604可以例如通过使用本体系结构来容纳线程以执行变换。在客户机1602和服务器1604之间的一种可能的通信能够以适合在支持并行化的解算机状态处理的两个或多个计算机进程之间传输的数据分组的形式进行。数据分组可包括例如cookie和/或相关联的上下文信息。系统1600包括可以用来使客户机1606和服务器1602之间通信更容易的通信框架1604（例如，诸如因特网等全球通信网络）。

[0147] 通信可经由有线（包括光纤）和/或无线技术来促进。客户机1602操作上被连接到可以用来存储对客户机1602本地的信息（例如，cookie和/或相关联的上下文信息）的一个或多个客户机数据存储1608。同样地，服务器1604可在操作上连接到可以用来存储对服务器1604本地的信息的一个或多个服务器数据存储1610。

[0148] 上面描述的包括所公开的本发明的各示例。当然，描述每一个可以想到的组件和/或方法的组合是不可能的，但本领域内的普通技术人员应该认识到，许多其它组合和排列都是可能的。因此，本发明旨在涵盖所有这些落入所附权利要求书的精神和范围内的更改、修改和变化。此外，就在说明书或权利要求书中使用术语“包括”而言，这一术语旨在以与术语“包含”在被用作权利要求书中的过渡词时所解释的相似的方式为包含性的。

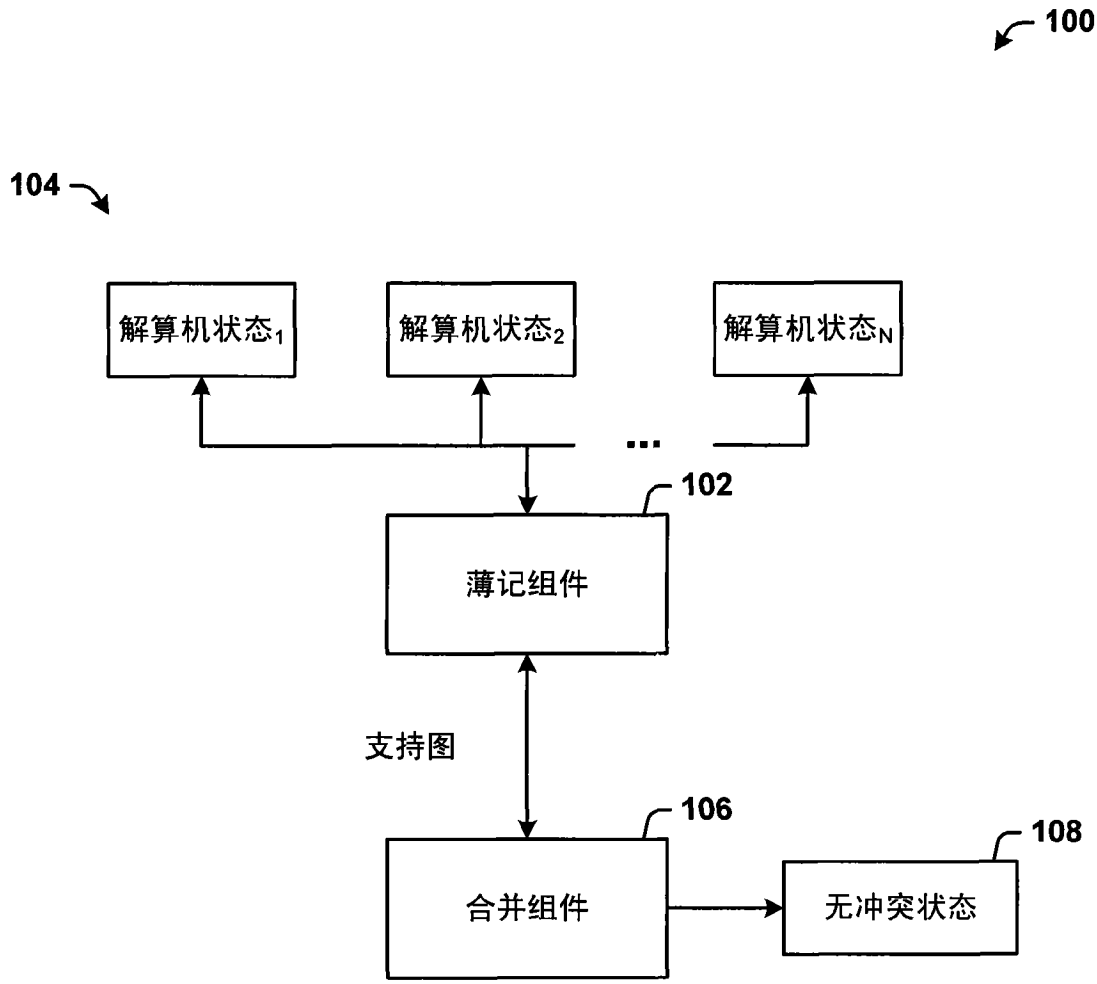


图 1

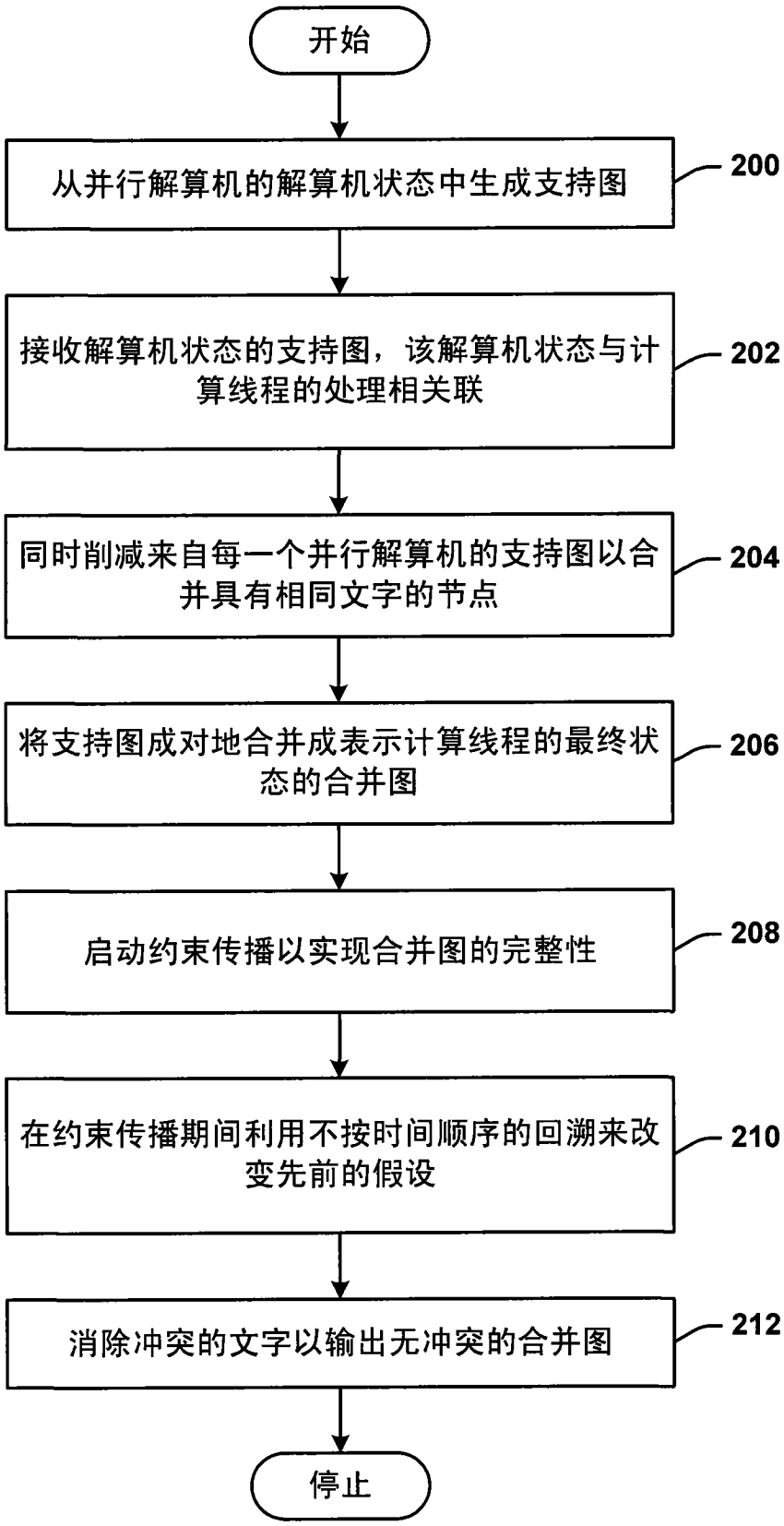

开始
从并行解算机的解算机状态中生成支持图
200
接收解算机状态的支持图，该解算机状态与计算线程的处理相关联
202
同时削减来自每一个并行解算机的支持图以合并具有相同文字的节点
204
将支持图成对地合并成表示计算线程的最终状态的合并图
206
启动约束传播以实现合并图的完整性
208
在约束传播期间利用不按时间顺序的回溯来改变先前的假设
210
消除冲突的文字以输出无冲突的合并图
212
停止

图 2

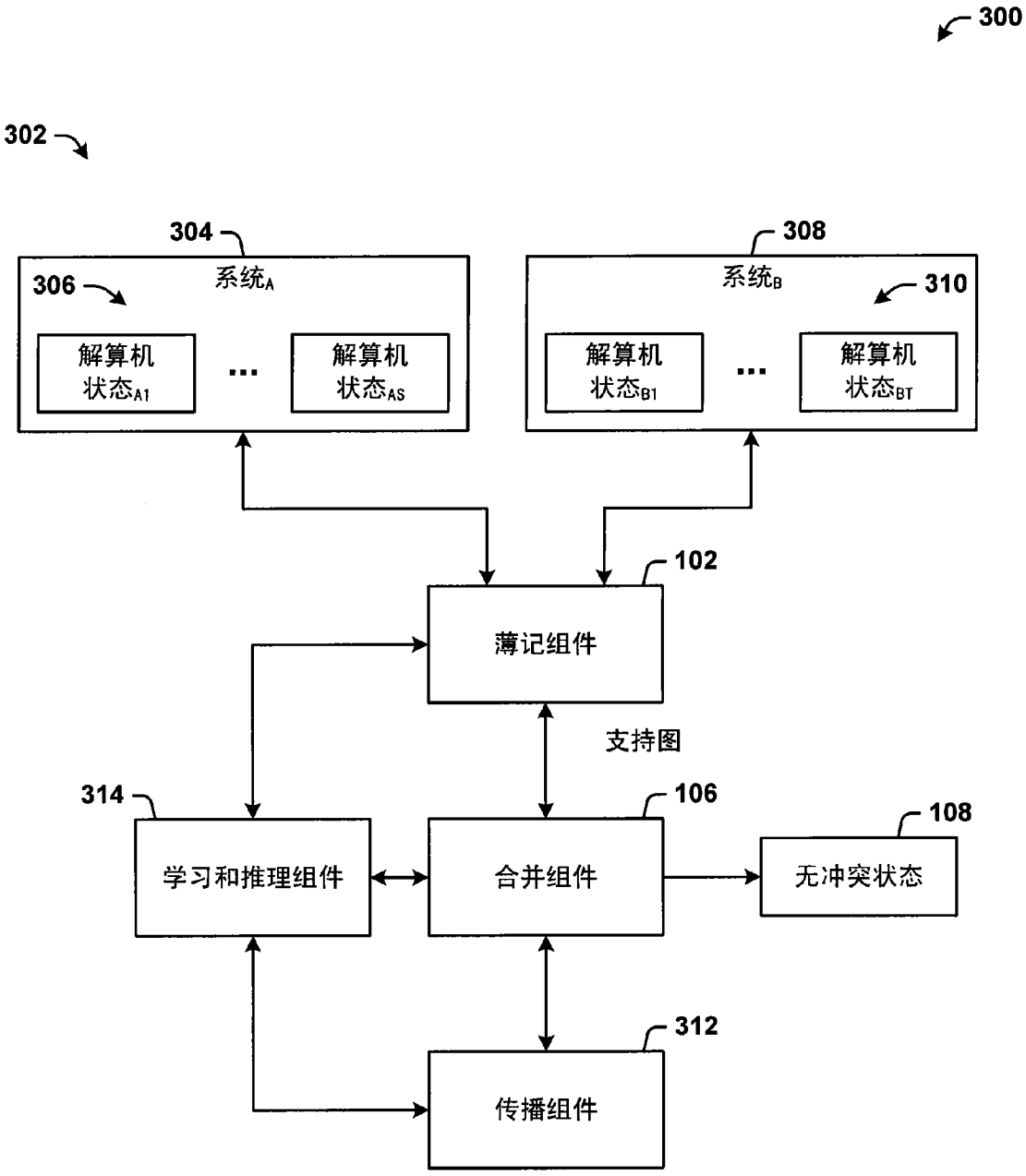

300
302
304
系统A
306
解算机状态A1
...
解算机状态AS
308
系统B
310
解算机状态B1
...
解算机状态BT
102
薄记组件
支持图
314
学习和推理组件
106
合并组件
108
无冲突状态
312
传播组件

图 3

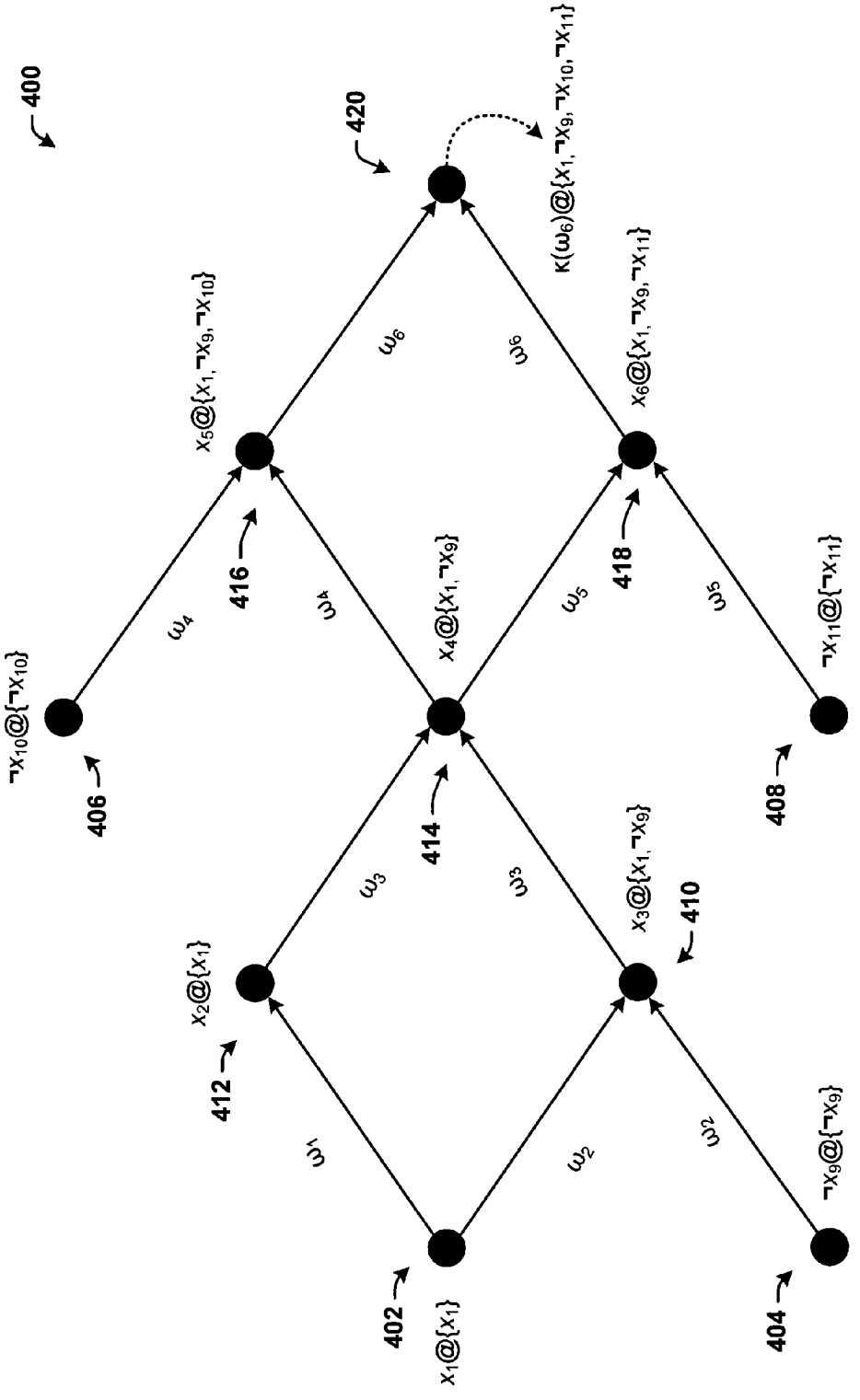

400
420
K(ω6)@{X1, ¬X9, ¬X10, ¬X11}
X5@{X1, ¬X9, ¬X10}
X6@{X1, ¬X9, ¬X11}
ω6
ω6
416
418
ω4
ω4
ω5
ω5
¬X10@{¬X10}
X4@{X1, ¬X9}
¬X11@{¬X11}
406
414
408
ω3
ω3
X2@{X1}
X3@{X1, ¬X9}
410
412
ω1
ω2
ω2
¬X9@{¬X9}
402
X1@{X1}
404

图 4

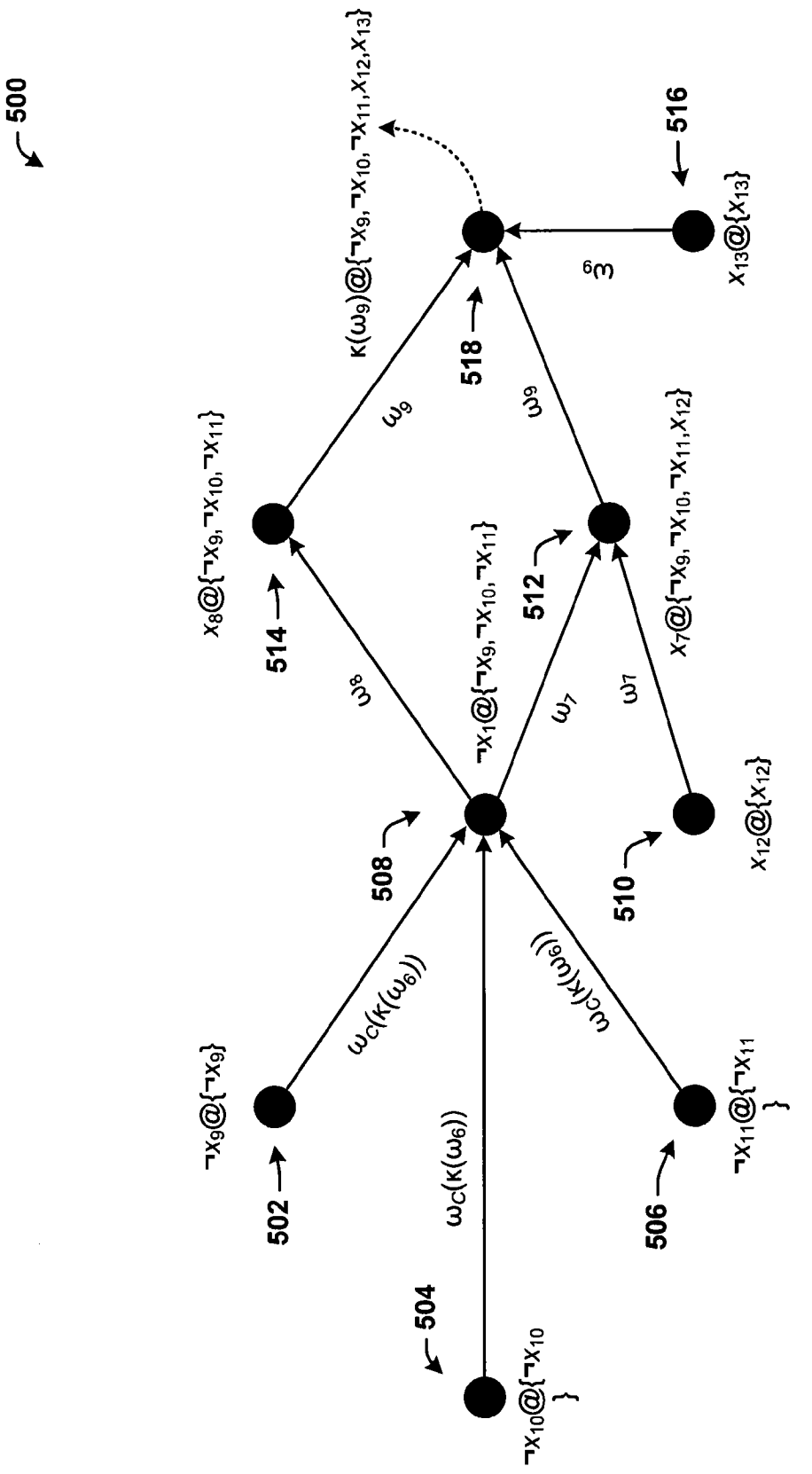

500
502
504
506
508
510
512
514
516
518
$\neg x_9@\{\neg x_9\}$
$\neg x_{10}@\{\neg x_{10}\}$
$\neg x_{11}@\{\neg x_{11}\}$
$x_{12}@\{x_{12}\}$
$x_{13}@\{x_{13}\}$
$\neg x_1@\{\neg x_9, \neg x_{10}, \neg x_{11}\}$
$x_8@\{\neg x_9, \neg x_{10}, \neg x_{11}\}$
$x_7@\{\neg x_9, \neg x_{10}, \neg x_{11}, x_{12}\}$
$\kappa(\omega_9)@\{\neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}\}$
$\omega_C(\kappa(\omega_6))$
$\omega_C(\kappa(\omega_6))$
$\omega_C(\kappa(\omega_6))$
ω_8
ω_7
ω_7
ω_9
ω_9
ω_9

图 5

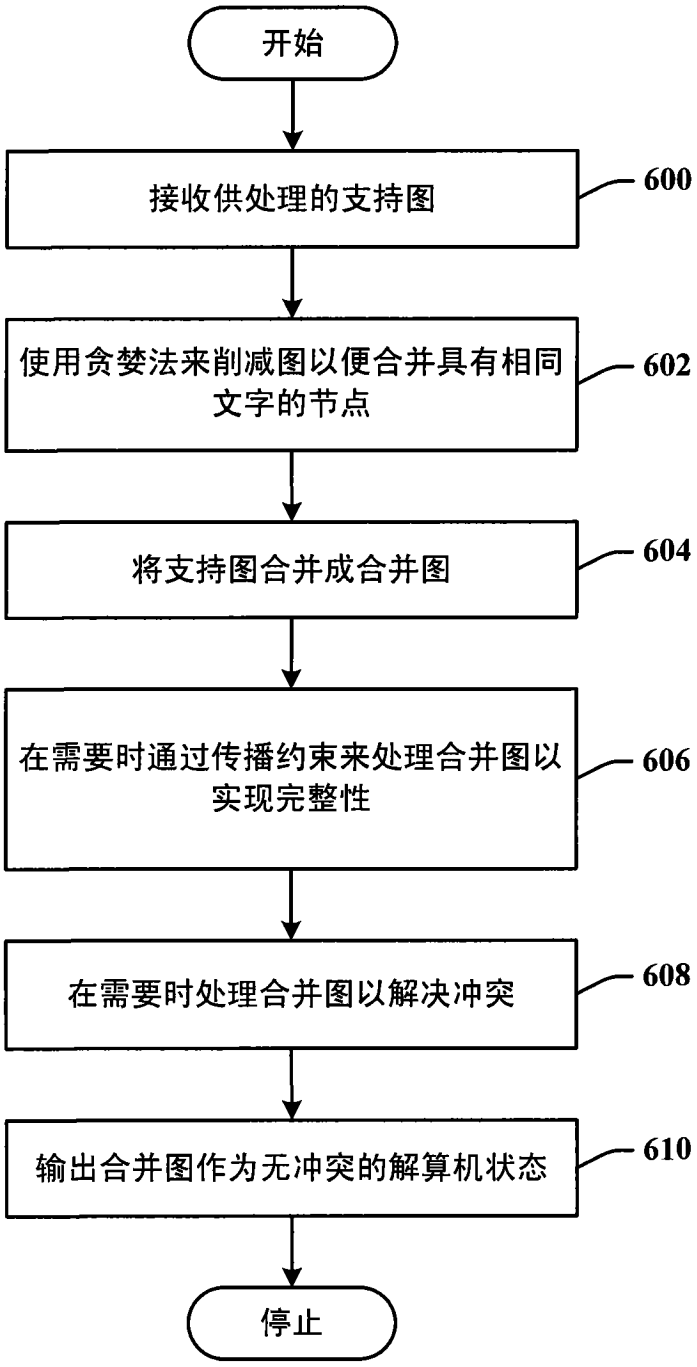

开始
接收供处理的支持图
600
使用贪婪法来削减图以便合并具有相同文字的节点
602
将支持图合并成合并图
604
在需要时通过传播约束来处理合并图以实现完整性
606
在需要时处理合并图以解决冲突
608
输出合并图作为无冲突的解算机状态
610
停止

图 6

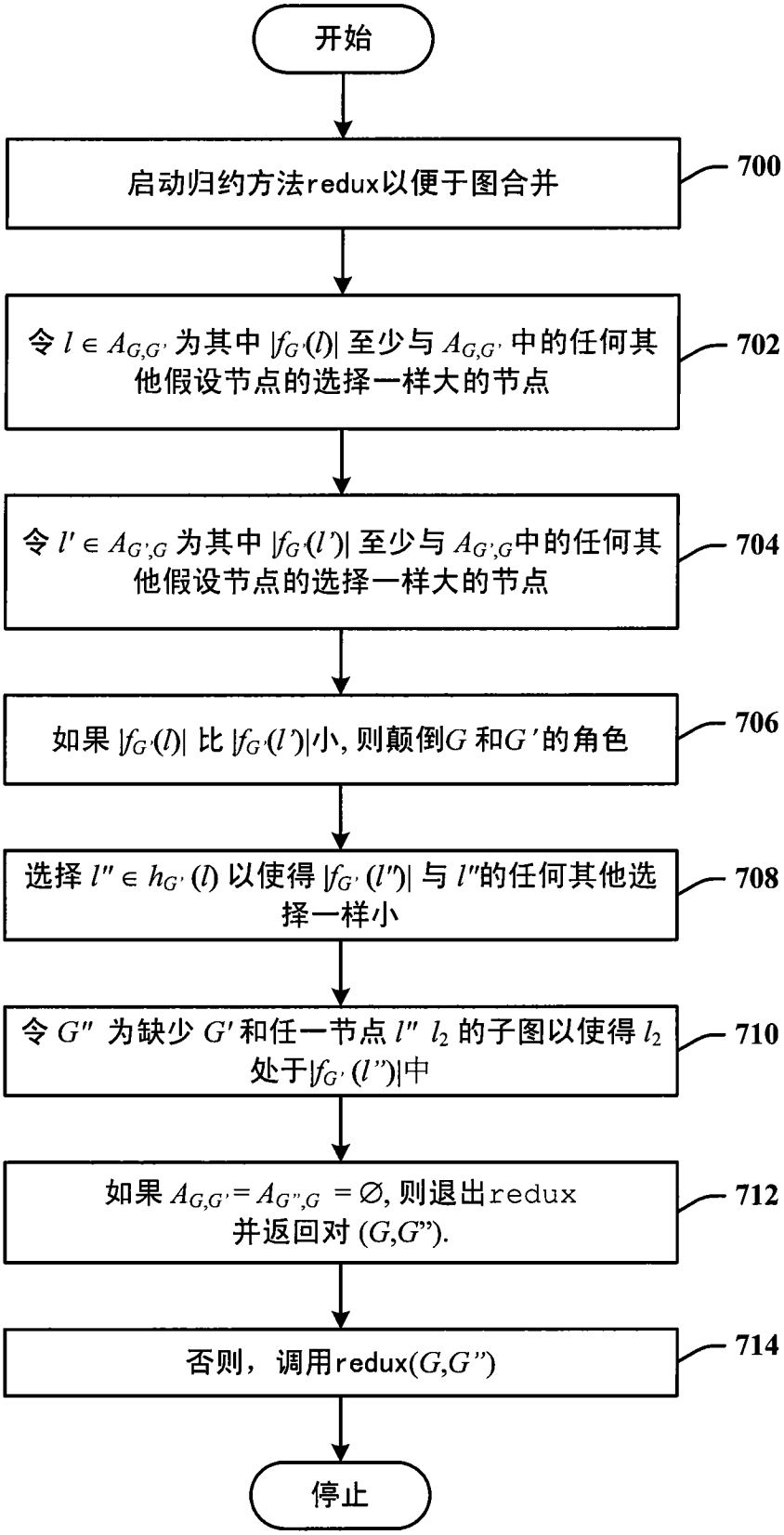

开始
启动归约方法redux以便于图合并
700
令 $l \in A_{G,G'}$ 为其中 $|f_{G'}(l)|$ 至少与 $A_{G,G'}$ 中的任何其他假设节点的选择一样大的节点
702
令 $l' \in A_{G',G}$ 为其中 $|f_{G}(l')|$ 至少与 $A_{G',G}$中的任何其他假设节点的选择一样大的节点
704
如果 $|f_{G'}(l)|$ 比 $|f_{G}(l')|$小, 则颠倒G 和G'的角色
706
选择 $l'' \in h_{G'}(l)$ 以使得 $|f_{G'}(l'')|$ 与 l''的任何其他选择一样小
708
令 G'' 为缺少 G'和任一节点 l'' l_2 的子图以使得 l_2 处于$|f_{G'}(l'')|$中
710
如果 $A_{G,G'} = A_{G'',G} = \varnothing$, 则退出redux并返回对 (G,G'').
712
否则，调用redux(G,G'')
714
停止

图 7

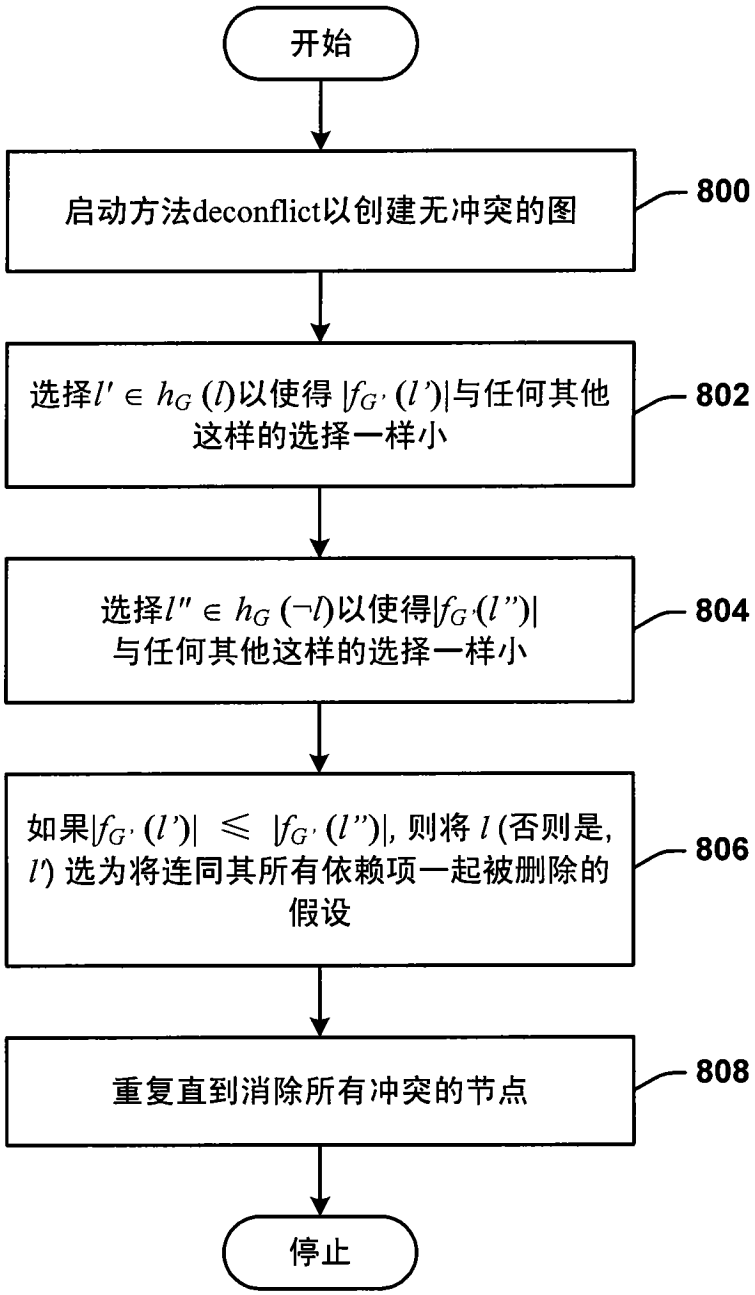

开始
启动方法deconflict以创建无冲突的图
800
选择$l' \in h_G(l)$以使得$|f_{G'}(l')|$与任何其他这样的选择一样小
802
选择$l'' \in h_G(\neg l)$以使得$|f_{G'}(l'')|$与任何其他这样的选择一样小
804
如果$|f_{G'}(l')| \leqslant |f_{G'}(l'')|$，则将$l$(否则是，$l'$)选为将连同其所有依赖项一起被删除的假设
806
重复直到消除所有冲突的节点
808
停止

图 8

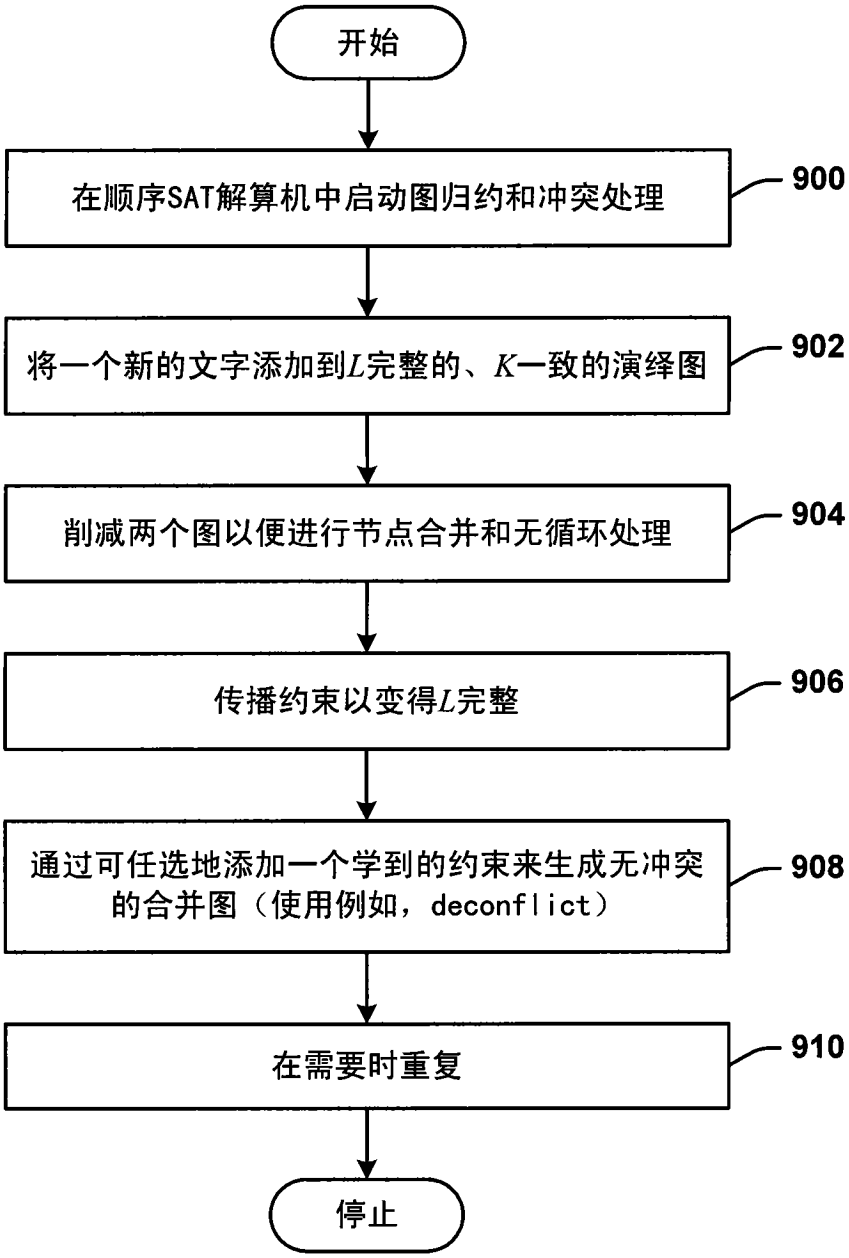

开始
在顺序SAT解算机中启动图归约和冲突处理
900
将一个新的文字添加到L完整的、K一致的演绎图
902
削减两个图以便进行节点合并和无循环处理
904
传播约束以变得L完整
906
通过可任选地添加一个学到的约束来生成无冲突的合并图（使用例如，deconflict）
908
在需要时重复
910
停止

图 9

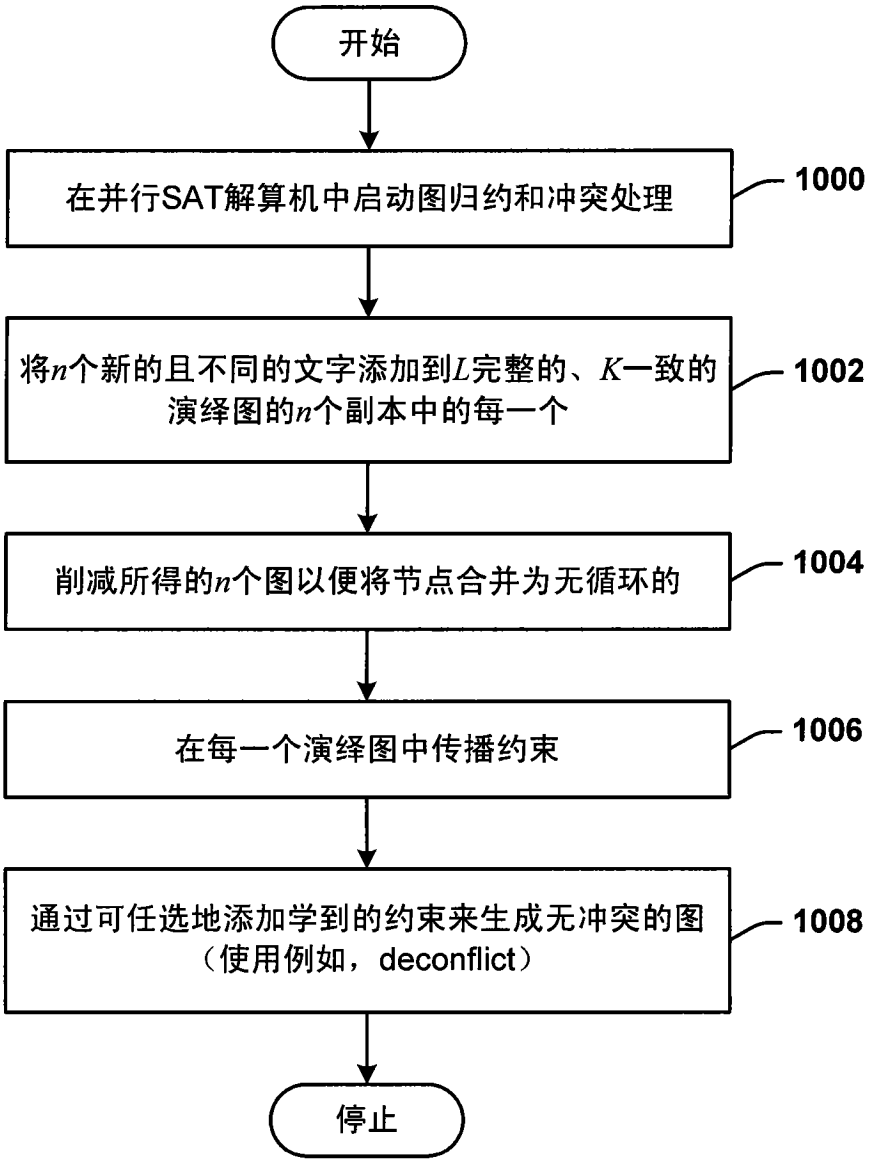

开始
在并行SAT解算机中启动图归约和冲突处理
1000
将n个新的且不同的文字添加到L完整的、K一致的演绎图的n个副本中的每一个
1002
削减所得的n个图以便将节点合并为无循环的
1004
在每一个演绎图中传播约束
1006
通过可任选地添加学到的约束来生成无冲突的图（使用例如，deconflict）
1008
停止

图 10

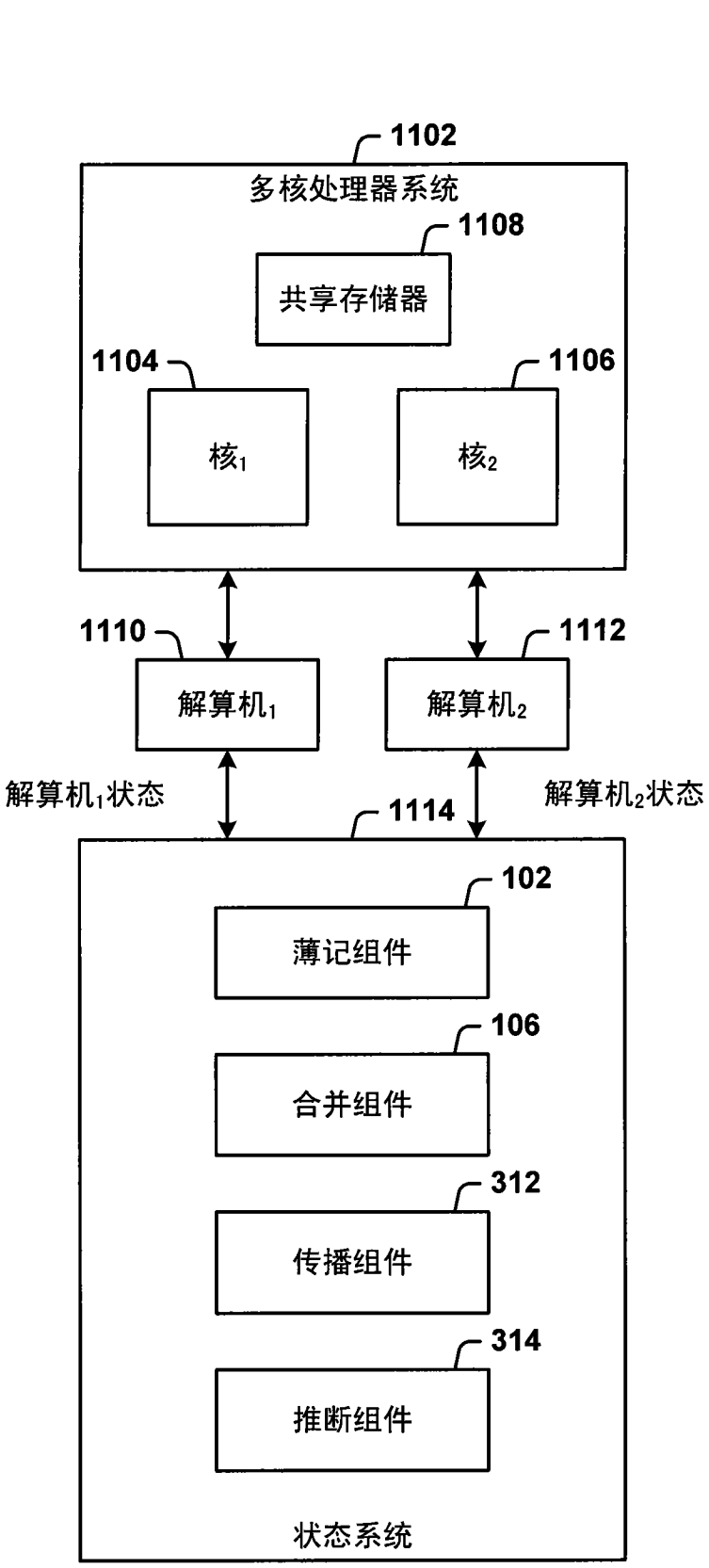

1100
1102
多核处理器系统
1108
共享存储器
1104
1106
核1
核2
1110
1112
解算机1
解算机2
解算机1状态
解算机2状态
1114
102
薄记组件
106
合并组件
312
传播组件
314
推断组件
状态系统

图 11

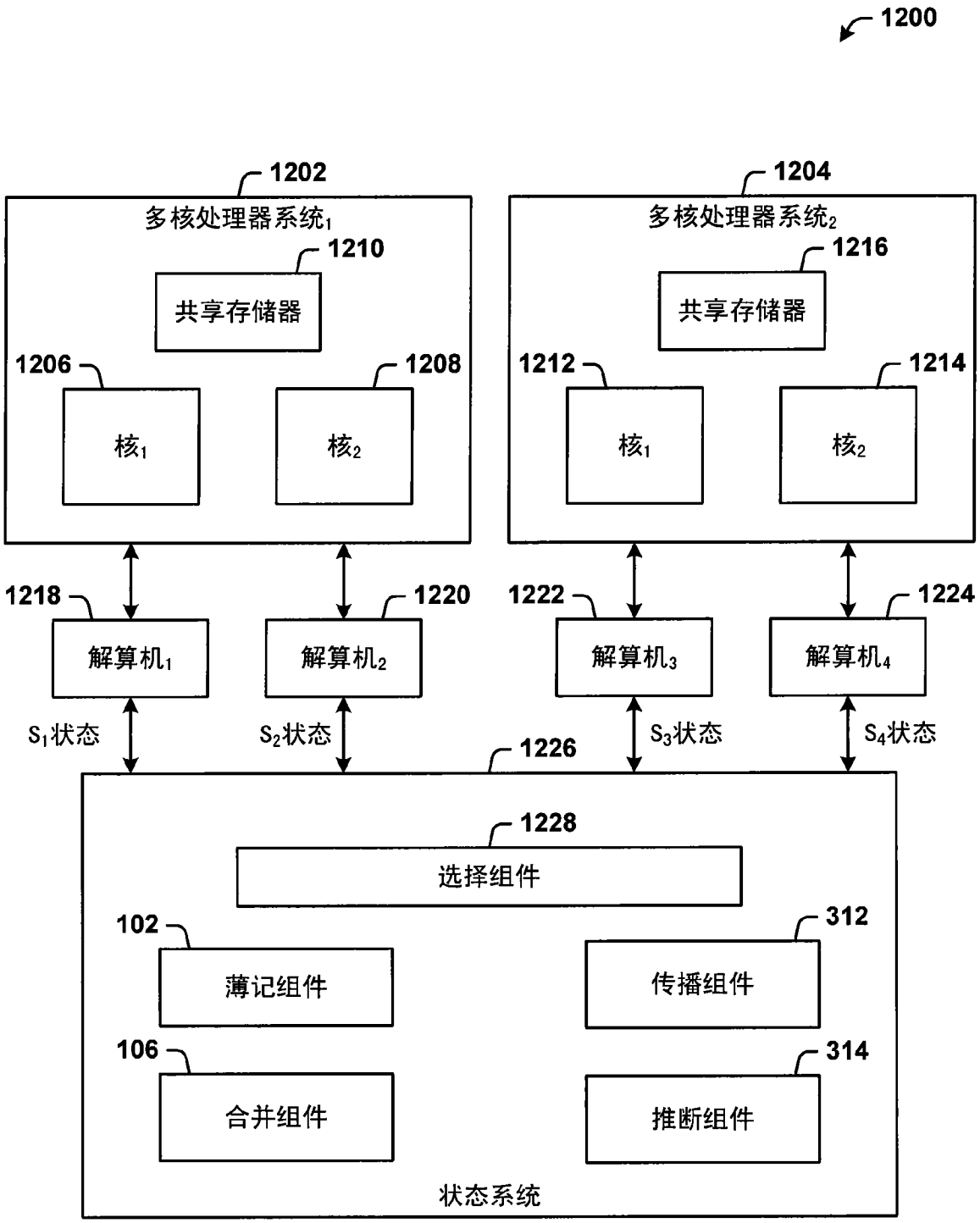

1200
1202
多核处理器系统1
1210
共享存储器
1206
核1
1208
核2
1204
多核处理器系统2
1216
共享存储器
1212
核1
1214
核2
1218
解算机1
1220
解算机2
1222
解算机3
1224
解算机4
S1状态
S2状态
S3状态
S4状态
1226
1228
选择组件
102
薄记组件
312
传播组件
106
合并组件
314
推断组件
状态系统

图 12

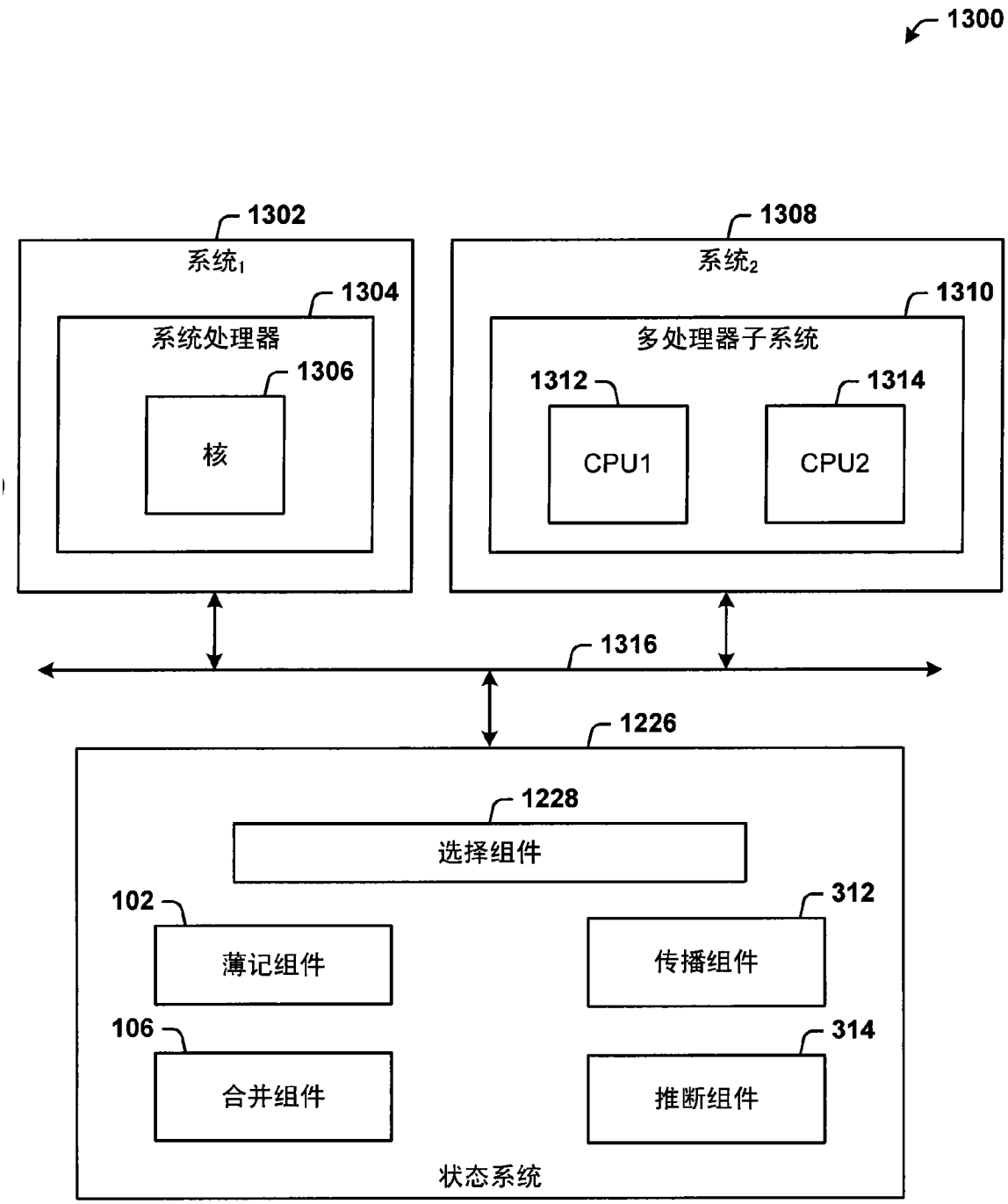

1300
1302
系统1
1304
系统处理器
1306
核
1308
系统2
1310
多处理器子系统
1312
CPU1
1314
CPU2
1316
1226
1228
选择组件
102
薄记组件
312
传播组件
106
合并组件
314
推断组件
状态系统

图 13

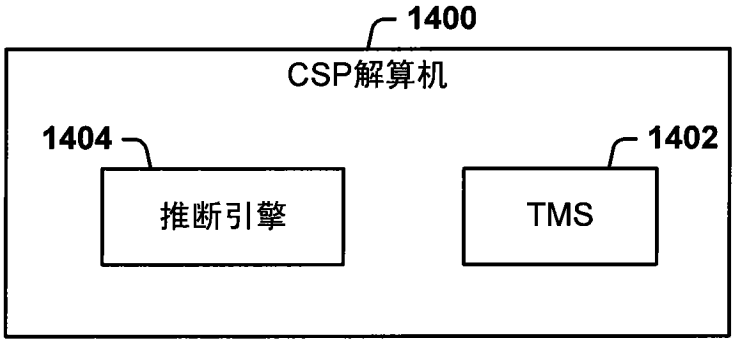


图 14

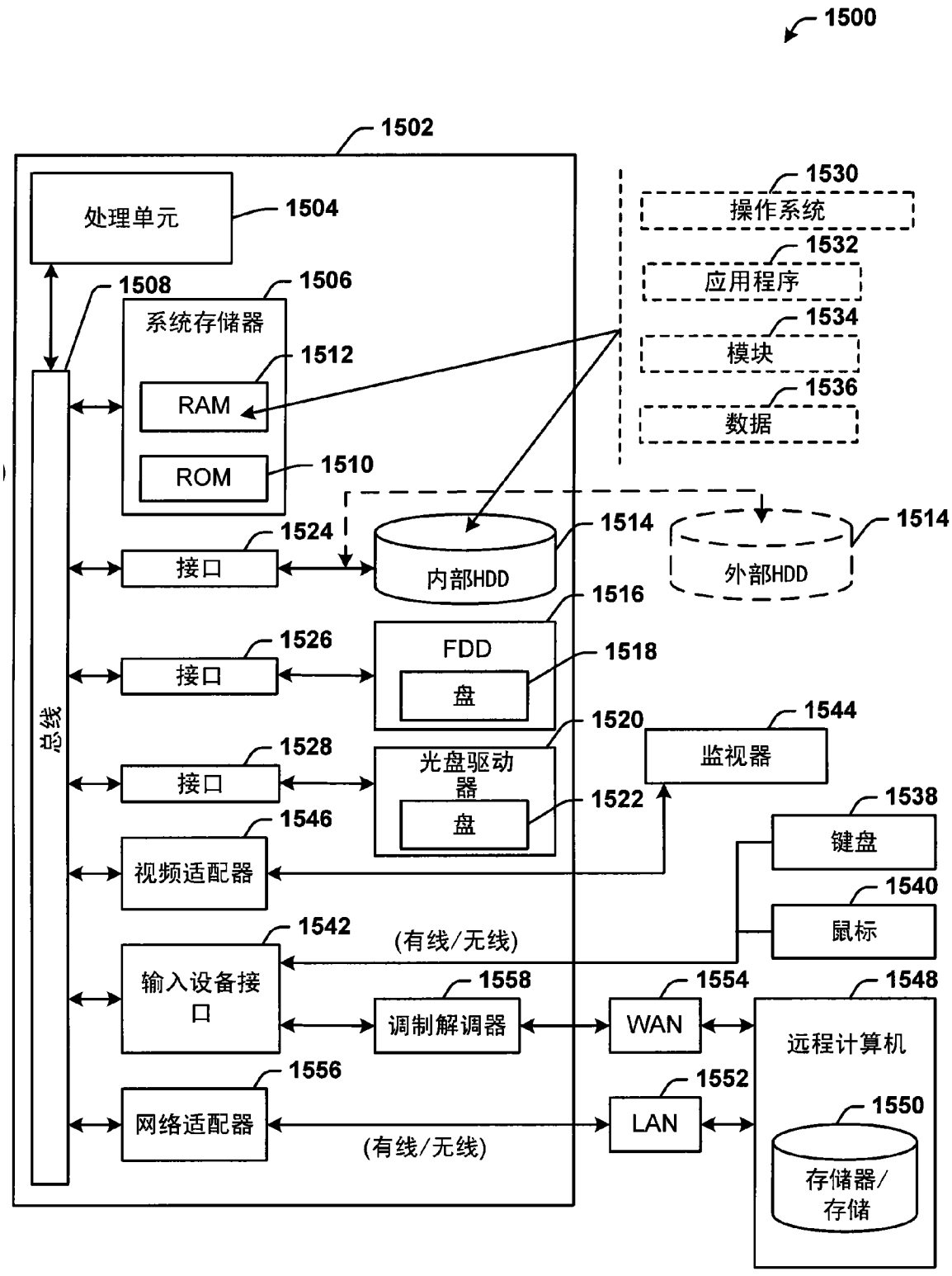

1500
1502
处理单元
1504
1508
1506
系统存储器
1512
RAM
ROM
1510
总线
1530
操作系统
1532
应用程序
1534
模块
1536
数据
1524
接口
内部HDD
1514
外部HDD
1514
1516
1526
接口
FDD
盘
1518
1528
接口
光盘驱动器
盘
1520
1522
1544
监视器
1538
键盘
1546
视频适配器
1540
鼠标
1542
(有线/无线)
输入设备接口
1558
调制解调器
1554
WAN
1548
远程计算机
1556
网络适配器
(有线/无线)
1552
LAN
1550
存储器/存储

图 15

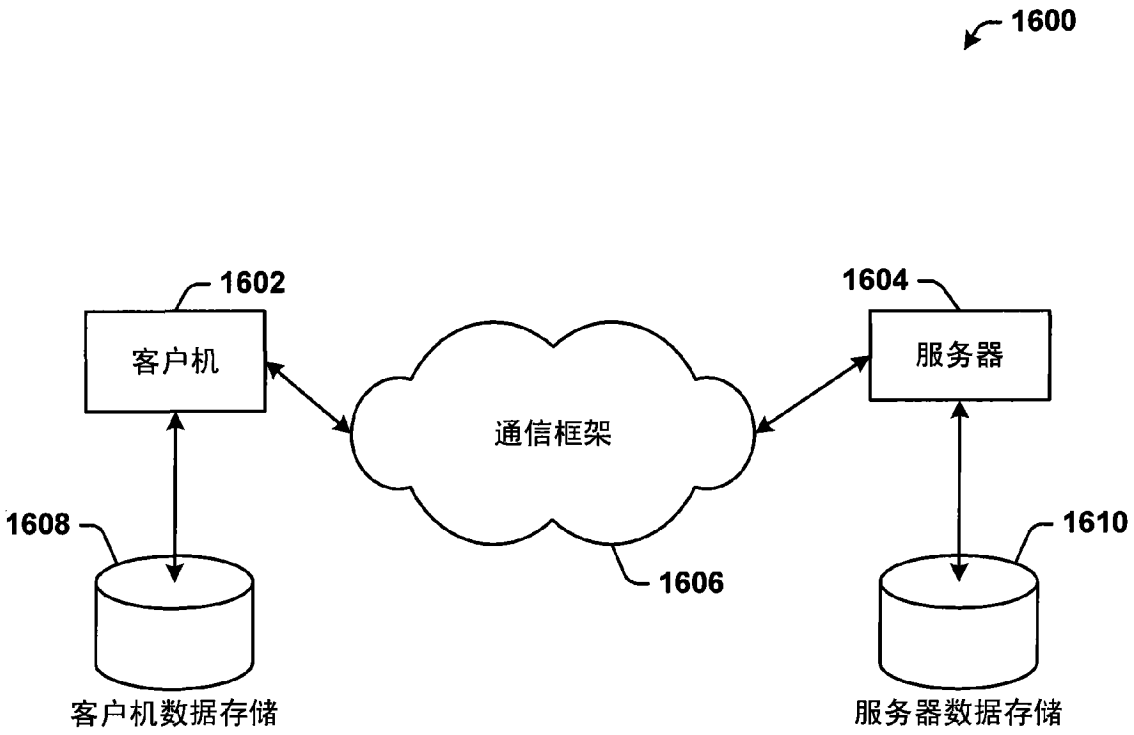


图 16