(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 103282877 B
(45)授权公告日 2017.03.29

(21)申请号 201180062500.2

(22)申请日 2011.12.06

(65)同一申请的已公布的文献号
申请公布号 CN 103282877 A

(43)申请公布日 2013.09.04

(30)优先权数据
12/978,557 2010.12.25 US

(85)PCT国际申请进入国家阶段日
2013.06.24

(86)PCT国际申请的申请数据
PCT/US2011/063466 2011.12.06

(87)PCT国际申请的公布数据
WO2012/087561 EN 2012.06.28

(73)专利权人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 D·J·萨格 R·萨桑卡 R·加伯 S·赖金 J·努兹曼 L·佩雷德 J·A·多莫 H-S·金 吴友峰 K·山田 T-F·奈 H·H·陈 J·鲍巴 J·J·库克 O·M·沙克 S·斯里尼瓦斯

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

(51)Int.Cl.
G06F 9/30(2006.01)
G06F 9/38(2006.01)

(56)对比文件
US 2010/274972 A1,2010.10.28,
US 6711667 B1,2004.03.23,
US 2010/0205599 A1,2010.08.12,
US 2007/0079281 A1,2007.04.05,
CN 1178941 A,1998.04.15,

审查员 邢白灵

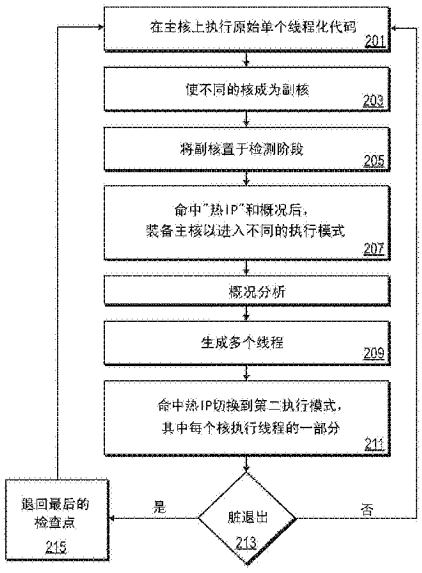
权利要求书1页 说明书45页 附图13页

(54)发明名称

用于将程序自动分解成多个并行线程的硬件和软件系统的系统、设备和方法

(57)摘要

描述了用于硬件或软件系统以将程序自动分解成多个并行线程的系统、设备和方法。在一些实施例中，系统和设备执行原始代码分解和/或所生成的线程执行的方法。



CN 103282877 B

1.一种执行代码的方法,包括:

在第一处理器核上执行原始源代码;

使用硬件包装器来:

i)将第二处理器核置于检测阶段,其中在所述检测阶段第二处理器核检测在第一处理器核上运行的软件的入口点,所述入口点指示切换到不同的与所述第一处理器核合作的执行模式,其中所述入口点和该软件的大部分动态执行对应,

i i)在所述第一处理器核中概况分析所述原始源代码,其中概况分析所述原始源代码包括针对设定的指令量生成关于负载、存储和分支的信息;

在所述第二处理器核中从所述原始源代码生成合作代码,从而由所述第一和第二处理器核合作地执行,其中所述合作代码是所述原始源代码的线程化版本,带有可能的入口点;

由所述第二处理器核检测所述入口点;以及

在所述第一和第二处理器核中执行所生成的合作代码,其中所述第一处理器核和第二处理器核支持两种逻辑处理器类型,并且其中所述逻辑处理器类型中的第一种是用于执行所生成的合作代码的劳工逻辑处理器。

用于将程序自动分解成多个并行线程的硬件和软件系统的系统、设备和方法

[0001] 优先权请求

[0002] 该部分继续申请要求题为“Systems,Methods,and Apparatuses for Parallel Computing(用于并行计算的系统、方法和设备)”的非临时专利申请S/N12/646,815的优先权，该申请S/N 12/646,815本身是部分继续申请并要求2009年11月24日提交的题为“System,Methods,and Apparatuses To Decompose A Sequential Program Into Multiple Threads,Execute Said Threads,and Reconstruct The Sequential Execution(将顺序程序分解成多个线程，执行所述线程并重构顺序执行的系统、方法和装置)”的非临时专利申请S/N 12/624,804的优先权，申请S/N 12/624,804要求2008年11月24日提交的题为“Method and Apparatus To Reconstruct Sequential Execution From A Decomposed Instruction Stream(从分解的指令流重构顺序执行的方法和设备)”的临时专利申请S/N 61/200,103的优先权。

发明领域

[0003] 本发明的实施例一般涉及信息处理领域，更具体地涉及在计算系统和微处理器中多线程执行的领域。

[0004] 背景

[0005] 在最近的几十年间，单线程处理器已经通过采用指令级并行性(ILP)显示出显著的性能改进。然而，这种类型的并行性有时难以利用并且需要复杂的硬件结构，这可导致过高的功耗和设计复杂性。此外，在复杂性和功率方面的这种增加提供逐渐减少的回报。芯片多处理器(CMP)已经呈现为有希望的可选方案，以便在合理的功率预算下提供进一步的处理器性能改进。

附图说明

[0006] 在附图各图中通过示例而不是限制说明了本发明，其中类似标记指示相似元件，且其中：

[0007] 图1图示动态线程切换执行架构操作的示例性实施例。

[0008] 图2示出根据一些实施例的DTSE操作的示例性方法。

[0009] 图3示出DTSE架构的实施例。

[0010] 图4示出根据一些实施例用于包装器(wrapper)的主要硬件块。

[0011] 图6示出DTSE硬件的实施例的更详细图示。

[0012] 图7示出根据一些实施例的XGC的使用。

[0013] 图8-11示出一些软件操作的示例。

[0014] 图12是示出根据本发明的实施例的核的示例性无序架构的框图。

[0015] 图13示出根据本发明一个实施例的系统的框图。

[0016] 图14示出根据本发明的实施例的第二系统的框图。

[0017] 图15示出根据本发明的实施例的第三系统的框图。

[0018] 详细描述

[0019] 在以下描述中，陈述了多个具体细节。然而，应当理解，本发明的实施例可在没有这些具体细节的情况下实践。在其他实例中，公知的电路、结构和技术未被详细示出，以免混淆对本描述的理解。

[0020] 在说明书中对“一个实施例”、“一实施例”、“示例实施例”等的参考指示所描述的实施例可包括特定特征、结构或特性，但并不一定每个实施例都包括该特定特征、结构或特性。此外，这样的短语不一定是指同一个实施例。此外，当参考实施例描述特定特征、结构或特性时，认为在本领域技术人员学识范围内，可以与其他实施例一起实施这样的特征、结构或特性，不论是否有明确描述。

[0021] 以下详细描述用于提供动态线程切换执行的系统、装置和方法的实施例。支持它的系统的实施例包括被硬件包装器和软件(动态线程切换软件)围绕的处理器核。在正常执行模式下，处理器如同硬件包装器和动态线程切换软件不存在那样地操作。例如，发生正常x86执行。在动态线程切换执行(DTSE)模式下，硬件包装器和软件一起工作以进行动态线程切换执行，包括流程的生成和修改等，如下详述。然而，首先提供高级概述。以下是诸如“流程”和“热”代码之类的一些主题的更详细解释。

[0022] 以下的细节对硬件包装器与软件的组合进行描述，该硬件包装器检查在全局提交前每个负载从正确的存储获得数据以及某些其它事项，该软件使用最近观察到的代码行为解析难以单独从代码确定的代码依赖性，因为硬件将检查并保证正确的执行。

[0023] 以下的细节描述在原始代码中对很多静态实例的标识，使得可以理解在原始指令的特定实例之间而非原始指令本身之间的依赖性。

[0024] 所描述的软件使用对限定的静态代码子集(称为“流程”)内依赖性的完整但可能不完全正确的理解，以在流程内将静态代码分成在跟踪之间没有依赖性(或仅有特别允许的依赖性)的分离的“跟踪”。

[0025] 另外，描述了包括以下中的一个或多个的维护操作：获取流程的附加概况，该流程由于意外的控制流程或不正确的负载结果或其它检查的失败，在退出前比期望的动态执行更短；删除在此方面突然变坏的流程；连续搜索对性能有意义的新或替换流程；以及当有新的概况信息以及对观察到不比原始代码表现更好的流程的删除时，将代码再分成跟踪。

[0026] DTSE硬件将形成顺序列表并将这些写入中级高速缓存。DTSE硬件具有寄存器，该寄存器具有与该跟踪元数据列表相关联的物理地址。我们可能需要考虑压缩负载和存储地址以使带宽保持下降。

[0027] 我们将需要能够跟踪执行负载或存储的逻辑处理器至负载执行以及至高级存储处理。

[0028] DTSE逻辑还获取退役时的分支方向位和所采用的间接分支的目标，包括返回。这用于闪速概况分析和支持寄存器状态恢复的数据。

[0029] I.高级概述

[0030] 如上所述，本文中详细描述的处理器可操作在传统模式中，其中每个核自己操作。当核之一“关闭”或“停止”时，处理器核还可用于在DTSE模式下概况分析、线程化和运行线程化的代码。虽然操作系统假设该核处于睡眠模式，但停止流程将核的所有权从OS切换到

DTSE模式。

[0031] 图1图示DTSE操作的示例性实施例。在图中，这控制成为副核的空闲核。在一些实施例中，当核接收到HALT/WMAIT指令时，发生对空闲核的控制。DTSE软件监视将被置于睡眠的核的所有线程，然后承担对该核的控制。

[0032] 副代码通过检测“热代码”开始，该“热代码”是与大部分动态执行相对应的主核(例如，运行100K连续指令的内核)上运行的软件的入口点。热代码检测通常在简单的硬件结构中完成。一旦已经检测到热入口点，将其装备用于概况分析。一旦(被主核)命中，热代码进入概况分析模式，这表示针对某些设定的指令量(诸如50000个指令)概况分析所有的负载、存储和分支。一旦完成概况分析，执行线程生成。线程生成分析该热代码，并且基于概况分析的信息生成两个线程用于线程化执行模式。一旦完成，则将入口点(热IP)装备作为DTSE模式入口点。下次原始代码(运行在主核上)命中入口点时，它切换点到其中两个核合作执行该代码的DTSE模式。如果流程退出热代码或如果检测到违背或外部事件(例如，中断)，则DTSE模式结束。违背包括例如，当两个核在同一时期将不同数据存储到相同的存储器地址时。一旦出现违背，线程化模式退回到最后的提交点并且回到原始代码。

[0033] 图2示出根据一些实施例的DTSE操作的示例性方法。在该实例中，利用两个核处理线程化代码。在201，主核(核0)执行原始单个线程化代码。

[0034] 在203，另一个核(核1)变成并用作副核。核能够以多种方式变成副核(工作线程)。例如，通过使用硬件动态方案，诸如抓取由OS置于睡眠(例如置于C状态)、由软件分配或由线程(由OS/驱动器或应用本身)置于睡眠的核，作为核的静态划分的结果，可将副核用作副核。

[0035] 当主核执行原始代码时，在205，副核将被置于检测阶段，其中它等待热区的热代码检测(通过硬件或软件)。在一些实施例中，热代码检测是检测频繁访问的热区并提供其入口IP(指令指针)的硬件表。一旦检测到这种热区入口IP，在207，主核被装备，使得它将触发对该IP的下一次调用的概况分析并将执行切换到原始代码的线程化版本。概况分析收集信息，诸如负载地址、存储地址和预定长度执行(例如，50000动态指令)的分支。

[0036] 一旦已经完成概况分析，则副核开始线程生成阶段(thread-gen)209。在该阶段，副核生成经概况分析的区的线程化版本，同时使用经概况分析的信息作为指导。线程生成提供原始代码的线程化版本以及可能的入口点。当在211，命中入口点之一(标记为“热IP”)时，主核和副核被重定向成执行代码的线程化版本，并且执行切换到不同的执行模式(有时称为“线程化执行模式”)。在该模式下，两个线程完全分开地操作，同时包装器硬件用于缓存存储器负载并且存储、检查可能的违背，并自动提交状态以提供向前发展同时维持存储器排序。

[0037] 该执行模式可按两种方式之一结束。它可在代码退出热区作为干净退出(执行没有问题)时或在发生违背作为脏退出时结束。在213，确定是哪种类型的退出。示例性的脏退出是存储/存储和负载/存储违背或在第二执行模式中未处理的例外情况(例如被零除的浮点例外、不可高速缓存的存储器类型存储等)。在第二执行模式退出时，主核返回到原始代码，而副核返回到检测模式，等待检测另一个热IP或命中已经生成的区的热IP。在干净退出(热区的退出)时，原始代码从退出点继续。在脏退出(例如，违背或例外)时，在215，主核返回到最后的检查点，并且继续那里的执行。在干净和脏退出时，寄存器状态从两个核合并并

移动到原始代码。

[0038] 示例性DTSE架构

[0039] 图3示出动态线程切换执行架构的实施例。以下详细讨论该系统的软件和硬件方面。如上所述，每个核301可先天地支持同时多线程(SMT)。这表示两个或更多个逻辑处理器可共享核的硬件。每个逻辑处理器独立地处理代码流，然而来自这些代码流的指令被随机混合以在同一硬件上执行。经常，来自不同逻辑处理器的指令在核的超标量硬件上同时执行。SMT核的性能和同一核上的逻辑处理器的数量增加。因此，由于逻辑处理器的数量增加，一些重要的工作负载将被更快速地处理。由于仅逻辑处理器的数量增加，其它工作负载可能不被更快速地处理。

[0040] 有的时候，系统中没有足够的软件线程利用所有的逻辑处理器。该系统自动分解可用软件线程中的一些或全部，将每个分解成将同时执行的多个线程(从单个线程至多个线程的动态线程切换)，从而利用多个(可能很多)逻辑处理器。由于仅逻辑处理器的数量增加，而未被更快速地处理的工作负载在其线程已经被分解成大量线程以便利用更多逻辑处理器时很可能会被更快速地处理。

[0041] 除“传统”核外，硬件包括动态线程切换逻辑，其包括用于维持全局存储器一致性303、全局退役307、全局寄存器状态309和软件的信息收集305的逻辑。该逻辑可被包括在核本身中或被单独提供。该逻辑可执行五种功能。第一种是收集关于运行代码的专用信息，称为概况分析。第二种是在原始代码运行时，硬件必须查看软件已经定义的执行命中热IP流地址。当这发生时，硬件强制核跳至软件已经定义的不同地址。这就是代码的线程化版本如何得到执行的。第三种是硬件必须与软件一起工作以时常有效地保存原始代码流的正确寄存器状态作为核的数据高速缓存内的全局提交点(Global Commit Points)。如果原始代码流被软件分解成多个线程，则可能没有逻辑处理器曾经具有原始程序的整个正确寄存器状态。伴随每个全局提交点的正确存储器状态也应是已知的。当必要时，与软件一起工作的硬件必须能够将寄存器和存储器的架构程序状态恢复到最后的全局提交点，如下所讨论的。第四，尽管软件在产生正确执行的代码方面表现良好，但有些事情软件不能总是100％正确。一个好的例子是：软件在生成代码的线程化版本时不能完美预测存储器地址。所以线程化代码有时获取负载的错误结果。硬件必须检查可能不正确的所有东西。如果一些东西不正确，则硬件必须与软件一起工作以修复程序状态。这通常通过将核状态恢复到最后的全局提交状态来完成。最后，如果原始代码流被分解成多个线程，则在原始代码中指定的对存储器的存储将被分布在多个逻辑处理器中并在这些逻辑处理器之间按随机顺序执行。动态线程切换逻辑必须确保任何其它代码流不能“查看”存储器中在正确执行的情况下按原始代码定义是不正确的状态。

[0042] 在这些示例性架构中，处理器核(“核”)通常具有两个SMT线程。一般而言，在DTSE模式中，每个核有比2个多得多的“逻辑处理器”。这些逻辑处理器被定义为两种类型之一：“主的”和“劳工的”。主逻辑处理器对软件(即操作系统和虚拟机监视器)是已知的。软件将核上的主逻辑处理器集看作它们实际所在的核上的SMT线程。主逻辑处理器完成SMT线程完成的大部分事情(如果不是全部事情的话)，尤其是获取中断。

[0043] 劳工逻辑处理器一般是外部软件不可见的。它们由“保留的存储器中”的DTSE软件使用并管理。该存储器是可物理寻址的、保留存储器系统。该保留存储器是现有系统存储器

的专用部分或单独的物理存储器。该保留存储器用于存储DTSE代码并且用作其工作存储器。在DTSE模式中生成的数据结构被存储在保留存储器中。“热”代码也被存储在该保留存储器中。跟踪数据、跟踪数据目录和跟踪元数据在保留存储器中。所有的这些都被物理寻址。

[0044] 劳工逻辑处理器不具有主逻辑处理器所具有的全部语境，结果它们被限于全部处理器功能的子集。劳工逻辑处理器仅执行保留存储器中流程的代码，并且将不执行可见存储器中的原始代码。

[0045] 主逻辑处理器可执行可见存储器中的原始代码或隐藏存储器中流程的代码。从DTSE角度看，主逻辑处理器可执行的隐藏存储器中流程的代码是单个跟踪代码。

[0046] 在操作中，操作系统在主逻辑处理器上运行线程，如同它现在所做的。主逻辑处理器全相同且被编组成“核”。有多个主逻辑处理器且每个的性能都非常高。主逻辑处理器的性能乘以操作系统可用的主逻辑处理器的数量远远超过芯片的吞吐能力。

[0047] 可能利用一部分所提供的主逻辑处理器来利用芯片的全部能力。另一方面，通过利用大量(优选地全部)所提供的主逻辑处理器，良好编码的大量并行程序可更高效地利用芯片的能力，因此实现更高性能。

[0048] 如果所运行的主逻辑处理器不利用芯片必须提供的所有资源，则DTSE软件将进行自动线程分解。它将在劳工逻辑处理器上运行并行化计算流程。这尤其包括重要SSE和AVX代码的分解以并行运行在吞吐引擎上。这包括在特殊的功能单元上运行能在这些特殊的功能单元上运行的代码。

[0049] 基本上，当主逻辑处理器上的线程实际在劳工逻辑处理器上运行计算流程时，主逻辑处理器不做任何事情。它实际上停止，但操作系统认为它做了所有的工作。如上详述，操作系统不知道劳工逻辑处理器。当计算流程运行在劳工上时，拥有它的主逻辑处理器等待两种事情之一发生：中断或计算流程中的流程退出。如果这些事情中的任一个发生，则主逻辑处理器接管。如果有中断进入中断流程，则它立即与劳工上的计算流程并行地执行该中断流程。在大多数情况下，中断流程完成整个中断处理，且主逻辑处理器返回等待，而对计算流程没有任何影响。当在其计算流程中有流程退出时，等待的主逻辑处理器在该点上构造寄存器状态，并且在原始代码中恢复。理想地，它将很快命中热代码入口点。这将开始新计算流程的执行，或者可能是主逻辑处理器上的系统流程。

[0050] 除以下详述的其它任务外，DTSE软件311可包括用于生成、维持和去除流程的例程。

[0051] 硬件“包装器”

[0052] 在一些实施例中，硬件包装器用于动态线程切换执行逻辑。包装器硬件支持以下功能中的一个或多个：1)检测热区(热代码根检测)；2)生成表征热区的信息(概况)；3)当执行事务时缓冲状态；4)在成功的情况下提交所缓冲的状态；5)在中止的情况下丢弃所缓冲的状态；6)检测一致性事件，诸如写-写和读-写冲突；7)预防交叉修改代码；和/或7)预防分页相关改变。以下将详细讨论这些功能中的每一个或者已经讨论过了。

[0053] 尽管在DTSE执行模式中，所生成的线程一起操作，但没有通信的途经。每个核执行其自身的线程，同时提交/跨度标记指示线程化代码中的与原始代码中的一些IP相对应的IP。DTSE硬件缓冲负载和存储信息，防止任何存储对外部可见(全局提交)。当两个核达到跨

度标记时，检查负载和存储是否有违背。如果没有检测到违背，则存储变为全局提交。存储的提交指示在违背/脏退出的情况下执行应跳至的检查点。

[0054] 图4示出根据一些实施例用于包装器(wrapper)的主要硬件块。如上所述，这由两个或更多个核401(示为属于一对，但可以更多)构成。违背检测、原子提交、热IP检测和概况分析逻辑403耦合到核401。在一些实施例中，该组被称为动态线程切换执行硬件逻辑。还耦合到核的是中级高速缓存405，其中合并第二执行模式的执行状态。另外，有末级高速缓存407。最后，有xMC保护数据高速缓存(guardata cache，XGC 409)，将参考图7对其进行详细讨论。

[0055] 应用的若干流程将由根检测硬件(未明确示出)中的硬件标识。每个流是具有一个或多个入口点的良好定义的二进制代码集。应用的流程集应覆盖该应用的大量动态执行，但由于流的并行化，静态流程的组合尺寸应该小。

[0056] 当原始代码被执行时，根检测硬件搜索单个IP值，称为标识新流程的“根”。通常，在执行处于流程中时该硬件将不进行搜索。在一些实施例中，硬件将保持64指令指针(IP)的列表。该列表从位置0排序到位置63，且每个位置可具有IP或为空。该列表开始为全空。

[0057] 如果有合格的分支到在位置N匹配列表中条目的IP，则位置N-1和N交换位置，除非N＝0。如果N＝0，则什么都不发生。更简单地，该IP在该列表中向上移动一个位置。

[0058] 如果有合格的分支到不匹配列表中条目的IP，则条目40至62下移1，至位置41至63。位置63的先前内容丢失。在位置40输入新IP。

[0059] 在一些实施例中，对哪些IP“合格”增加到列表、或“合格”匹配已经在列表上的IP因此导致提升存在限制。第一限制是仅所采用的向后分支的目标是合格的。调用和返回是不合格的。如果所采用的向后分支执行“热”作为流程的一部分并且它不离开该流程，则它的目标是不合格的。如果所采用的向后分支的目标命中热代码入口点高速缓存，则它是不合格的。基本上，已经在流程中的IP不应被置于列表中。

[0060] 在一些实施例中，存在软件可设置的两个“排除”区域。每个区域由该排除区域的IP上的下界和上界描述。注意该机构可被设置成仅接受特定区域中的IP。第二限制是排除区中的IP不合格进入列表。

[0061] 在一些实施例中，在命中列表中的指令之后没有小于16384动态指令的指令合格来添加，然而，可允许用16384动态指令窗内的新IP替换列表中的最后IP命中。基本上，流程的目标是动态地对最少50000个指令取平均。列表中的IP是这一流程的可能根。因此，接下来的16000动态指令被视为列表中已经存在的流程的一部分。

[0062] 在一些实施例中，硬件将栈16保持较深。调用可以循环地递增栈指针，返回可以递减栈指针，但它不包装。即，在调用时，栈指针总是递增。但有推入深度计数器。它不能超过16。如果返回将使推入深度变负，则返回不会递减栈指针和推入深度计数器。每个指令递增栈中的所有位置。在推入时，新的栈顶被清除。栈位置在最大计数64K处饱和。因此，另一个限制是没有IP合格来添加到列表，除非栈顶饱和。其原因是避免错误环。假设存在一个过程，其包含总是迭代两次的环。从全部代码调用该过程。然后该过程中的向后分支经常被命中。尽管这看上去很热，但它是来自所有位置的逻辑不相关工作，且不会导致良好的流程。该过程中调用这个过程的IP是期望的。外部过程是优选的，但内部过程不是，除非内部过程大到足以包含流程。

[0063] 在一些实施例中，如果IP，I被添加到列表或被提升(由于命中匹配)，则在接下来的1024动态指令内没有指令合格来匹配I。该规则的目的是防止过高评价紧密环。这种环中的向后分支命中很多，但每次命中不表示很多工作。

[0064] 列表中的顶部IP被视为表示非常活跃的代码。

[0065] 典型的过载负载将具有若干流程来获得高动态覆盖。完全按重要性的顺序找到这些不是关键的，但较佳的是通常大致按重要性的顺序产生这些流程以便较早获得最大性能增益。应找到用于构建流程的合理位置。这将成为热代码，然后它出界以找到下一流程用于工作。很可能，将找到若干流程。

[0066] 一般而言，流程并不脱节且可重叠。但是，至少每个流程的根不在先前找到的流程中。它实际仍可在稍后找到的流程中。这足以保证没有两个流程是相同的。尽管上面已经使用的特定数字，但那些仅仅是说明性的。

[0067] 这种架构至少有两类流程：系统流程和计算流程。计算流程在劳工逻辑处理器的集群上执行。系统流程在主逻辑处理器上执行。

[0068] 计算流程可以是但不必需是通过线程分解从单个代码流形成的多个跟踪。计算流程完全由单个集群中的劳工逻辑处理器执行，因为整个流程必须在被检查后按程序顺序全局地提交(但通常被全局提交为提交跨度的序列而不是单片。)执行流程的所有劳工可自由加载并存储到相同的地址。在不能做什么方面，计算流程是限制最多的。

[0069] 系统流程也是DTSE软件产生的代码，并且也驻留在隐藏的存储器中。在系统流程中有很多不能做的事情。它被限制，但比计算流程限制得要少很多。系统流程是计算流程之间的串行代码或者是对计算流程独立的并行线程。

[0070] 系统流程的最重要用途之一是中断处理。有时，DTSE软件创建流程，其入口点是最受欢迎的中断向量目标。该代码将覆盖来自这些向量目标的最受欢迎的路径。如果中断在系统流程中被完全处理，从向量目标至中断返回(IRET)回到中断代码而无需系统流程退出，则不需要打扰执行“被中断”的流程的劳工。如果整个中断处理序列不能如此被处理，则必须在执行“被中断”的流程的劳工中强制流程退出。上下文切换将总是导致中断流程退出，因此导致“被中断”的计算流程中的流程退出。

[0071] 如果中断可被保持在该中断流程中，则寄存器永远不保存。中断系统流程是单个全局提交跨度。或者它将完成至IRET并且所有的提交作为一单元，或者状态将被恢复到中断向量的目标且执行在原始代码中从这里重新开始。状态恢复包括停止计算流程中的全局提交(和进一步的执行)，并且从计算流程中的最后全局提交点恢复寄存器状态。

[0072] 隐藏存储器中的流程中的代码可以被超快速且高效地线程切换，因为线程切换被正确地编程为代码。原始代码不能具有被编程的线程切换。涉及原始代码的线程切换缓慢且低效。

[0073] 针对能做什么，限制计算流程。一些原始代码不合格来进入计算流程。对计算流程合格的一些代码可进入限制较少的系统流程。然而，仍有一些代码甚至不合格来进入系统流程。这必须保留原始代码。

[0074] 在一些实施例中包括无限量的原始代码(如果需要)与两类流程复用。

[0075] 一旦检测到热代码根IP(入口IP)，装备主核，使得在该IP的下一命中时，该核将开始概况分析原始代码。在概况分析时，以上信息(分支、负载和存储)按程序动态顺序存储到

存储器中的缓冲器。缓冲器稍后由线程生成软件使用以指导线程生成—消除不使用的代码(基于分支)、指导优化并检测负载/存储关系。在一些实施例中,将用于冲突检查的同一硬件(以下描述)用于缓冲负载、存储来自退役的分支信息并使其溢出到存储器。在其它实施例中,将微操作插入执行的程序,这将所需信息直接存储在存储器中。

[0076] 为了表征热区(概况分析),线程化执行模式软件需要以下信息中的一个或多个:1)对于分支,它需要a)对于条件分支,采用/未采用信息,以及b)对于间接分支,分支目标;2)对于负载,a)负载地址和b)存取尺寸;3)对于存储,a)存储地址和b)存储尺寸。

[0077] 在一些实施例中,排序缓冲器(OB)将被维持用于概况分析。这是因为负载、存储和分支无序地执行,但概况分析数据需要按顺序。OB在尺寸上类似于重排序缓冲器(ROB)。负载在分配时将其地址和尺寸写入OB。存储在STA(存储地址)分配期间将进行相同的动作(STA分配早于存储退役,该分配的目的是将虚拟存储地址转换成物理地址)。分支将写"至"字段,它可用于直接和间接分支。当这些负载、存储和分支从ROB退役时,将从OB复制其对应信息。热代码概况分析使用包装器硬件可缓冲事务状态并且稍后提交它的事实。它将使用提交所缓冲的状态的相同数据路径,以从OB向写组合高速缓存(稍后描述)复制数据,然后提交它。概况分析信息将被写入特殊存储器位置中的专用缓冲器,以便稍后由线程化执行软件使用。

[0078] 在一些实施例中,DTSE软件将IP写在寄存器中并装备它。一旦命中该IP,硬件将获取概况数据并将其写入存储器中的缓冲器。在流程根IP在执行期间由硬件报告后,遇到某一数量(例如,10000)分支的分支方向历史。该列表是按本地退役顺序的每分支一位。动态线程切换执行软件在退役时获得所采用分支的目标。它报告分支方向流中嵌入的间接分支的目标。同时,硬件将报告负载和存储的地址和尺寸。

[0079] 当应用随时间改变其行为时,只要应用执行,就监视该执行。当流看起来似乎变得"太"短时,它们应被生长或删除。另外,可在任何给定时间找到、再生成或合并新的流程。在一些实施例中,动态线程切换执行系统的硬件将报告每个流程的平均全局提交指令和循环。如果有任何问题,软件将需要考虑它并且有时也通过临时禁用流程来获得原始代码的数据。在多数实例中,软件不运行"热"代码,除非显然这是一个净赢。如果不清楚"热"代码是净赢,则软件应禁用它。这可逐个流程地进行,或者软件可针对该工作负载关闭所有的东西。

[0080] 软件将继续接收分支未命中预测数据和分支方向数据。另外,由于全局队列的段为满或等待流程覆盖,软件将获得关于线程停止的报告。这些可指示遥遥领先运行的负载下跟踪(稍后讨论)。它还将获得高速缓存未命中的核停止时间。例如,获得很多高速缓存未命中停止时间的核A可解释为什么核B遥遥领先地运行。所有这些可用于针对该流程更好地进行跟踪的负载平衡。硬件还将报告具有最高高速缓存未命中率的负载的全标识。这可帮助软件重新分布高速缓存未命中。

[0081] 在一些实施例中,软件将在每个流程执行中获得循环或指令的报告。这将标识太小的流程,因此具有过度的覆盖开销。

[0082] 图5示出根据实施例的跨度执行。当线程化执行软件生成用于热代码的线程时,它试图以尽可能小的复制来这样做。根据原始静态代码,创建两个或更多线程。这些线程被跨越。生成对原始代码的跨度标记同步,并且在跨度标记边界处检查违背(诸如以上描述的那

些)。可在每个跨度完成时提交存储器状态。如图所示,在命中热IP时,执行模式切换到线程化执行。与先前的一般图示不同的是每个线程具有跨度。在每个跨度之后,进行检查(chk)。在示例中,在第二跨度执行之后,检查(chk2)找到违背。由于该违背,代码退回到最后的检查点(它可以在线程之后或在热IP命中之前)。

[0083] 如上所述,当热代码区退出(干净退出)或违背情况(脏退出)时,线程化执行模式将退出。在干净退出时,退出点将指示跨度和提交点,以便提交所有的存储。在干净和脏退出时,原始代码将进入最后检查点(提交)的相应原始IP。寄存器状态需要从两个核的状态合并。为此,线程生成器将需要在每次提交时更新寄存器检查点信息。例如这可通过插入特殊存储来完成,该特殊存储将来自每个核的相关寄存器存储在硬件缓冲器或存储器中。在退出时,寄存器状态将从两个核合并入原始(主)核。应注意,寄存器合并存在其它可选退出方案,例如寄存器状态可从缓冲的负载和存储信息(如在生成时由线程生成器确定)检索。

[0084] 在图6中示出DTSE硬件实施例的更详细图示。它在左侧描述推测执行在右侧描述一致性。在一些实施例中,除MLC 617和核601外的所有形成违背检测、原子提交、热IP检测和概况分析逻辑403的一部分。该执行是推测的,因为在线程化模式中,核601中所生成的线程一起操作,它们彼此不通信。

[0085] 在一些实施例中,硬件包装器包括单个物理保护高速缓存,其保护原始代码,在保留的存储器(又一次未示出)中存在该原始代码的某种形式的副本。在一些实施例中,这是16K线、8路集合相关高速缓存,其中每个线覆盖原始代码的4096字节。分辨率将是64字节块。高速缓存线包含在与线范围对齐的4096字节内的开始块和结束块。块号是6位。该高速缓存可具有在具有相同标签的相同集合中的多条线。因此,当读取物理保护高速缓存时,相同集合中的多条线可被命中高达8条线。这提供了在相同的4096字节中的表示若干分离跨度(其间具有未保护的间隙)的能力。

[0086] 软件311应确保在物理保护高速缓存中不会一次以上地表示64字节块。当进行对命中64字节块的检查时,还检查块是否位于4096字节粒度地址命中的高达8条线中。因为有16K线,所以它们以14位线号编号。64字节块将未命中该高速缓存或命中独特的14位线号。

[0087] 在任何使无效监听中查找物理保护高速缓存。命中后,就检索该命中的14位线号,任选地提供被命中的64字节块的6位号。

[0088] 对于每个流程,软件将确保在其处理代码前其原始代码被覆盖在物理保护高速缓存中。这可涉及扩展已经在高速缓存中的线的跨度或增加新线。

[0089] 软件将获得覆盖该流程所需的线号的列表。例如,软件将该列表减小至单个28位号。线号是14位。我们将2位用于线号中的每一位,有效位和数据位,因此28位用于14位线号。在每个位位置,如果整个列表在该位是什么上达成一致,它变为所得的数据位且有效位变为1。如果列表中对于位位置中的值有不同意见,则所得的数据位变为0且有效位变为0。这28位是该流程的签名。

[0090] 每个核在其热代码入口点高速缓存(未示出)中具有对所有流程的签名。并且,具体地,它具有当前暂时地全局提交流程的一个签名。在保留的存储器中有热代码入口点表,然而在一些实施例该表的部分被高速缓存。

[0091] 如果采用的分支被预测,则在最宽范围的高速缓存中查找其目标。如果它未命中,则请求重新填充。如果它命中且没有入口点,则那是它的结束。如果有入口点,则查找下一

较小范围的高速缓存。最后，它或者命中入口点、不命中入口点，或者不确定但取得对高速缓存的重新填充。

[0092] 如果采用的分支目标在保留的存储器中，则没有热代码入口点高速缓存查找。

[0093] 我们获得所采用分支的标识。如果我们命中入口点，则当该分支退役但仍被采用时，我们强制跳跃至所指示的目标且劳工也强制跳跃至所指示的入口点。

[0094] 检查每个临时全局提交流程。如果有线匹配，则将有流程退出。否则可允许它提交。这继续，直到在热代码入口点高速缓存被清洗之前进入的所有流程完成。一旦该检查处于适当位置，则全局提交可重新开始。

[0095] 必须检查热代码入口点高速缓存中的所有签名。如果有线匹配，则入口点被收回认证。当认证代码被调用时，它将查看标志。这表示在它能认证任何入口点之前它必须处理保留的存储器代理(未示出)中的工作列表。它读取被命中的线号。它知道每个线号的准确跨度。它准确知道每个流程所需的高速缓存线。它删除或者检查已经被命中的每个流程。它更新其整个热代码入口点表。然而它可进行其认证工作。

[0096] 每个核601执行其自身的线程，而跨度标记指示线程化代码中与原始代码中相同IP对应的位置(IP)(在前面的图中示出跨度标记)。当在该模式中执行时，硬件缓冲负载并存储信息，防止任何存储对外部可见(全局提交)。该信息可被存储在多个高速缓存中。这些被示为推测负载数据高速缓存(SLC)603、负载存储排序缓冲器(LSOB)605和推测存储高速缓存(SSC)607。尽管这些被示为分离的，但在一些实施例中它们是单个实体的多个部分。另外，如所示，可以不按核为基础配置该存储。

[0097] 在一些实施例中，在捕捉该数据之前，它穿过排序缓冲器(OB)602，该排序缓冲器维持如前详述的所执行的负载和存储的适当退役顺序。当高级(较旧)存储被写入数据高速缓存且被置于SLC 603中时，存储的物理地址和尺寸对于DTSE逻辑可用。SLC 603还检测使无效监听。类似地，使负载地址和尺寸对于DTSE逻辑可用且被存储在SLC中。还将负载和存储写入负载存储排序缓冲器(LSOB)605，以保持每个线程中负载和存储之间的排序信息。LSOB 605维持数据和掩码(有效字节)。

[0098] 最终DTSE逻辑获得按适当顺序的退役负载地址连同其存储地址。在一些实施例中，DTSE逻辑在执行时获得负载地址和尺寸。如果是这种情况，则DTSE逻辑将使它们老化(age)并过滤它们以获得适当排序的退役负载地址和尺寸。

[0099] 数据高速缓存

[0100] 当两个核达到跨度标记时，利用违背检查组件(如下详述的存储正确性高速缓存(store correctness cache，SCC)611和负载正确性高速缓存(load correctness cache LCC)613)在负载和存储中检查违背。典型地，每个劳工有SCC 611和LCC 613。对于其劳工，用负载和存储地址写入SCC 611。利用所有其它合作劳工的存储地址读取它。它示出在单个对齐跨度中由代码写入的字节。典型地这种情况是独立的物理结构。利用来自LSOB 605的所有劳工的存储地址写入LCC 613。利用其劳工的负载地址读取它。典型地，这是保留存储器中的数据结构。如果没有检测到违背，则存储变为全局提交。存储的提交指示在随后跨度中的违背的情况下执行应跳至的检查点。

[0101] 存在可能发生的若干违背。第一是来自外部实体(例如另一个核)的使无效监听，它使诸核之一使用的数据无效。因为一些值是假设的(推测执行)，这可能是错误的，所以执

行不得不中止并且原始代码将返回最后的检查点。当在不同线程上的两个存储写入同一跨度中的同一地址时，存储/存储违背可能发生。在一些实施例中，因为在单个跨度中的不同线程之间没有排序，所以无法知道哪个存储在原始程序顺序中较晚，且线程化执行模式中止并返回到原始执行模式。如果在不同线程中的存储和负载在同一跨度中使用存储器中的同一地址，则存储/负载违背可能发生。因为线程之间没有通信，所以负载可能丢失通过存储而存储的数据。应注意，通常不允许负载命中在任何过去的跨度中由其它核存储的数据。这是因为核独立地执行，且在其它核到达存储之前负载可能已经执行(一个核可能提前其它核很多跨度)。可能发生自修改代码或交叉修改代码，其中原始代码已经由程序中的存储或一些其它代理(例如核)修改。在这种情况下，线程化代码可变得陈旧。由于性能优化和架构折衷而可能发生其它违背。这种违背的示例是L1数据高速缓存单元未命中，其命中丢弃的推测存储(如果这不被硬件支持)。另一个示例是线程生成器做出假设，而稍后被检测是错误的(断言硬件块609)。

[0102] 一旦保证没有发生违背，则缓冲的存储可被提交并全局可见。这自动发生，否则存储排序可能被损坏(存储排序是处理器必须遵守的存储器排序架构的一部分)。

[0103] 当执行线程化模式时，所有的存储将不使用“规则”数据路径，但将两者写入(执行存储的核的)第一级高速缓存，它将充当私有、非一致暂存器(scratchpad)，并且写入专用的数据存储。数据存储(以上的高速缓存和缓冲器)中的信息将包括存储的地址、数据和数据尺寸/掩码。允许存储组合，而存储来自同一提交区。

[0104] 当硬件决定提交状态(在违背已经被检查到)时，所有的存储需要从数据存储(例如，SSC 607)排出并且变为一致的可监听状态。这通过将来自数据存储的存储移至写组合高速缓存(WCC)615来完成。在数据复制监听其间，使无效将被发送到所有其它的一致性代理，所以存储将获取它们改变的高速缓存线的所有权。

[0105] 写组合高速缓存615组合来自工作在同一优化区中的不同代理(核和线程)的存储并且使这些存储变为全局可见状态。一旦来自所有核的所有存储被组合入WCC 615，它变得可监听。这将原子提交(维持存储器排序规则)提供给中级高速缓存217。

[0106] 通过清除数据存储中的一些“有效”位而使缓冲状态在中止中被丢弃，从而去除所有经缓冲的状态。

[0107] 由于原始程序被分成两个或更多个并发线程的事实，可使用一致性检查。如果软件优化器没有正确地对负载和存储消除歧意，则可能发生错误结果。以下的硬件构建块用于检查读-写和写-写冲突。负载-正确性-高速缓存(LCC)613保持在优化区中执行的负载的地址和数据尺寸(或掩码)。它用于确保来自另一个逻辑核的存储不会与来自经优化区的负载抵触。在跨度违背检查时，每个核将其存储写入其它核的LCC 613(设置该核写入的每个字节的有效位)。LCC 613然后保持其它核的存储地址。然后每个核通过以下动作检查其自身的负载：在其LSOB(负载存储排序缓冲器)605上迭代，复位其存储写入的每个字节的有效位以及检查其未命中的每个负载，该字节具有设置为1的有效位(表示该字节由其它核写入)。命中有效位1的负载被表示为违背。存储-正确性-高速缓存(SCC)611保持在优化区中执行的存储的地址和掩码。来自该高速缓存的信息与合作逻辑核的LSOB 605中的条目进行比较，以确保没有冲突不被检测到。在跨度违背检查时，SCC 611被复位。每个核将其存储从其LSOB 605写入其它核的SCC 611。然后相对于已经在其SCC 611中的其它核的存储，每个

核检查其存储(来自LSOB)。如果存储命中来自其它的存储,则检测到违背。应注意一些存储可通过线程生成器复制。这些存储必须由SCC 611正确处理以防止错误违背检测。另外,推测-负载-高速缓存(SLC)603保卫来自优化区的负载,以防御来自逻辑核的监听无效,这些逻辑核在线程化执行方案描述下不合作,但可并发运行同一应用的其它线程或访问共享数据。在一些实施例中,本文描述的线程化执行方案实现"全有或全无"策略,且优化区中的所有存储器事务应被看作好像全部在单个时间点(提交时间)一起执行。

[0108] 在DTSE执行模式中,每个核的L1数据高速缓存作为暂存区工作。存储不应对监听做出响应(以防止任何存储全局可见),且推测数据(未检查/提交的数据)不应被写回中级高速缓存。在退出该模式时,应从数据高速缓存中丢弃可能已经退回的所有推测数据。注意,由于一些实现折衷,可能需要使所有存储或加载的数据无效,该数据在线程化模式中已经执行。在数据高速缓存中,所有全局提交点的原始程序数据块和跟踪数据块均存储在相同的位置。它们均通过块的原始程序地址来寻址。

[0109] 重要的是注意上述示例易于概括,以包括在DTSE执行模式中合作的两个以上的核。同样由于停止或同步核执行,而在硬件周围发生一些违背(例如,一些负载/存储违背)。

[0110] 在一些实施例中,提交不在每个跨度上完成。在这种情况下,将在每个跨度上完成违背检查,但每隔几个跨度完成提交一次(以减少寄存器检查点开销)。

[0111] 在一些实施例中,数据高速缓存及其条目不是典型的。利用所支持的劳工量,数据高速缓存需要一些地址空间映射形式的帮助。例如,可利用x位地址空间映射来扩展每个数据高速缓存标签(诸如假设每核4个劳工时的5位值、如果每核2个劳工的3位值或如果每核仅1个劳工的仅2位值)。

[0112] 在5位变量中,位4指示主逻辑处理器可访问该高速缓存线。每个劳工逻辑处理器具有在字段<3:0>中的一位。这指示相应的劳工逻辑处理器是否可访问该线(例如,0001表示劳工逻辑处理器1能访问该线,而其它则不能)。如果它在地址空间映射中的位未被设置,则劳工逻辑处理器将获得数据高速缓存未命中。如果在地址空间映射中的主位未被设置,则主逻辑处理器将获得数据高速缓存未命中。

[0113] 假设每次存取将最多命中线中的一条线,数据高速缓存可包括在同一地址处的多个线(按多路)。这表示对于地址空间映射中的5位中的每一位,在同一地址处该位被设置的情况下,在数据高速缓存中有不多于1条线。

[0114] 在读取具有相同标签的多路的数据高速缓存时,将选择请求逻辑处理器可访问的独特路,如果有的话。如果没有,则它是数据高速缓存未命中。

[0115] 在高级存储处理(最旧存储的处理)时,高级存储可仅进行到执行存储的逻辑处理器可访问的高速缓存线。此外,如果劳工将高级存储写入数据高速缓存,则目标线将其地址空间映射设置成仅可由进行该存储的逻辑处理器访问。出于此目的,所有的主逻辑处理器被视为相同。

[0116] 另外,在一些实施例中,存在与每个数据高速缓存线逻辑相关联的"当前位"。它不需要物理接近数据高速缓存或数据高速缓存标签。对于劳工逻辑处理器可访问的所有高速缓存线,在劳工逻辑处理器中的全局提交标记变高级时清除当前位。对线的任何成功写入将其设置为"当前"。对中级高速缓存的线写回将其设置为"当前"。在来自中级高速缓存的重新填充时,如果为它脏且在中级高速缓存中非当前,则线必须被设置为"非当前"。否则,

在数据高速缓存中将其设置为当前。针对来自中级高速缓存的响应，以此方式设置当前位，既使没有数据；MLC仅仅授予所有权。

[0117] 不常见的一件事是中级高速缓存形成脏和非当前。这被返回到数据高速缓存并且设置数据高速缓存中的当前位的状态，既使没有数据；MLC仅仅授予所有权。

[0118] 将高级存储写入其当前位关闭的脏数据高速缓存线的尝试在该存储被置于数据高速缓存之前导致写回，但不是使无效。进行数据高速缓存线的写回设置其当前位。并且在其调用的写回之后成功写入高级存储(如果适当)将该线设置为“当前”。

[0119] 如果高级存储的写入命中干净数据高速缓存线(E状态)，该干净数据高速缓存线的当前位关闭，则向中级高速缓存发信号(如果它是脏)以将该线的副本写回到末级高速缓存，并且获取新跟踪数据块的新物理地址。中级高速缓存的脏数据不进入相同的全局提交跨度，因为该脏数据现在进入数据高速缓存。作为将高级存储写入它的结果，数据高速缓存线获取当前。

[0120] 在一些实施例中，存在用于每个劳工逻辑处理器的元数据缓冲器(未示出)。元数据被顺序地写入缓冲器。元数据缓冲器应能将此写入中级高速缓存。跟踪执行导致被写入保留存储器的跟踪数据以及被写入保留存储器的元数据。该过程的剩余部分是从保留存储器读回元数据，检查它，然后合并来自各跟踪的从保留存储器读取的跟踪数据，以及使其作为原始程序地址空间的新状态而可见。

[0121] 在中级高速缓存217中，所有全局提交点的原始程序数据块和跟踪数据块均存储在相同的位置。它们均通过块的原始程序地址来寻址。

[0122] 在中级高速缓存处，对保持中级高速缓存线的当前真实物理地址的标签存在扩展。需要真实的物理地址用于从中级高速缓存向末级高速缓存的写回。它不用于中级高速缓存中的多数其它操作。在一些实施例中，真实的物理地址是跟踪数据块号。它不多于16位。

[0123] 在一些实施例中，与中级高速缓存相关联的硬件寄存器保持每个劳工逻辑处理器的跟踪数据空间的基本地址和限制以及每个跟踪的最后分配的跟踪数据块。该硬件还保持当前全局提交号。

[0124] 存在与每个中级高速缓存线逻辑相关联的“当前位”。它不需要物理接近中级高速缓存或中级高速缓存标签。对于劳工逻辑处理器可访问的所有高速缓存线，在劳工逻辑处理器中的全局提交标记变高级时清除该位。

[0125] 在一些实施例中，用6位(假设我们支持每核4个劳工)地址空间映射扩展每个中级高速缓存标签。该标签的尺寸依赖于每核的劳工数量。

[0126] 地址空间映射支持在中级高速缓存中同时以相同地址(包括相同标签)具有多条线。它规定哪些逻辑处理器可访问该线。在读取具有相同标签的多路的中级高速缓存时，将选择请求逻辑处理器可访问的独特路，如果有的话。如果没有，则它是中级高速缓存未命中。

[0127] 在某些实施例中，中级高速缓存中的每条线可具有暂时提交位和暂时解除位。

[0128] 提交将包括对于所有暂时提交高速缓存线刷新清除真实物理地址上的有效位并且设置主访问、以及对于所有暂时解除高速缓存线刷新清除线有效位。

[0129] 在一些实施例中，存在具有各个DTSE集群的若干检查器状态机。每个检查器状态

机可工作在线程上，所以所提供的状态机的数量确定集群可同时工作的线程数量。检查器状态机从同一线程上工作的所有跟踪读取元数据列表，并且通过存储正确性高速缓存和负载正确性高速缓存处理它们。它还对复制回列表进行写入。

[0130] 在一些实施例中，复制回状态机将来自复制回列表的高速缓存线分布至合并状态机，且中级高速缓存进行来自不同跟踪的存储数据的实际合并并将结果置于原始程序地址空间中。

[0131] 在一些实施例中，准确地存在每中级高速缓存一个合并状态机。当必要时，该状态机进行来自若干跟踪的跟踪数据存储的数据的实际合并。

[0132] 一旦负载被执行，必须可证明除完成负载的线程外系统中的任何代理的存储不可命中该负载直到负载被全局提交。应有根据全局提交之前的任何负载或存储的核退役，确保在该线程执行中合作的一些中级高速缓存使目标原始程序地址线至少处于共享状态。该线可在合作中级高速缓存之间传送(主-从方法)，但必须维持连续占有。这表示对于获得该线所有权的其它代理，当没有其它的合作中级高速缓存具有该线时，合作中级高速缓存中的至少一个将获得使无效监听。

[0133] 典型地，每个DTSE集群具有占有高速缓存。该结构指示高速缓存线，对于该高速缓存线我们可肯定集群中的一些中级高速缓存将从环或数据高速缓存中获得监听。典型地，占有高速缓存位于保留存储器中，但一部分可利用检查器来进行高速缓存。

[0134] 每个劳工逻辑处理器具有推测访问高速缓存。推测访问高速缓存表示已经由劳工逻辑处理器本地退役但还未全局提交的负载和存储。

[0135] 全局提交标记是对特殊地址的存储。该数据将类号和指针给予提交指针描述符。

[0136] 提交等待是对特殊地址集的负载。它可出现在任何位置。取决于该地址集中的精确地址，它可以是在最后的全局提交点或某一类最后全局提交点处对提交的等待或其它可能的选择。用于该负载的该地址可请求最后的全局提交存储器状态。

[0137] DTSE将不对提交等待做出响应，因此在停止的请求劳工中保持执行直到指定的全局提交已经发生。如果最后的全局提交存储器状态被请求，则DTSE逻辑将进一步使请求劳工中的所有跟踪数据无效。然后，DTSE逻辑将返回用于提交等待的数据。这将使该劳工解除停止并允许它继续。

[0138] 如果最后的全局提交存储器状态被请求，则在该等待中指定的全局提交点之后该劳工中的所有存储将丢失。所以，一般而言，实际上仅在等待全局提交点之后但在该劳工进行更多的存储之前使用该选项。典型地，等待将紧随其等待的全局提交点的全局提交标记之后。

[0139] 等待有三个用途：等待可用于保护寄存器状态免受改变，因此避免保持一些寄存器的必要性；当负载有时命中不同跟踪中的存储时，这向负载提供获取正确结果的能力；以及它还可用于防止失控推测。

[0140] II.操作

[0141] 数据存储器状态由数据本身和元数据表示。它们不仅被分开存储而且被不同地存储。主逻辑处理器和劳工逻辑处理器二者均具有元数据。在没有元数据的情况下，存储器状态对于劳工逻辑处理器没有意义。元数据仅用于主逻辑处理器的概况分析。

[0142] 关于保留存储器中的数据空间，在一些实施例中，每个跟踪在保留存储器中具有

其自身的数据空间，如同每个劳工逻辑处理器一样。主逻辑处理器不具有跟踪数据空间。

[0143] 在流程的入口，所有的跟踪数据空间是未分配的。在流程执行时，根据需要循环地、顺序地分配高速缓存线尺寸块。每次分配针对特定的全局提交跨度。当这些块被分配的跨度被全局提交时，这些块按顺序解除分配。对于每个劳工逻辑处理器有一分配指针和一解除分配指针。这些指针包含块号，而不是地址。块号空间将是对于该分配存在的多个物理块数量的倍数。

[0144] 在分配中，块号不能包装。当所有的块号已经分配一次，该跟踪不能分配更多的跟踪数据块。如果需要更多的分配，这将导致流程退出。

[0145] 另一方面，物理块的分配可包装多次，直到不包装块号的极限。中级高速缓存中具有其真实物理地址中的解除分配块号的线将不会写回到末级高速缓存。类似地，如果在中级高速缓存中有未命中且用于重新填充的真实物理地址是解除分配的块号，则重新填充将替代地来自原始程序地址。这一线在其中不能具有推测存储。如果有推测存储，它最近可能已经被重新分配。

[0146] 可见，应有足够的物理块以分配用于跟踪数据，以覆盖跟踪在它们执行的代码的逻辑位置中相差的量加上一些以覆盖检查和全局提交的时间。块号空间需要覆盖流程从开始至流程退出的整个执行。

[0147] 如果在跟踪中有状态恢复，则利用分配的原始开始点，将执行跟踪的劳工的跟踪数据空间初始化为空。

[0148] 分配在跟踪被执行时与劳工逻辑处理器相关联的逻辑中发生。复制回状态机将该空间中特定位置与全局提交过程中稍后的存储相关联。

[0149] 对于使用中的每个合并状态机，复制回状态机将合并工作队列写在存储器中。这描述了合并状态机应访问的跟踪数据块。因此这通过存储器，所以这将被尽可能地保持压缩。

[0150] 用于每一个逻辑处理器的元数据被分开存储。在一些实施例中，用于逻辑处理器的元数据是程序正确地址、尺寸和负载/存储指示符的线性阵列。它包含对齐标记和全局提交标记以及复制的存储标记。它还可包含分支方向、间接分支的目标。它还可以是在中级高速缓存处接收的使无效记录

[0151] 当核加载到数据的原始程序地址时，硬件将捕捉地址和尺寸并将其添加至元数据列表。如果劳工从主逻辑处理器可访问和拥有的数据高速缓存线加载，则(如果设计方便的话)在强制对中级高速缓存的写回后如果中级高速缓存是脏，它使其共享。如果这不方便，则它替代地使该线不是主逻辑处理器可访问的且不是当前的。这些动作中的任一个将确保通过不与该劳工合作的逻辑处理器向该线通知任何未来的存储。

[0152] 在执行模式中来自劳工的高级存储不需要数据高速缓存中的所有权来写入数据高速缓存。高级存储可被写入共享状态中的线，但该线必须可由跟踪访问。在任何事件中，如果来自Catalina劳工的高级存储被写入数据高速缓存线，则该线变得仅可由该劳工访问。不管该线以前是什么，它已经被转换成跟踪数据。注意，如果线为脏，则在一些实施例中，它可能已经在高级存储被写入之前被强制写回，除非它已经是跟踪数据且是当前的。

[0153] 因此，当处理至该线的高级存储时，不请求对数据高速缓存线的所有权。

[0154] 因为这真是对跟踪数据的存储，中级高速缓存将不需要或请求原始程序地址线的

所有权。它将具有跟踪数据线的所有权。

[0155] 这是重要的，因为其它跟踪可存储至该线。合并引擎将需要获得原始程序地址线的所有权以进行提交。其它中级高速缓存可使线共享并由跟踪读取(而不是通过主逻辑处理器)，但不能拥有它。由于此处描述的规定，它们不需要拥有它且将不请求拥有它。

[0156] 这表示当一个中级高速缓存被用于进行合并且正提交，并且它要求原始程序地址线的所有权时，其它跟踪可保持该线为共享状态并对其进行读取和写入(如所述，对其写入将其转换成跟踪数据)。

[0157] 在数据高速缓存未命中时，照常从中级高速缓存请求该线。

[0158] 当核向数据的原始程序地址存储时，硬件将捕捉地址和尺寸并将其添加至元数据列表。

[0159] 在数据高速缓存中，所有全局提交点的原始程序数据块和跟踪数据块均位于相同的位置。它们均通过块的原始程序地址来寻址。

[0160] 在全局提交点标记变为高级时，对于该劳工可访问的所有高速缓存线清除当前位。对线的任何成功写入将其设置为“当前”。对中级高速缓存的线写回将其设置为“当前”。在来自中级高速缓存的重新填充时，如果为它脏且在中级高速缓存中非当前，则线必须被设置为“非当前”。否则，在数据高速缓存中将其设置为当前。针对来自中级高速缓存的响应，以此方式设置当前位，既使没有数据。

[0161] 劳工将高级存储写入其当前位关闭的脏数据高速缓存线的尝试在该存储被置于数据高速缓存之前导致写回，但不是使无效。设置一位，且该写回告诉中级高速缓存分配新的跟踪数据块。进行数据高速缓存线的写回设置其当前位。并且在其调用的写回之后成功写入高级存储(如果适当)将该线设置为“当前”。

[0162] 有权访问该线的劳工将高级存储写入主逻辑处理器可访问的脏数据高速缓存线的尝试在该存储被置于数据高速缓存之前导致写回，但不是使无效。设置一位，且该写回告诉中级高速缓存分配新的跟踪数据块。进行数据高速缓存线的写回设置其当前位。并且在其调用的写回之后成功写入高级存储(如果适当)将该线设置为“当前”。

[0163] 如果有权访问该线的劳工将高级存储写入主逻辑处理器可访问的干净数据高速缓存线，则向中级高速缓存发信号以将该线写回到末级高速缓存(如果该线是脏)且无条件获取新跟踪数据块的新物理地址。成功写入高级存储将线设置为“当前”。

[0164] 如果劳工的高级存储写入命中干净数据高速缓存线(E状态)，该干净数据高速缓存线的当前位关闭，则向中级高速缓存发信号(如果它是脏)以有条件地将该线的副本写回到末级高速缓存，并且获取新跟踪数据块的新物理地址。中级高速缓存的脏数据不进入相同的全局提交跨度，因为该脏数据现在进入数据高速缓存。作为将高级存储写入它的结果，数据高速缓存线获取当前。

[0165] 如果有高级存储且在数据高速缓存中线不是E或M状态，照常对来自中级高速缓存的线进行针对所有权请求的读取。

[0166] 数据高速缓存可请求中级高速缓存“写回”线并且分配新跟踪数据块。如果在中级高速缓存中线是脏的，则它被写回并且新跟踪数据块被分配。既使在中级高速缓存中线是干净的，请求也可指定无条件分配新跟踪数据块。如果新跟踪数据块被分配，则取代曾经在这里的块。数据是相同的。

[0167] 在对中级高速缓存的所有写回时，数据高速缓存“当前位”(在由于此写回而设置它之前)被发送到中级高速缓存。数据高速缓存写回有两个原因：这可能是数据高速缓存的传统驱逐或新的高级存储已经命中并非当前的脏数据高速缓存线。如果新的高级存储已经命中并非当前的脏数据高速缓存线，则数据高速缓存发送被断言的位。

[0168] 中级高速缓存将分配新跟踪数据块。

[0169] 在一些实施例中，在用于每个跟踪的保留存储器中有一高速缓存，即跟踪数据目录。线中的数据仅仅是跟踪数据块号和全局提交号。不与分配新跟踪数据块相接合，仅在跟踪可访问但主逻辑处理器不可访问的脏跟踪数据块被驱逐时，该高速缓存被写入。

[0170] 在一些实施例中，存在另一个结构，其镜像跟踪数据目录，称为跟踪数据目录概要。如果存在劳工逻辑处理器的中级高速缓存未命中，则从跟踪数据目录概要读取覆盖跟踪数据目录中其位置的位。通常它将是无效的，指示跟踪数据目录中的未命中。在这种情况下，对中级高速缓存的重新填充来自块的原始程序地址。该数据可能已经在中级高速缓存中作为主逻辑处理器可访问的版本。在这种情况下，可简单地使该版本对于劳工也可访问。如果概要显示命中的可能性，则如果证明实际上是命中，跟踪数据目录必须被访问以获得最当前的跟踪数据块号。逻辑可将此转换成块的地址。然后经由环请求该块，像未命中那样。

[0171] 分配新的跟踪数据块是一种特殊形式的从中级高速缓存至末级高速缓存的写回。如果中级高速缓存想要将线写回到末级高速缓存，且该线不是主逻辑处理器可访问的，但不具有有效的真实物理地址，则在写回可进行之前为其分配新的跟踪数据块。写回进行到通过分配新的跟踪数据块获得的物理地址。在本说明书的较早部分处理分配新的跟踪数据块的过程。

[0172] 在每个跟踪中，执行将分离地命中全局提交点。此处有对特殊地址的存储，该特殊地址用作全局提交点标记。它给予新全局提交号递增。其自身跟踪中的全局提交号前进。将这保持为与中级高速缓存相关联的DTSE逻辑的一部分。如上所指出的，这将导致存储进行到新的数据块。并且与数据高速缓存和中级高速缓存相关联的“当前位”被清除。

[0173] 存储变为仅在全局提交点被全局观察。负载和存储两者被全局推测，因此易受使无效监听命中的影响，直到下一全局提交点。

[0174] 跟踪设置指示它已经完成该提交跨度的标志。检查器状态机等待来自所有跟踪的该标志。

[0175] 如上所述，每个劳工逻辑处理器具有存储正确性高速缓存。该高速缓存强调在同一对齐跨度中一个劳工中的存储必须不能命中另一个劳工中的存储除非它们是同一存储的规则。它还强调在同一对齐跨度中一个劳工中的存储必须不能与另一个劳工中的存储冲突的规则。它强调在负载偶然首先处理且存储稍后处理的情况。如果存储被首先处理且负载稍后，这被负载正确性高速缓存捕捉。

[0176] 如上详述，每个劳工逻辑处理器具有负载正确性高速缓存。负载正确性高速缓存标识从进入流程至当前的暂时提交点已经被存储的所有字节。具体地，它告诉哪个劳工是该字节的逻辑最后写入者。

[0177] 复制回状态机工作在复制回列表上以将所有指示的存储合并到原始程序地址位置并使它们全局可见。复制回列表表示不是按程序顺序的存储聚集。因此必须使在复制回

列表中表示的所有存储原子地全局可见。外部世界必须看见这些存储中的全部或一个都看不到。

[0178] 如果由于诸如仅执行分支违背或存储违背或负载违背之类的内部问题而有流程退出，则状态将被恢复到流程退出前的最后全局提交点。这将等待复制回状态机完成，所以它确实是违背前的逻辑最后全局提交点。

[0179] 如果由于诸如命中全局推测负载的使无效监听之类的某些外部问题而有流程退出，则所涉及的指令可能在已经被释放到复制回状态机的跨度中。除非已知这不是这种情况，否则这必然导致立即退出而不全局提交复制回状态机当前工作的复制回列表。

[0180] 从数据高速缓存的角度看，合并状态机类似于另一种数据高速缓存。如果合并状态机访问高速缓存线，则如果在数据高速缓存中该线是脏的，这监听数据高速缓存并且将强制写回。

[0181] 经修改的块可跨越多个中级高速缓存，所以我们具有很少的同步：

[0182] 阻断对原始程序地址块的所有访问(仅对跟踪数据的访问被读取并且良好)。暂时位用于此。

[0183] 当运行优化(线程化)代码时，由于优化代码或者通过同时运行的不相关代码(或者甚至通过相同代码)生成的存储，原始代码可改变。为了进行提防，使用XMC保护数据高速缓存(XGC)409。该高速缓存保持所有页的地址(按页粒度)，它们被访问以便生成在监听使无效命中的情况下使用的优化代码和优化区ID。区ID指示所有的静态代码线，所有的静态代码线的联合触摸到被保护的区域(高速缓存线)。图7示出根据一些实施例的XGC 409的使用。

[0184] 在执行优化代码之前，代码保证XGC中的所有条目存在并且未被监听或取代。在这种情况下，允许执行优化代码。

[0185] 如果在优化代码被执行的时段中，另一个逻辑核改变原始页之一中的数据，则XGC将接收监听使无效消息(类似于一致性领域中的任意其它高速缓存代理)并且将通知核之一它必须中止执行与给定页相关联的优化代码并且使其保持的使用该页的任何优化代码入口点无效。

[0186] 当在线程化执行模式中执行时，相对于XMC 409检查每个存储，以提防自修改代码。如果存储命中热代码区，违背将被触发。

[0187] 在一些实施例中，线程生成器进行一些假设，稍后将检查这些假设的正确性。这主要是性能优化。这一假设的示例是调用-返回配对。线程生成器可假设返回将回到其调用(大多数时间它是正确的)。因此，线程生成器可将整个调用函数置于一个线程且允许以下代码(在返回后)在其它线程中执行。因为在返回执行前以下代码将开始执行，且栈被查找，所以执行可能是错误的(例如，当函数将返回地址写入栈，盖写原始返回地址)。为了提防这些情况，每个线程可将断言写入断言硬件块。如果两个线程就断言达成一致，则断言被满足。断言必须被满足以便提交跨度。

[0188] III.软件

[0189] 动态线程切换执行(DTSE)软件使用由硬件收集的概况分析信息来限定称为“流程”的代码的重要静态子集。在一些实施例中，该软件具有其自己的工作存储器空间。流程中的原始代码在该工作存储器中被再造。可通过软件改变工作存储器中的代码复制。原始

代码被精确的保持为其原始形式、在存储器中的其原始位置。

[0190] DTSE软件可将DTSE工作存储器中的流程分解成能够在多逻辑处理器上执行的多个线程。如果在没有该动作的情况下逻辑处理器未被完全利用，则这将会进行。通过硬件可以做的五件事来使这成为可能。

[0191] 在任一情况下，DTSE软件将代码插入DTSE工作存储器中的流程中以控制较大量逻辑处理器的(可能的SMT)硬件处理器上的处理。

[0192] 在一些实施例中，硬件该继续概况分析运行的代码，包括DTSE软件已经处理的流程。DTSE软件对改变的行为做出响应以修订它对代码的先前处理。因此软件将系统地(如果缓慢)改进其处理的代码。

[0193] 限定流程

[0194] 当DTSE硬件在其列表的顶部具有新IP时，软件将IP作为新流程的概况根。软件将指导硬件从该概况根获取概况。在实施例中，在一个连续的冲击中，硬件将使概况在下次执行命中概况根IP时开始并且之后扩展大致50000动态指令。用所有负载和存储的地址、直接分支的方向和间接分支的目标填充存储器中的缓冲器。包括返回。由此，软件可在静态代码的概况根处开始，并且跟踪穿过静态代码的概况分析路径。可找到分个分支的实际目标，并且所有负载和存储的目标地址是已知的。

[0195] 由该概况路径命中的所有静态指令被限定为在流程中。由该概况路径命中的每个控制流程路径被限定为在流程中。未被该概况路径命中的每个控制流程路径被限定为离开流程。

[0196] 在一些实施例中，DTSE软件将指导硬件再次从相同的根获取概况。已经不再流程中的新指令或新路径被添加到流程。软件在其获得概况时将停止请求更多概况，这不会将指令或路径添加到流程。

[0197] 流程维护

[0198] 在已经限定流程之后，它可被监视和修订。这包括在已经为其生成新代码之后，可能在多个线程中。如果流程被修订，通常这表示代码(可能是线程化代码)应被重新生成。

[0199] 老化无效流程

[0200] 在一些实施例中，为每个流程保持“指数老化的”平均流程长度L。在实施例中，L被初始化为500000。当流程被执行时，使流程中执行的指令数为N。然后计算：L＝0.9*(L+N)。如果对于流程L低于设定数(假设100000)，则该流程被删除。这还表示这些指令合格来再次成为热IP，除非它们在一些其它流程中。

[0201] 合并流程。

[0202] 在一些实施例中，当流程被执行时，如果在动态指令的设定数(例如，25000)之前存在流程退出，则其热代码入口点被设定成获得概况而不是执行热代码。下次热代码入口点被命中时，对于从该入口点开始的若干指令(例如，50000)获取概况。这添加到该流程的概况集合。

[0203] 任何新指令和新路径被添加到该流程。在一些实施例中，利用集合中的新概况再次进行流程分析和代码生成。

[0204] 拓扑分析

[0205] 在一些实施例中，DTSE软件执行拓扑分析。该分析可包括以下活动中的一个或多

个。

[0206] 基本块

[0207] DTSE软件将流程的代码分成基本块。在一些实施例中，仅将概况分析中已经观察到的连接点保持为连接点。所以既使在流程中存在具有明确目标的分支且该目标在流程中的情况下，如果该连接点未在概况分析中观察到出现，该连接点仍将被忽略。

[0208] 在概况分析中未观察到发生的所有控制流程路径(边沿)被标记为“离开流程”。这包括被观察到总是采用的分支的下降(未采用的分支)方向。

[0209] 在概况中单调的分支，包括无条件分支不结束基本块除非目标是连接点。调用和返回结束基本块。

[0210] 在进行以上之后，DTSE软件现在具有基本块的集合和基本块之间的“边缘”的集合。

[0211] 拓扑根

[0212] 在一些实施例中，每个概况用于引导该流程的静态代码的遍历。在每次调用的该遍历中，调用目标基本块标识符被推至栈，并在每次返回，栈被弹出(popped)。

[0213] 既使整个代码流可能具有平衡的调用和返回，该流程是从动态执行的片断开始的，该片段具有更多或更少的随机开始和结束点。没有理由认为该调用和返回在流程中平衡。

[0214] 如果有的话，遇到的每个基本块被标记为在由栈顶部上的基本块标识符所标识的过程中。

[0215] 来自概况根的代码最初不在任何过程中。稍后在概况中，可能再次遇到该代码，其中它将被标识为在过程中。更加可能的是有一些代码不再任何过程中。

[0216] 拓扑分析的质量取决于用于拓扑分析的根。典型地，为了得到良好的拓扑分析，根应在限定在流程中的静态代码的最外过程中，即在“不在任何过程中”的代码中。被硬件找到的概况根可能不是概况根。因此DTSE软件限定拓扑分析所使用的拓扑根。

[0217] 在一些实施例中，对于不在任何过程中的基本块，代码的子集R被标识，使得从R中的任何指令开始，但仅使用该流程的边缘，即，在至少一个概况中被观察到被采用的边缘，存在至流程中的每个其它指令的路径。R可能为空。如果R是空，则限定拓扑根是概况根。如果R不是空，则拾取R中数字最低的IP值作为拓扑根。从此处开始，任何提及的“根”表示拓扑根。

[0218] 过程内联

[0219] 传统的过程内联用于消除调用和返回开销的目的。DTSE软件保持关于代码行为的信息。取决于代码调用什么，过程中的代码表现不同。因此，在一些实施例中，DTSE软件对于该过程的每次不同静态调用保持单独的关于过程中代码的信息表。

[0220] 该代码的中间阶段是不可执行的。当完成分析时，DTSE软件将生成可执行代码。在该中间状态，没有过程中的用于内联的代码的复制。过程内联将多个名字分配给过程的代码并且保持关于每个名字的单独信息。

[0221] 在一些实施例中，这是递归的。如果外部过程A从3个不同的地点调用过程B，并且过程B从4个不同的地点调用过程C，则有用于过程C的12种不同行为。DTSE软件将保持关于过程C中的代码的12个不同信息表，对应于对该代码的12个不同调用路径，和用于该代码的

12个不同名字。

[0222] 当DTSE软件生成用于该流程的可执行代码时，可能有该代码的远小于12个静态副本。具有相同代码位的多个副本不是感兴趣的，且在多数情况下，调用和返回开销较小。然而，在一些实施例中，DTSE软件为至该代码的每个调用路径保持单独的行为信息。DTSE保持的行为信息的示例首先是指令依赖性、以及负载和存储目标及分支可能性。

[0223] 在一些实施例中，DTSE软件将假设如果在原始代码中有静态调用指令，则从所调用的过程返回将总是进入调用之后的指令，除非观察到这在概况分析中未发生。然而，在一些实施例中，检查到这在执行时正确。DTSE软件生成的代码将对此进行检查。

[0224] 在一些实施例中，在DTSE生成的最终的可执行代码中，对于原始代码中的调用指令，存在将架构返回地址推入程序的架构栈的指令。注意，这不能通过所生成的代码中的调用指令来完成，因为所生成的代码在完全不同的位置且将会将错误的值推到栈上。推到栈上的该值对于热代码几乎无用。用于程序的数据空间将始终保持正确。如果生成多个线程，则哪个线程来完成是没有区别的。它应在某个位置某些时间完成。

[0225] 在一些实施例中，如果过程非常小，则DTSE软件可选择将在特定线程中进行的过程的部分的物理副本物理地放在线中。否则，此处没有物理副本，且将有某种类型的控制传递指令，以进入该代码。这将在“代码生成”下进行更多的描述。

[0226] 在一些实施例中，在DTSE生成的最终的可执行代码中，对于原始代码中的返回指令，存在将来自程序的架构栈的架构返回地址弹出(pop)的指令。DTSE软件相信会是该返回的目标的架构(而不是热代码)返回目标IP将对代码是已知的。在一些情况下这是热代码中的直接常数。在其它情况下，这存储在DTSE存储器中，有可能在栈结构中。这不是程序的数据空间的一部分。从栈弹出的值必须与DTSE软件认为会是该返回的目标的IP进行比较。如果这些值不同，则流程退出。如果生成多个线程，则哪个线程来完成是没有区别的。它应在某个位置某些时间完成。

[0227] 在一些实施例中，如果过程非常小，则DTSE软件将在特定线程中进行的过程的部分的物理副本物理地放在线中。否则，此处没有物理副本，且将有某种类型的控制传递指令，以进入该线程中的热代码返回目标。这将在“代码生成”下进行更多的描述。

[0228] 后边沿

[0229] 在一些实施例中，DTSE软件将找到该流程的最小后边沿集合。最小后边沿集合是从一个基本块至另一个的边沿的集合，使得如果这些边沿被切割，则将没有闭合的环路径。该集合应在如果任何边沿从集合中去除则将有闭合环路径的意义上最小。在一些实施例中，存在如果集合中的所有后边沿被切割则代码仍完全连接的性质。有可能从根至基本块的整个集合中的每个指令。

[0230] 每个过程单独完成。因此，对此忽略调用边沿和返回边沿。

[0231] 单独地，可在一些实施例中执行递归调用分析。这通过探索嵌套调用树来完成。从顶部开始，如果存在对在已经在该路径中的嵌套调用树中的路径上的任何过程的调用，则存在递归调用。递归调用是环，且从该调用限定后边沿。所以单独地，调用边沿可被标记为“后边沿”。

[0232] 在一些实施例中，算法在根处开始并且跟踪从基本块至基本块的所有路径。基本块的内部不重要。另外，已经被限定的后边沿不被遍历。如果在从根P的任何线性路径上遇

到已经在P中的基本块S，则在S结束的该边沿被限定为后边沿。

[0233] 限定分支再收敛点

[0234] 在一些实施例中，存在一些未被预测的分支，因为它们被视为单调的。如果该分支在执行中进入错误路径，则它是分支未命中预测。不仅如此，它离开流程。在处理流程中的代码时，出于所有目的，这些分支被视为完美单调(即，完全无条件的分支)。

[0235] 间接分支将具有已知目标列表。本质上，它是多目标条件分支。DTSE软件可将此编码为比较和分支的顺序串，或者利用反弹表。在任一编码中，还有一个目标：离开该流程。这基本上是终端处的单调分支。如果这进入错误路径，则我们离开该流程。至已知目标的多路分支具有与直接条件分支相同的再收敛点，并且建立相同的路径。并且当然，未预测的单调的最后解决分支(last resort branch)按不作为分支来处理。

[0236] 调用和返回(如所述)是特殊的，且不是“分支”。返回是再收敛点。在过程P中的不具有以某些其它方式限定的再收敛点的任何分支具有“返回”作为其再收敛点。P可具有在很多位置编码的返回。出于成为再收敛点的目的，返回的所有编码实例被视为相同。对于过程的任何静态实例，所有编码的返回精确地进入相同位置，该位置对于过程的该静态实例是独特的。

[0237] 给出所有这些，对于所有事件分支的再收敛点应能够被找到。在一些实施例中，仅基本块的入口点可以是再收敛点。

[0238] 对于分支B，可找到在收敛点R，使得在从B至R的所有控制流程路径中，后边沿遍历的总数最小。给出对于分支B的再收敛点集，该再收敛点集在从B至再收敛点的所有路径上具有相同数量的后边沿，在从B至再收敛点的完整路径集上具有最少指令的再收敛点通常是优选的。

[0239] 在一些实施例中，在分析其间保持两个参数：后边沿限制和分支限制。两者被初始化为0。在一些实施例中，过程通过还没有限定的再收敛点的所有的分支并且执行以下动作中的一个或多个。对于每个这类分支B，在分支B开始，向前遵循所有的控制流程路径。如果任何路径离开流程，停止继续该路径。如果所遍历的不同后边沿的数量超过后边沿极限，则该路径不再继续，且会越过该极限的后边沿不再被遍历。对于每个路径，收集该路径上的基本块集。找到所有这些集合的交集。如果该交集是空，则该搜索是不成功的。从该交集，拾取集合的成员R，对于该成员R在从B至R的所有路径上的所有指令的总数最小。

[0240] 现在，确定从B至R的所有路径中总共有多少“可见的”后边沿。如果该数大于后边沿极限，则R被拒绝。然后对于可见后边沿的总数测试下一个具有较大总指令数的可能再收敛点。最后，找到满足后边沿极限的再收敛点或没有更多的可能性。如果找到一个，则确定在从B至R的所有路径上还没有再收敛点的分支的总数。如果它超过分支极限，则拒绝R。最终满足后边沿极限和分支极限的R将被找到，或者没有可能性。良好的R是B的再收敛点。

[0241] 在一些实施例中，一旦分支B的再收敛点已经被找到，对于找到再收敛点的其余算法，通过B的任何正向控制流程遍历将直接跳至其再收敛点，而不查看分支和其再收敛点之间的细节。通过再收敛点的任何向后控制流程遍历将直接跳至其匹配分支，而不查看分支及其再收敛点之间的细节。本质上，控制流程从分支收缩到其再收敛点向下至单个点。

[0242] 在一些实施例，如果找到再收敛点，则后边沿极限和分支极限均被复位，并且考虑还不具有再收敛点的所有分支。如果成功找到再收敛点，则使一些东西不可见。现在可找到

之前不成功的分支的再收敛点，甚至在后边沿极限和分支极限的下限值处。

[0243] 在一些实施例中，如果没有找到再收敛点，则尝试下一分支B。当还不具有再收敛点的所有分支已经被不成功地尝试后，分支极限递增并且再次尝试分支。在一些实施例中，如果由于分支极限没有可能的再收敛点被拒绝，则将分支极限复位为0，递增后边沿极限并再次尝试。

[0244] 一般而言，在从分支B至其再收敛点R的控制流程路径上可以有不具有再收敛点的其它分支C，因为分支极限设置成大于0。对于每个这样的分支C，C获得分配给它的B具有的相同再收敛点，即R。分支B和所有这样的分支C的集合被限定为“分支组”。这是全具有相同再收敛点的一组分支。在一些实施例中，在使从分支至再收敛点的整个事件“不可见”之前，这被维护。如果这不作为组来维护，则一旦分支之一获得分配的再收敛点，则为组中的其它分支找到再收敛点所必需的所有路径变得不可见，更不必说还没有再收敛点的那些其它分支变得不可见。

[0245] 在一些实施例中，所有分支具有限定的再收敛点。“线性路径中后边沿的数量”表示不同后边沿的数量。如果在线性路径中相同的后边沿出现多次，它仍仅作为一个后边沿计数。如果基本块E是分支B的限定的再收敛点，这不会使其不合格来成为分支D的限定的再收敛点。

[0246] 整体展开

[0247] 在一些实施例中执行整体展开。在整体展开中，创建代码的有限量的静态复制以允许暴露特定形式的并行性。

[0248] 在这些实施例中，整个流程被复制N次用于每个分支嵌套级。N的良好值可以是在最终代码中期望的跟踪数量，然而其它数量可能具有一些优点。该复制提供了在工作在环的不同迭代上的多个(可能是所有)跟踪中具有相同代码的机会。它不会使任何环的不同迭代进入不同跟踪。一些环将通过迭代分离，而一些将在环内以细粒度逐指令地分离。更普遍地，环将以逐指令为基础以两种方式分离。期望发生什么，将发生什么。它仅允许通过迭代分离。

[0249] 在目前的情况下，仅有环体的一个静态副本。如果仅有一个静态副本，则在没有动态复制的情况下它不能在多个跟踪中，这可能起相反作用。为了允许该代码在多个跟踪中，以便在不同的控制流程路径(不同的迭代)上使用，可以有多个静态副本。

[0250] 嵌套

[0251] 在从组中的分支至组限定的再收敛点的路径中具有至少一个可见后边沿的分支组被限定为“环”。在再收敛点分析中限定对特定分支组什么“可见”或不“可见”。此外，不在从任何可见分支至其再收敛点的路径上的任何后边沿也被限定为“环”。

[0252] 仅被限定成后边沿的环被限定成具有从其后边沿开始，经由该后边沿回到其后边沿的开始的路径，作为它的“从其分支至它们的再收敛点的路径”。

[0253] 给出不同的环A和B，如果B的组中的所有分支在从A中的分支至A的限定再收敛点的路径中，则B嵌套在A中。限定为不在从分支至其再收敛点的路径上的后边沿的环被限定为不嵌套在任何其它环内，但其它环可被嵌套在它中，且通常是这样的。

[0254] 仅被限定成后边沿的环与该后边沿相关联。其它环与从环的分支至环再收敛点的路径中的可见后边沿相关联。在再收敛点分析中限定什么对特定的分支组“可见”或不“可

见”。

[0255] 可将以下定理和引理中的一个或多个施加到嵌套的实施例。

[0256] 定理1:如果B嵌套在A中,则A不嵌套在B中。

[0257] 假设B嵌套在A中。则在B中有分支,且B中的所有分支在从A至其再收敛点的路径上。如果A不包含分支,则通过限定,A不能嵌套在B中。如果A中的分支X在从B中的分支至其再收敛点的路径上,则X是B的一部分或者它对B不可见。如果X是B的一部分,则A的全部是B的部分且环A和B不是不同的。所以X必须对B不可见。这表示A必须在B之前限定其收敛点,使得A的分支对B不可见。因此B对A不可见。B中的全部分支在从A至其再收敛点的路径上且可见。使使得B是A的部分,所以A和B不是不同的。X不能如所假设的。

[0258] 引理1:如果分支B2在从分支B1至其再收敛点的路径上,则从B2至其再收敛点的整个路径也在从B1至其再收敛点的路径上。

[0259] 从B1至其再收敛点R1的路径通向B2。因此它跟随来自B2的所有路径。如果B1已经再收敛,则B2已经再收敛。如果我们还未达到指定用于B2的“再收敛点”,则R1是较好的点。再收敛点算法将找到最佳点,所以它必须已经找到R1。

[0260] 定理2:如果环B的一个分支在从环A中的分支至其再收敛点的路径上,则B嵌套在A中。

[0261] 令X是B中的分支,X在从A中的分支至A的再收敛点RA的路径上。通过引理1,从X至其再收敛点RB的路径在从A至RA的路径上。环B是在从X至RB的路径上所有分支的集合。它们均在从A至RA的路径上。

[0262] 定理3:如果B嵌套在A中,且C嵌套在B中,则C嵌套在A中。

[0263] 令X是具有再收敛点RC的C中的分支。则X在从B中的分支Y至B的再收敛点RB的路径上。通过引理1,从X至RC的路径在从Y至RB的路径上。B中的分支Y在从A中的分支Z至A的再收敛点RA的路径上。通过引理1,从Y至RB的路径在从Z至RA的路径上。

[0264] 因此,从X至RC的路径在从Z至RA的路径上。所以,X一定在从Z至RA的路径上。这对于C中的所有X都是正确的。所以C嵌套在A中。

[0265] 定理4:后边沿与一个且仅一个环“相关联”。

[0266] 不在从可见分支至其再收敛点的路径上的后边沿本身是环。如果后边沿在从可见分支至其再收敛点的路径上,则该分支所属的分支组具有至少一个后边沿,因此是环。

[0267] 假设有与环L相关联的后边沿E。令M是不同的环。如果L或M是不具有分支的环,即它们仅仅是单个后边沿,则定理是正确的。所以假设L和M具有分支。顺序地限定再收敛点。如果M的再收敛点被首先限定,且E在从M至其再收敛点的路径上,则E可能已经被隐藏。稍后它不会对L可见。如果首先限定L的再收敛点,则E将被隐藏且稍后不会对M可见。

[0268] 非定理5:在流程中执行超过一次的所有代码在一些环中不是对的。

[0269] 不在任何环中但被执行多次的流程中的代码的示例是在分支中结束的两个基本块。分支的一个臂的目标是第一基本块而分支的另一个臂的目标是第二基本块。分支的再收敛点是至第二基本块的入口点。在第一基本块中的代码在环中,但在第二基本块中的代码不在环中,即它不在从环分支至其再收敛点的任何路径上。

[0270] “反向后边沿”是与环分支组相关联的后边沿,使得从该后边沿向前前进,在该环分支组中的任何分支之前,环分支组的再收敛点被命中(且可能绝对不命中该环分支组中

的任何分支)。如果后边沿对环分支组可见且在从该环分支组中的分支至该环分支组的再收敛点的路径上,则后边沿与环分支组“相关联”。

[0271] 注意,在具有退出环的环分支的经典环中,穿过后边沿的路径首先命中环分支然后其再收敛点。如果后边沿是反向的后边沿,则穿过该后边沿的路径首先命中再收敛点然后命中环分支。

[0272] 定理6:如果不在任何环中的流程中有被执行超过一次的指令,则该流程包含反向后边沿。

[0273] 令I是流程中被执行超过一次的指令。假设I不在任何环中。假设在流程中没有反向后边沿。

[0274] 在从I返回I的路程中必然有某种路径P。在该路径中有至少一个后边沿E。

[0275] 假设有分支B,该分支B是与E相关联的环的一部分。这表示B是分支组的一部分。E对该分支组可见,且E在从该组中的分支至其再收敛点的路径上。

[0276] 从E向前前进在P上,除非有另一个分支。如果有另一个分支C,则C在从B至B的再收敛点的路径上,因此C在该相同的分支组中。C在P中。因此,在P中有该环的环分支。如果没有C,则P被跟随并到达I。如果在B的再收敛点之前到达I,则I在环中,与假设相反。所以应在达到I之前到达B的再收敛点。并且这在到达任何分支之前。所有从后边沿开始的路径在其命中另一个环分支之前命中再收敛点。

[0277] 另一方面,假设有在P中的环分支C。如果再收敛点不在P中,则所有的P在环中,尤其是I。所以再收敛点也在P中。所以C、E和再收敛点R均在路径P上。顺序必须是至E然后C然后R,因为任何其它顺序将给我们反向的后边沿。如果P上有超过一个分支,诸如可到P上的某处的分支X。但至少一个环分支必须在E和R之间。C是该环分支。

[0278] C具有另一个臂。应有从C的其它臂至R的路径。从C开始的所有路径在E之前到R,则E不在从C至R的任何路径上。因此,从C至R的整个结构对B不可见,并且C不会是该环的环分支。因此从C开始的某个路径必须在R之前通过E。但这是不可能的。该路径必须在边沿E之前的某些位置连接P。它不论在哪里,它都将成为再收敛点R。结论是从其它角度看,P上唯一可能的顺序E然后C然后P事实上不可能。

[0279] 在一些实施例中,利用以上理论中的一个或多个,可向环分配独特的嵌套级。不具有嵌套于其中的其它环的环具有嵌套级0。包含它们的环是嵌套级1。存在具有最高嵌套级的环。这为该流程限定嵌套级。注意环嵌套仅在过程中。在每个过程中它从0开始。由于过程内联,这适合。流程的嵌套级是流程中所有过程上的最大嵌套级。

[0280] 因为每个后边沿属于一个且仅一个环,所以后边沿的嵌套级可被限定成其所属的环的嵌套级。

[0281] 在一些实施例中,DTSE软件将复制整个流程N^U次,作为单元,其中U是流程的环嵌套级。N是每个环嵌套级展开的路数。

[0282] 在一些实施例中,因为这是每个相同代码的N^U个精确副本,软件没有理由实际复制该代码。位将精确地相同。代码在概念上被复制N^U次。

[0283] 流程的静态副本可由具有U位的数字来命名。在实施例中,数字是底数N。最低位数字与嵌套级0相关联。下一位数字与嵌套级1相关联。每个数字对应于嵌套级。

[0284] 在一些实施例中,对于展开的复制名中的每位数字D,DTSE软件使在D的值为0的所

有副本中，具有与D相关联的嵌套级的每个后边沿到D的值为1的副本中相同IP，但所有其它位数字是相同的。这使在D的值为1的所有副本中，具有与D相关联的嵌套级每个后边沿到D的值为2的副本中相同IP，但所有其它位数字是相同的。依此类推，直到副本N-1。软件使在D的值为N-1的所有副本中，具有与D相关联的嵌套级的每个后边沿到D的值为0的副本中相同IP，但所有其它位数字是相同的。

[0285] 其实施例是当前展开静态副本数和用于在遍历流程时它如何改变的算法。该算法是如果级L的后边沿沿正向方向遍历，则第L数字模N递增。如果沿向后方向遍历级L的后边沿，则递减第L数字模N。这就是先前的复杂段落所说的。在一些实施例中，DTSE软件不具有指针或表示它的任何东西。它仅具有该简单的当前静态副本号和计数算法。

[0286] 因此，在一些实施例中，DTSE软件已经通过因数N展开所有的环。它整体、单次进行，而不真实地理解任何环或单独查看它们。它真实知晓的一切是每个后边沿的嵌套级，以及这些中的最大值，即流程的嵌套级。

[0287] 在这些实施例中，因为没有目标IP改变，所以对代码中的任何位都没有改变。所改变的是在同一IP处的指令的每个静态实例可具有不同的依赖性。每个静态实例依赖于不同的其它指令，且不同的其它指令依赖于它。对于由其IP限定的每个指令，期望针对其静态实例中的每一个单独记录其依赖性的能力。当遍历任何控制路径时，展开副本计数器将适当改变状态以始终告诉现在指令的哪些展开副本被查看。

[0288] 分支再收敛点。

[0289] 在一些实施例中，如果在控制流程图遍历中，命中作为环L的成员的分支B，则B所属的分支组的标识符被推到栈上。如果在控制流程图遍历中，命中一分支，该分支的分支组已经在栈的顶部，则什么都不做。如果在控制流程图遍历中，对于在栈X(展开前限定的)顶部上的分支，命中再收敛点，则到该展开嵌套级的版本0，且弹出该栈。这说明X的版本0将是展开环的实际再收敛点。

[0290] 在一些实施例中，有例外。如果被遍历的L的最后后沿是反向后沿且L(展开前限定的)的再收敛点X被命中，且L在栈的顶部，则栈被弹出，但应维持一些展开版本，而不是到版本0。在这种情况下，该展开嵌套级X的版本0被限定成L的再收敛点。

[0291] 在退出时，环L总是到嵌套级L的版本0(除了当L具有反向后边沿以外)。

[0292] 以上描述如何向前跟随控制流程图的实施例。当在一些实施例中证明更需要遵循向后分支控制流程图而不是前向。在一些实施例中，这与嵌套过程相同。

[0293] 向后，L的再收敛点首先被命中。复杂点是这可能是多个环的收敛点且还用于不是环的分支组。问题是哪个结构被倒入。的确有很多路径进入到该点。如果倒入环，则它应在当前点之下的嵌套级1处。在此嵌套级处仍可有很多环以及非环分支组。可进行挑选跟随哪个路径。如果挑选被倒入的环L，则有N个路径用于跟随进入N个展开副本。在一些实施例中，它们之一被挑选。现在已知被倒入的代码的静态副本。可被找到的那些是相应分支组中的分支。该信息被推到栈上。

[0294] 在一些实施例中，如果不在当前嵌套级的展开副本0中，则倒入该环的后边沿。所以，当到达获得后边沿的最后机会时，该路径被知晓。直到那时，有所有可能性。如果在当前嵌套级的展开副本0中，则在一些实施例中做出不采用任何后边沿的附加选择和环外的撤出。如果环被撤出，则弹出栈。

[0295] 在一些实施例中，每次获得该环的后边沿，则在该嵌套级模N处递减副本号。

[0296] 通常在其嵌套级的静态副本0处进入环，且它总是退出至其嵌套级的静态副本0。

[0297] 记住，这些是分析该代码的软件内部的操作；不是执行该代码。在很多实施例中，执行不具有这种栈。将生成代码以全部到正确的位置。对于生成进入到所有正确位置的代码的软件，它自己需要知道如何遍历流程。图8-11示出这些操作中的一些的示例。图8示出具有三个基本块的示例，该三个基本块具有两个后边沿。这形成两级嵌套简单环。C的入口是B中分支的收敛点。从C退出的目标是C中分支的再收敛点。图9示出整个流程已经被复制。此处示出它的一部分。我们嵌套的环有4个副本，副本00、副本01、副本10和副本11。Cx的入口是Bx中分支的收敛点。从Cx退出的目标是Cx中分支的再收敛点。对于每个x这些是不同的。图10示出后边沿和至再收敛点的边沿已经利用以上讨论中的操作中的一个或多个进行修改。至C00的入口现在是环B00-B01的再收敛点。至C10的入口点现在是环B10-B11的再收敛点。较外部的环，静态副本00和10均到共同的再收敛点。有共同的再收敛点也是C01和C11的目标。对此不感兴趣，因为C01和C11是死代码。没有办法到达该代码。事实上，从该段代码退出总是在来自C00或C01的静态副本00中。在图11中，已经去除死代码和死路径以更清楚地示出它如何工作。注意至该代码仅有一个活入口，它在静态副本00中，且从该代码仅有一个活出口，它在静态副本00中。在一些实施例中，DTSE软件将不具体“去除”任何代码。仅有该代码的一个副本。没有要去除的东西。软件理解基本块A和C需要仅两个名字00和01下的依赖信息，而不是4个名字下。基本块B需要四个名字下的依赖信息。

[0298] 较大数字N增加准备代码的工作量，但它还可潜在地增加利用较少动态复制的并行性。在一些实施例中，DTSE软件可增加N以进行更好的工作，或减小N以凭借较小的工作产生代码。一般而言，匹配最终跟踪数的N将以合理的工作量给予最多的并行性。一般而言，比它大的N将给予略好的结果但工作多很多。

[0299] 环展开提供指令I对于一些迭代在一个跟踪中被执行的可能性，而对于不同迭代，同一指令I的不同静态版本在不同的跟踪中同时执行。此处强调“指令”，因为跟踪分离是以指令接指令为基础进行的。指令I可按此方式处理，而该环中紧接I的指令J可能完全不同地处理。可针对跟踪0中的所有迭代执行指令J，而该环中紧接I和J的K可针对跟踪1中的所有迭代执行。

[0300] 允许来自同一环的不同迭代的指令在不同跟踪中执行的环展开是有用的工具。它揭示很多代码中的显著并行性。另一方面，环展开揭示在很多代码中根本没有并行性。这是DTSE可使用的唯一的一个工具。

[0301] 此外，对于DTSE软件中的分析，通常没有理由复制用于展开的任何代码，因为位将是相同的。展开对代码产生多个名字。每个名字具有它自己的特性表。每个名字可具有不同的行为。这可以揭示并行性。甚至稍后将生成的可执行代码将不具有很多副本，即使在分析期间该代码有很多名字。

[0302] 线性静态复制

[0303] 在一些实施例中，对于整体展开，整个流程已经被复制多次。在其顶部，在一些实施例中，按需要将整个流程复制多次。副本被命名为S0、S1、S2…。

[0304] 每个分支B在流程中被复制在每个静态副本S0、S1、S2…中。B的每个副本是一般分支B的实例。类似地，B具有在S0、S1、S2…中已经复制的再收敛点。所有的副本是一般分支B

的一般再收敛点的实例。所复制的后边沿均被标记为后边沿。

[0305] 在一些实施例中，没有代码被复制。在这些实施例中，在代码中的所有东西还获得另一级别的多个名字。每个名字获得存储信息的位置。

[0306] 在一些实施例中，在流程的所有“S”副本中的所有边沿获得改变至正确的一般基本块的目标，但未被分配给特定的“S”副本。所有的后边沿获得其改变至特殊的S0副本的目标。

[0307] 在一些实施例中，流程S0、S1、S2…的副本按数字顺序被一一通过。对于Sk，不是后边沿的每个边沿E(原点在流程副本Sk中)将用于其目标的特定副本分配为最低“S”号副本，使得它不与任何其它边沿共享目标。

[0308] 最后，将没有边沿，它不是后边沿，它与任何其它边沿共享目标基本块。当然，后边沿将频繁地与其它(可能是很多其它)后边沿共享目标基本块。

[0309] 如同在整体展开的情况下，在一些实施例中，退出环的边沿的目标“S”实例通过进入环再收敛点而修改，如下。

[0310] 在一些实施例中，如果在控制流程图遍历中，命中作为环L的成员的分支B，则B所属的分支组的标识符被推到栈上的当前“S”实例号。在一些实施例中，如果在控制流程图遍历中，命中一分支，该分支的分支组已经在栈的顶部，则什么都不做。在一些实施例中，如果在控制流程图遍历中，在栈顶部的环的一般收敛点的实例被命中，则I弹出栈且实际到从栈弹出的“S”实例号。

[0311] 这说明在环中，环的每次迭代在“S”实例号0处开始，但在退出该环时，到“S”实例，其中该环被进入。

[0312] 注意可使用与整体展开所使用的相同的栈。如果使用相同的栈，将字段添加到“S”实例的每个栈元素。

[0313] 再次，这些是分析该代码的软件内部的操作；不是执行该代码。执行不具有这种栈。将生成代码以全部到正确的位置。对于生成进入到所有正确位置的代码的软件，它自己需要知道如何遍历流程。

[0314] 将有第一流程副本Sx，它是不能从副本S0到达的。不需要这个以及所有较高编号的副本。除此之外，每次存活，静态副本S1、S2通常具有大量不能从S0到达的死代码。不能从此处到达的要素将不生成发射的可执行代码。

[0315] 依赖性分析

[0316] 多结果指令

[0317] 已经讨论在一些实施例中，原始调用指令可能已经被推入取代，且原始返回可能已经被弹出和比较取代。

[0318] 一般而言，在分析中不需要多结果指令。在一些实施例中，这些将被分成多个指令。在很多情况下(但为了确定不是全部)，这些或类似指令可在代码生成时重新构造。

[0319] 推入和弹出是显然的示例。推入是存储和递减栈指针。弹出是加载和递增栈指针。通常期望将栈指针修改和存储器操作分离。有很多其它指令具有可被分离的多个结果。在一些实施例中，这些指令被分离。

[0320] 分离这些的常见原因是很可能所有的线程需要跟踪栈指针变化，但它不必复制在每一线程中推入的数据的计算。

[0321] 不变值

[0322] 硬件支持和机构

[0323] 在一些实施例中，DTSE硬件具有软件可用的若干“断言寄存器”。每个“断言寄存器”可至少保持两个值：实际值和被断言值，且每个值有一个有效位。在一些实施例中，断言寄存器是所有核和硬件SMT线程的全局资源。

[0324] 在一些实施例中，DTSE软件从任何核中的任何硬件SMT线程，在任何时间写入任何断言寄存器的实际值部分或断言值部分。

[0325] 在一些实施例中，为了全局提交对给定断言寄存器的断言值的写入，目标断言寄存器的实际值部分必须有效，且两个值必须匹配。如果实际值不是有效的或者值不匹配，则硬件将导致脏流程退出，且状态将恢复到最后的全局提交状态。

[0326] 断言寄存器为一个核中的一个逻辑处理器A上运行的代码提供使用不是由该逻辑处理器或核实际计算的值的能力。在一些核中的一些逻辑处理器B中，该值必须在逻辑上较早地计算，而不必在物理上较早地计算，并且被写入断言寄存器的实际值部分。在A中运行的代码可假设任何值并将其写入相同断言寄存器的断言值。断言值的写入之后的代码肯定知道被写入断言值的值精确匹配在对断言值的写入的逻辑位置处被写入实际值的值，不管该代码被放在何处。

[0327] 当DTSE软件很可能但不确定知道一值而不进行对该值的所有计算并且该值用于很多事情时，这是有用的。它提供了在多个核中的多个逻辑处理器中使用该值但仅在一个核中的一个逻辑处理器中正确计算它的可能性。在DTSE软件关于该值是正确的情况下，本质上对断言操作没有成本。如果DTSE软件关于该值不正确，则没有正确性问题，但对所导致的流程退出可能有大的性能成本。

[0328] 栈指针

[0329] 栈指针和基址指针通常被频繁使用。不太可能执行非常有用的代码而不使用栈指针和基址指针中的值。因此，通常每个DTSE线程中的代码将使用这些寄存器中的值的大多数。例如栈指针的实际值通常取决于对栈指针的变化的长依赖性链。在一些实施例中，DTSE软件可通过插入对断言寄存器的实际值部分的写入然后是对该断言寄存器的断言值写入假设值，来打破该长依赖性链。然后是不直接依赖于实际值的写入或之前的任何东西的值。

[0330] 对于原始代码中的过程调用和返回，DTSE软件将正常地假设紧跟返回之后的栈指针和基址指针的值与其刚好在调用之前相同。

[0331] 在一些实施例中，刚好在调用(原始指令)之前，可将伪指令插入。这是不会生成代码但具有类似指令的表的指令。标记伪作为栈指针和基址指针的消费者。

[0332] 在从过程返回之后，指令被插入以将栈指针和基址指针复制到2个断言寄存器的实际值部分。这些插入的指令被标记为这些值的消费者。

[0333] 之后，在一些实施例中，指令被插入以将栈指针和基址指针复制到这些断言寄存器的断言值部分。这些插入的指令被标记为不消耗这些值，但产生这些值。这些指令被直接标记为依赖于伪。

[0334] 类似地，对于不明显进行不平衡栈变化的很多环，假设在每次迭代开始时，栈指针和基址指针的值是相同的。在一些实施例中，作为消费者的伪被插入在该环的最初入口。实际值的副本被插入并标识为消费者，随后是断言值的副本，标识为生产者。使断言值的副本

直接依赖于伪。

[0335] 很多其它用途可由此形成。注意为了使用断言，不要求值不变。仅要求很多步骤评价可被可能正确的短得多的评价取代。

[0336] 断言比较失败由硬件报告。如果在某些实施例中观察到断言失败，DTSE软件将去除损坏的断言寄存器使用并且在没有失败断言的情况下再次处理代码。

[0337] 注意即使这样，也很可能生成错误代码。线程能结束过程中对栈指针的一些而非全部变化。因此能假设在过程结束时栈指针的错误值。这不是正确性问题。断言将捕捉它，但断言将总是或频繁地失败。如果线程不具有过程的所有栈指针变化，则我们希望它不具有它们中的任何一个。这不是直接强制的。

[0338] 对实际值写入的线程将具有对栈指针的全部变化。这不是常见问题。在一些实施例中，如果有在执行时报告的断言失败，则去除该断言。

[0339] 在一些实施例中，DTSE软件能具体地检查一些但并非全部对线程中假设的不变的改变。如果检测到该有问题的情况，则去除断言。或者，可将值保存在伪的位置，并且在断言值的写入位置再次加载。

[0340] 控制依赖性

[0341] 在一些实施例中，每个概况用于跟踪穿过完全复制代码的线性路径。该概况限定每个分支或跳跃的一般目标，且完全复制的代码中的可用路径限定作为目标的特定实例。因此该跟踪将通过指令的特定实例。概况是线性列表，但它迂回通过完全复制的静态代码。一般而言，它将命中相同的指令实例多次。单独地对于每个分支的每个静态实例，记录其输出边沿中的每一个被采用多少次。

[0342] 如果还未看到来自分支的实例的边沿在任何概况中被采用，则该边沿离开流程。这将致使不能到达一些代码。将分支的单调实例标记为“仅执行的”分支。先前这些中的很多被标识。一般分支可以是单调的。在这种情况下，该一般分支的所有实例是“仅执行的”分支。现在，即使一般分支不是单调的，该分支的某些静态实例可以是单调的。这些实例也是“仅执行的”分支。

[0343] 没有其它的指令实例依赖于“仅执行的分支”。特定的分支实例是或不是“仅执行的”。

[0344] 在一些实施例中，对于一般分支B的每个非仅执行的实例，在所有路径上跟踪向前，停止在B的一般再收敛点的任何实例处。该路径上的所有指令实例被标记为具有对B的该实例的直接依赖性。在一些实施例中，这对于所有的一般分支B进行。

[0345] 可存在具有“离开流程”作为输出边沿的分支，但具有一个以上的其它边沿。这通常用于间接分支。概况分析已经标识间接分支的可能目标中的一些，但通常假设由未被标识的目标。如果间接分支到概况分析中未标识的目标，则这是“离开流程”。

[0346] 在这些情况下，DTSE软件将这打破成至已知目标的分支和是“离开流程”或不是“离开流程”的两路分支。“离开流程”或不“离开流程”的分支是典型的单调“仅执行”分支。

[0347] 直接依赖性

[0348] 在一些实施例中，每个指令实例的直接控制依赖性已经被记录。

[0349] 对于每个指令实例，其“寄存器”输入被标识。这包括执行该指令所需的全部寄存器值。这可包括状态寄存器、条件代码和暗示寄存器值。

[0350] 在一些实施例中，进行从所有可能的路径上的每个指令实例向回的跟踪以找到所需“寄存器”值的所有可能源。源是特定的指令实例，而不是一般指令。特定的指令实例从特定的指令实例获取值。可以有多个源用于指令实例的单个所需值。

[0351] 概况是分支目标、负载地址和尺寸以及存储地址和尺寸的线性序列。DTSE软件应具有至少一个概况来进行依赖性分析。若干概况是可用的。

[0352] 在一些实施例中，每个概况用于跟踪穿过完全复制代码的线性路径。该概况限定每个分支或跳跃的一般目标，且完全复制的代码中的可用路径限定作为目标的特定实例。因此该跟踪将通过指令的特定实例。概况是线性列表，但它迂回通过完全复制的静态代码。一般而言，它将命中相同的指令实例多次。

[0353] 负载从存储频繁加载若干字节。原则上，每个字节是单独的依赖性问题。实际上，当然这可被优化。在一些实施例中，对于每个负载的每个字节，在概况中从负载按反向顺序向回看以找到对该字节最后的在前存储。存在负载指令的相同实例和存储的精确实例。在一些实施例中，该存储实例被记录为该负载实例中的直接依赖性。负载实例可直接依赖于很多存储实例，即使对于同一字节。

[0354] 超级链

[0355] 没有其它指令实例直接依赖于其上的每个指令实例是“超级链的生成器”。

[0356] 超级链是在依赖性下包括一个超级链生成器的静态指令实例集的过渡闭包。即，开始超级链作为包含超级链发生器的集合。在一些实施例中，超级链中依赖于任何指令实例的任何指令实例被添加到该集合。在一些实施例中，这是连续递归，直到超级链包含超级链中的任何指令实例依赖的每个指令实例。

[0357] 在所有的超级链已经从所标识的超级链生成器形成之后，仍可能有不在任何超级链中的一些指令实例。在一些实施例中，不在任何超级链中的任何指令实例被挑选并被指定成为超级链生成器，并且其超级链形成。如果仍有不在任何超级链中的指令实例，则挑选任何这样的指令实例作为超级链生成器。这继续直到每个指令实例在至少一个超级链中。

[0358] 注意很多指令实例将在多个甚至很多超级链中。

[0359] 在一些实施例中，超级链集是依赖性分析的最终产物。

[0360] 跟踪形成。

[0361] 基本跟踪分离

[0362] 在一些实施例中，如果期望N个跟踪，则同时分离N个跟踪。

[0363] 最初种子生成

[0364] 在一些实施例中，找到最常的超级链(即“主链(backbone)”)。

[0365] 对于每次跟踪，在一些实施例中，找到具有最多的不在“主链”中且不在任何其它跟踪中的指令的超级链。这是该跟踪的最初种子。

[0366] 在一些实施例中，在跟踪集周围执行一或二次迭代。对于每次跟踪，在一些实施例中，找到具有最多的不在任何其它跟踪中的指令的超级链。这是该跟踪的下一迭代种子并且替换我们之前具有的种子。对于该细化，如果它真的成为最独特的选择，它可能是(或可能不是)允许“主链”成为种子的好想法。

[0367] 典型地，仅仅是对跟踪“播种”的开始，而非结束。

[0368] 跟踪生成

[0369] 在一些实施例中，挑选跟踪T，它被动态估计为最短。然后将超级链置于该跟踪中。

[0370] 在一些实施例中，将按还不在任何跟踪中的动态指令的估计数量从最小至最大的顺序回顾超级链。

[0371] 在一些实施例中，对于每个超级链，如果与将其放入任何其它跟踪相比，它将导致将其放入跟踪T的复制的一半或更少，则它这样放置并且跟踪生长的开始返回。否则跳过该超级链并且尝试下一超级链。

[0372] 如果已经达到超级链的列表末端而没有将一个放入跟踪T中，则跟踪T需要新种子。

[0373] 新种子

[0374] 在一些实施例汇总，从T以外的所有跟踪去除所有“已生长的”超级链，在这些跟踪中留下所有种子。跟踪T临时保留其“已生长的”超级链。

[0375] 在一些实施例中，从未放置的超级链的当前池，找到具有最大数量的(动态估计的)不在T以外的任何跟踪中的指令的超级链。该超级链是跟踪T中的附加种子。

[0376] 然后从跟踪T去除所有“已生长的”超级链。已经从所有其它跟踪去除“已生长的”超级链。现在所有的跟踪仅包含其种子。在每个跟踪中可能有多个甚至很多种子。

[0377] 从此处，可执行跟踪生长。

[0378] 获取良好种子有助于质量跟踪分离。最长的超级链可能是具有全部的“主链”指令集的一个，这些指令很可能在所有的跟踪中结束。很可能不限定与众不同的指令集。因此，这不是最初被选为种子。

[0379] 在一些实施例中，相反，寻找具有尽可能多不同于“主链”的指令的超级链。这具有更好的机会与众不同。每个连续跟踪获得尽可能与“主链”不同的种子，以便也具有最好的机会与众不同，并且尽可能与现有跟踪不同。

[0380] 在一些实施例中，这再次是迭代的。如果有用于每个跟踪的东西，则如果可能则尝试使每个跟踪更与众不同。每个跟踪中种子的选择被再次视为尽可能与其它跟踪不同。

[0381] 从此处开始，可以有双管齐下的方法。

[0382] “生长”旨在绝对递增。它仅向跟踪中已经存在的添加一点点，且仅当非常清楚它真的属于该跟踪时。“生长”不造成大的跳跃。

[0383] 在一些实施例中，当显然的递增生长停止时，进行至新活动中心的跳跃。为此，添加至在该跟踪中种子的集合。

[0384] 通过添加种子完成大跳跃。生长填充明确伴随种子的东西。一些流程将具有非常好的连续性。从最初的种子开始的递增生长可非常好地工作。一些流程将具有阶段。每个阶段具有种子。然后跟踪将非常好地递增地填充。

[0385] 在一些实施例中，为了找到跟踪T的新种子，除它们的种子外的所有其它跟踪变空。此处可能具有对新种子的不期望偏见。然而我们想要在跟踪T中保持我们有的所有东西。这是已经自然地与T相关联的要素。我们想要的是找到一些不同的东西进入T。它不会帮助我们使我们无论如何将获得的一些东西成为种子。我们需要通过生长我们可能未获得的一些东西来作为种子进行添加。

[0386] 当返回到生长时，在一些实施例中，过程开始清洗。在种子的不同情况下，生长可采用显著不同的过程，且这些种子可被优化。

[0387] 在一些实施例中，生长被执行片刻，正如同用于找到种子需要什么的机制。在流程具有不同阶段的情况下，可能需要在所有不同阶段中的种子。但阶段不是已知的或者需要多少种子。在实施例中，这就是如何找到它。因为“试验”生长仅仅是发现需要什么种子的一种方式，所以它仅仅是投掷方式。当有所需种子的全集时，使高质量“生长”填充每个跟踪中所进行的。

[0388] 原始跟踪代码

[0389] 在一些实施例中，对于每个跟踪，完全复制的流程是开始点。从此处开始，来自该跟踪的代码的每个指令实例(不在分配给该跟踪的任何超级链中)被删除。这是该跟踪的原始代码。

[0390] 一旦限定跟踪的原始代码，则不会进一步使用超级链。超级链仅存在用于确定可将哪些指令实例从每个跟踪的代码中删除。

[0391] 在这点上，所有的跟踪包含所有的完全复制的基本块。在现实中，仅有一般的基本块且它具有很多名字。对于它名字中的每一个，它具有其指令的不同子集。对于每个名字，它具有去往其它一般基本块的不同名字的输出边沿。一些输出边沿是后边沿。一般而言，在其一些或甚至全部名字下的很多基本块将不包含指令。

[0392] 基本块的每个名字具有其自身的输出边沿。每个空基本块实例具有输出边沿。可能在或可能不在基本块的某些名字中的分支和跳跃并不正确地支持该基本块的该名字的输出边沿。存在不包含跳跃或分支指令的基本块的实例(名字)，然而存在该基本块的该实例的输出边沿。存在的分支和跳跃仍具有原始代码目标IP。这也是固定的。目标IP必须被改变以支持输出边沿，但这还未完成。并且对于很多基本块的实例，甚至控制传递指令(跳跃)必须被插入在末端以支持输出边沿。

[0393] 在这一点，所有的跟踪精确地具有相同的控制流程结构和精确相同的基本块实例。它们都是相同的东西，仅仅具有对于每个跟踪不同的指令删除。然而，跟踪的删除可以很大，从整个结构倒空所有指令。例如，环中的所有指令可能已经从跟踪完全消失。

[0394] 跨度标记

[0395] 跨度标记指令是特殊指令，在一些实施例中是对DTSE寄存器的存储，它还指示哪些其它跟踪也在代码的相同位置具有该跨度标记。这稍后将被填充。在生成可执行代码之前，这将不是已知的。

[0396] 在一些实施例中，以展开副本号数字级别的展开副本0为目标的任何后边沿(不是反向后边沿)获得插入在后边沿上的跨度标记。这是仅包含跨度标记的新基本块。改变后边沿以实际上以该新基本块为目标。该新基本块仅具有一个无条件的输出边沿，该输出边沿去往后边沿的先前目标。

[0397] 在一些实施例中，来自这些跨度标记的所有边沿目标将跨度标记仅插入在连接点之前。该新跨度标记不在自后边沿上的跨度标记开始的路径上。它在去往该连接点的所有其它路径上。该跨度标记还是仅包含跨度标记且仅具有去往连接的1个无条件输出边沿的新基本块。

[0398] 在一些实施例中，对于具有反向后边沿的每个分支，该分支的再收敛点添加了跨度标记作为基本块中的第一指令。

[0399] 所有的跨度标记将在所有的跟踪上匹配，因为所有的跟踪具有相同的基本块和相

同的边沿。在可执行代码生成时，相同的跨度标记将从一些跟踪消失。可能必须保持跟踪哪些跨度标记在跟踪上匹配，所以当它们中的一些消失时这将是已知的。

[0400] 可执行代码生成

[0401] 所生成的可执行代码不具有在DTSE软件内使用的标识的静态副本名或信息表。在一些实施例中，它是以地址次序顺序地执行的正常X86指令，除非至不同地址的分支或跳跃被执行。

[0402] 该代码是“池”。它不属于任何特定跟踪或其它的任何东西。如果代码的一部分具有正确的指令序列，则任何跟踪可在跟踪的任何位置使用它。在“池”中，如果所需的代码已经存在，则不需要再次生成相同代码的另一个副本。

[0403] 当然有问题，即一旦在一些代码中开始执行，该代码本身确定将被执行的所有未来代码。假设有某种代码C，它匹配两个不同使用U1和U2的所需指令序列，但在完成C的执行之后，U1需要执行指令序列X，而U2需要执行指令序列Y，且X和Y是不同的。这可能有问题。

[0404] 对于DTSE代码生成，存在该问题的至少两个解决方案。

[0405] 在一些实施例中，第一解决方案是在DTSE软件中生成代码的静态副本所采用的方式使得频繁(但不总是)出现这种情况，即诸如U1和U2之类的不同使用(其需要诸如C之类的相同代码序列一段时间)事实上在此之后将永远想要相同的代码序列。

[0406] 在一些实施例中，第二解决方案是匹配诸如U1和U2之类的多种使用的诸如C之类的代码段可成为DTSE子例程。在子例程内U1和U2使用相同的代码C，但在从该子例程返回后，U1和U2可以不同。而且，代码分析软件创建代码的静态副本所采用的方式使其通常显而易见且易于形成这种子例程。这些子例程对于原始程序不是已知的。

[0407] 构建块

[0408] 代码已经被构造成自然落入圆丘(hammock)。圆丘是成为DTSE子例程的自然候选。

[0409] DTSE子例程不是原始程序已知的过程。注意，DTSE子例程的返回地址通常不被放入架构栈中。除它对于程序不正确外，所有的执行核将共享相同的架构栈，然而，一般而言，它们执行圆丘的不同版本并且需要不同的返回地址。

[0410] 期望使用调用和返回指令来去往DTSE子例程并从其返回，因为硬件非常精确地具有到分支预测返回的特殊结构。在一些实施例中，栈指针在调用前被改变成指向DTSE私有栈，并在执行代码前改回程序栈指针。然后它被改回私有栈指针以便返回。私有栈指针值必须被保存在统一寻址但对于每个逻辑处理器不同的位置中。例如，一般的寄存器是这种存储。但它们用于执行程序。DTSE硬件可提供被统一寻址但访问逻辑处理器专用存储的寄存器。

[0411] 如所指出的，它通常没有必要形成子例程，因为共享代码序列的使用事实上将永远从该点执行相同代码。如果其用户同意“永远”自该点的代码，则将不使可共享的代码序列成为子例程。

[0412] 如果对于圆丘的版本的所有使用在圆丘之后去往相同的代码，在该点通常不需要返回。只要对所有的用户相同，共同代码可被扩展。当用户不再同意执行代码时，需要返回。

[0413] 仅当期望执行足够长以合理减少调用和返回的成本时，才使圆丘成为子例程。如果这不是真的，则不使其成为子例程。

[0414] 内联过程

[0415] 过程被“内联”，生成它们的“副本”。这是递归的，所以仅利用几个调用级和几个调用点，可以有大量的“副本”。另一方面，过程是DTSE子例程的良好候选。可能，在最常见的情况下，对于过程的很多“副本”，它们均证明是相同的(而不是对于不同跟踪为不同指令子集)。或者，证明仅有几个实际不同的版本(而不是对于不同跟踪为不同指令子集)。所以过程变为一个或仅几个DTSE子例程(而不是对于不同跟踪为不同指令子集)。

[0416] 整体环展开

[0417] 在一些实施例中，总是在环的展开副本0中输入环。环被限定为具有单个退出点，即该环的展开副本0中的环分支组的一般共同的再收敛点。这使其成为圆丘。因此，总是使环成为子例程。

[0418] 机会主义子例程

[0419] 分支树的部分可作为在树中重复的圆丘出现。其试验示例是分支的树，且线性静态复制有效地解码成很多线性代码段。许多这些线性代码段包含相同的代码序列一段时间。线性代码序列可总是子例程。

[0420] 代码汇编

[0421] 在一些实施例中，对于每个跟踪，拓扑根是开始点，且从此处处遍历所有的可到达代码和所有可到达边沿。在遍历时生成代码。先前解释了如何从特定的基本块实例前往特定的基本块实例。

[0422] 特定跟踪中的基本块的实例可能不具有指令。然后不生成代码。然而，可以有来自应被维护的该基本块实例的多个输出边沿。

[0423] 如果跟踪中的基本块的实例具有多个输出边沿，但从该跟踪中的该实例删除用于选择输出边沿的分支或间接跳跃，则该跟踪将不包括在该(删除的)分支实例与其再收敛点之间的任何指令。在一些实施例中，遍历不应遵循该跟踪中基本块的该实例的多个输出边沿中的任一个，但相反应直接去往该跟踪中该基本块的末端处的(删除)分支或跳跃的再收敛点。

[0424] 如果有来自基本块实例的单个输出边沿，则不管是否有分支或跳跃，都遵循该边沿。

[0425] 如果在该跟踪的基本块实例的末端处有在多个输出边沿之间进行选择的分支或间接跳跃，则遍历遵循该多个输出边沿。

[0426] 在一些实施例中，当跟踪中的遍历遇到包括用于该跟踪的一个或多个指令的基本块实例时，将存在代码。池中已经存在的代码可被使用或者可将新代码添加到池。在任一种情况下，将待使用代码置于特定地址。然后，该路径上最后生成的代码被固定以去往该地址。可能的情况是，在该路径上的最后在前代码之后，可顺序地放置该代码。然后此处不需要获得任何东西。否则，最后的在前指令可能已经成为分支或跳跃。然后，其目标IP需要被固定以去往正确的位置。该路径上最后的在前代码可能不是分支或跳跃。在这种情况下，需要插入至正确目的地的无条件跳跃。

[0427] 在跟踪中，多数基本块实例通常不可到达。

[0428] 所生成的代码不必且不应具有大量盲目生成的中间形式的静态副本。所生成的代码仅需要具有在每个可到达路径上的正确的指令序列。

[0429] 在遍历中间形式的边缘时，它可从一个静态副本至另一个。静态副本在所生成的

代码中不被区分。一般的想法是尽可能方便地到达正确的指令序列，例如如果已经有具有正确指令序列的代码，则使环闭合回到已经为正确的原始IP生成的代码。另一个示例是去往为不同的静态副本生成的、但具有正确的指令序列的代码。

[0430] 当去往已经在那里的代码时，出现问题。可能是，现有的指令序列在一段时间是正确的，但之后它不再匹配。对于两种不同的情况，代码可去往相同的原始IP，但从相同原始IP需要的代码序列对于两种情况是不同的。

[0431] 线性静态复制

[0432] 在一些实施例中，线性静态复制创建代码的“副本”，以防止在下一个后边沿前，控制流在非环分支的一般再收敛点处物理再接合。这是基本的，直到包含环的下次迭代或包含环的退出。这些趋向于导致很多代码“副本”的分支树。

[0433] 在多数而非全部情况下，在分支的一般再收敛点之后已经被保持分离的代码不会变得不同，而是对于不同跟踪的不同指令子集(期望的不同)。在代码生成时，可将其一起放回(对于对不同跟踪形成的不同指令子集单独地进行)，因为在一般再收敛点处，并且永远从那开始，指令序列相同。副本已经消失。如果代码的可能的很多副本不全相同，它们可能仅落入几种不同的可能性，所以很多静态副本实际上导致所生成代码的仅几个静态副本。

[0434] 即使对于分支B的副本全离开且所生成的代码完全再收敛，精确地如同原始代码那样(除了对不同跟踪形成指令子集以外)，从该静态复制未获得好处不是正确的。该代码是传送依赖性的通道。如果它未被分离，它将产生限制并行性的错误依赖性。必须分离它。除此之外，由于跟踪分离，在B的一般再收敛点之后的代码副本有时(尽管不是通常)是不同的。

[0435] 整体环展开

[0436] 在一些实施例中，整体环展开创建用于嵌套环的代码的很多“副本”。例如，如果有4级嵌套环且仅有2路展开，则有最内部环体的16个副本。很不可能的是，这16个副本均不同。完全相反。环展开具有远小于50％的机会提供任何有用的好处。流程的大部分展开且通常全部展开是没有收益的。没有收益的展开，即大部分展开，通常导致该环的所有副本是相同的(除了对于不同跟踪形成不同指令子集)。因此，在代码生成时，大部分且通常是全部展开被再次一起放回。但有时，几个副本是不同的且对并行性有益。

[0437] 如果来自展开的环体的两个副本是相同的，则在代码生成时，该环的后边沿将去往相同的位置，因为所需的以下指令序列永远相同。该环的展开副本已经消失。如果这是内部环，则在由外部环创建的其很多副本中，这以相同方式发生。

[0438] 如果外部环具有有益展开，则非常可能的是内部环在外部环的多个副本中不是不同的，即使在外部环的副本中有区别。环自然地倾向于形成圆丘。很可能内部环成为子例程。仅有一个它的副本(而不是对于不同跟踪形成不同指令子集)。将从外部环的存活的多个副本中调用它。

[0439] 内联过程

[0440] 在一些实施例中，过程被“内联”，生成它们的“副本”。这是递归的，所以仅利用几个调用级和几个调用点，可以有大量的“副本”。另一方面，过程是DTSE子例程的理想候选。可能，在最常见的情况下，过程的很多“副本”，它们均证明是相同的(而不是对于不同跟踪形成不同指令子集)。或者，证明仅有几个实际不同的版本(而不是对于不同跟踪形成不同

指令子集)。所以过程变为一个或仅几个DTSE子例程(而不是对于不同跟踪形成不同指令子集)。

[0441] 如果过程未被“内联”则它将形成错误依赖性。因此,即使过程被重构为仅一个DTSE子例程(每个跟踪),仍期望它被完全“复制”用于依赖性分析。除此之外,由于跟踪分离,过程的“副本”有时但不经常是不同的。

[0442] 复制存储

[0443] 非常相同的指令最终可出现在多个跟踪中,其中它将被冗余地执行。出现这种情况是因为没有从多个跟踪删除该指令。因为对任何指令这都能发生,所以可以有出现在多个跟踪中的存储,其中它们将被冗余地执行。

[0444] 在一些实施例中,DTSE软件标记在多个跟踪中冗余的同一存储的情况。存储可获得特殊前缀或可在其前面是复制存储标记指令。在一些实施例中,复制存储标记指令可以是对DTSE寄存器的存储。复制存储标记无论采用何种形式必须指示其它跟踪将冗余地执行该相同存储。

[0445] 对齐标记

[0446] 在一些实施例中,如果DTSE硬件检测到从一个以上的跟踪至相同对齐跨度中的相同字节的存储,则它将宣称背离并导致状态恢复到最后的全局提交状态并且流程退出。当然,预期标记的复制存储。DTSE硬件将匹配冗余地执行的所标记复制存储,并且它们将被提交作为单个存储。

[0447] 跨度标记是对齐跨度分隔符。所标记的复制存储是对齐跨度分隔符。对齐标记是对齐跨度分隔符。

[0448] 在一些实施例中,对齐标记是特殊的指令。它是对DTSE寄存器的存储并且指示其它跟踪具有相同的对齐标记。

[0449] 如果存在对多个跟踪中的相同字节的存储,只要冲突存储在不同的对齐跨度中,则硬件可按程序顺序适当地放置这些存储。

[0450] DTSE硬件知道来自相同跟踪的存储器访问的程序顺序。仅当不同跟踪在不同对齐跨度中时,硬件知道不同跟踪中存储器访问的程序顺序。在一些实施例中,如果硬件发现负载可能需要来自不在同一跟踪中执行的存储的数据,则它将宣称违背并导致状态恢复到最后的全局提交状态,并且流程退出。

[0451] 在一些实施例中,DTSE软件将相同形式的对齐标记置于出现在多个跟踪中的存储之间,已经看到该多个跟踪命中相同字节。DTSE软件放置对齐标记,使得看到命中与存储相同的地址的任何负载将被适当地排序至硬件。

[0452] 状态保存和恢复

[0453] 在一些实施例中,在每个跨度标记处建立全局提交点。跨度标记本身将标识符发送到硬件。在一些实施例中,DTSE软件构建表。如果必须恢复状态至最后的全局提交点,则软件将从硬件获得标识符并且在表中查找该全局提交点。DTSE软件将该全局提交点的原始代码IP放置在表中,连同不频繁改变且在代码准备时已知的此代码位置处的其它状态,例如,代码运行的环。其它信息可以是可能已经从最后的全局提交点改变的寄存器。可能此处有一个至软件代码的指针以恢复该状态,因为该代码可被定制用于不同的全局提交点。

[0454] 在一些实施例中,将代码添加到每个跨度标记以保存任何需要保存的数据,使得

在需要时可恢复状态。这可能包括至少一些寄存器值。

[0455] 在一些实施例中，可能定制到全局提交点的代码被添加以恢复状态。将代码的指针置于表中。

[0456] 相对频繁地遇到全局提交点，但状态恢复较不频繁。有利的是当必须执行实际状态恢复时，以甚至大大增加工作为代价最小化全局提交点处的工作。

[0457] 因此，对于依赖性分析和跟踪分离的一些实施例，代码均被扩展至很多“副本”。在可执行代码生成时，它主要被再次一起放回。

[0458] 逻辑处理器管理

[0459] 可利用一组核实现DTSE，该组核具有多个同时多线程化硬件线程，例如每个核2个同时多线程化硬件线程。DTSE系统可创建更多的逻辑处理器，使得每个核看似具有例如四个逻辑处理器而不是仅两个。此外，DTSE系统可有效地管理核资源以实现逻辑处理器。最后，如果DTSE已经将一些代码流分解成多个线程，则这些线程可运行在逻辑处理器上。

[0460] 为了在具有例如2个同时多线程化硬件线程的核上实现例如四个逻辑处理器，在一些实施例中，DTSE系统将为例如两个逻辑处理器(其在核硬件中不具有其状态)保持处理器状态。DTSE系统将时常地切换每个同时多线程化硬件线程中的状态。

[0461] DTSE将生成用于每个软件线程的代码。视具体情况，DTSE可完成线程分解以从单个原始代码流形成若干线程，或DTSE可仅从单个原始代码流形成单个线程。无论如何，对于单个原始代码流以相同方式生成代码。在跟踪分离时，可将代码分成一个以上的线程，或跟踪分离可将所有的代码放入相同的单个跟踪。

[0462] 在生成可执行代码之前，可在代码上完成附加工作，包括指令的添加，以实现逻辑处理器管理。

[0463] 在一些实施例中，DTSE硬件将提供统一寻址的至少一个存储位置，但事实上对于执行访问的每个同时多线程化硬件线程这将访问不同的存储。在实施例中，这是处理器通用寄存器，诸如RAX。这由运行在任何核上的任何同时多线程化硬件线程访问，因为“RAX”但存储位置以及因此的数据对于执行对“RAX”的访问的每个同时多线程化硬件线程是不同。在实施例中，处理器通用寄存器用于运行程序代码，所以DTSE需要一些其它的DTSE硬件将提供的同时多线程化硬件线程专用存储。对于每个DTSE逻辑模块的同时多线程化硬件线程，这将是例如一个或几个寄存器。

[0464] 具体地，在一些实施例中，同时多线程化硬件先用专用存储寄存器ME将包含至当前运行在该同时多线程化硬件线程中的逻辑处理器的状态保存表的指针。在此位置的表将包含某些其它信息，诸如至运行的下一逻辑处理器的保存区的指针以及至运行在该同时多线程化硬件线程保存表上的先前逻辑处理器的指针。

[0465] 对于所有的原始代码流，对于所有的线程，DTSE生成的全部代码在相同的地址空间中。因此，任何原始代码流的任何生成的代码可跳至任何原始代码流的所生成代码。DTSE专用数据也全在相同的地址空间中。一般而言，程序数据空间在每个原始代码流的不同地址空间内。

[0466] 有效线程切换。

[0467] 在一些实施例中，DTSE将在为其生成代码的每个线程中插入HT切换入口点和退出点。因此，在硬件部分讨论这种入口点的使用。

[0468] HT切换入口点

[0469] 在一些实施例中，HT切换入口点处的代码将从ME读取至其本身的保存表的指针，然后是至下一逻辑处理器保存表的指针。从该表，可获得下一HT切换入口点的IP以转到后续被处理的入口点。代码可使用特殊的指令，该指令将该地址推入分支预测器中的返回预测栈。任选地，在该地址且可能在附加的地址发出预取。这全部是对下一HT切换的设置，将在该当前HT切换入口点之后进行下一HT切换。现在需要设置返回预测器，所以下一HT切换将被正确地预测。如果在下一HT切换之后，可能有I高速缓存未命中，在这点行应发出预取，以在下一HT线程切换时使该I流在I高速缓存中。代码然后将在该点从其自身的保存表中读取其所需状态，并且在该HT切换入口点之后重新开始执行代码。这可包括当需要时加载CR3、EPT和段寄存器。有利的是，例如，使共享相同的同时多线程化硬件线程的逻辑处理器具有相同的地址空间，因为它们均运行来自相同过程的线程，使得它不必将这些寄存器重新加载在HT切换上，尽管这不是必须的。

[0470] HT切换退出点

[0471] 在一些实施例中，在HT切换退出点处的代码将从ME读取至其自身的保存表的指针。它将存储所需的状态用于恢复到其自身的保存表。它然后将从其自身的保存表读取至下一逻辑处理器的保存表的指针，以将其运行并写入到ME。它读取要去的下一HT切换入口点的IP，并将它推到栈上。它进行返回指令，以执行至所需HT切换入口点的完全预测跳跃。

[0472] 注意，在HT切换退出点处的代码在其再次获得要运行的同时多线程化硬件线程时，具有对IP的控制，它在该IP处重新开始。它可将它想在IP中的任何东西放在其自身的保存表中。

[0473] 有效的不可预测间接分支

[0474] 在一些实施例中可由DTSE通过改变间接分支以仅计算分支目标来有效地完成不可预测间接分支。它之后是HT切换退出点，但存储至保存表的所计算的分支目标。

[0475] 当该线程切换回时，它将自然地去往间接分支的正确目标。这可完成，且对于间接分支或HT切换，无分支未命中预测和无I高速缓存未命中。

[0476] 对于逻辑处理器的切换资源

[0477] 在一些实施例中有特殊的指令或前缀，停止获取直到分支报告。就在分支或间接跳跃之前，可插入该指令。

[0478] 当停止获取直到分支报告被解码时，只要其它同时多线程化硬件线程正前进，对于该I流的指令获取停止，且在该I流的下一后续指令之后没有指令将被解码。如果其它同时多线程化硬件线程不前进，则该指令被忽略。以下指令应是分支或间接跳跃。它被加标签。分支和跳跃在执行时报告它们被正确预测或未命中预测。当加标签的分支报告时，对于该I流的指令获取和解码被重新开始。当在该同时多线程化硬件线程中的任何分支报告未命中预测时，指令获取和解码重新开始。

[0479] 在一些实施例中，有特殊的指令或前缀，停止获取直到负载报告。可在负载之后的某一时间插入该指令。它具有操作数，将使该操作数为负载的结果。停止获取直到负载报告指令实际执行。它将报告它何时执行而不被取消。有两种形式的停止获取直到负载报告指令：有条件和无条件。

[0480] 无条件停止获取直到负载报告指令在被解码时将停止指令获取和解码。有条件停

止获取直到负载报告指令在被解码时仅当其它同时多线程化硬件线程前进时停止该I流上的指令获取和解码。当指令报告未取消的执行时，两种形式的指令重新开始该I流上的指令获取和解码，且对于该I流没有显著的数据高速缓存未命中。

[0481] 代码分析

[0482] 如果该执行实例被未命中预测或正确预测，则闪速概况分析将针对每个单独的分支或跳跃执行实例进行指示。它将指示获取I高速缓存未命中、二级高速缓存未命中和对DRAM未命中的指令执行实例。如果该执行实例获得数据高速缓存未命中、二级高速缓存未命中或对DRAM的未命中，则它将针对每个负载执行实例进行指示。

[0483] DTSE软件进行的所有形式的静态复制还可用于逻辑处理器管理。在一些实施例中，负载、分支和间接跳跃的所有静态实例获取未命中数。在这些实施例中指令的静态实例获取高速缓存未命中数。

[0484] 相同指令的不同静态实例(通过原始IP)经常具有不同的未命中行为，因此通常最好使用指令的静态实例。指令的实例越多，每个实例的未命中率数为高或低的机会越好。中间未命中率数更难以处理。

[0485] 尽管尽最大努力且尽管与只使用IP相比有很大改进，然而可能仍有很多指令实例具有中间范围的未命中数。在一些实施例中，分组是处理中间范围未命中数的一种方式。各自具有中间范围的未命中预测率的小分支树可在穿过该树的执行路径上的某处呈现大概率的某种未命中预测。类似地，各自具有中间范围的高速缓存未命中率的若干负载的顺序串可呈现在至少一个负载上大概率的未命中。

[0486] 环展开是编组机制。在环迭代中的单个负载可具有中间范围高速缓存未命中率。如果将若干环迭代上负载的多次执行作为一组，则它可呈现在这些迭代中的至少一个中的高概率的高速缓存未命中。迭代内的多个负载自然与编组的多个迭代组合在一起。

[0487] 在一些实施例中，DTSE软件创建组，使得每个组具有相对高概率的某类未命中。有时组可被压缩。对于分支树尤其如此。分支树中的稍后分支可通过静态复制分支之前使用但现在在该分支之后的指令来向上移动。这将树中的分支更紧密地压紧。

[0488] 如果组仅仅非常可能获取分支未命中预测，它一般不值得HT切换。在一些实施例中，停止获取直到分支报告在该路径上的最后组分支之前被插入到组外的路径上。执行路径上的组中的分支将被解码，然后解码将停止，只要其它同步多线程化硬件线程前进。这将核资源给予其它同时多线程化硬件线程。如果组中没有未命中预测，当执行路径上的最后组分支报告时，将再次开始获取和解码。否则，分支报告未命中预测之后，获取将在正确的目标地址重新开始。这不是非常完美的，因为分支可能不按顺序报告。

[0489] 然而，HT切换用于具有高概率的未命中预测的间接分支，如之前描述的。

[0490] 类似地，如果组仅仅非常可能获取数据高速缓存未命中，它一般优选不进行HT切换。如果可能，在一些实施例中，组中的负载将被移动，使得所有负载在任何负载的第一消费者之前。在一些实施例中，使有条件停止获取直到负载报告指令依赖于组中最后的负载并被置于负载之后但在任何消费者之前。

[0491] 如果数据高速缓存数据高速缓存未命中几乎确定，但它仅仅是数据高速缓存未命中，可使用无条件停止获取直到负载报告指令。

[0492] 通常，组中的负载一般不被放在任何消费者之前。例如，如果组是环的展开迭代，

则这不起作用。在这种情况下，期望使组足够大，使得至少一个且优选为若干个数据高速缓存未命中几乎不可避免。如果组是环的展开迭代，则这一般可实现。在一些实施例中，生成预取集以覆盖组中的负载。首先放置预取，然是HT切换然后是代码。

[0493] 具有高概率的二级高速缓存未命中、D流或I流的组被证明且HT切换。首先放置预取，然是HT切换，然后是代码。

[0494] 甚至大约30％的DRAM未命中可能性可证明HT切换是正当的。在这些实例中，在一些实施例中，首先进行预取，然后是HT切换。仍然优选的是更多分组以使未命中概率更高，且如果可覆盖若干未命中则更好。

[0495] 在一些实施例中，在HT切换发生时，其它同时的多线程化硬件线程上的工作被“覆盖”。目标是总是使一个同时的多线程化硬件线程进行实际工作。

[0496] 如果一个同时的多线程化硬件线程进行实际工作，而其它停止获取，在运行同时多线程化硬件线程的任何时间存在问题风险。所以一般而言，不会长时间仅依赖单个工作的同时多线程化硬件线程。另外，长的停止获取通常不是期望的。如果进行得太长，在一些实施例中进行HT切换，所以当其遇到阻碍时工作的同时多线程化硬件线程被另一个支持。

[0497] 示例性计算机系统和处理器

[0498] 图12是示出根据本发明的实施例的核的示例性无序架构的框图。然而，上述指令也可实现在有序架构中。在图12中，箭头指示两个或更多个单元之间的耦合，且箭头的方向指示这些单元之间的数据流的方向。该架构的组件可用于处理以上详述的指令，包括这些指令的获取、解码和执行。

[0499] 图12包括耦合到执行引擎单元1210和存储器单元1215的前端单元1205；执行引擎单元1210还耦合到存储器单元1215。

[0500] 前端单元1205包括耦合到二级(L2)分支预测单元1222的一级(L1)分支预测单元1220。这些单元允许核获取并执行指令而不等待分支被解析。L1和L2分支预测单元1220和1222耦合到L1指令高速缓存单元1224。L指令高速缓存单元1224保持将可能由执行引擎单元1210执行的指令或一个或多个线程。

[0501] L1指令高速缓存单元1224耦合到指令转换后备缓冲器(ITLB)1226。ITLB 1226耦合到指令获取和预解码单元1228，该指令获取和预解码单元1228将字节流分成分立指令。

[0502] 指令获取和预解码单元1228耦合到指令队列单元1230以存储这些指令。解码单元1232解码包括上述指令的排队指令。在一些实施例中，解码单元1232包括复杂解码器单元1234和三个简单解码器单元1236、1238和1240。简单解码器可处理大多数(如果不是全部的话)x86指令，其解码成单个微操作。复杂解码器可解码映射到多个微操作的指令。解码单元1232还可包括微代码ROM单元1242。

[0503] L1指令高速缓存单元1224还耦合到存储器单元1215中的L2高速缓存单元1248。指令TLB单元1226还耦合到存储器单元1215中的二级TLB单元1246。解码单元1232、微代码ROM单元1242和环流检测器(LSD)单元1244各自耦合到执行引擎单元1210中的重命名/分配器单元1256。LSD单元1244检测何时执行软件中的环，停止预测分支(及可能不正确预测环的最后分支)以及其外部的流指令。在一些实施例中，LSD 1244高速缓存微操作。

[0504] 执行引擎单元1210包括耦合到退役单元1274和统一调度器单元1258的重命名/分配器单元1256。重命名/分配器单元1256在任何寄存器重命名之前确定所需资源并分配可

用资源用于执行。该单元还将逻辑寄存器重命名至物理寄存器文件的物理寄存器。

[0505] 退役单元1274还耦合到执行单元1260且包括记录器缓冲器单元1278。该单元在指令完成时使指令退役。

[0506] 统一调度器单元1258还耦合到物理寄存器文件单元1276，物理寄存器文件单元1276耦合到执行单元1260。该调度器在处理器上运行的不同线程之间共享。

[0507] 物理寄存器文件单元1276包括MSR单元1277A、浮点寄存器单元1277B和整数寄存器单元1277C，且可包括未示出的附加寄存器文件(例如，混叠在MMX打包整数平面寄存器文件550上的标量浮点栈寄存器文件545)。

[0508] 执行单元1260包括三个混合标量和SIMD执行单元1262、1264和1272；负载单元1266；存储地址单元1268；存储数据单元1270。负载单元1266、存储地址单元1268和存储数据单元1270执行负载/存储和存储器操作，且各自进一步耦合到存储器单元1215中的TLB单元1252。

[0509] 存储器单元1215包括耦合到数据TLB单元1252的二级TLB单元1246。数据TLB单元1252耦合到L1数据高速缓存单元1254。L1数据高速缓存单元1254还耦合到L2高速缓存单元1248。在一些实施例中，L2高速缓存单元1248还耦合到存储器单元1215内部和/或外部的L3和更高级高速缓存单元1250。

[0510] 以下是适用于执行本文详述的指令的示例性系统。对于膝上计算机、台式机、手持PC、个人数字助理、工程师工作站、服务器、网络设备、网络集线器、交换器、嵌入式处理器、数字信号处理器(DSP)、图形设备、视频游戏设备、机顶盒、微控制器、蜂窝电话、便携式媒体播放器、手持设备以及各种其它电子设备，其它业内已知的系统设计和配置也是适用的。一般而言，本文中公开的各种能够合并处理器和/或其它执行逻辑的系统或电子设备一般是适用的。

[0511] 现在参考图13，所示出的是根据本发明一实施例的系统1300的框图。系统1300可包括耦合至图形存储器控制器中枢(GMCH)1320的一个或多个处理元件1310、1315。附加的处理元件1315的任选性在图13中通过虚线来表示。

[0512] 每个处理元件可以是单核，或可替代地包括多核。处理元件可任选地包括除处理核之外的其它片上元件，诸如集成存储器控制器和/或集成I/O控制逻辑。此外，对于至少一个实施例，处理元件的(多个)核可多线程化，因为它们对每个核可包括一个以上的硬件线程上下文。

[0513] 图13示出GMCH 1320可耦合至存储器1340，该存储器1340可以是例如动态随机存取存储器(DRAM)。对于至少一个实施例，DRAM可以与非易失性高速缓存相关联。

[0514] GMCH 1320可以是芯片组或芯片组的一部分。GMCH 1320可以与(多个)处理器1310、1315进行通信，并控制处理器1310、1315和存储器1340之间的交互。GMCH 1320还可担当(多个)处理器1310、1315和系统1300的其它元件之间的加速总线接口。对于至少一个实施例，GMCH 1320经由诸如前端总线(FSB)1395之类的多点总线与(多个)处理器1310、1315进行通信。

[0515] 此外，GMCH 1320耦合至显示器1345(诸如平板显示器)。GMCH 1320可包括集成图形加速器。GMCH 1320还耦合至输入/输出(I/O)控制器中枢(ICH)1350，该输入/输出(I/O)控制器中枢(ICH)1350可用于将各种外围设备耦合至系统1300。在图13的实施例中作为示

例示出了外部图形设备1360以及另一外围设备1370，该外部图形设备1360可以是耦合至ICH 1350的分立图形设备。

[0516] 替代地，系统1300中还可存在附加或不同的处理元件。例如，附加(多个)处理元件1315可包括与处理器1310相同的附加处理器、与处理器1310异类或不对称的附加(多个)处理器、加速器(诸如例如图形加速器或数字信号处理(DSP)单元)、现场可编程门阵列或任何其它处理元件。按照包括架构、微架构、热、功耗特征等等优点的度量谱，物理资源1310、1315之间存在各种差别。这些差别会有效显示为处理元件1310、1315之间的不对称性和异类性。对于至少一个实施例，各种处理元件1310、1315可驻留在同一管芯封装中。

[0517] 现在参照图14，所示出的是根据本发明一实施例的第二系统1400的框图。如图14所示，多处理器系统1400是点对点互连系统，并且包括经由点对点互连1450耦合的第一处理元件1470和第二处理元件1480。如图14所示，处理元件1470和1480中的每一个都可以是多核处理器，包括第一和第二处理器核(即，处理器核1474a与1474b以及处理器核1484a与1484b)。

[0518] 替代地，处理元件1470、1480中的一个或多个可以是除处理器之外的元件，诸如加速器或现场可编程门阵列。

[0519] 虽然仅以两个处理元件1470、1480来示出，但应理解本发明的范围不限于此。在其它实施例中，在给定处理器中可存在一个或多个附加处理元件。

[0520] 第一处理元件1470还可包括存储器控制器中枢(MCH)1472和点对点(P-P)接口1476和1478。类似地，第二处理元件1480可包括MCH 1482与P-P接口1486和1488。处理器1470、1480可以经由使用点对点(PtP)接口电路1478、1488的点对点(PtP)接口1450来交换数据。如图14所示，MCH 1472和1482将处理器耦合到相应的存储器，即存储器1442和存储器1444，这些存储器可以是本地附连到相应处理器的主存储器部分。

[0521] 处理器1470、1480可各自经由使用点对点接口电路1476、1494、1486、1498的单独PtP接口1452、1454与芯片组1490交换数据。芯片组1490还可经由高性能图形接口1439与高性能图形电路1438交换数据。本发明的实施方式可以置于具有任意数目的处理核的任意处理器中，或置于图14的PtP总线代理中的每一个中。在一个实施例中，任意处理器核可包括本地高速缓存存储器(未示出)或者以其它方式关联于本地高速缓存存储器(未示出)。此外，共享高速缓存(未示出)可被包括于在这两个处理器的外部但经由p2p互连与这些处理器连接的任一处理器中，从而如果一处理器被置于低功率模式，则任一个或这两个处理器的本地高速缓存信息可被存储在该共享的高速缓存中。

[0522] 第一处理元件1470和第二处理元件1480可分别经由P-P互连1476、1486和1484耦合到芯片组1490。如图14所示，芯片组1490包括P-P接口1494和1498。此外，芯片组1490包括将芯片组1490与高性能图形引擎1448耦合的接口1492。在一个实施例中，总线1449可被用于将图形引擎1448耦合到芯片组1490。替代地，点对点互连1449可耦合这些部件。

[0523] 芯片组1490又经由接口1496耦合至第一总线1416。在一个实施例中，第一总线1416可以是外围部件互连(PCI)总线或诸如PCI Express总线或另一第三代I/O互连总线之类的总线，虽然本发明的范围不限于此。

[0524] 如图14所示，各种I/O设备1414可连同总线桥1418一起耦合到第一总线1416，总线桥1418将第一总线1416耦合到第二总线1420。在一个实施例中，第二总线1420可以是低引

脚数(LPC)总线。多个设备可耦合至第二总线1420，包括例如键盘/鼠标1422、通信设备1426以及数据存储单元1428(诸如盘驱动器或其它大容量存储设备，在一个实施例中其可包括代码1430)。此外，音频I/O 1424可耦合至第二总线1420。注意，其它体系结构是可能的。例如，代替图14的点对点架构，系统可实施多点总线或另一此类架构。

[0525] 现在参照图15，所示出的是根据本发明实施例的第三系统1500的框图。图14和15中的类似元件使用类似附图标记，且在图15中省略了图14的某些方面以避免混淆图15的其它方面。

[0526] 图15示出处理元件1470、1480可分别包括集成存储器和I/O控制逻辑(“CL”)1472和1482。对于至少一个实施例，CL 1472、1482可包括诸如以上结合图13和14所描述的存储器控制器中枢逻辑(MCH)。此外。CL 1472、1482还可包括I/O控制逻辑。图15示出不仅存储器1442、1444耦合至CL 1472、1482，而且I/O设备1514也耦合至控制逻辑1472、1482。传统I/O设备1515耦合至芯片组1490。

[0527] 本文中公开的机构的实施例可按照硬件、软件、固件或此类实现方法的组合来实现。本发明的实施例可被实现为在包括至少一个处理器、数据储存器系统(包括易失性和非易失性存储器和/或储存元件)、至少一个输入设备以及至少一个输出设备的可编程系统上执行的计算机程序。

[0528] 可将诸如图14中所示的代码1430的程序代码应用于输入数据以执行本文中所描述的功能，并产生输出信息。可按照已知方式将输出信息应用于一个或多个输出设备。为了此应用的目的，处理系统包括具有诸如例如数字信号处理器(DSP)、微控制器、专用集成电路(ASIC)或微处理器之类的处理器的任意系统。

[0529] 程序可按照高级过程或面向对象的高级编程语言来实现，以与处理系统通信。程序代码在需要时还可按照汇编或机器语言来实现。实际上，本文中描述的机制在范围上不限于任何特定编程语言。在任何情况下，该语言可以是编译或解释语言。

[0530] 至少一个实施例的一个或多个方面可以由存储在机器可读介质上的代表性数据来实现，该数据表示处理器中的各种逻辑，其在被机器读取时使得该机器生成执行本文描述的技术的逻辑。被称为“IP核”的这些表示可以被存储在有形的机器可读介质上，并被提供给各个顾客或生产设施以加载到实际制造该逻辑或处理器的制造机器中。

[0531] 此类机器可读存储介质可包括但不限于通过机器或设备制造或形成的粒子的有形排列，包括存储介质，诸如：硬盘；包括软盘、光盘、压缩盘只读存储器(CD-ROM)、可重写压缩盘(CD-RW)以及磁光盘的任何其它类型的盘；诸如只读存储器(ROM)之类的半导体器件；诸如动态随机存取存储器(DRAM)、静态随机存取存储器(SRAM)之类的随机存取存储器(RAM)；可擦除可编程只读存储器(EPROM)；闪存；电可擦除可编程只读存储器(EEPROM)；磁卡或光卡；或适于存储电子指令的任何其它类型的介质。

[0532] 因此，本发明的实施例也包括非瞬态有形机器可读介质，该介质包含诸如HDL之类的设计数据，该设计数据限定本文中描述的结构、电路、装置、处理器和/或系统特征。此类实施例也可被称为程序产品。

[0533] 本文公开的指令的某些操作可由硬件组件执行，且可体现在机器可执行指令中，该指令用于导致或至少致使电路或其它硬件组件以执行该操作的指令编程。电路可包括通用或专用处理器、或逻辑电路，这里仅给出几个示例。操作也可任选地由硬件和软件的组合

来执行。执行逻辑和/或处理器可包括响应于从机器指令导出的机器指令或一个或多个控制信号以存储指令指定的结果操作数的专用或特定电路或其它逻辑。例如，本文公开的指令的实施例可在图13、14和15的一个或多个系统中执行，且指令的实施例可存储在将在系统中执行的程序代码中。

[0534] 上述描述旨在说明本发明的优选实施例。根据上述讨论，还应当显而易见的是，在发展迅速且进一步的进展难以预见的此技术领域中，本领域技术人员可在安排和细节上对本发明进行修改，而不背离落在所附权利要求及其等价方案的范围内的本发明的原理。例如，方法的一个或多个操作可组合或进一步分开。

[0535] 可选实施例

[0536] 尽管已经描述了将自然执行本文公开的指令的实施例，但本发明的可选实施例可通过运行在执行不同指令集的处理器(例如，执行美国加利福亚州桑尼维尔的MIPS技术的MIPS指令集的处理器、执行加利福亚州桑尼维尔的ARM保持的ARM指令集的处理器)上的仿真层来执行指令。同样，尽管附图中的流程图示出本发明的某些实施例的特定操作顺序，按应理解该顺序是示例性的(例如，可选实施例可按不同顺序执行操作、组合某些操作、使某些操作重叠等)。

[0537] 在以上描述中，为解释起见，阐明了众多具体细节以提供对本发明的实施例的透彻理解。然而，将对本领域技术人员明显的是，在没有这些具体细节中的一些的情况下，也可实践一个或多个其他实施例。提供所描述的具体实施例不是为了限制本发明而是为了说明本发明的实施例。本发明的范围不是由前面提供的具体示例来确定的，而是仅由所附权利要求来确定的。

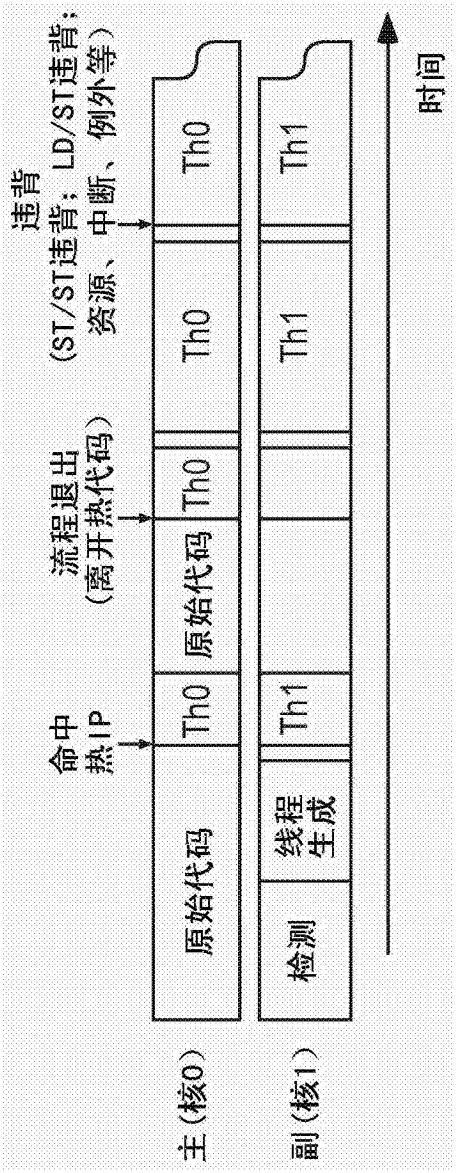

违背
(ST/ST违背；LD/ST违背；
资源、中断、例外等)
流程退出
(离开热代码)
命中
热IP
原始代码
Th0
原始代码
Th0
Th0
检测
线程
生成
Th1
Th1
Th1
主(核0)
副(核1)
时间

图1

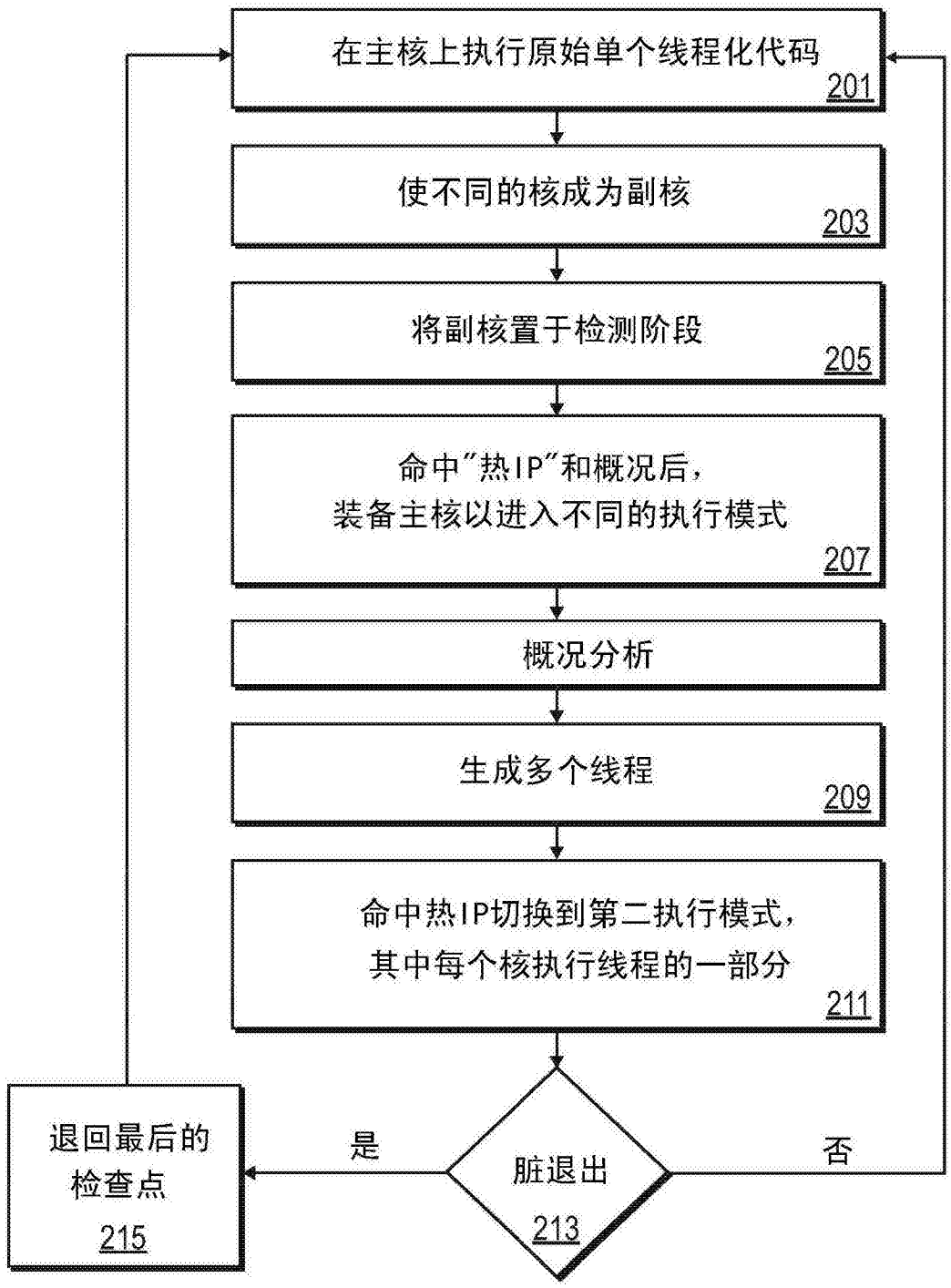

在主核上执行原始单个线程化代码
201
使不同的核成为副核
203
将副核置于检测阶段
205
命中"热IP"和概况后，
装备主核以进入不同的执行模式
207
概况分析
生成多个线程
209
命中热IP切换到第二执行模式，
其中每个核执行线程的一部分
211
脏退出
213
是
否
退回最后的
检查点
215

图2

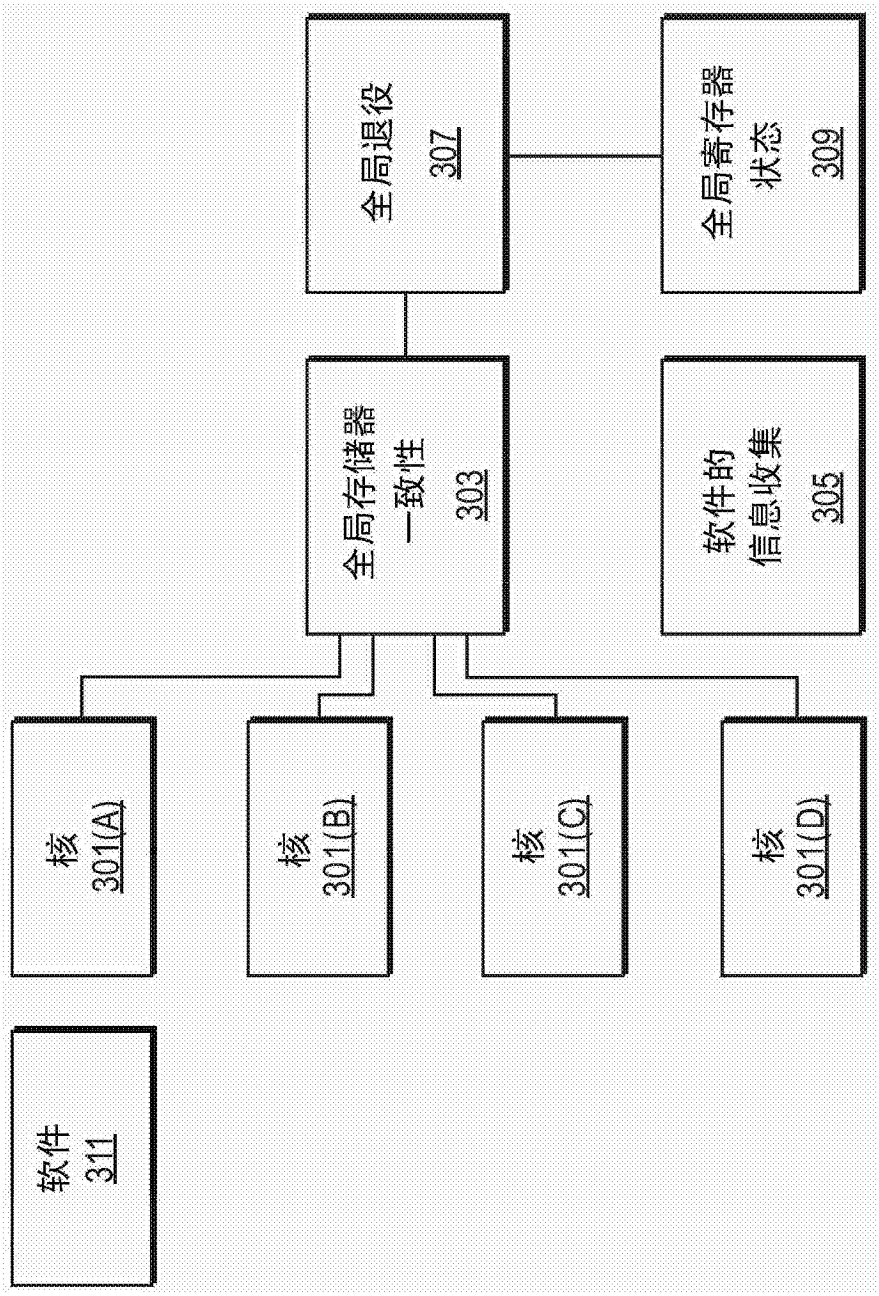

全局退役
307
全局寄存器
状态
309
全局存储器
一致性
303
软件的
信息收集
305
核
301(A)
核
301(B)
核
301(C)
核
301(D)
软件
311

图3

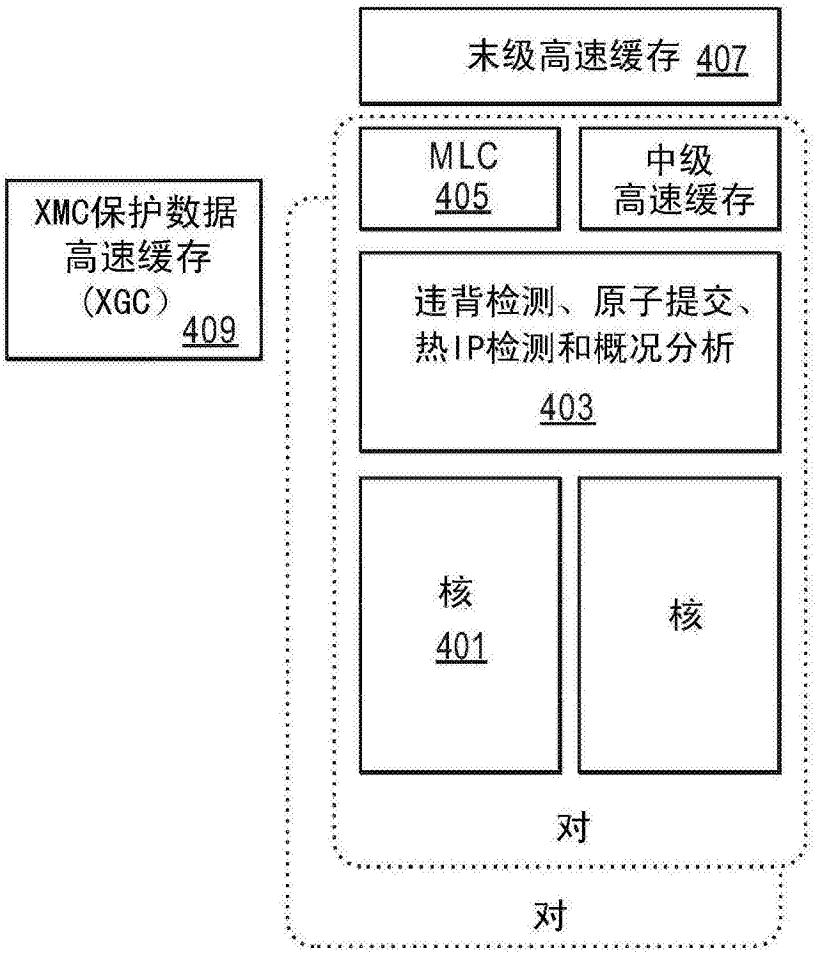

末级高速缓存 407
MLC 405
中级高速缓存
XMC保护数据高速缓存（XGC） 409
违背检测、原子提交、热IP检测和概况分析 403
核 401
核
对
对

图4

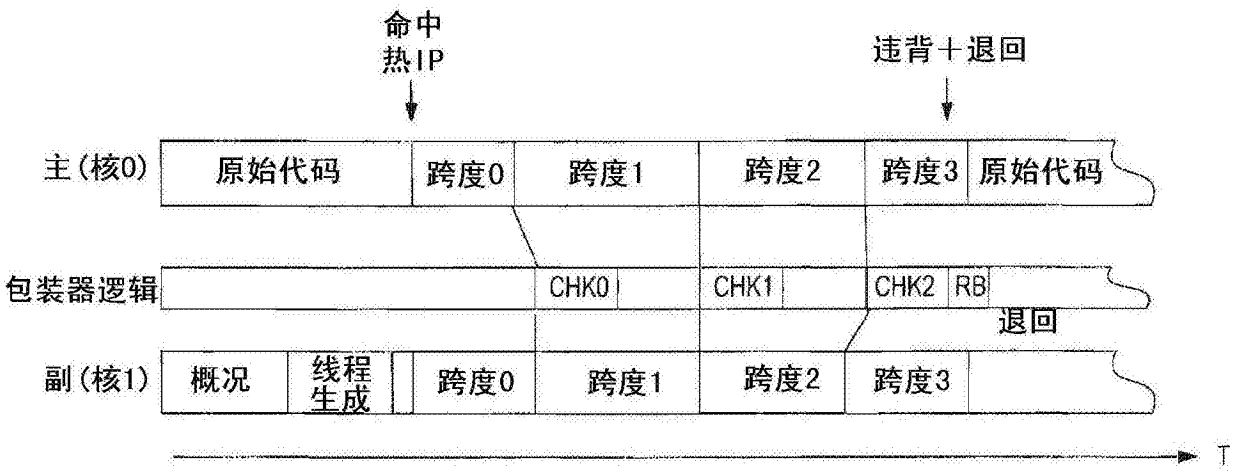

命中热IP
违背＋退回
主(核0)
原始代码
跨度0
跨度1
跨度2
跨度3
原始代码
包装器逻辑
CHK0
CHK1
CHK2
RB
退回
副(核1)
概况
线程生成
跨度0
跨度1
跨度2
跨度3
T

图5

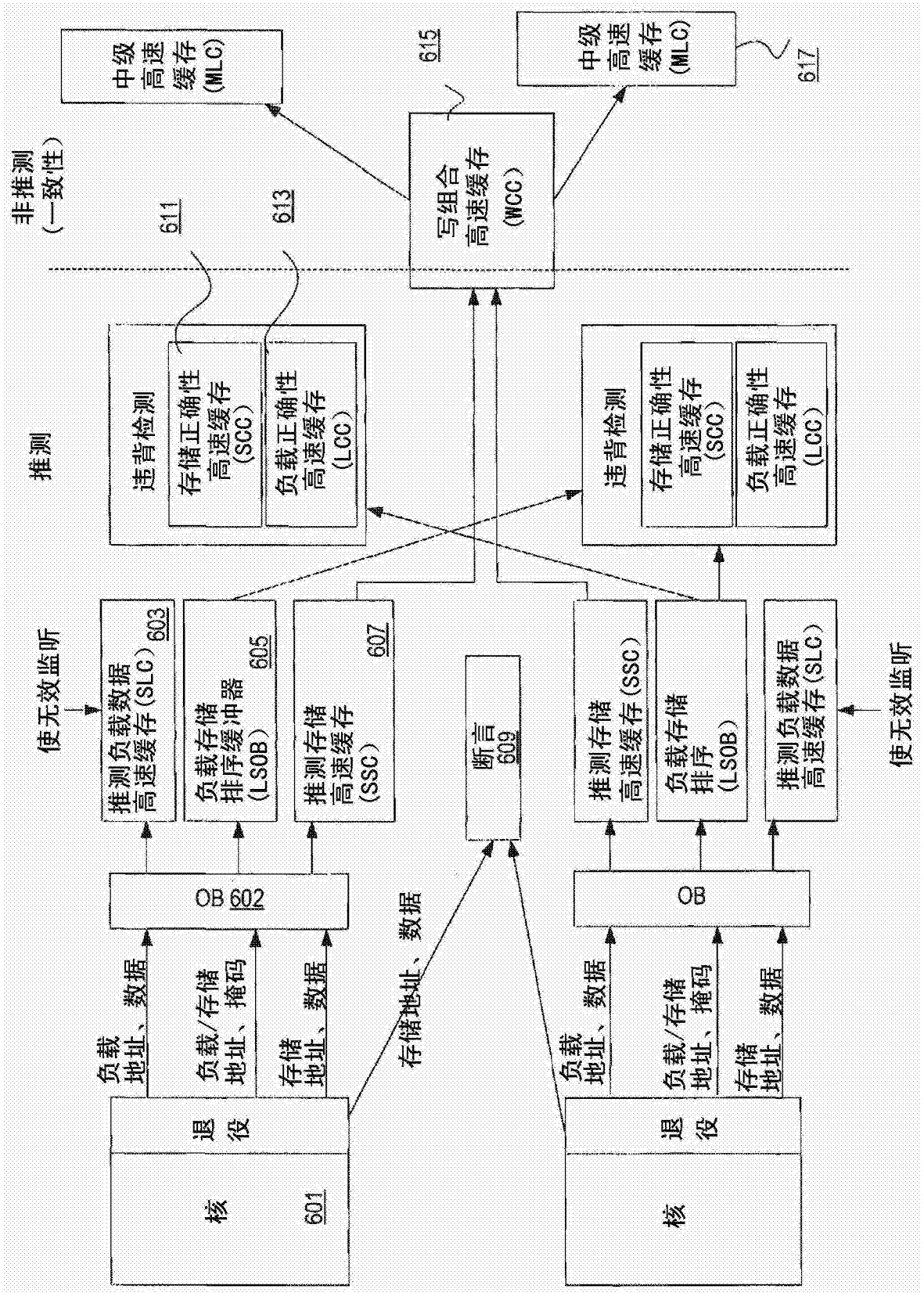

中级高速缓存(MLC)
615
中级高速缓存(MLC)
617
非推测(一致性)
611
613
写组合高速缓存(WCC)
推测
违背检测
存储正确性高速缓存(SCC)
负载正确性高速缓存(LCC)
违背检测
存储正确性高速缓存(SCC)
负载正确性高速缓存(LCC)
使无效监听
推测负载数据高速缓存(SLC) 603
负载存储排序缓冲器(LSOB) 605
推测存储高速缓存(SSC) 607
断言 609
推测存储高速缓存(SSC)
负载存储排序(LSOB)
推测负载数据高速缓存(SLC)
使无效监听
OB 602
OB
负载地址、数据
负载/存储地址、掩码
存储地址、数据
存储地址、数据
负载地址、数据
负载/存储地址、掩码
存储地址、数据
退役
核 601
退役
核

图6

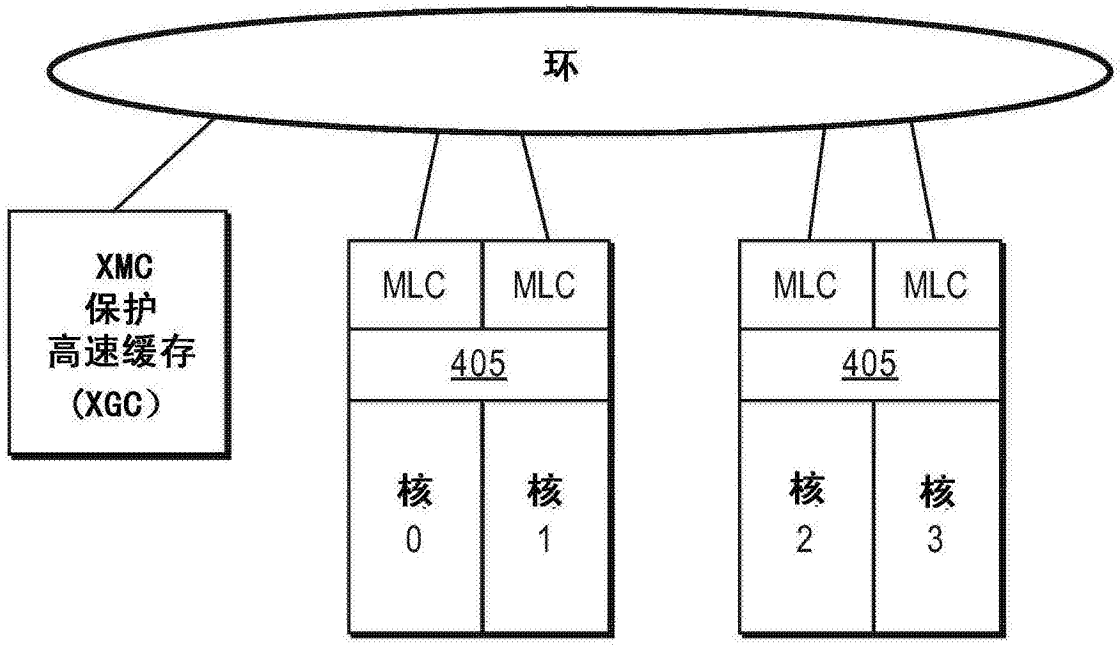


图7

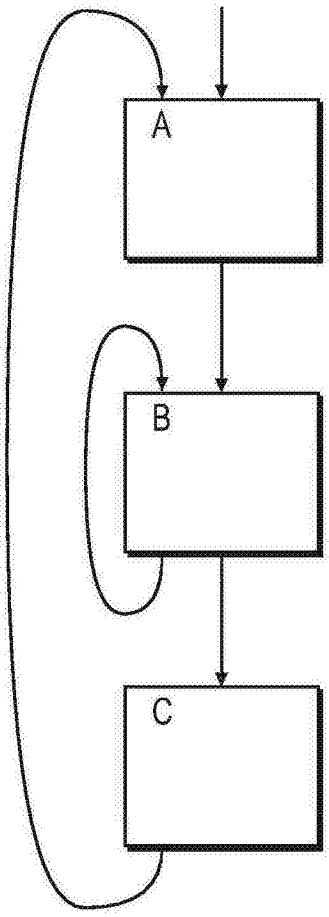


图8

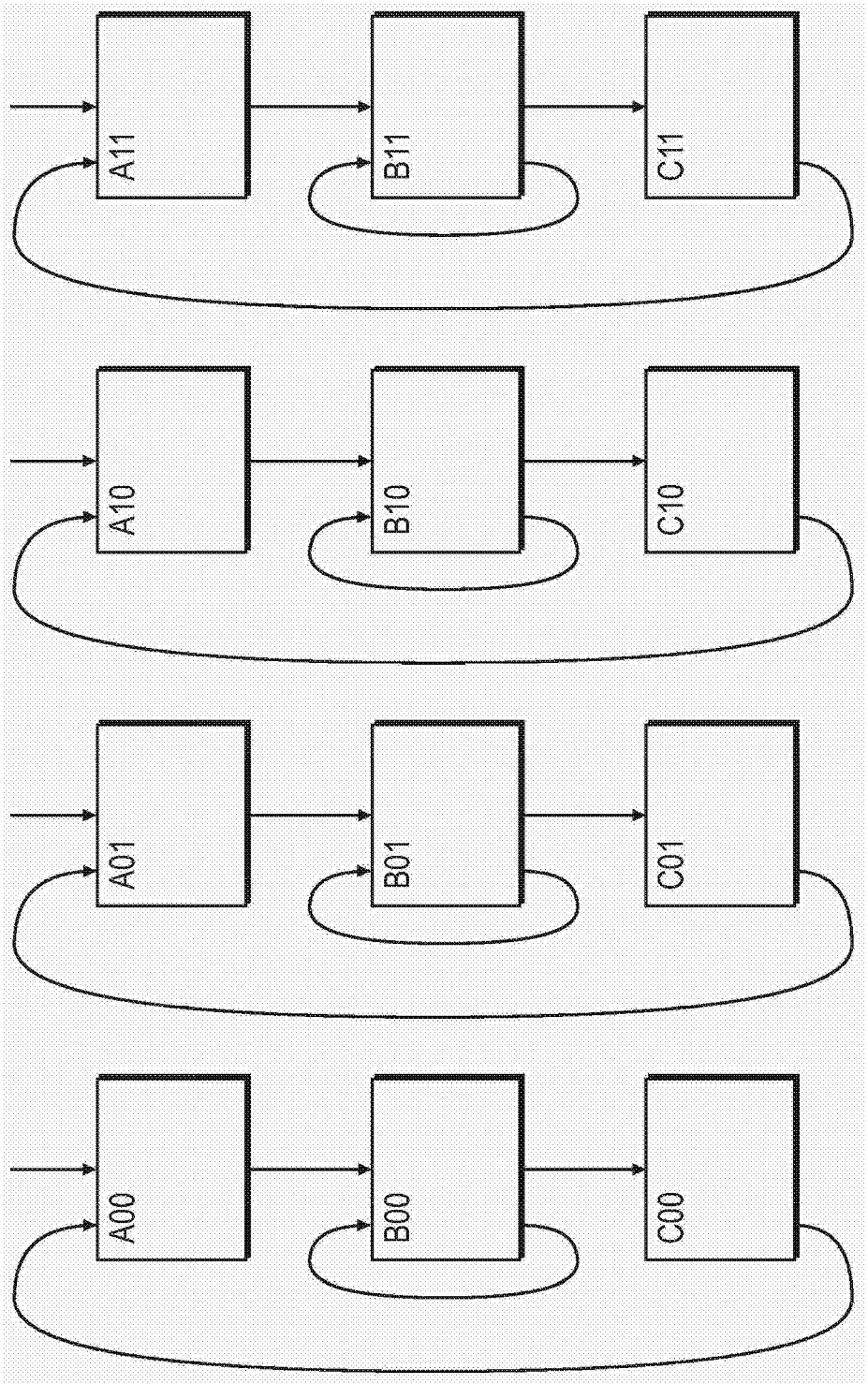


图9

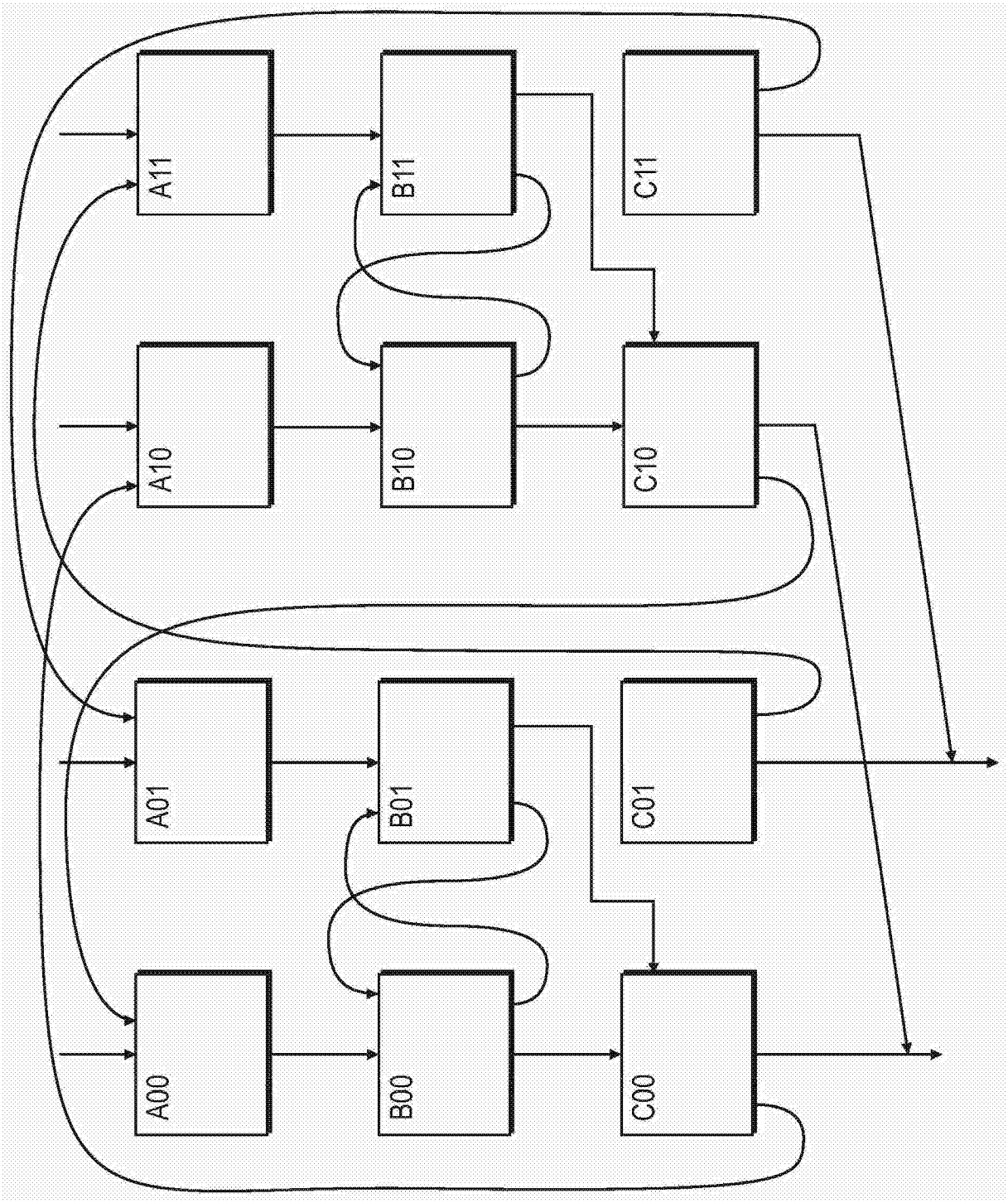


图10

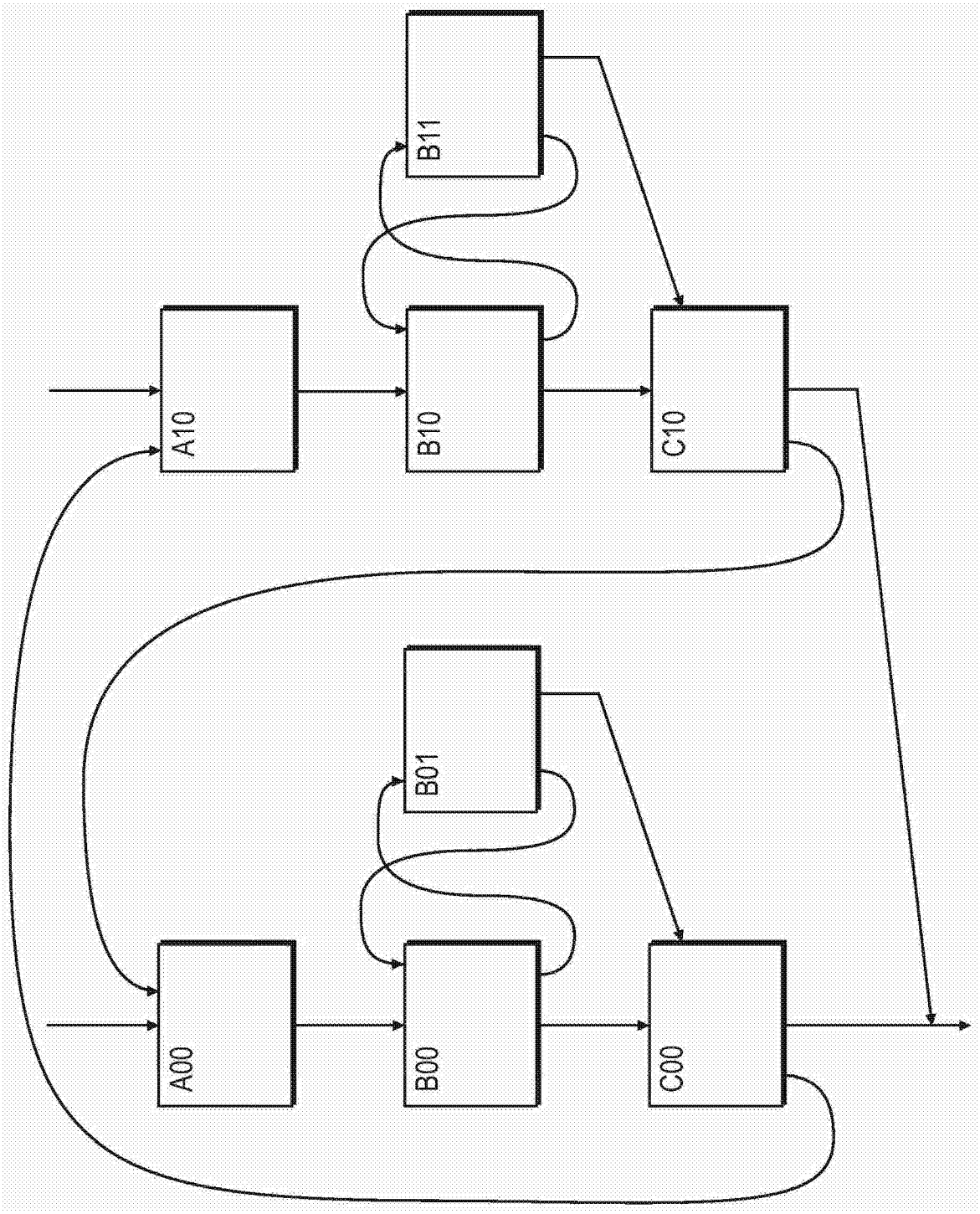


图11

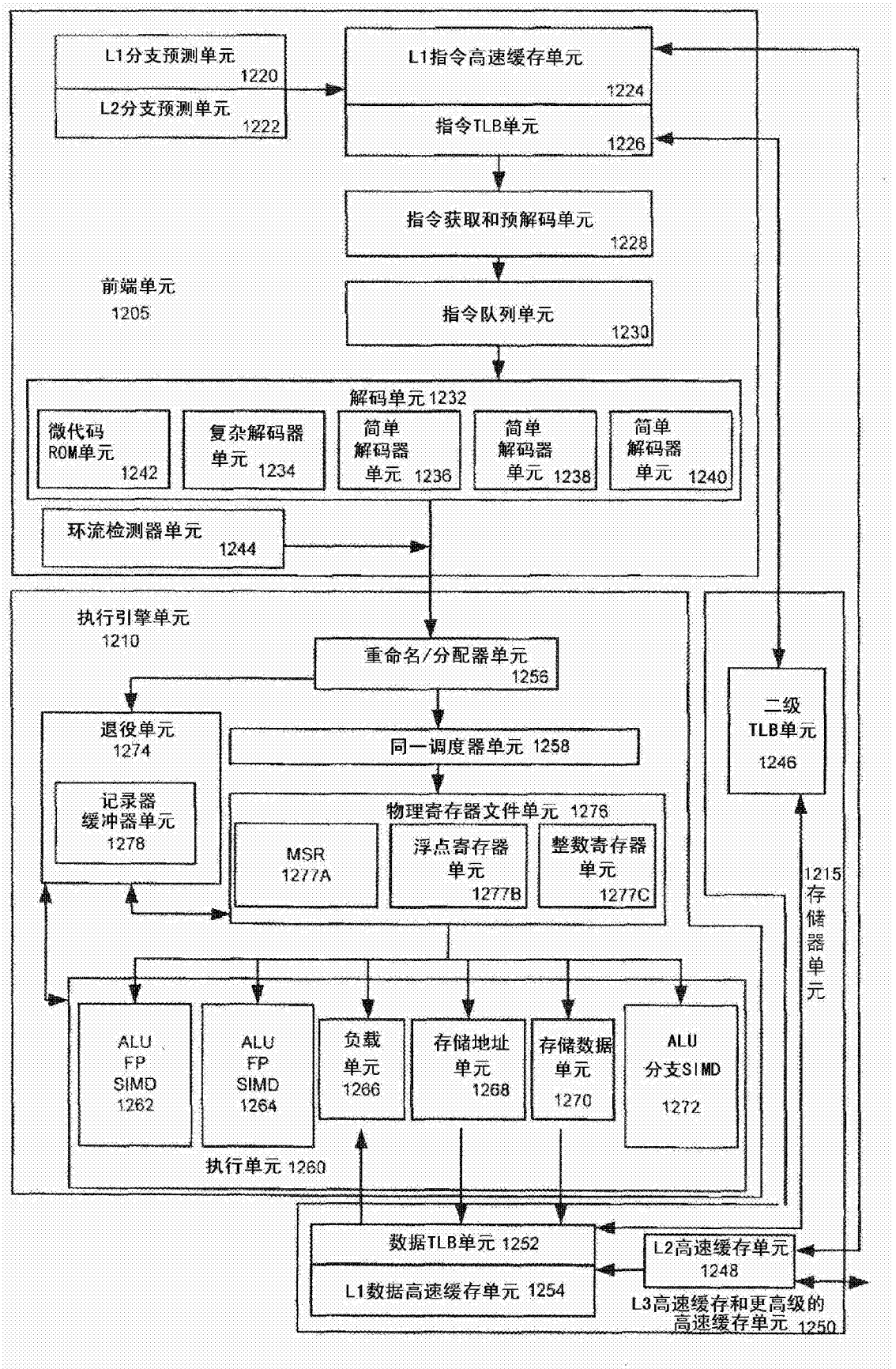

L1分支预测单元
1220
L2分支预测单元
1222
L1指令高速缓存单元
1224
指令TLB单元
1226
指令获取和预解码单元
1228
前端单元
1205
指令队列单元
1230
解码单元1232
微代码
ROM单元
1242
复杂解码器
单元 1234
简单
解码器
单元 1236
简单
解码器
单元 1238
简单
解码器
单元 1240
环流检测器单元
1244
执行引擎单元
1210
重命名/分配器单元
1256
退役单元
1274
记录器
缓冲器单元
1278
同一调度器单元 1258
物理寄存器文件单元 1276
MSR
1277A
浮点寄存器
单元
1277B
整数寄存器
单元
1277C
二级
TLB单元
1246
1215
存储器单元
ALU
FP
SIMD
1262
ALU
FP
SIMD
1264
负载
单元
1266
存储地址
单元
1268
存储数据
单元
1270
ALU
分支SIMD
1272
执行单元 1260
数据TLB单元 1252
L1数据高速缓存单元 1254
L2高速缓存单元
1248
L3高速缓存和更高级的
高速缓存单元 1250

图12

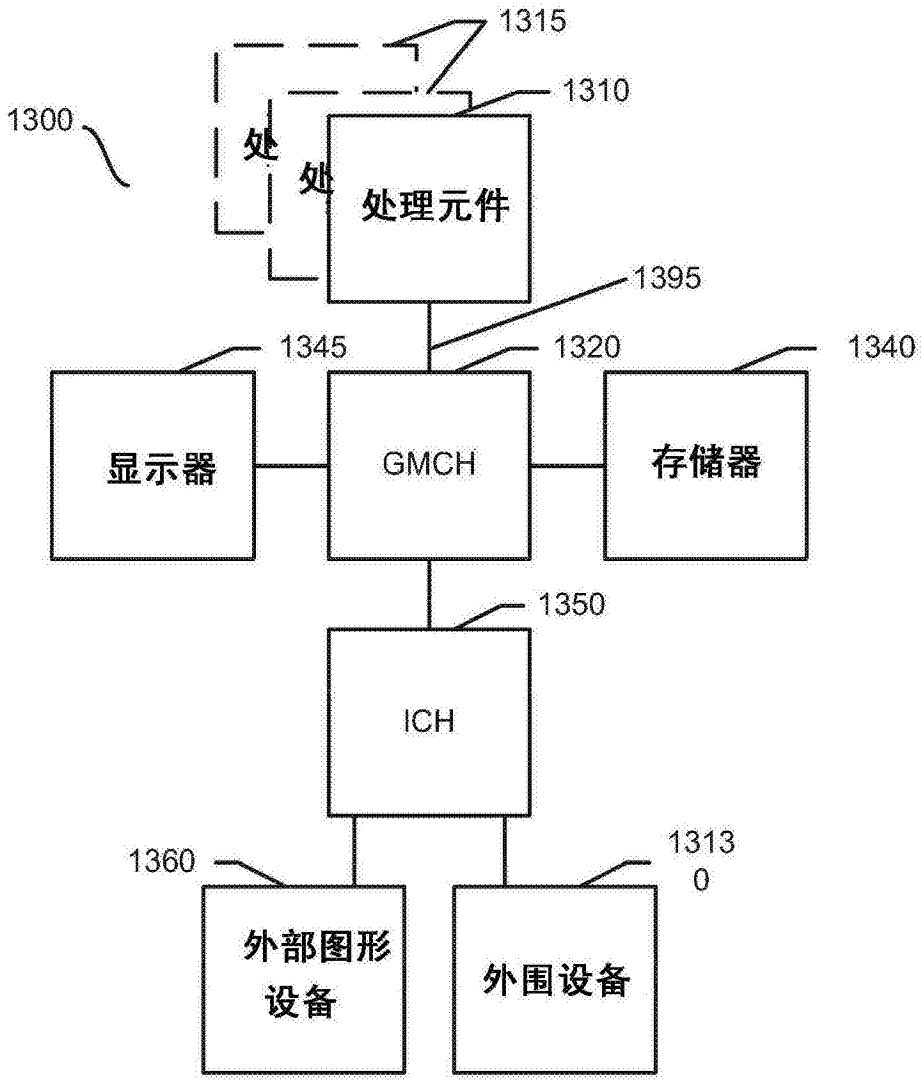

1315
1310
1300
处
处
处理元件
1395
1345
1320
1340
显示器
GMCH
存储器
1350
ICH
1360
1313
0
外部图形
设备
外围设备

图13

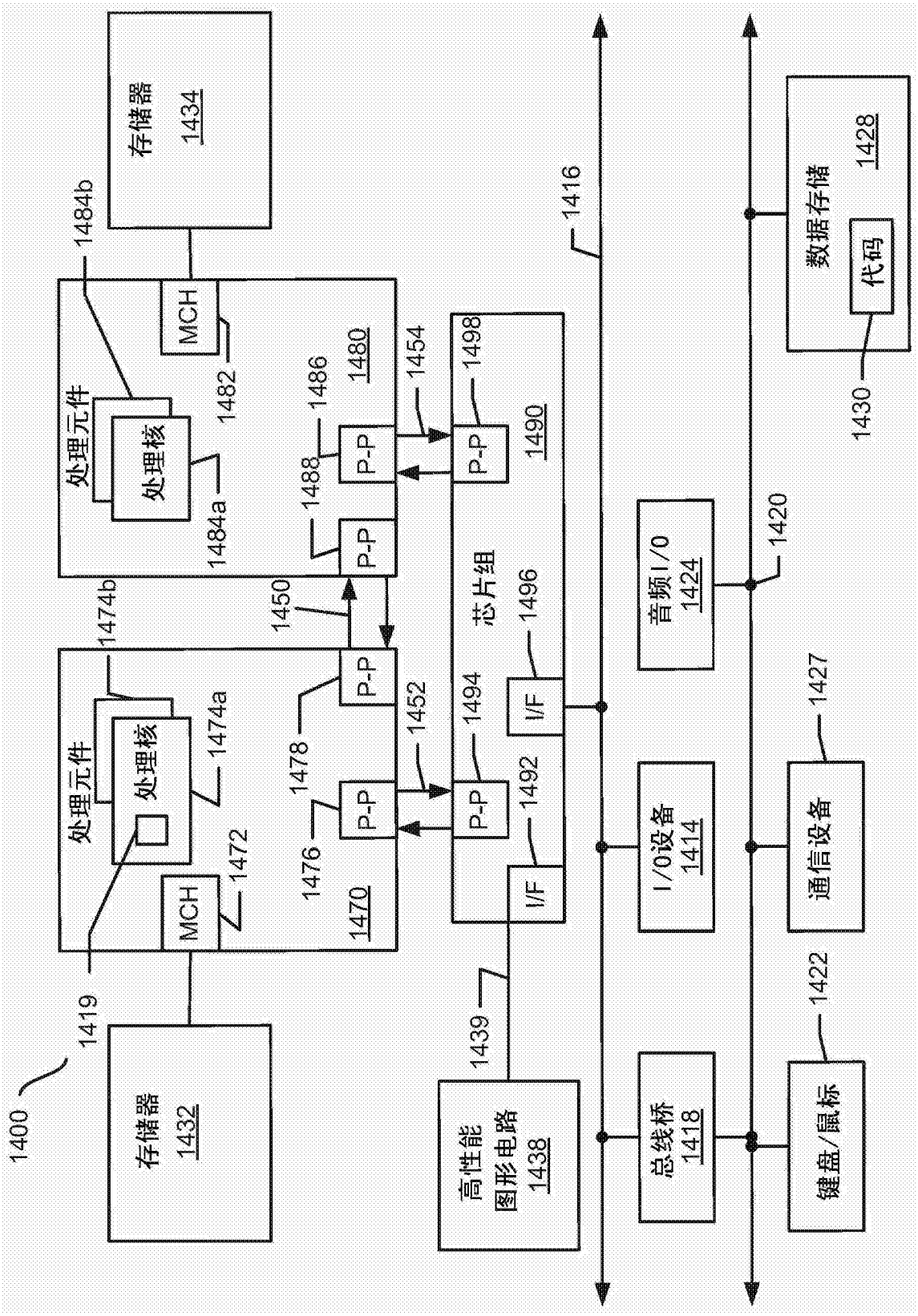

1400
1419
存储器 1432
MCH 1472
处理元件 1474b
处理核 1474a
1484b
存储器 1434
MCH 1482
处理元件
处理核 1484a
1470
1476
1478
P-P
1452
1450
1480
1486
1488
1454
1498
1490
芯片组
1494
1492
1496
I/F
1439
高性能图形电路 1438
1416
I/O设备 1414
总线桥 1418
音频I/O 1424
1420
通信设备 1427
键盘/鼠标 1422
数据存储 1428
代码 1430

图14

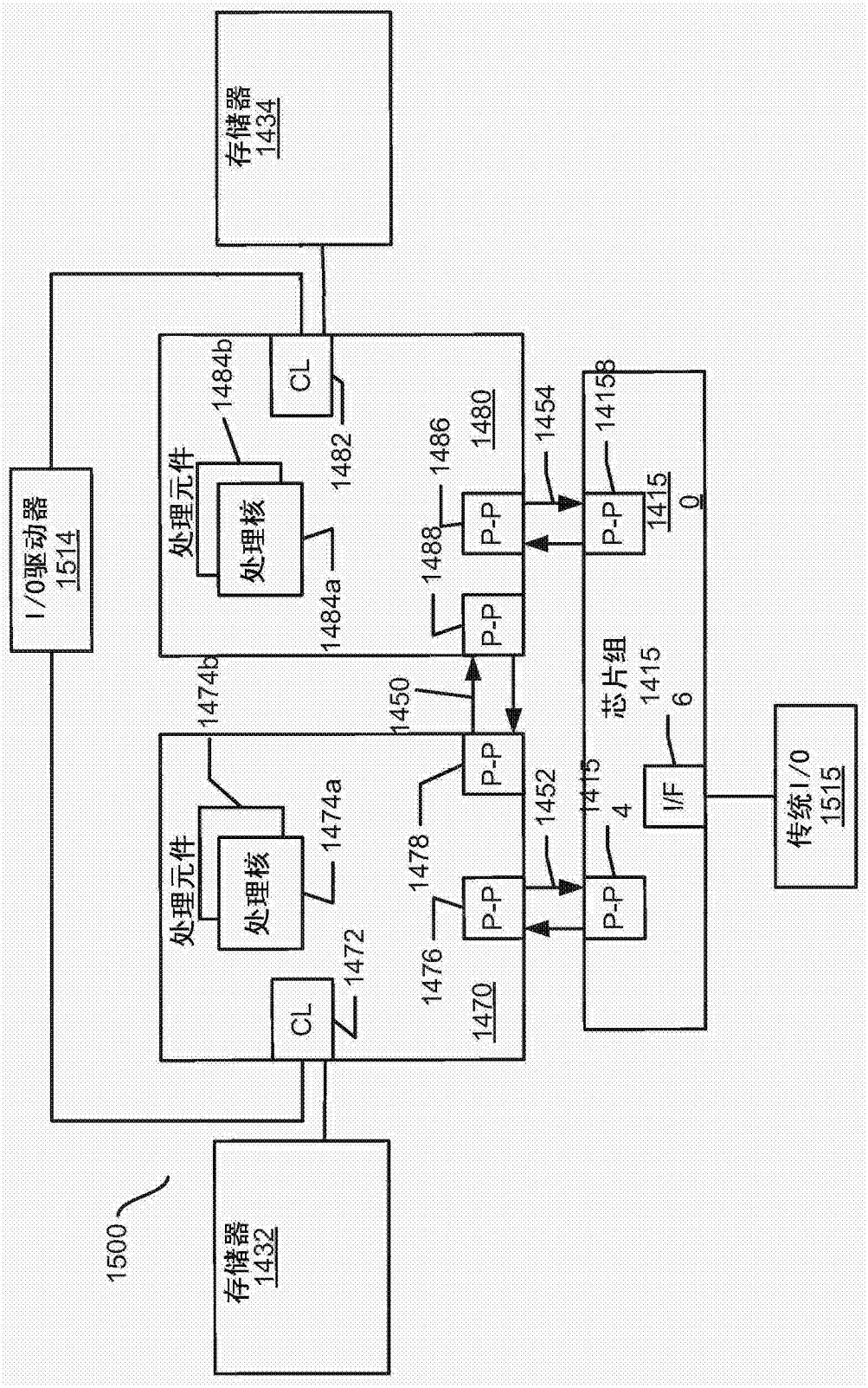

存储器
1434
I/O驱动器
1514
处理元件
处理核
1484b
1484a
CL
1482
1480
1486
1488
P-P
1454
14158
1415
0
1474b
1450
1474a
1478
1452
1415
4
芯片组
1415
6
I/F
传统I/O
1515
1476
1470
1472
1500
存储器
1432

图15