(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 111095222 B
(45)授权公告日 2023.09.15

(21)申请号 201880060330.6
(22)申请日 2018.06.22
(65)同一申请的已公布的文献号
申请公布号 CN 111095222 A
(43)申请公布日 2020.05.01
(30)优先权数据
62/559,780 2017.09.18 US
15/915,930 2018.03.08 US
(85)PCT国际申请进入国家阶段日
2020.03.17
(86)PCT国际申请的申请数据
PCT/US2018/038875 2018.06.22
(87)PCT国际申请的公布数据
WO2019/055094 EN 2019.03.21

(73)专利权人 微软技术许可有限责任公司
地址 美国华盛顿州
(72)发明人 J·莫拉
(74)专利代理机构 北京世辉律师事务所 16093
专利代理师 李峥宇

(51)Int.Cl.
G06F 11/36(2006.01)
G06F 11/34(2006.01)
(56)对比文件
CN 105408877 A,2016.03.16
CN 103885894 A,2014.06.25
US 2011145530 A1,2011.06.16
审查员 刘升

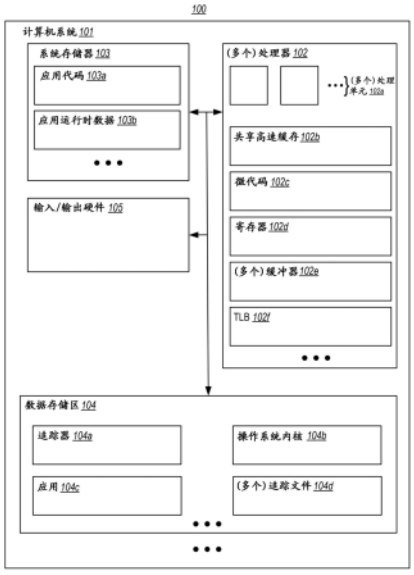
权利要求书3页 说明书27页 附图16页

(54)发明名称

使用高速缓存一致性协议数据的基于高速缓存的追踪记录

(57)摘要

使用高速缓存一致性协议(CCP)数据执行基于高速缓存的追踪记录。实施例检测到发生了引起高速缓存行与后备存储区之间的交互的操作，针对引起该操作的处理单元启用了记载，高速缓存行是记载中的参与者，并且CCP指示存在要被记载到追踪的数据。然后，实施例使该数据被记载到追踪，该数据可用于重放该操作。



CN 111095222 B

1.一种数据处理设备,包括:

多个处理单元;

包括多个高速缓存行的高速缓存存储器,所述多个高速缓存行被用来对来自一个或多个后备存储区的数据进行高速缓存并且由多个处理单元共享,其中所述一个或多个后备存储区与所述多个高速缓存行中的数据之间的一致性根据高速缓存一致性协议CCP被管理;以及

存储的控制逻辑,所述存储的控制逻辑配置所述数据处理设备以至少执行以下操作:

确定至少以下条件已被满足:

(i)操作已引起所述多个高速缓存行中的特定高速缓存行与所述一个或多个后备存储区之间的交互;

(ii)针对引起所述操作的所述多个处理单元中的特定处理单元,记载被启用;

(iii)所述特定高速缓存行是记载中的参与者;以及

(iv)所述CCP指示基于所述操作存在要被记载到追踪的数据;以及

至少基于确定所述条件已被满足,引起所述数据被记载到所述追踪,所述数据可用于重放所述操作。

2.根据权利要求1所述的数据处理设备,其中所述存储的控制逻辑还配置所述数据处理设备以更新与所述特定高速缓存行相关联的一个或多个计账比特,以指示所述特定高速缓存行在所述操作之后是否保持作为记载中的参与者。

3.根据权利要求2所述的数据处理设备,其中与所述特定高速缓存行相关联的所述一个或多个计账比特包括以下中至少一项:(i)单个比特,(ii)多个比特,其中每个比特对应于所述多个处理单元之一,或(iii)存储处理器索引值的多个比特。

4.根据权利要求2所述的数据处理设备,其中与所述特定高速缓存行相关联的所述一个或多个计账比特被存储在一个或多个保留的高速缓存行中,所述一个或多个保留的高速缓存行与被用来对来自一个或多个后备存储区的数据进行高速缓存的高速缓存行是分离的。

5.根据权利要求1所述的数据处理设备,其中使所述数据被记载到所述追踪包括:将所述数据写入到缓冲器,并且其中基于存储器总线活动,将数据从所述缓冲器刷新到所述追踪文件被推迟。

6.根据权利要求1所述的数据处理设备,其中所述存储的控制逻辑还配置所述数据处理设备以通过参考关联高速缓存中的组和路来记载至少一个高速缓存逐出。

7.根据权利要求1所述的数据处理设备,其中记载的所述数据包括不同的CCP状态之间的转变。

8.根据权利要求1所述的数据处理设备,其中记载的所述数据包括以下中至少一项:从写入状态到读取状态的转变,从写入状态到写入状态的转变,或从读取状态到写入状态的转变。

9.根据权利要求1所述的数据处理设备,其中使用所述CCP来标识存在要被记载到追踪的数据包括:标识从读取状态到读取状态的转变不需要被记载到所述追踪。

10.根据权利要求1所述的数据处理设备,其中针对每个处理单元的数据被记载到至少一个分离的数据流。

11.根据权利要求1所述的数据处理设备,其中针对两个或更多处理单元的数据被记载到相同的数据流,但是用处理单元标识符来标记。

12.根据权利要求1所述的数据处理设备,其中要被记载到所述追踪的所述数据包括排序信息。

13.根据权利要求1所述的数据处理设备,其中要被记载的所述数据包括由包围区写入到所述特定高速缓存行的数据,并且其中使所述数据被记载到所述追踪包括:

当引起所述特定高速缓存行与所述一个或多个后备存储区之间的所述交互的所述操作对应于与所述包围区交互的线程时,使所述数据被记载到与所述线程相对应的追踪数据流中;或

当引起所述特定高速缓存行与所述一个或多个后备存储区之间的所述交互的所述操作对应于所述包围区时,使所述数据被记载到与对应于所述线程的所述追踪数据流分离。

14.一种在计算环境中实现的方法,所述计算环境包括多个处理单元和高速缓存存储器,所述高速缓存存储器包括多个高速缓存行,所述多个高速缓存行被用来对来自一个或多个后备存储区的数据进行高速缓存并且由所述多个处理单元共享,其中所述一个或多个后备存储区与所述多个高速缓存行中的数据之间的一致性根据高速缓存一致性协议CCP被管理,所述方法用于使用高速缓存一致性协议数据执行基于高速缓存的追踪记录,所述方法包括:

确定至少以下条件已被满足:

(i)操作已引起所述多个高速缓存行中的特定高速缓存行与所述一个或多个后备存储区之间的交互;

(ii)针对引起所述操作的所述多个处理单元中的特定处理单元,记载被启用;

(iii)所述特定高速缓存行是记载中的参与者;以及

(iv)所述CCP指示基于所述操作存在要被记载到追踪的数据;以及

至少基于确定所述条件已被满足,引起所述数据被记载到所述追踪,所述数据可用于重放所述操作。

15.根据权利要求14所述的方法,还包括:更新与所述特定高速缓存行相关联的一个或多个计账比特,以指示所述特定高速缓存行在所述操作之后是否保持作为记载中的参与者。

16.根据权利要求14所述的方法,其中使所述数据被记载到所述追踪包括:将所述数据写入到缓冲器,并且其中基于存储器总线活动,将数据从所述缓冲器刷新到所述追踪文件被推迟。

17.根据权利要求14所述的方法,其中记载的所述数据包括不同的CCP状态之间的转变。

18.根据权利要求14所述的方法,其中记载的所述数据包括以下中至少一项:从写入状态到读取状态的转变,从写入状态到写入状态的转变,或从读取状态到写入状态的转变。

19.根据权利要求14所述的方法,其中使用所述CCP来标识数据是否要被记载到追踪包括:标识从读取状态到读取状态的转变不需要被记载到所述追踪。

20.一种计算机可读存储介质,用于在计算设备处使用,所述计算设备包括多个处理单元和高速缓存存储器,所述高速缓存存储器包括多个高速缓存行,所述多个高速缓存行被

用来对来自一个或多个后备存储区的数据进行高速缓存并且由所述多个处理单元共享,其中所述一个或多个后备存储区与所述多个高速缓存行中的数据之间的一致性根据高速缓存一致性协议CCP被管理,所述计算机可读存储介质其上存储有计算机可执行指令,所述计算机可执行指令由一个或多个处理单元可执行以引起所述计算设备至少执行以下操作:

确定至少以下条件已被满足:

(i)操作已引起所述多个高速缓存行中的特定高速缓存行与所述一个或多个后备存储区之间的交互;

(ii)针对引起所述操作的所述多个处理单元中的特定处理单元,记载被启用;

(iii)所述特定高速缓存行是记载中的参与者;以及

(iv)所述CCP指示基于所述操作存在要被记载到追踪的数据;以及

至少基于确定所述条件已被满足,引起所述数据被记载到所述追踪,所述数据可用于重放所述操作。

使用高速缓存一致性协议数据的基于高速缓存的追踪记录

背景技术

[0001] 当在软件应用的开发期间编写代码时，开发人员通常花费大量时间来“调试”代码以查找运行时和其他源代码错误。在这样做时，开发人员可以采取几种方法来重现和本地化源代码错误，诸如基于不同的输入观察程序的行为，插入调试代码(例如，打印变量值、跟踪执行的分支等)，临时移除代码部分等。跟踪到运行时错误以查明代码错误可能会占用很大一部分的应用开发时间。

[0002] 为了帮助开发人员进行代码调试过程，已经开发了许多类型的调试应用(“调试器”)。这些工具给开发人员提供追踪(trace)、可视化和更改计算机代码的执行的能力。例如，调试器可以可视化代码指令的执行，可以在代码执行期间的不同时间呈现代码变量值，可以使开发人员能够更改代码执行路径，和/或可以使开发人员能够在感兴趣的代码元素上设置“断点”和/或“观察点”(当在执行期间到达所述点时导致代码的执行被挂起)，等等。

[0003] 新兴形式的调试应用启用“时间旅行”、“反向”或“历史性”调试。通过“时间旅行”调试，程序(例如，诸如线程之类的可执行实体)的执行由追踪应用记录/追踪到一个或多个追踪文件中。然后，(多个)这些追踪文件可以被用于稍后重放程序的执行，以进行前向和后向分析。例如，“时间旅行”调试器可以使开发人员能够设置前向断点/观察点(如常规调试器)以及反向断点/观察点。

发明内容

[0004] 本文中的实施例通过利用处理器的共享高速缓存及其高速缓存一致性协议(CCP)来增强“时间旅行”调试记录，以便确定什么数据应被记载到追踪文件中。与现有方法相比，这样做可以将追踪文件的大小减小几个数量级，从而显著降低追踪记录的开销。

[0005] 在一些实施例中，在包括以下的计算环境中实现：(i)多个处理单元，以及(ii)包括多个高速缓存行(line)的高速缓存存储器，该多个高速缓存行被用来对来自一个或多个后备存储区(store)的数据进行高速缓存并由多个处理单元共享。根据高速缓存一致性协议来管理一个或多个后备存储区与多个高速缓存行中的数据之间的一致性。

[0006] 这些实施例包括使用CCP数据执行基于高速缓存的追踪记录。这些实施例包括：确定操作已引起多个高速缓存行中的特定高速缓存行与一个或多个后备存储区之间的交互；确定针对引起该操作的多个处理单元中的特定处理单元，记载被启用；确定特定高速缓存行是记载中的参与者；并且确定CCP指示存在要被记载到追踪的数据。至少基于这些确定，实施例使该数据被记载到追踪。该数据可用于重放该操作。

[0007] 提供本发明内容来以简化的形式介绍一系列概念，这些概念将在下面的详细描述中被进一步描述。本发明内容既不旨在标识所要求保护的主题的关键特征或必要特征，也不旨在被用于帮助确定所要求保护的主题的范围。

附图说明

[0008] 为了描述可以获得本发明的上述及其他优点和特征的方式，将通过参考在附图中

图示出的本发明的特定实施例来呈现对以上简要描述的本发明的更具体描述。应理解这些附图仅描绘了本发明的典型实施例，并且因此不应被认为是对本发明范围的限制，本发明将通过使用附图以附加的特征和细节来进行描述和解释，在附图中：

[0009] 图1图示出了示例计算环境，该示例计算环境利于使用高速缓存一致性协议(CCP)数据经由共享高速缓存来记录代码执行的“比特准确(bit-accurate)”追踪；

[0010] 图2图示出了共享高速缓存的示例；

[0011] 图3图示出了用于使用CCP数据执行基于高速缓存的追踪记录的示例方法的流程图；

[0012] 图4A图示出了示例共享高速缓存，该共享高速缓存用一个或多个附加计账比特来扩展其每个高速缓存行；

[0013] 图4B图示出了共享高速缓存的示例，该共享高速缓存包括一个或多个保留的高速缓存行，用于存储应用于常规高速缓存行的计账比特；

[0014] 图5图示出了关联高速缓存映射的示例；

[0015] 图6A图示出了表，其示出了共享高速缓存中的单个行上的四个处理单元的示例读取和写入活动；

[0016] 图6B图示出了表，其示出了基于图6A中所示的读取和写入活动的示例跟踪(tracked)高速缓存一致性状态；

[0017] 图6C图示出了表，其示出了基于图6A中所示的读取和写入活动而存储在共享高速缓存的计账比特(即单元比特、索引比特和/或标志比特)中的示例数据；

[0018] 图6D图示出了表，其示出了可以结合图6A中所示的读写活动而被写入到追踪文件中的示例日志数据；

[0019] 图7A图示出了一个示例，在其中取决于处理器如何被跟踪，可以从追踪中省略一些读取->读取转变；

[0020] 图7B图示出了省略图7A中突出显示的读取->读取转变的记载数据的示例；

[0021] 图7C图示出了表，其示出了如果使用“索引比特”并且索引在读取时被更新则可能被记录的示例记载数据；

[0022] 图8A图示出了包括两个处理器的示例计算环境，每个处理器包括四个处理单元以及L1-L3高速缓存；

[0023] 图8B图示出了表，其示出了由图8A的一些处理单元执行的示例读取和写入操作；

[0024] 图9A图示出了表，其示出了由两个处理单元进行的示例读取和写入；

[0025] 图9B图示出了图示表的示例，该表比较提供CCP单元信息加上高速缓存行标志比特的环境与提供CCP索引信息加上高速缓存行标志比特的环境何时制成日志条目；

[0026] 图10A图示出了存储器地址的不同部分及其与关联高速缓存的关系的示例；和

[0027] 图10B图示出了在关联高速缓存中记载高速缓存未命中和高速缓存逐出的示例。

具体实施方式

[0028] 本文的实施例通过利用处理器的共享高速缓存及其高速缓存一致性协议来增强“时间旅行”调试记录，以便确定什么数据应被记载到追踪文件中。与现有方法相比，这样做可以将追踪文件的大小减小几个数量级，从而显著降低追踪记录的开销。

[0029] 图1图示出了示例计算环境100，该示例计算环境100利于使用高速缓存一致性协议数据经由共享高速缓存来记录代码执行的“比特准确”追踪。如所描绘的，实施例可以包括或利用专用或通用计算机系统101，其包括计算机硬件，诸如例如一个或多个处理器102、系统存储器103、一个或多个数据存储区104、和/或输入/输出硬件105。

[0030] 本发明范围内的实施例包括用于承载或存储计算机可执行指令和/或数据结构的物理和其他计算机可读介质。这样的计算机可读介质可以是计算机系统101可以访问的任何可用介质。存储计算机可执行指令和/或数据结构的计算机可读介质是计算机存储设备。承载计算机可执行指令和/或数据结构的计算机可读介质是传输介质。因此，通过示例而非限制，本发明的实施例可以包括至少两种明显不同种类的计算机可读介质：计算机存储设备和传输介质。

[0031] 计算机存储设备是存储计算机可执行指令和/或数据结构的物理硬件设备。计算机存储设备包括各种计算机硬件，诸如RAM、ROM、EEPROM、固态驱动器(“SSD”)、闪存、相变存储器(“PCM”)、光盘存储装置、磁盘存储装置或其他磁存储设备或者任何其他可以被用来以计算机可执行指令或数据结构的形式存储程序代码并且可以由计算机系统101访问和执行以实现本发明公开的功能性的(多个)硬件设备。因此，例如，计算机存储设备可以包括所描绘的系统存储器103、可以存储计算机可执行指令和/或数据结构的所描绘的数据存储区104、或诸如处理器上存储装置之类的其他存储装置，如稍后讨论的。

[0032] 传输介质可以包括网络和/或数据链路，其可以被用来以计算机可执行指令或数据结构的形式承载程序代码，并且其可以由计算机系统101访问。“网络”被定义为一个或多个数据链路，其使得能够在计算机系统和/或模块和/或其他电子设备之间传送电子数据。当信息通过网络或另一通信连接(硬连线、无线、或硬连线或无线的组合)而被传送或提供给计算机系统时，计算机系统可以将连接视为传输介质。上述的组合也应被包括在计算机可读介质的范围内。例如，输入/输出硬件105可以包括连接网络和/或数据链路的硬件(例如，网络接口模块(例如，“NIC”))，该网络和/或数据链路可以被用来以计算机可执行指令或数据结构的形式承载程序代码。

[0033] 此外，在到达各种计算机系统组件时，计算机可执行指令或数据结构形式的程序代码可以被自动从传输介质传输到计算机存储设备(反之亦然)。例如，通过网络或数据链路接收的计算机可执行指令或数据结构可以被缓冲在NIC(例如，输入/输出硬件105)内的RAM中，然后最终传输到系统存储器103和/或计算机系统101处的易失性较小的计算机存储设备(例如，数据存储区104)。因此，应当理解，计算机存储设备可以被包括在也(或者甚至主要地)利用传输介质的计算机系统组件中。

[0034] 计算机可执行指令例如包括指令和数据，当在(多个)处理器102处被执行时，这些指令和数据使得计算机系统101执行特定功能或功能组。计算机可执行指令可以是例如二进制文件、中间格式指令(诸如汇编语言)或者甚至源代码。

[0035] 本领域技术人员将理解，本发明可以在具有许多类型的计算机系统配置的网络计算环境中被实践，该计算机系统配置包括个人计算机、台式计算机、膝上型计算机、消息处理器、手持式设备、多处理器系统、基于微处理器或可编程的消费类电子产品、网络PC、小型计算机、大型计算机、移动电话、PDA、平板电脑、寻呼机、路由器、交换机等。本发明也可以在分布式系统环境中被实践，在该分布式系统环境中，通过网络(通过硬连线数据链路、无线

数据链路、或通过硬连线和无线数据链路的组合)链接的本地和远程计算机系统均执行任务。这样,在分布式系统环境中,计算机系统可以包括多个组成计算机系统。在分布式系统环境中,程序模块可以位于本地和远程存储器存储设备中。

[0036] 本领域技术人员还将理解,本发明可以在云计算环境中被实践。云计算环境可以是分布式的,尽管这不是必需的。当被分布时,云计算环境可以在组织内进行国际分布和/或具有在多个组织上拥有的组件。在本说明书和所附权利要求书中,“云计算”被定义为用于支持对可配置计算资源(例如,网络、服务器、存储装置、应用和服务)的共享池的按需网络访问的模型。“云计算”的定义不限于在被适当部署时可以从这种模型获得的其他众多优势中的任何一种。

[0037] 云计算模型可以由各种特性组成,诸如按需自助服务、广泛的网络访问、资源池化、快速弹性、测量的服务等等。云计算模型也可以采用各种服务模型的形式,诸如例如软件即服务(“SaaS”)、平台即服务(“PaaS”)和基础设施即服务(“IaaS”)。还可以使用诸如私有云、社区云、公共云、混合云等不同的部署模型来部署云计算模型。

[0038] 一些实施例,诸如云计算环境,可以包括一种系统,该系统包括一个或多个主机,每个主机能够运行一个或多个虚拟机。在操作期间,虚拟机模拟操作计算系统,从而支持操作系统以及可能的一个或多个其他应用。在一些实施例中,每个主机包括管理程序,该管理程序使用从虚拟机的角度抽象的物理资源来模拟用于虚拟机的虚拟资源。管理程序还可以在虚拟机之间提供适当的隔离。因此,从任何给定虚拟机的视角来看,管理程序提供了虚拟机正在与物理资源进行接口连接的错觉,即使虚拟机仅与物理资源的外观(例如,虚拟资源)进行接口连接。物理资源的示例包括处理能力、存储器、磁盘空间、网络带宽、媒体驱动器等。

[0039] 如所图示,数据存储区104可以存储表示应用程序的计算机可执行指令和/或数据结构,该应用程序诸如例如是追踪器104a、操作系统内核104b和应用104c(例如,作为由追踪器104a追踪的主题的应用,以及一个或多个追踪文件104d)。当这些程序正在执行(例如,使用(多个)处理器102)时,系统存储器103可以存储对应的运行时数据,诸如运行时数据结构、计算机可执行指令等。因此,图1将系统存储器103图示为包括运行时应用代码103a和应用运行时数据103b(例如,每个与应用104c相对应)。

[0040] 追踪器104a可用于记录诸如应用104c之类的应用的比特准确追踪,并将追踪数据存储在(多个)追踪文件104d中。在一些实施例中,追踪器104a是独立的应用,而在其他实施例中,追踪器104a被集成到另一个软件组件中,诸如操作系统内核104b、管理程序、云结构等。由于(多个)追踪文件104d被描绘为被存储在数据存储区104中,所以(多个)追踪文件104d也可以独占地或临时地被记录在系统存储器103中或某个其他存储设备处。

[0041] 图1包括(多个)处理器102的内部硬件组件的简化表示。如所图示,每个处理器102包括多个处理单元102a。每个处理单元可以是物理的(即,物理处理器核)和/或逻辑的(即,由支持超线程的物理核所呈现的逻辑核,其中多于一个应用线程在物理核处执行)。因此,例如,即使处理器102在一些实施例中可以仅包括单个物理处理单元(核),它也可以包括由该单个物理处理单元所呈现的两个或更多个逻辑处理单元102a。

[0042] 每个处理单元102a执行由应用(例如,追踪器104a、操作系统内核104b、应用104c等)定义的处理器指令,并且从预定义的处理器指令集架构(ISA)中选择这些指令。每个处

理器102的特定ISA基于处理器制造商和处理器模型而变化。常见的ISA包括来自INTEL公司的IA-64和IA-32架构，来自超微半导体公司的AMD64架构以及来自安谋国际科技股份公司(ARM HOLDINGS,PLC)的各种高级RISC机器(“ARM”)架构，但是大量其他的ISA存在并且可以被本发明使用。通常，“指令”是可由处理器执行的最小的外部可见(即，在处理器外部)的代码单元。

[0043] 每个处理单元102a从共享高速缓存102b获得处理器指令，并且基于共享高速缓存102b中的数据，基于寄存器102d中的数据和/或在没有输入数据的情况下来执行处理器指令。通常，共享高速缓存102b是小量(即，相对于系统存储器103的典型数量而言为小)的随机存取存储器，其存储后备存储区的各部分的在处理器上的副本，诸如系统存储器103和/或另一个高速缓存。例如，当执行应用代码103a时，共享高速缓存102b包含应用运行时数据103b的各部分。如果(多个)处理单元102a需要尚未被存储在共享高速缓存102b中的数据，那么发生“高速缓存未命中”，并且该数据从系统存储器103中被获取(可能是从共享高速缓存102b“逐出”一些其他数据)。

[0044] 典型地，共享高速缓存102b包括多个“高速缓存行”，其每一个都存储来自后备存储区的存储器组块。例如，图2图示出了共享高速缓存200的至少一部分的示例，该共享高速缓存200包括多个高速缓存行203，每个高速缓存行203包括地址部分201和值部分202。每个高速缓存行203的地址部分201可以在该行所对应的后备存储区(例如系统存储器103)中存储地址，并且值部分202可以最初存储从后备存储区接收的值。值部分202可以由处理单元102a修改，并且最终被逐出回到后备存储区。如省略号所指示的，共享高速缓存200可以包括大量高速缓存行。例如，当代的INTEL处理器可以包含层1高速缓存，其包括512个或更多高速缓存行。在此高速缓存中，每条高速缓存行典型地可用于存储64字节(512比特)值，以参考8字节(64比特)存储器地址。

[0045] 存储在每个高速缓存行203的地址部分201中的地址可以是物理地址，诸如系统存储器103中的实际存储器地址。备选地，存储在每个高速缓存行203的地址部分201中的地址可以是虚拟地址，这是指派给物理地址以提供抽象的地址。这样的抽象可以被使用，例如，以利于在(多个)处理器102处执行的不同过程之间的存储器隔离。当虚拟地址被使用时，处理器102可以包括转变后备缓冲器(TLB)102f(通常是存储器管理单元(MMU)的一部分)，其维护物理和虚拟存储器地址之间的映射。

[0046] 共享高速缓存102b可以包括代码高速缓存部分和数据高速缓存部分。例如，当执行应用代码103a时，共享高速缓存102b的代码部分存储被存储在应用代码103a中的处理器指令的至少一部分，并且共享高速缓存102b的数据部分存储应用运行时数据103b的数据结构的至少一部分。通常，处理器高速缓存被划分为分离的层次/层(例如，层1(L1)、层2(L2)和层3(L3))，而一些层次(例如L3)可能与(多个)处理器102相分离地存在。因此，共享高速缓存102b可以包括这些层之一(L1)，或者可以包括多个这些层。

[0047] 当多个高速缓存层被使用时，(多个)处理单元102a直接与最低层(L1)交互。在大多数情况下，数据在层之间流动(例如，在一次读取时，L3高速缓存与系统存储器103交互并将数据提供给L2高速缓存，而L2高速缓存继而又将数据提供给L1高速缓存)。当处理单元102a需要执行写入时，高速缓存进行协调以确保已经影响了在(多个)处理单元102a之间所共享的数据的那些高速缓存不再具有该数据。使用高速缓存一致性协议(稍后讨论)执行此

协调。

[0048] 高速缓存可以是包含性的、独占的、或者包括包含的和独占的行为。例如，在包含性高速缓存中，L3层将在其下方的L2层中存储数据的超集(superset)，而L2层存储在它们下方的L1层的超集。在独占高速缓存中，这些层可能是不相交的——例如，如果L1高速缓存所需的数据存在于L3高速缓存中，则它们可以交换信息，诸如数据、地址等。

[0049] 每个处理单元102还包括微代码102c，其包括控制处理器102的操作的控制逻辑(即，可执行指令)，并且其通常用作处理器的硬件和由处理器102暴露给执行应用的处理器ISA之间的解译器。微代码102c可以被体现在诸如ROM、EEPROM等的处理器上的存储装置中。

[0050] 寄存器102d是基于硬件的存储位置，其基于(多个)处理器102的ISA而被定义，并从处理器指令中被读取和/或被写入处理器指令。例如，寄存器102d通常被用来存储从共享高速缓存102b获取的值以供指令使用，存储执行指令的结果和/或存储状况或状态——诸如执行指令的一些副作用(例如，值符号改变、值达到零、进位的发生等)、处理器周期计数等。因此，一些寄存器102d可以包括“标志”，其被用来发信号通知由执行处理器指令引起的某些状态改变。在一些实施例中，处理器102还可以包括控制寄存器，其被用来控制处理器操作的不同方面。

[0051] 在一些实施例中，(多个)处理器102可以包括一个或多个缓冲器102e。如下文将讨论的，(多个)缓冲器102e可以被用作针对追踪数据的临时存储位置。因此，例如，(多个)处理器102可以将追踪数据的各部分存储在(多个)缓冲器102e中，并且在适当的时间(诸如当存在可用存储器总线带宽时)将该数据刷新到(多个)追踪文件104d。在一些实现中，(多个)缓冲器102e可以是共享高速缓存102b的一部分。

[0052] 如上所述，拥有共享高速缓存102b的处理器根据高速缓存一致性协议(“CCP”)来操作高速缓存。特别地，CCP定义了当各种处理单元102a在共享高速缓存102b中读取数据以及写入数据时如何在共享高速缓存102b中的数据与后备数据存储区(例如，系统存储器103或另一高速缓存)之间保持一致性，以及如何确保各种处理单元102a总是从共享高速缓存102b中的给定位置读取有效数据。CCP通常与处理器102的ISA所定义的存储器模型相关并启用该存储器模型。

[0053] 公共CCP的示例包括MSI协议(即，已修改、共享和无效)、MESI协议(即已修改、独占、共享和无效)以及MOESI协议(即已修改的、占有、独占、共享和无效)。这些协议中的每一个都为共享高速缓存102b中的个体位置(例如，线)定义状态。“已修改”高速缓存位置包含已在共享高速缓存102b中被修改的数据，并且因此可能与后备存储区(例如，系统存储器103或另一高速缓存)中的对应数据不一致。当从共享高速缓存102b中逐出具有“已修改”状态的位置时，公共CCP需要高速缓存以保证其数据被写回到后备存储区，或者另一个高速缓存接管了这一责任。“共享”高速缓存位置包含未从后备存储区中的数据进行修改、以只读取状态存在并且由(多个)处理单元102a共享的数据。共享高速缓存102b可以逐出该数据而无需将其写入到后备存储区。“无效”高速缓存位置不包含有效数据，并且可以被认为是空的且可用于存储高速缓存未命中的数据。“独占”高速缓存位置包含与后备存储区匹配的数据，并且仅由单个处理单元102a使用。可以随时(即，响应读取请求)将其更改为“共享”状态，或者在对其写入时可以将其更改为“已修改”状态。“占有”高速缓存位置由两个或更多处理单元102a共享，但是其中一个处理单元具有对其进行更改的独占权。当该处理进行更

改时，它直接或间接地通知其他处理单元——因为所通知的处理单元可能需要基于CCP实现进行无效或更新。

[0054] 不同的CCP跟踪高速缓存一致性并使该高速缓存一致性数据可用于追踪器104a的粒度可以变化。例如，在频谱的一端处，一些CCP按每个高速缓存行以及每个处理单元来跟踪高速缓存一致性。因此，这些CCP可以跟踪每个高速缓存行的状态——在它与每个处理单元相关时。如将在下面结合图6A-图6D的示例中所展示的，这意味着单个高速缓存行可以具有关于其状态的信息——在它与每个处理单元102a相关时。其他CCP的粒度较小，并且仅跟踪该级别高速缓存行的高速缓存一致性(并且缺少每个处理单元的信息)。在频谱的另一端处，由于一个处理器一次只能独占地占有(独占、已修改等)一行，因此处理器制造商出于效率考虑可以选择仅跟踪该级别高速缓存行的高速缓存一致性。作为中间粒度的示例，CCP可以跟踪每个高速缓存行的高速缓存一致性，以及具有当前高速缓存行状态的处理单元的索引(例如，针对四处理单元处理器的索引为0、1、2、3)。

[0055] 实施例利用处理器的共享高速缓存102b来有效地记录应用104c和/或操作系统内核104b的执行的比特准确追踪。这些实施例是基于对处理器102(包括共享高速缓存102b)形成半封闭或准封闭系统的观察而被建立的。例如，一旦用于过程的数据的各部分(即代码数据和运行时应用数据)被加载到共享高速缓存102b中，则处理器102可以——在没有任何输入的情况下——作为半封闭或准封闭系统自行运行一段时间。特别地，处理单元102a中的一个或多个使用存储在共享高速缓存102b的数据部分中的运行时数据并使用寄存器102d来执行来自共享高速缓存102b的代码部分的指令。

[0056] 当处理单元102a需要一些信息涌入时(例如，由于其正在执行、将要执行或可能执行的指令访问共享高速缓存102b中尚未存在的代码或运行时数据)，将发生“高速缓存未命中”，并且该信息从系统存储器103被带入到分片(shard)高速缓存102b中。例如，如果当所执行的指令在应用运行时数据103b内的存储器地址处执行存储器操作时发生数据高速缓存未命中，则来自该存储器地址的数据被带入到共享高速缓存102b的数据部分的高速缓存行之一中。类似地，如果当指令在系统存储器103中存储的存储器地址应用代码103a处执行存储器操作时发生代码高速缓存未命中，则来自该存储器地址的代码被带入到共享高速缓存102b的代码部分的高速缓存行之一中。然后，处理单元102a使用共享高速缓存102b中的新信息继续执行，直到新信息再次被带入到共享高速缓存102b中(例如，由于另一个高速缓存未命中或未高速缓存的读取)。

[0057] 发明人观察到，为了记录应用执行的比特准确表示，追踪器104a可以记录足够的数据，以便能够在该应用的(多个)线程的执行期间将信息的涌入再现到共享高速缓存102b中。执行此操作的第一种方法是：通过记载所有高速缓存未命中和未高速缓存的读取(即，来自硬件组件和不可高速缓存存储器的读取)以及每片数据被带入到共享高速缓存102b中的执行期间的时间(例如，使用被执行的指令的计数或某个其他计数器)，来记录被带入到共享高速缓存102b中的所有数据。

[0058] 第二种方法是：跟踪和记录被每个处理单元102a“消耗的”高速缓存行——其导致比第一种方法小得多的追踪文件。如本文所使用的，当处理单元知道其当前值时，它已经“消耗”了高速缓存行。这可能是因为处理单元是写入高速缓存行的当前值的单元，或者是因为处理单元对高速缓存行执行了读取。该第二种方法涉及对共享高速缓存102b的扩展，

其使得处理器102能够为每个高速缓存行标识消耗了该高速缓存行的一个或多个处理单元102a。

[0059] 根据本文的实施例，第三种方法是利用处理器的CCP来确定“消耗的”高速缓存行的子集以记录在(多个)追踪文件104d中，并且这仍将使得共享高速缓存102b的活动能够被再现。与第一种方法和第二种方法相比，该第三种方法产生小得多的追踪文件——从而大大降低了追踪开销。

[0060] 本文中的一些实施例记录与处理单元/线程相对应的追踪数据流。例如，(多个)追踪文件104d可以包括用于每个处理单元的一个或多个单独的追踪数据流。在这些实施例中，每个追踪数据流中的数据分组可能缺少该数据分组所应用于的处理单元的标识，因为该信息是基于追踪数据流本身而固有的。在这些实施例中，如果计算机系统101包括多个处理器102(即，在不同的处理器插槽内)，则(多个)追踪文件针对不同处理器102中的每个处理单元102a而可以具有一个或多个不同的追踪数据流。多个数据流甚至可以被用于单个线程。例如，一些实施例可以将一个数据流与被线程使用的处理单元相关联，并且将一个或多个附加数据流与被线程使用的每个共享高速缓存相关联。

[0061] 在其他实施例中，(多个)追踪文件104d可以包括用于处理器102的单个追踪数据流，并且可以在每个数据分组中标识该数据分组应用于哪个处理单元。在这些实施例中，如果计算机系统101包括多个处理器102，则(多个)追踪文件104d可以包括用于多个处理器102中的每一个处理器的单独的追踪数据流。不管(多个)追踪文件的布局如何，针对每个处理单元102a的数据分组通常独立于其他处理单元而被记录，从而使得在不同处理单元102a处执行的不同线程能够被独立地重放。然而，追踪文件可以包括一些信息——无论它是表达的还是固有的——这些信息在不同线程之间提供了部分排序。

[0062] 图3图示出了用于使用CCP数据执行基于高速缓存的追踪记录的方法300的流程图。方法300可以包括由处理器102在追踪器104a追踪应用104c和/或操作系统内核104b时执行的动作。处理器102做出的动作可以基于处理器102中的硬编码逻辑、软编码逻辑(即，微代码102c)和/或诸如追踪器104a、操作系统内核104b或管理程序之类的其他软件应用。尽管图3图示出了一系列动作，但是应当理解，实施例可以按任何顺序执行这些动作中的许多动作，甚至可以并行执行一些动作。这样，方法300中所示的动作序列是非限制性的。

[0063] 如所描绘的，方法300包括检测高速缓存与后备存储区之间的交互的动作301。在一些实施例中，动作301包括检测引起多个高速缓存行中的特定高速缓存行与一个或多个后备存储区之间的交互的操作。例如，当在处理单元102a之一处执行应用104c或操作系统内核104b的线程时，处理单元可以引起共享高速缓存102b中的行与后备存储区(例如，系统存储器103或另一高速缓存)之间的交互。可以例如由处理器102基于执行其微代码102c来执行该检测。

[0064] 方法300还包括标识引起交互的处理单元的动作302。在一些实施例中，动作302包括标识多个处理单元中引起该操作的特定处理单元。例如，基于执行微代码102c，处理器102可以标识哪个处理单元102a引起了在动作301中检测到的操作。

[0065] 方法300还包括确定是否针对处理单元启用了记载的动作303。在一些实施例中，动作303包括使用一个或多个记载控制比特来确定针对特定处理单元启用了记载。例如，处理器102可以基于一个或多个记载控制比特来确定在动作302中标识的处理单元是否已启

用了记载。使用(多个)记载控制比特的记载使能不同处理单元的记载可以被动态启用和禁用。因此,通过使用(多个)记载控制比特,追踪器104a可以动态地控制哪个或哪些线程正被追踪,和/或不同线程的执行的哪个或哪些部分正被追踪。

[0066] (多个)记载控制比特的特定形式和功能可以变化。在一些实施例中,例如,(多个)记载控制比特是诸如控制寄存器之类的寄存器102d之一的一部分。在这些实施例中,单个记载控制比特可以对应于一个处理单元102a,或者对应于多个处理单元102a。因此,寄存器102d包括单个记载控制比特(例如,对应于所有处理单元或特定处理单元或处理单元的子集),或者可以潜在地包括多个记载控制比特(例如,每个对应于一个或多个处理单元)。在其他实施例中,(多个)记载控制比特包括地址空间标识符(ASID)和/或过程上下文标识符(PCID),或以其他方式与之相关联,该地址空间标识符和/或过程上下文标识符(PCID)与引起高速缓存与后备存储区之间的交互的指令相对应。因此,例如,方法300可以仅在处理单元执行与一个或多个特定ASID/PCID相关联的指令时才追踪处理单元。以这种方式,方法300可以仅记录指定的(多个)地址空间和/或特定的(多个)过程上下文。组合也是可能的。例如,(多个)记载控制比特可以被存储在一个或多个寄存器102d中,但是其可以基于当前的ASID/PCID值而被设置/清除。不管(多个)记载控制比特的形式如何,一些实施例都可以能够设置/清除在上下文切换处的(多个)记载控制比特,从而使得方法300能够仅追踪特定的线程。

[0067] 方法300还包括确定高速缓存行是否参与记载的动作304。在一些实施例中,动作304包括:至少基于针对特定处理单元启用的记载来确定特定高速缓存行是否是记载中的参与者。例如,处理器102可以确定在动作301中检测到的操作中所涉及的高速缓存行是否涉及在记载中。如稍后将更详细讨论的,存在若干可以用于检测的机制,诸如使用共享高速缓存102b内的比特或使用高速缓存路锁定(way-locking)。

[0068] 方法300还包括使用CCP来标识存在要被记载到追踪的数据的动作305。例如,处理器102可以查阅其CCP以确定由于该操作而发生的高速缓存状态中的哪些转变,以及那些转变是否需要记载数据。稍后结合图6A-图9B给出使用CCP来标识追踪数据的详细示例。

[0069] 方法300还包括使用CCP将适当的数据记载到追踪的动作306。在一些实施例中,动作306包括:使数据被记载到追踪,该数据可用于重放该操作。当数据要被记载到(多个)追踪文件时,一个或多个数据分组可以被添加到适当的追踪数据流——诸如对应于特定处理单元的追踪数据流或通常对应于处理器102的追踪数据流。如果适当的追踪数据流通常对应于处理器102,则一个或多个数据分组可以标识特定的处理单元。注意,如果追踪数据流通常对应于处理器102,则数据流中本身的数据分组的固有顺序将提供一些附加排序信息,该信息在多个数据流被使用的情况下可能是不可用的。

[0070] 注意,当共享高速缓存102b包括多个高速缓存级别时,在一些实施例中,方法300在与系统存储器103交互的高速缓存级别处操作,因为它是处理高速缓存未命中的高速缓存级别。在此级别处进行操作使得每个处理单元102a的高速缓存活动能够得以表示,而不会造成冗余(即,不止一次表示一个单元的活动)。因此,例如,如果计算机系统101包括两个处理器102(即,两个处理器插槽),并且每个插槽包括一个“包含性”L3高速缓存以及在L3高速缓存下方的“包含性”L2高速缓存,则在一些实施例中,方法300在L3高速缓存上操作。方法300也可以在多个高速缓存级别处操作。例如,如果计算机系统101包括一个处理器102

(即,一个处理器插槽)并且包括用于该插槽的一个“独占”L3高速缓存以及在L3高速缓存下方的“包含性”L2高速缓存,则方法300可以在其上操作的是L3和L2高速缓存二者。下面讨论了展现混合的包含性/独占行为的高速缓存中的记载的其他示例。

[0071] 如以上结合动作304所提及,处理器102可以使用几种机制来确定高速缓存行是否是“记载中的参与者”。一种是用一个或多个附加“计账比特”来扩展共享高速缓存102b的每一行,这些“计账比特”可以被用作标志、用作处理单元标识符或用作处理器索引。用于控制这些“计账比特”的逻辑可以成为处理器的微代码102c的一部分。

[0072] 为了说明此实施例,图4A图示出了与图2的共享高速缓存200相似的示例共享高速缓存400a,其用一个或多个附加计账比特401来扩展其每个高速缓存行404。因此,每个高速缓存行404包括(多个)计账比特401、常规地址比特402和值比特403。

[0073] 在一些实现中,每个高速缓存行的(多个)计账比特401包括单个比特,其用作被处理器102用来指示高速缓存行是否正在参与追踪记载的标志(即,开启或关闭)。如果处理器的CCP具有足够的粒度(例如,如果CCP跟踪每个高速缓存行的一致性状态——在它与每个处理单元相关时,或者参考占有高速缓存行的一致性状态的处理单元的索引),则此单个比特可以足以利于记录稳健的完全确定的追踪(即,保证追踪的执行的完全可重构性的追踪)。

[0074] 在其他实现中,每行的(多个)计账比特401包括多个比特。可以以若干方式使用多个比特。使用一种在本文中被称为“单元比特”的方法,每个高速缓存行的(多个)计账比特401可以包括等于处理器102的处理单元102a的数目(例如,如果处理器102支持超线程,则是逻辑处理单元的数目,或者如果不支持超线程,则是物理处理单元的数目)的单元比特的数目。处理器102可以使用这些单元比特来跟踪哪个或哪些特定处理单元已经消耗了高速缓存行(或者如果高速缓存行尚未被消耗,则指出没有一个处理单元已经消耗了它)。因此,例如,由两个处理单元102a共享的共享高速缓存102b可以包括用于每个高速缓存行的两个单元比特。结合添加到每个高速缓存行的这些单元比特,实施例扩展处理器的微代码102c以利用这些单元比特来代表每个处理单元跟踪高速缓存行中的当前值是否已被记载(即,在追踪文件104d中),或者以其他方式让处理单元已知。如果处理器的CCP具有较粗的粒度(例如,如果CCP仅在高速缓存行的级别跟踪一致性状态),则这些单元比特可以提供附加信息以利于稳健的追踪。例如,如果高速缓存行被CCP标记为共享或独占,则单元比特可以被用来标识哪个或哪些处理单元共享高速缓存行,或者哪个处理单元具有独占性。

[0075] 使用在本文中被称为“索引比特”的另一种方法,每个高速缓存行的(多个)计账比特401可以包括多个索引比特以及“保留”值(例如-1),多个索引比特足以表示对参与记载的计算机系统101的(多个)处理器102的每个处理单元102a的索引。例如,如果计算机系统101的(多个)处理器102包括128个处理单元102a,则可以通过每个高速缓存行仅使用七个索引比特的索引值(例如,0-127)来标识这些处理单元。在一些实施例中,保留一个索引值(例如,“无效”)以指示没有处理器记载高速缓存行。因此,这将意味着七个索引比特加上保留值实际上将能够表示127个处理单元102a。例如,二进制值0000000-1111110可能对应于索引位置0-126(十进制),并且二进制值1111111(例如十进制的-1或127,取决于解释)可能对应于“无效”,以指示没有处理器记载了对应的高速缓存行——尽管此注释取决于实现可以变化。因此,处理器102可以使用单元比特来跟踪高速缓存行是否正在参与追踪记载(例

如,-1以外的值),并用作消耗了高速缓存行的特定处理单元(例如,最近消耗它的处理单元)的索引。该第二种方法具有在共享高速缓存102b中以很少的开销支持大量的处理单元的优点,其缺点是粒度小于第一种方法(即一次仅标识一个处理单元)。再次,如果处理器的CCP具有较粗的粒度(例如,如果CCP仅在高速缓存行的级别跟踪一致性状态),则这些索引比特可以提供附加信息以利于稳健的追踪。例如,如果高速缓存行被CCP标记为共享或独占,则索引比特可以被用来标识共享该高速缓存行的至少一个处理单元,或者哪个处理单元具有独占性。

[0076] 可以被处理器102使用来确定高速缓存行是否是记载中的参与者的另一种机制可以采用结合图4A讨论的概念,但是不用附加的计账比特(多个)计账比特410来对每个高速缓存行进行扩展。相反,该机制保留一个或多个高速缓存行404用于存储计账比特。图4B图示出了共享高速缓存400b的示例,该共享高速缓存400b包括存储存储器地址402和值403的常规高速缓存行405,以及用于存储应用于常规高速缓存行405的计账比特的保留的一个或多个高速缓存行406。保留的(多个)高速缓存行406的比特被分配到不同的计账比特组,其每一个对应于常规高速缓存行405的不同的一个。这些计账比特组可以用作标志比特、单元比特或索引比特,这取决于实现。

[0077] 可以被(多个)处理器102使用来确定高速缓存行是否是记载中的参与者的另一种机制是利用关联的高速缓存和路锁定。由于处理器的共享高速缓存102b通常比系统存储器103小得多(通常小数个数量级),并且因此,系统存储器103中的存储器位置通常远比共享高速缓存102b中的行多得多。这样,每个处理器定义了一种用于将系统存储器的多个存储器位置映射到高速缓存中的(多个)行的机制。处理器通常采用两种通用技术之一:直接映射和关联映射。使用直接映射,系统存储器103中的不同存储器位置仅被映射到高速缓存中的一行,使得每个存储器位置只能被高速缓存到高速缓存中的特定行中。

[0078] 另一方面,使用关联映射,系统存储器103中的不同位置可以被高速缓存到共享高速缓存102b中的多行之一。图5图示出了关联高速缓存映射的示例。在此,高速缓存502的高速缓存行504逻辑上被划分成两个高速缓存行的不同地址组,每个地址组包括两个高速缓存行504a和504b的第一组(被标识为索引0)以及两个高速缓存行504c和504d的第二地址组(被标识为索引1)。地址组中的每个高速缓存行与一个不同的“路(way)”相关联,以使高速缓存行504a由索引0、路0标识,高速缓存行504b由索引0、路1标识,依此类推。如进一步描绘的那样,存储器位置503a、503c、503e和503g(存储器索引0、2、4和6)被映射到索引0。这样,系统存储器中的这些位置中的每一个都可以被高速缓存到组内索引0处的任何高速缓存行(即,高速缓存行504a和504b)。所描绘的映射的特定模式仅出于说明和概念目的,并且不应被解释为可以将存储器索引映射到高速缓存行的唯一方式。

[0079] 关联高速缓存通常被称为N路关联高速缓存,其中N是每个地址组中“路”的数目。因此,图5的高速缓存502可以被称为2路关联高速缓存。处理器通常实现N路高速缓存,其中N是2的幂(例如2、4、8等),通常选择4和8的N值(尽管本文的实施例不限于任何特定的N值或N值的子集)。值得注意的是,1路关联高速缓存通常等效于直接映射高速缓存,因为每个地址组仅包含一个高速缓存行。另外,如果N等于高速缓存中的行数,则将其称为完全关联高速缓存,因为它包括一个包含高速缓存中所有行的地址组。在完全关联的高速缓存中,任何存储器位置都可以被高速缓存到高速缓存中的任何行。

[0080] 注意,图5表示系统存储器和高速缓存的简化视图,以便说明一般原理。例如,尽管图5将个体存储器位置映射到高速缓存行,但是应当理解,高速缓存中的每一行通常将与多个可寻址位置有关的数据存储在系统存储器中。因此,在图5中,系统存储器(501)中的每个位置(503a-503h)实际上可以表示多个可寻址存储器位置。另外,映射可以在系统存储器501中的实际物理地址与高速缓存502中的行之间,或者可以使用虚拟地址的中间层。

[0081] 关联高速缓存可以被用于确定高速缓存行是否是通过使用路锁定的记载中的参与者。出于某种目的,路锁定在高速缓存中锁定或保留某些路。特别地,本文的实施例利用路锁定来为正被追踪的线程保留一个或多个路,使得锁定/保留的路独占地被用于存储与该线程的执行有关的高速缓存未命中。因此,再次参考图5,如果为追踪的线程锁定了“路0”,那么高速缓存行504a和504c(即索引0、路0和索引1、路0)将独占地被用于与该线程的执行有关的高速缓存未命中,其余的高速缓存行将被用于所有其他高速缓存未命中。因此,为了确定特定的高速缓存行是否是记载中的参与者,处理器102仅需要确定高速缓存行是否是为正被追踪的线程所保留的路的一部分。

[0082] 图6A-图6D图示出了在图1、图2、图4A、图4B和图5的上下文中应用图3的方法300的具体示例600。图6A图示出了第一表600a,其示出共享高速缓存102b中的单个行上的四个处理单元102a(即,P0-P3)的读取和写入活动。图6B图示出了第二表600b,其基于这些读取和写入来指示跟踪的高速缓存一致性状态(例如,如使用处理器的CCP所跟踪的)的一个实施例。图6C图示出了第三表600c,其示出了如果根本使用计账比特,则可能被存储在共享高速缓存102b的计账比特中(如结合图4A和图4B所描述的)的内容。尽管通常仅一种类型的计账比特被使用(即,每行单元比特、每行索引比特或每行标志比特),但是为了说明完整起见,表600c示出了单元比特603、索引比特604和标志比特605的每一个。最后,图6D图示出了第四表600d,其示出了日志数据606的示例类型,其可以结合每个操作而潜在地被写入到(多个)追踪文件104d。

[0083] 为了描述简单起见,表600a描绘了一次仅单个处理单元102a的操作,但是应当理解,本文描述的原理适用于存在并发活动的情况(例如,由同一高速缓存行的两个或更多处理单元进行的并发读取)。另外,结合图6A-图6D描述的示例假定跟踪对于处理单元P0-P2是启用的,而对处理单元P3是禁用的。例如,如上所述,这可以被控制对应于每个处理单元的比特,诸如控制寄存器的多个比特。

[0084] 最初,为了易于描述,该示例将使用简化的高速缓存行状态,这些简化的高速缓存行状态是从以上讨论的CCP(即MSI、MESI和MOESI)中使用的高速缓存行状态(即,已修改、占有、独占、共享和无效)中得出的。在这种简化中,这些状态映射为“读取”状态(即,已从其中读取高速缓存行)或“写入”状态(即,已向其中写入高速缓存行)。下面的表1示出了这些映射的一个示例。注意,这些映射仅被用作示例,并且是非限制性的。例如,可能存在除本文讨论的CCP和状态以外的CCP和状态,并且,鉴于本文的公开内容,本领域的普通技术人员将认识到,可以用许多不同的CCP进行类似的映射。

[0085]

协议状态	映射状态
已修改	写入

[0086]

占有	读取
独占	写入
共享	读取
无效	无映射-高速缓存行被认为是空的

[0087] 表1

[0088] 值得注意的是,取决于从处理器102可获得什么数据和/或基于实现选择,实施例可以以不同的级别记载CCP数据。例如,基于“映射的”CCP状态(诸如表1中所示那些)、基于由处理器102使得可见的实际CCP状态(例如,已修改、占有、独占、共享和/或无效)、和/或甚至基于典型地不可由处理器102使得可见的较低级别的“原始”CCP数据,CCP数据可以被记载。

[0089] 转到图6A-图6D,表600a包括示出标识符(ID)的第一列601,其被用来指定操作之间的全局顺序。表600a还包括四个附加列602a-602d,其每一个对应于处理单元之一。尽管为简单起见,该示例使用全局ID,但应理解,实际上每个处理单元正常将使用其自己独立的标识符集对操作进行排序。这些ID可以包括指令计数(IC)或用于指定操作之间的顺序的任何其他适当的机制,诸如“跳转计数”加程序计数器。请注意,该示例以与MSI、MESI和MOESI CCP相一致的方式使用存储器,但为简单起见,它仅使用“已修改”、“共享”和“无效”状态。然而,应注意,一些CCP可以提供它们自己的唯一和/或单调递增的ID,这些ID也可以被记录在追踪中(例如,在每个分组中,或在偶然的分组中),以强烈地对追踪条目进行排序。即使CCP没有提供这样的ID,插槽计时器的值(例如TSC)或另一可排序ID潜在地可以被使用。

[0090] 如表600a中所示,处理单元P0在标识符ID[0]处执行读取,这引起数据DATA[1]被带入到高速缓存行中的高速缓存未命中。对应地,表600b示出:处理器的CCP指出高速缓存行现在已由P0“共享”。表600c示出:如果单元比特603被使用,则它们指示处理单元P0已消耗(即读取)高速缓存行(而处理单元P1-P3却尚未),如果索引比特604被使用,则它们指示P0已消耗高速缓存行,并且如果标志比特605被使用,则它指示某个处理单元已消耗高速缓存行。给定此状况,在动作303中,处理器102将确定记载针对P0是启用的,并且在动作304中,它将确定高速缓存行参与了记载(即,使用单元比特603、索引比特604、标志比特605或路锁定)。因此,在动作306中,如果必要,处理器102将利用CCP将适当的数据记载到(多个)追踪文件。在此,由于高速缓存行从无效(空)状态变为读取(表600a)/共享(表600b)状态,因此数据应被记载。如表600d的日志数据606中所示,处理器102可以在必要时记下处理单元(P0)(即,取决于数据分组是被记载为每个处理单元的单独数据流还是被记载为单个数据流);高速缓存行地址(@);指令计数或某个其他计数;以及被带入到高速缓存行中的数据(DATA[1])。如上面所讨论的,尽管指令计数典型地将是特定于处理单元的值,但是为了简化起见,表600d参考对应的全局ID(在这种实例中为IC[0])来参考指令计数。

[0091] 注意,在一些实施例中,高速缓存行地址(@)和数据(例如,DATA[1])可以被压缩在(多个)追踪文件104d内。例如,通过参考(明确或隐式地)先前记录的存储器地址中的“高”比特来避免记录存储器地址的“高”比特,从而可以压缩存储器地址。可以通过将数据值的比特归组为多个组(每个组包括多个比特)并将每个组与对应的“标志”比特相关联来压缩数据。如果一组等于特定模式(例如全0,全1等),则可以设置标志比特,而不必将该组比特

存储在追踪中。

[0092] 接下来，表600a示出：在ID[1]处，处理单元P1在高速缓存行上执行读取，从而读取数据DATA[1]。表600b示出：处理器的CCP指出高速缓存行现在由P0和P1“共享”。表600c示出：处理单元P0和P1已消耗高速缓存行(单元比特603)，P1已消耗高速缓存行(索引比特604)，或者某个处理单元已消耗高速缓存行(标志比特605)。注意，索引比特604仍然参考P0而不是P1也是正确的。表600d示出：使用CCP，处理器102确定该操作的记录要被记载。如图所示，处理器102可以记下处理单元(P1)；高速缓存行地址(@)；指令计数(IC[1])；高速缓存行已从读取(共享)状态变为读取(共享)状态；并且P0对先前的高速缓存行具有先前的访问权，但是现在P0和P1具有访问权。

[0093] 接下来，表600a示出：在ID[2]处，处理单元P0执行对高速缓存行的写入，写入数据DATA[2]。表600b示出：处理器的CCP指出高速缓存行现在被P0“修改”，而对于P1“无效”。表600c示出：仅处理单元P0已消耗高速缓存行(即，更新了其值)(单元比特603)，P0已消耗高速缓存行(索引比特604)，或者某个处理单元已消耗高速缓存行(标志比特605)。表600d示出：使用CCP，由于高速缓存行已被写入/修改，因此处理器102确定该操作的记录需要被记载。如图所示，处理器102可以记下处理单元(P0)；高速缓存行地址(@)；指令计数(IC[2])；高速缓存行已从读取(共享)状态变为写入(已修改)状态；并且P0和P1对先前的高速缓存行具有先前的访问权，但是现在只有P0具有访问权。

[0094] 注意，可以使用表600b中所示的CCP状态找到有关哪个或哪些处理单元对高速缓存行具有先前的访问权的信息。然而，注意，一些CCP可能没有维持足够的信息来这样做(例如，仅在高速缓存行级别跟踪一致性状态的CCP)。备选地，如果使用单元比特603，则可以从单元比特得出该信息。对应地，图6D中所示的日志数据606假定是维持该信息的稳健CCP，或者单元比特603的使用。

[0095] 如果不使用这些中的任何一个(例如，如果CCP不那么稳健，并且如果使用索引比特604、标志比特605或路锁定而不是单元比特603)，则日志数据606可以是不够彻底或更大。作为第一示例，如果CCP仅在高速缓存行级别跟踪一致性状态，并且如果索引比特604被使用，则这二者可以被用来标识高速缓存行状态无效(对于所有处理单元)，其是已修改的(以及使其修改的处理单元的索引)，它是独占的(以及使其独占的处理单元的索引)或它是共享的(并且所有处理单元都具有访问权)。这可以导致更简单的硬件实现，其缺点是，当是时候将高速缓存行从共享更改为已修改或独占时，必须通知所有处理单元，而不是仅通知更细粒的CCP知道的处理单元以共享高速缓存行。作为第二示例，索引比特604可以被用来标识访问高速缓存行的最后处理单元。然后，如果高速缓存是包含性的(即，在L2或L1高速缓存级别处的访问之后隐藏了如此多的读取)，那么即使处理单元正在读取同一高速缓存行，L3高速缓存也可能会看到来自相同处理单元的相对较少的重复请求。记载每个索引改变以进行读取->读取，然后进行读取->写入、写入->写入和写入->读取日志，该索引也给出了与单元比特603的使用相同的数据，但代价是可能稍大的追踪。作为第三示例，每个高速缓存行可以包括单个标志比特，但是CCP可以参考占有高速缓存行的一致性状态的处理单元的索引来跟踪每个高速缓存行的一致性状态。在这里，与单元比特被使用或者CCP跟踪个体处理单元相比，该追踪可能记录更多的高速缓存行移动，但是该追踪仍然可以是完全确定的。在下文中结合图9A和图9B呈现了当具有关于每个处理单元的信息时与仅具有关于处

理器索引的信息时的追踪文件大小的简要比较。

[0096] 返回图6A,表600a示出:在ID[3]处,处理单元P1执行从高速缓存行的读取,读取数据DATA[2]。表600b示出:处理器的CCP指出高速缓存行现在由P0和P1“共享”。表600c示出:处理单元P0和P1已消耗高速缓存行(单元比特603),P1已消耗高速缓存行(索引比特604),或者某个处理单元已消耗高速缓存行(标志比特605)。注意,索引比特604仍然参考P0而不是P1也是正确的。表600d示出:使用CCP,由于高速缓存行已从写入(已修改)状态变为读取(共享)状态,因此处理器102确定该操作的记录需要被记载。如图所示,处理器102可以记下处理单元(P1);高速缓存行地址(@);指令计数(IC[3]);高速缓存行已从写入(已修改)状态变为读取(共享)状态;并且P0对先前的高速缓存行具有先前的访问权,但是现在P0和P1具有访问权。

[0097] 接下来,表600a示出:在ID[4]处,处理单元P0再次执行对高速缓存行的写入,这一次是写入数据DATA[3]。表600b示出:处理器的CCP指出高速缓存行再次被P0“修改”,而对于P1“无效”。表600c示出:仅处理单元P0已消耗高速缓存行(单元比特603),P0已消耗高速缓存行(索引比特604),或者某个处理单元已消耗高速缓存行(标志比特605)。表600d示出:使用CCP,由于高速缓存行已被写入/修改,因此处理器102确定该操作的记录需要被记载。如图所示,处理器102可以记下处理单元(P0);高速缓存行地址(@);指令计数(IC[4]);高速缓存行已从读取(共享)状态变为写入(已修改)状态;并且P0和P1对先前的高速缓存行具有先前的访问权,但是现在只有P0具有访问权。

[0098] 接下来,表600a示出:在ID[5]处,处理单元P2执行从高速缓存行的读取,读取数据DATA[3]。表600b示出:处理器的CCP指出高速缓存行现在由P0和P2“共享”。表600c示出:处理单元P0和P2已消耗高速缓存行(单元比特603),P2已消耗高速缓存行(索引比特604),或者某个处理单元已消耗高速缓存行(标志比特605)。注意,索引比特604仍然参考P0而不是P2也是正确的。表600d示出:使用CCP,由于高速缓存行已从写入(已修改)状态变为读取(共享)状态,因此处理器102确定该操作的记录需要被记载。如图所示,处理器102可以记下处理单元(P2);高速缓存行地址(@);指令计数(IC[5]);高速缓存行已从写入(已修改)状态变为读取(共享)状态;并且P0对先前的高速缓存行具有先前的访问权,但是现在P0和P2具有访问权。

[0099] 接下来,表600a示出:在ID[6]处,处理单元P1执行从高速缓存行的读取,并且还读取数据DATA[3]。表600b示出:处理器的CCP指出高速缓存行现在由P0、P1和P2“共享”。表600c示出:处理单元P0、P1和P2已消耗高速缓存行(单元比特603),P1已消耗高速缓存行(索引比特604),或者某个处理单元已消耗高速缓存行(标志比特605)。注意,索引比特604仍然参考P0或P2而不是P1也是正确的。表600d示出:使用CCP,处理器102确定该操作的记录要被记载。如图所示,处理器102可以记下处理单元(P1);高速缓存行地址(@);指令计数(IC[6]);高速缓存行已从读取(共享)状态变为读取(共享)状态;并且P0和P2对先前的高速缓存行具有先前的访问权,但是现在P0、P1和P2具有访问权。

[0100] 接下来,表600a示出:在ID[7]处,处理单元P3执行从高速缓存行的读取,并且还读取数据DATA[3]。表600b示出:处理器的CCP指出高速缓存行现在由P0、P1、P2和P3“共享”。表600c示出:单元比特603、索引比特604或标志比特605均未被更新。这是因为记载对于P3是禁用的,并且出于追踪目的,因此没有通过执行读取来“消耗”高速缓存行。表600d示出:未

记载任何数据。这是因为在动作303中，处理器102将确定针对P3没有启用记载。

[0101] 接下来，表600a示出：在ID[8]处，处理单元P3执行写入高速缓存行，写入数据DATA[4]。表600b示出：处理器的CCP指出高速缓存行现在对于P0、P1和P2“无效”，并由P3“修改”。表600c示出：单元比特603、索引比特604和标志比特605全部反映了没有被任何处理单元消耗的高速缓存行。这是因为记载针对P3是禁用的，因此出于追踪目的，在它执行写入时它并未“消耗”高速缓存行；此外，该写入使针对其他处理单元的高速缓存行中的值无效。表600d示出：未记载任何数据。再次，这是因为在动作303中，处理器102将确定针对P3没有启用记载。

[0102] 接下来，表600a示出：在ID[9]处，处理单元P0执行对高速缓存行的写入，写入数据DATA[5]。表600b示出：处理器的CCP指出高速缓存行现在被P0“修改”，而对于P3“无效”。表600c示出：没有任何处理单元消耗过高速缓存行。这是因为没有结合此操作制成日志条目——如表600d中所反映的。无需制成日志条目，因为写入的数据将通过P0的线程的指令的正常执行来再现。然而，在这种情境下，可选地条目可以被写入到追踪(即，写入到未由启用了记载的处理单元记载的高速缓存行)，以向追踪的消费者提供额外的数据。在这种情境下，日志条目可能被视为对高速缓存行值的读取加上对DATA[5]的写入。

[0103] 接下来，表600a示出：在ID[10]处，处理单元P2执行从高速缓存行的读取，读取数据DATA[5]。表600b示出：处理器的CCP指出高速缓存行现在由P0和P2“共享”。表600c示出：处理单元P2已消耗高速缓存行(单元比特603)，P2已消耗高速缓存行(索引比特604)，或者某个处理单元已消耗高速缓存行(标志比特605)。表600d示出：使用CCP，处理器102确定该操作的记录需要被记载，因为值高速缓存行先前尚未被记载(即，在ID[9]处它未被记载)。如图所示，处理器102可以记下处理单元(P2)；高速缓存行地址(@)；指令计数(IC[10])；被带入到高速缓存行中的数据(DATA[5])；并且P2对高速缓存行具有访问权。取决于特定CCP和计账比特提供的信息，可能也记载P0也对高速缓存行具有访问权。

[0104] 接下来，表600a示出：在ID[11]处，处理单元P1执行从高速缓存行的读取，也读取数据DATA[5]。表600b示出：处理器的CCP指出高速缓存行现在由P0、P1和P2“共享”。表600c示出：处理单元P1和P2已消耗高速缓存行(单元比特603)，P1已消耗高速缓存行(索引比特604)，或者某个处理单元已消耗高速缓存行(标志比特605)。注意，索引比特604仍然参考P2而不是P1也是正确的。表600d示出：使用CCP，处理器102确定该操作的记录需要被记载。如图所示，处理器102可以记下处理单元(P1)；高速缓存行地址(@)；指令计数(IC[11])；高速缓存行已从读取(共享)状态变为读取(共享)状态；并且P2对先前的高速缓存行具有先前的访问权，但是现在P1和P2具有访问权。注意，值(DATA[5])不需要被记载，因为在ID[10]处它由P2记载。

[0105] 接下来，表600a示出：在ID[12]处，处理单元P0执行从高速缓存行的读取，还读取数据DATA[5]。表600b示出：处理器的CCP仍然指出高速缓存行现在由P0、P1和P2“共享”。表600c示出：处理单元P0、P1和P2已消耗高速缓存行(单元比特603)，P0已消耗高速缓存行(索引比特604)，或者某个处理单元已消耗高速缓存行(标志比特605)。注意，索引比特604仍然参考P1或P2而不是P0也是正确的。表600d示出：使用CCP，处理器102可以确定该操作的记录要被记载。在这种情况下，处理器102可以记下处理单元(P0)；高速缓存行地址(@)；指令计数(IC[12])；高速缓存行已从读取(共享)状态变为读取(共享)状态；并且P1和P2对先前的

高速缓存行具有先前的访问权，但是现在P0、P1和P2具有访问权。没有值(DATA[5])被记载，因为其可从P2获得。

[0106] 备选地，处理器102可能仅在ID[12]处参考P0，因为P0已经具有高速缓存行的值(即，因为它在ID[9]处写入了该值)。甚至可能避免在ID[12]处的任何记载，因为可以在重放时使用试探法来恢复该值(即DATA[5])，而无需在追踪中参考P0的信息。然而，那些技术在计算上可能是昂贵的，并且会降低系统检测重放何时“脱轨”的能力。一个示例试探法是认识到在处理单元之上的存储器访问通常是强烈有序的(基于CCP数据)，因此对于给定的存储器位置，重放可以使用在这些单元之上的最后一个值。

[0107] 接下来，表600a示出：在ID[13]处高速缓存行被逐出。结果，表600b示出CCP条目为空，表600c示出计账比特反映出没有处理单元已消耗高速缓存行，并且表600d示出没有数据被记载。

[0108] 注意，尽管为了完整起见，日志数据606列出了所有结束访问状态(即，哪些处理单元现在对高速缓存行具有访问权)，但是该信息可能是隐式的，并且可以通过省略该信息来减小追踪文件的大小。例如，在从写入->读取的转变中，在读取之后具有访问权的处理单元的列表始终是先前具有访问权的处理单元加上执行了读取的处理单元。在从读取->写入的转变或从写入->写入的转变中，在写入之后具有写入访问权的处理单元的列表始终是执行了写入的处理单元。在从读取->读取的转变中，在读取之后具有访问权限的处理单元的列表始终是转变之前具有访问权限的处理单元加上执行了读取的处理单元。

[0109] 通常，为了生成完全确定的追踪文件，CCP将指示在处理单元上(例如，从P0到P1)的所有转变(即，写入->读取，写入->写入，读取->写入和读取->读取)被记载。然而，无需记载同一处理单元内(例如，从P0到P0)的转变。这些不需要被记载，因为它们将通过在该处理单元处执行的线程的正常执行而被再现。

[0110] 应当理解，使用诸如在以上示例中记载的数据之类的数据，以及对进行记录的处理器102所使用的CCP的进一步知识，对发生在每个线程处的操作的完全排序可以被重构，并且不同处理单元之间的操作的至少部分排序可以被重构。因此，无论是索引过程还是通过追踪文件的重放，上述每个操作都可以被重构——即使它们并未全部被明确地记录在(多个)追踪文件104d中也是如此。

[0111] 在一些实施例中，追踪器104a可以在(多个)追踪文件104d中记录附加数据分组，以便增强在处理单元之上的操作排序的记载。例如，追踪器104a可以记录一些事件排序信息，诸如单调递增数字(MIN)(或某个其他计数器/计时器)，以便提供具有在线程之上的MIN(或其他计数器/计时器)的事件的完整排序。这些MIN可以被用来标识与事件相对应的数据分组，这些事件被定义为在线程上是“可排序的”。这些事件可以基于“追踪存储器模型”而被定义，该模型定义了线程可以如何通过共享存储器进行交互以及它们在存储器中的数据共享使用。作为另一个示例，追踪器104a可以(周期性地或随机地)基于定义的确定性算法和定义的一组寄存器(例如，程序计数器、堆栈、通用寄存器等)来记录处理器状态的哈希。作为又一示例，追踪器104a可以(周期性地或随机地)强行记载高速缓存行数据。作为又一示例，追踪器104a可以在记载哈希的追踪“转变”分组中包括其隐式承载的全部或部分数据(例如，几比特)。因此，当在重放时重构该隐式数据时，可以对隐式数据的(多个)适当部分进行哈希处理并将其与这些转变分组进行匹配，以帮助标识其排序。例如，如果CCP无法跟

踪与高速缓存行相关联的处理器索引(如果高速缓存行处于共享状态),则这可能是有用的。

[0112] 当追踪器104a在(多个)追踪文件104d中记录附加数据分组以增强排序时,可能会省略记录在处理单元之上的一些转变。例如,可能会省略记录在线程之上的一些读取->读取转变。在一些情形下,这可能会导致“减弱”的非确定性追踪——因为一些读取的排序可能无法基于追踪和CCP而被完全重构——但是附加排序信息(例如,MIN、处理器状态的哈希、额外的高速缓存行数据)可以帮助减少重放期间的搜索空间,以找到不会使追踪的重放“脱轨”的读取的有效排序。省略在线程之上的一些读取->读取转变的好处包括追踪大小和对处理器101的潜在简化修改,以利于追踪。

[0113] 图7A图示出了一个示例,其中取决于处理器如何被跟踪,可能从追踪中省略一些读取->读取转变。类似于图6A,图7A包括具有全局ID 701的表700a,以及对应于三个处理单元(P0-P2)的三列(702a-702c)。省略一些读取->读取转变是基于两个观察而被建立的。首先,需要对写入进行排序;然而,两次接连写入之间的所有读取(例如,在ID[3]-ID[7]处的读取)将读取相同的值,因此这些读取之间的顺序无关紧要(并且因此省略那些读取->读取转变的追踪可以是确定性的)其次,在重放时让读取与写入“相交”(即,对同一高速缓存行的读取和写入以不正确的顺序被重放)意味着未使用正确的数据进行重放;然而,具有避免犯此错误的数据(例如MIN等)将有助于标识有效的排序。

[0114] 在表700a中所示的示例中,处理单元P2仅对共享数据执行读取,而那些共享的读取仅从其他读取中“窃取”(例如,假设ID[9]留下了共享的高速缓存行)。如果针对任何读取->读取转变(即ID[4]-ID[7]和ID[10])都没有制成日志条目,则追踪中将没有信息来正确放置P2的读取。基于写入,可以得出结论:P2永远不读取值DATA[1](即,由于在ID[2]处的写入不从P2窃取),并且缺少针对P2的读取->读取转变的日志条目(即,ID[4],ID[7]和ID[10]),对于P2可以得出的所有结论是,P2在ID[2]和ID[8]之间至少进行了一次读取。然而,如果操作针对ID[4]和ID[10]的日志条目,则可能不需要记载的其余读取(即,ID[5]-ID[7],如图7B中所示)可以被定位。这些读取中的每一个都与最后一次记载的读取(即,在ID[4]处)属于同一互写入部分。因此,可以基于写入从中窃取了什么来定位这些读取(并且如果没有操作从读取中窃取,那么在其之后直到下一个被记载的分组才存在写入)。

[0115] 鉴于表700a,图7B图示出了表700b,其示出了记载数据——省略了图7A中突出显示的读取->读取转变,如果使用“单元比特”,则该数据可能会被记录。图7C图示出了表700c,其示出了如果使用“索引比特”并且在读取时更新索引则可能会被记录的记载数据。

[0116] 如上面简单提及的,一些高速缓存包括包含性层和独占层(即,非完全包含性高速缓存)。本文所述的记载技术适用于这些高速缓存以及纯包含性或独占高速缓存。作为示例,图8A图示出了包括两个处理器801a/801b(例如,对应插槽中的两个处理器)的计算环境800a。每个处理器801包括四个处理单元802a/802b(例如,物理或逻辑处理单元)。每个处理器801还包括三层高速缓存,包括L1层803a/803b、L2层804a/804b和L3层805a/805b。如图所示,每个高速缓存包括四个L1高速缓存803——每一个对应于处理单元802之一。另外,每个高速缓存包括两个L2高速缓存804——每一个对应于处理单元802中的两个。另外,每个高速缓存包括用于处理器801中的所有处理单元802的一个L3高速缓存805。处理单元和一些高速缓存被各个地标识——例如,处理器801a的处理单元802a被标识为A0-A3,L2高速缓存

被标识为A4和A5，并且L3高速缓存被标识为A6。类似的标识符被用于处理器801b中的对应组件。与处理单元A0、A1、A2、B0和B1相关联的星号(*)指示针对这些处理单元，记载被启用。

[0117] 在计算环境800a中，高速缓存可以展现出包含性和独占行为的混合。例如，当仅处理单元A0正在使用高速缓存行时，处理器801a的A6 L3高速缓存存储高速缓存行可能是低效率的。相反，在这种情况下，高速缓存行可以被存储在A0的L1高速缓存和A4 L2高速缓存中，但不能被存储在A1的L1高速缓存或A5 L2高速缓存或更低级别的高速缓存中。为了释放空间，一些高速缓存可以允许A6 L3高速缓存在这种情形下逐出该高速缓存行。发生这种情况时，A1可以从A4 L2高速缓存中获取高速缓存行，这在包含性高速缓存中是正常的。然而，由于A6 L3高速缓存或A5 L2高速缓存中不存在高速缓存行，因此一些高速缓存实现也可以允许高速缓存行的横向移动，诸如从A0的L1高速缓存到A2或A3的L1高速缓存。这可能给使用CCP进行追踪带来一些挑战。以下示例说明了在这种情形下如何实现使用CCP进行追踪。

[0118] 图8B图示出了包括表800b，其示出了由一些处理单元802执行的示例读取和写入操作。表800b的格式类似于表600a的格式。鉴于计算环境800a和表800b，现在给出三个不同的记载示例，每个示例使用不同的高速缓存行为。在以下针对使用CCP进行记载的原理的上下文中描述这些示例：

[0119] (1)通常，当地址(高速缓存行)从“未被记载”变为“被记载”时(即，基于在动作304中确定高速缓存行参与记载)时，记载数据；

[0120] (2)通常，当高速缓存行从“被记载”变为“未被记载”或“逐出”时，避免记载(尽管如果该数据被记载，日志仍然有效)。然而，记载逐出是有效的。这样做会增加追踪的大小，但会提供附加信息，该附加信息可以有助于标识追踪数据流之间的排序，可以有助于标识追踪的重放何时已“脱轨”，并且可以提供附加追踪分析。关于追踪分析，记载逐出可以提供有关如何使用高速缓存的更多信息，可以被用来标识已执行代码的性能特性，还可以帮助查明给定高速缓存行在其期间存储特定值的时间窗口。稍后结合图10A和图10B讨论用于记载逐出的实施例；

[0121] (3)当高速缓存行在核上移动时记载移动，或者以提供新信息的方式对一致性状况进行高速缓存；

[0122] (4)当处理单元进行写入时，使针对所有其他处理单元的高速缓存行无效。如果尚未针对处理单元记载高速缓存行，则系统将无法记载高速缓存行或将写入视为：(i)读取(即，记载高速缓存行并打开记载跟踪)，以及(ii)写入，假定高速缓存行中的存储器对处理单元是可读的。对于处理器来说，关闭记载而不记载写入可能是合法的，但效率较低——但是，这会丢失需要在重放时被重构的信息，并且平均而言，追踪大小可能效率很低，因为记载参考比稍后记载数据的完整高速缓存行更便宜；

[0123] (5)(例如，按照上述原则2)过度记载(over-log)以帮助以后重构追踪是有效的。尽管这会增长追踪大小，但其不会影响正确性。例如，一些读取->读取转变可以被省略(如以上结合图7A-图7C所描述的)，但是任何以写入开始或结束的跨核转变应被显式或隐式记载。在另一个示例中，实施例可以在任何时候将附加数据分组(例如，其提供附加的排序信息和/或哈希)添加到追踪。在又一示例中，实施例可以记载在其CCP转变到写入状态之后写入何时第一次被提交给高速缓存行(即，由于推测性执行可能引起高速缓存行转变到写入状态，但实际上未提交任何写入给它)。记载这些写入可以利于以后每核追踪流的分离。在

又一个示例中，实施例可以记载间接跳转或在分离追踪数据流时有助于快速减小搜索空间的其他信息；以及

[0124] (6)非完整日志(即未记载所有转变的日志)仍然可以被用来重放追踪。这可以在重放时带来额外的计算成本以计算未命中的片段。

[0125] 在第一示例中，CCP跟踪每个处理单元的高速缓存行状况(即，每个核具有其自己的读取和写入状况)。在此示例中，高速缓存的行为与包含性高速缓存的行为非常相似，除了以下之外：可能存在在记载时不可用、跨越高速缓存(cross-cache)或跨越插槽(cross-socket)移动的数据。为简便起见，在这些示例中，处理单元802被称为“核”，并且处理器801a和801b被称为处理器A和B或插槽A和B。此外，简化的记载符号“ID：核：来自：转变(即从->到)被用来表示可以被记载的数据的类型。该符号在行内被更详细地解释。对于第一个示例，记载可以包括：

[0126] 在ID[0]处，“0:A0:R[DATA]->[1]”——即，在ID[0]处，按照上述原理1记载核A0读取了DATA[1]。

[0127] 在ID[1]处，“1:B0:R[DATA]->[1]”——即，在ID[1]处，也按照上述原理1记载核B0读取了DATA[1]。如果处理器B中的高速缓存不知道A0已经记载了数据，那么处理器B自己记载它。备选地，如果处理器B中的高速缓存知道A0已经记载了DATA[1]，那么该日志条目可以包括“1:B0:R[A0]->R”。

[0128] 在ID[2]处，“2:A1:R[A0]->R”——即，在ID[2]处，记载核A1进行了读取->读取转变，并且A0具有访问权。由于高速缓存行状态与处理器B共享，因此该条目可以是“2:A1:R:[A0,B0]->R”——即，在ID[2]处，记载A1进行了读取->读取转变，并且A0和B0具有访问权。由于跨越插槽通常比在插槽内进行记载更为昂贵，因此对于读取->读取转变，优选第一个日志条目。然而，当记载去往/来自跨越插槽的写入时，记载也跨越插槽。

[0129] 在ID[3]处，一些实施例任何内容也不记载。备选地，由于A2核尚未记载任何内容，并且它所做的第一件事就是写入，因此这可以被记载为读取->写入。无论哪种方式，由于发生写入，因此其他核使其高速缓存行状态无效。在ID[3]处记载读取->写入的成本(例如，就追踪数据而言)典型地比在ID[4]处记载实际数据的成本低，因此在此处记载可能是有益的。在这种情况下，日志条目可以包括“3:A2:R[A0,B1,B0]->W”——即，核A2进行了读取->写入转变，并且核A0、B1和B0具有访问权。

[0130] 在ID[4]处发生了什么取决于在ID[3]处记载了什么。如果在ID[3]中没有记载任何内容，那么数据被记载(即“4:A2:R[DATA]->[2]”)。另一方面，如果在ID[3]中分组被记载，那么没有任何可记载的内容。

[0131] 在ID[5]处，存在跨越核的读取。然而，如果A2核仍具有已修改的高速缓存行(或等效的)，那么高速缓存行将服务该请求(无法从存储器中为其服务)。在那种情况下，插槽B将知道这出自插槽A，并且可以避免重新记载数据；它可以被记载为“5:B0:W[A2]->R”。如果高速缓存从主存储器中得到数据(如果插槽A能够更新主存储器并针对该行共享其高速缓存一致性状态，则这可能是这种情况)，那么该条目可能为“5:B0:R[DATA]->2”。

[0132] 在ID[6]处，操作是正常读取。就像在ID[2]处的读取一样，插槽B可能知道或不知道插槽A的数据。如果知道，则日志条目可以包括“6:B1:R[B0,A2]->R”；否则它可以包括“6:B1:R[B0]->R”。

[0133] 在ID[7]处，如果针对B0的高速缓存行尚未被逐出，则不存在要记载的任何内容。如果其已被逐出，则处理器B将数据记载为出自另一核，或者记载高速缓存行数据。在完全包含性高速缓存中通常不会发生对一个核的逐出，但是对插槽中的其他核则并非如此。在完全包含性高速缓存中，如果插槽中的任何核在其L1高速缓存中具有高速缓存行，那么L3具有高速缓存行，因此针对一个核而非另一个核，无法逐出高速缓存行。

[0134] 在ID[8]处，由于自此以来A0核没有记载任何内容，并且记载的第一个操作是写入，因此这类似于在ID[3]处的操作。处理器A可以将这记载为读取->写入；备选地，但可能不太优选地，处理器A不能记载任何内容。如果分组被记载，则其内容将取决于插槽A是否可以看到插槽B而变化。如果它无法看到，则该分组可以包括“8:A0:R[A2]->W”，但是如果它可以看到，则该分组可以包括“8:A0:R[B0,B1,A2]->W”。

[0135] 在ID[9]处，如果在ID[8]处分组被记载，则不存在可记载的内容(因为它是对已记载的高速缓存的写入)，但是如果其尚未被记载，针对其他核的高速缓存行状态典型是无效的。

[0136] 在ID[10]处，记载取决于在ID[8]处记载了什么。如果在ID[8]处没有数据被记载，那么它需要在此处完成，因此分组可以包括“10:A1:R[DATA]->[4]”。如果在ID[8]处分组被记载，则这是正常的写入->读取分组(例如，“10:A1:W[A0]->R”)。

[0137] 在ID[11]处，读取->读取转变被记载。如果在ID[8]处分组被记载，那么A0在核的源列表上(例如，“11:A2:R[A0,A1]->R”)；否则，A0不在列表中(例如，“11:A2:R[A1]->R”)。

[0138] 在ID[12]处，如果插槽B可以看到插槽A，则这是读取->读取分组(例如，“12:B0:R[A0,A1,A2]->R”)。如果它无法看到，则它是完整的数据日志(例如，“12:B0:R[DATA]->[4]”)。

[0139] 在ID[13]处，数据出自B0，加上插槽A——如果它可见(例如，“13:B1:R[A0,A1,A2,B0]->R”)。如果在ID[8]处写入未被记载，则该列表可忽略核A0。

[0140] 在ID[14]处，如果已在ID[8]处分组被记载，确实已记载，则无需记载任何内容。否则，A0将从A1和A2中得到数据，潜在地加上插槽B——如果它可以被看到的话。这样，该分组可以包括“14:A0:R[A1,A2,B0,B1]->R”。

[0141] 注意，尽管该示例将插槽一起记载，但是以隔离的方式记载每个插槽是正确的，类似于可以以隔离的方式记载线程的方式。这可能会导致较大的追踪，但是它具有不必改变处理器中的任何跨越插槽通信机制的优点。

[0142] 而且，在任何时刻，高速缓存行都可能被逐出，这将意味着数据需要从另一个核进行收集或被重新记载。例如，如果在ID[11]之前A0使其高速缓存行被逐出，那么A2将从A1得到值。如果A1和A0两者均被逐出，那么处理器A可能需要将高速缓存行值记载到针对A2的追踪中。

[0143] 最后，一些处理器可能知道数据出自另一个插槽，但是不知道该插槽中的哪个核。在那些情况下，处理器可以将优先级(源)记载为插槽ID，记载数据本身，或记载插槽ID和数据的哈希(即，帮助跨越插槽访问排序，但不必将整个数据记载到追踪)。

[0144] 在第二示例中，CCP使用索引而不是分离跟踪每个核的高速缓存一致性。在这种环境中，可以跨越插槽或在插槽内跟踪索引。由于跨越插槽与插槽内通信的执行，后一种情况(插槽内)可能是更实用的。当在插槽内跟踪索引时，当数据跨越插槽移动时，追踪可能需要

记载一些内容。这可以包括记载来自另一插槽的索引(但是这对于确定性追踪而言可能不一定足够唯一),记载高速缓存行值的一个或多个部分的哈希或在发送插槽的追踪上记载分组以指示该数据已发送。

[0145] 当在使用非完全包含性高速缓存时跟踪核索引时,当L1高速缓存可能具有不在L3高速缓存中的数据时,产生复杂化。因此,例如,假设以下事件序列:(i)A0在其L1高速缓存中得到一行(因此索引比特指向A0);(ii)A1在其L1高速缓存中得到该行(因此索引比特指向A1);(iii)L3高速缓存逐出该行;(iv)A1将该行从L1高速缓存中逐出;以及(v)A2从其L1高速缓存中的A0得到高速缓存行。在这里,尽管A2从A0获取了高速缓存行,但索引并未指向A0。这使记载映射到追踪变得复杂。一些解决方案可以包括:添加额外的信息(如上所述),诸如高速缓存行数据的一个或多个部分的哈希;周期性地添加冗余信息,像通用寄存器的哈希,等等。记载逐出也可能有帮助,但是这可能会大大增长追踪文件大小并使记载变得复杂(例如,记载不在L2或L3高速缓存中的L1高速缓存逐出,但不记载在L2或L3高速缓存中的L1高速缓存逐出)。

[0146] 在一些实施例中,当数据从L3高速缓存移动到子级L2或L1高速缓存时,仅在索引改变时才制成日志条目。例如,假设A0在其L1高速缓存中具有该行(因此索引比特指向A0),那么A1在其L1高速缓存中得到该行(A1处的索引),然后两者都逐出该高速缓存行,但公共L2(或L3)仍然具有它。如果L2高速缓存服务于A1,那么没有可记载的内容。如果L2高速缓存服务于A0,那么在已知A0已经具有该数据的情况下,无需制成日志条目;但是如果未知(或无法确定)A0是否已具有该数据,那么处理器可能需要记载读取->读取。

[0147] 表800d呈现了表800b的操作的日志,假定插槽独立地进行记载,通过索引执行跟踪,没有额外的隐藏逐出,并且所有影响CCP并且在打开记载时发生的写入被记载(例如,如果存在由同一核进行的接连写入,并且另一个核或其他外部实体的写入之间不存在访问权,则需要记载一次写入)。对于第二个示例,记载可以包括:

[0148] 对于ID[0],“0:A0:R[DATA]->[1]”。

[0149] 对于ID[1],“1:B0:R[DATA]->[1]”——即,回顾,每个插槽被分离记载。

[0150] 对于ID[2],“2:A1:R[A0]->R”。

[0151] 对于ID[3],“3:A2:R[A1]->W”。

[0152] 对于ID[4],什么也没有。

[0153] 对于ID[5],“5:B0:R[DATA]->[2]”。这是因为在ID[3]处的写入使在所有插槽之上的行无效,并且插槽被独立追踪(如上所陈述)。

[0154] 对于ID[6],“6:B1:R[B0]->R”。

[0155] 对于ID[7],如果针对B0的高速缓存行尚未被逐出,则无需记载任何内容。

[0156] 对于ID[8]:“8:A0:R[A2]->W”,因为记载比特打开(并且尽管该核之前尚未记载数据)。此条目展示了如何用索引仅了解对插槽中的最后一个占有者。

[0157] 对于ID[9],没有可记载的内容。

[0158] 对于ID[10],“10:A1:W[A0]->R”。

[0159] 对于ID[11],“11:A2:R[A1]->R”。

[0160] 对于ID[12],“12:B0:R[data]->[4]”。这是因为在ID[8]处高速缓存行在所有插槽之上均无效。

[0161] 对于ID[13],“13:B1:R[B0]->R”。

[0162] 对于ID[14],“14:A0:R[A2]->R”。注意,在ID[11]处索引已被更新为A2。还应注意,由于索引不承载该信息,同时在每个处理器状态(单元比特)能够承载该信息之前,不知道该核已经具有该数据(即ID[9])。

[0163] 在第三示例中,环境800a中的高速缓存无法保持跟踪哪个核对高速缓存行具有最后的共享(读取)访问权。因此,在此示例中,无法跟踪最后一个读取器的索引,因为没有比特可以这样做。在这里,CCP可以使用一个索引值(其未映射到任何核)来发信号通知共享行,使用另一个索引值来发信号通知无效行,以及使用处理器索引用于“已修改”状态(例如,使用MSI协议)。在这个第三示例中,记载可以包括记载分组中的高速缓存的索引,而不是核的索引。父级到子级的移动无需被记载,但可以被记载为额外数据。如果父级到子级的移动未被记载,那么可能需要提供父级到子级的高速缓存层次结构以用于对日志进行解释。

[0164] 如上面所提及的,在一些环境中,高速缓存的每个高速缓存行可以包括单个标志比特,但是处理器的CCP可以参考占有高速缓存行的一致性状态的处理单元的索引,来跟踪每个高速缓存行的一致性状态。如所提及的,这产生了完全确定的追踪,但是与具有每个处理单元的信息的情况(例如,每个处理单元都进行跟踪的CCP,结合每个高速缓存行的标志比特)相比,可以产生更大的追踪。图9A和图9B图示出了在这两种情形下记载可能如何不同(即,CCP单元信息加上高速缓存行标志比特与CCP索引加上高速缓存行标志比特)。图9A图示出了表900a,其示出了两个处理单元(P0和P1)的读取和写入,并且图9B图示出了表900b,其比较了可以在这两个环境中可以何时制成日志条目。在这些示例中,假定标志比特开始清除,并且单元/索引比特指示没有处理单元对高速缓存行具有访问权。

[0165] 最初,如果CCP跟踪单元信息并且高速缓存行使用标志比特,则记载可以如下进行。如表900b中所示,在ID[0]处不需要记载任何内容,因为它是在尚未被记载的高速缓存行上的写入(备选地,在写入之前的值可以被记载,并且标志比特可以被翻转)。在此时,CCP可以指出P0和P1对高速缓存行都不具有访问权。在ID[1]处,针对P1可以记载高速缓存行数据。该标志比特可以被打开,并且CCP可以指出P1对高速缓存行具有访问权。在ID[2]处,可以记载一个读取->读取分组,其中P0取自P1的高速缓存行(由于标志比特打开,因此这被记载,并且CCP被用来确定P0没有访问权)。该标志比特已打开,并且CCP指出P0现在也对高速缓存行的状态具有访问权。在ID[3]处,不需要记载任何内容(高速缓存行已在针对该核的日志中)。该确定是因为标志比特打开,并且CCP指示P1已经对高速缓存行具有访问权。在ID[4]处,可以针对P0记载读取->写入分组。这是因为标志比特打开,并且P0已对高速缓存行具有访问权。由于这是写入,因此CCP可以使针对所有其他处理器的高速缓存行无效(即P0具有访问权而P1没有访问权)。在ID[5]处,可以针对P1记载写入->读取分组。这是因为该标志比特打开,但是P1在追踪中不具有数据(如CCP所指示的)。注意,在ID[4]处和ID[5]处的两个参考分组小于在ID[4]处不记载任何内容,然后必须在ID[5]处记载数据。CCP指出,除了P0之外,P1现在也对高速缓存行具有访问权。

[0166] 现在,如果CCP仅跟踪索引信息,并且高速缓存行使用标志比特,则记载可以如下进行。如表900b中所示,在ID[0]处不需要记载任何内容,因为标志比特关闭,并且这是写入。如前所述,如果存储器可以被P0读取,则这可以备选地被记载为读取加写入。在ID[1]

处,可以针对P1记载高速缓存行数据。标志比特可以被打开,并且CCP并更新索引以指到P1。在ID[2]处,针对P0可以记载读取->读取分组。这是因为标志比特已打开并且索引在P1上。CCP可以将索引更新为P0。在ID[3]处,针对P1可以记载读取->读取分组。注意,这种情况现在与ID[2]是不能区分的,因为在两种情况下,索引都在另一个处理器上,标志比特均打开,并且高速缓存行处于共享状态。CCP可以将索引更新为P1。在ID[4]处,针对P0可以记载读取->写入分组。标志比特打开,因此分组通过参考进行记载。这将CCP的索引更新为P0。ID[5],针对P1可以记载写入->读取分组。标志比特打开,分组通过参考进行记载。高速缓存行移至共享状态,因此CCP将索引更新为P1。如表900b中所示,索引情况下导致比单元情况下更大的追踪文件,但仍会产生完全确定的追踪。

[0167] 本文中的一些实施例已经指示:就追踪文件大小而言,(如果可能的话)记录参考另一处理单元拥有的数据的数据分组可能是有益的,而不是稍后记录高速缓存行数据(例如,每一个前面示例中的ID[4])。从通过参考进行记录也可以产生其他益处。例如,在重放时,当有一系列通过参考的日志条目时,可以推断出高速缓存行数据中没有发生外部干预。这是因为在重新记载完全高速缓存行数据时,意味着高速缓存行被逐出或无效。因此,即使在严格不需要日志条目的情形下,通过参考包括日志条目也可以提供有关缺少外部干预的隐式信息,这些信息可能是在重放时或用于调试的有用信息。

[0168] 在一些实现中,记载在追踪条目(例如,上面的“@”条目)中的地址包括物理存储器地址。在这些实现中,处理器102可以将TLB 102f的一个或多个条目记录到(多个)追踪文件104d中。这可以作为不同处理单元的追踪数据流的一部分,也可以作为一个或多个附加追踪数据流的一部分。这将使得重放软件能够稍后将这些物理地址映射到虚拟地址。

[0169] 另外,由于物理地址有时可以被认为是“秘密”信息(例如,当在用户模式级别进行记录时),因此一些实施例记录了实际物理地址的某种表示,而不是物理地址本身。该表示可以是将其标识符唯一地映射到物理地址而无需透露物理地址的任何表示。一个示例可以是每个物理地址的哈希。当使用这些表示并将TLB 102f的条目记录到(多个)追踪文件104d中时,处理器102记录这些表示和虚拟地址之间的映射,而不是物理地址到虚拟地址的映射。

[0170] 如所提及的,处理器102可以包括一个或多个缓冲器102e。在实际将那些条目写入到(多个)追踪文件102f之前,这些缓冲器可以被用作追踪文件条目的临时存储位置。因此,当动作305使数据被记载到追踪时,动作305可以包括将数据写入到(多个)缓冲器102e。在一些实施例中,处理器102采用推迟的记载技术以便减少在处理器102和存储器总线上写入追踪数据的影响。在这些实施例中,处理器102可以将追踪数据存储到(多个)缓冲器102e中,并且推迟对(多个)追踪文件102f的写入,直到存储器总线上存在可用带宽,或者(多个)缓冲器102e已满。

[0171] 如还提及的,一些实施例可以记载高速缓存逐出。图10A和图10B图示出了一些实施例,如何利用关联高速缓存的属性以有效的方式(即,就追踪文件大小而言)记载高速缓存逐出。最初,图10A图示出了存储器地址的不同部分及其与关联高速缓存的关系的示例1000。如图所示,存储器地址包括第一多个比特1001,其是地址的低比特,并且典型地为零。因为存储器地址典型地与存储器地址大小(例如32比特、64比特等)对准,所以第一多个比特1001为零。因此,第一多个比特1001的数目取决于存储器地址的大小。例如,如果存储器

地址是32比特(即,2^5比特),那么第一多个比特1001包括五个比特(使得存储器地址是32的倍数),如果存储器地址是64比特(即,2^6比特),那么第一多个比特1001包括六个比特(使得存储器地址是64的倍数),等等。存储器地址还包括第二多个比特1002,其可以被处理器102用来确定关联高速缓存中的特定地址组,在该特定地址组中应存储存储器地址的数据。例如,在图10A的示例1000中,第二多个比特1002包括三个比特,这将对应于具有八个地址组的关联高速缓存。因此,第二多个比特1002的数目取决于关联高速缓存的特定几何形状。存储器地址还包括第三多个比特1003,其包括存储器地址的其余高比特。

[0172] 在图10A的上下文中,图10B图示出了在关联高速缓存中记载高速缓存未命中和高速缓存逐出的示例1004。最初,示例1004示出了三个存储器地址1005(即,地址@1024)、1006(即,地址@2112)和1007(即,地址@2048)。图10B还图示出了具有八个组的关联高速缓存1010,每个组包括四种路。这些组和路的二进制标识在列1008(组)和1009(路)中示出,并在括号中示出对应的十进制表示。因此,例如,高速缓存1010中的高速缓存行(0,0)——即组0,路0——以二进制形式被示出为组“000”(列1008)和路“00”(列1009);高速缓存1010中的高速缓存行(0,1)——组0,路1——以二进制形式被示出为组“000”(列1008)和路“01”(列1009);依此类推,直到高速缓存1010中的高速缓存行(8,3)——即组8,路3——以二进制形式被示出为组“111”(列1008)和路“11”(列1009)。

[0173] 现在,假设在地址1005(即@1024)上存在第一高速缓存未命中。这里,由于其第二多个比特1002为“000”,所以处理器102可以确定其将与地址1005相对应的数据存储在高速缓存1010的组0中。组0中的特定路典型地由处理器特定的逻辑来选择。然而,出于示例1004的目的,假设数据被存储在路0(如箭头所示)1011a中。结合此高速缓存未命中,由追踪器104a记录的日志数据可以包括存储器地址(即,@1024)和数据被存储的路(即,路0)。注意,可以使用任何数目的压缩技术来减少在追踪中存储存储器地址所需的比特数。该组(即,组0)不需要被记载,因为它可以从存储器地址的第二多个比特1002中被获得。

[0174] 接下来,假设在地址1006(即,@2112)上存在第二高速缓存未命中。这次,第二多个比特1002是“010”,因此处理器102可以确定它将与地址1006相对应的数据存储在高速缓存1010的组2中。同样,组2中的特定路典型地由处理器特定的逻辑来选择。然而,出于示例1004的目的,假设数据被存储在路0(如箭头所示)1011b中。结合此高速缓存未命中,由追踪器104a记录的日志数据可以包括存储器地址(即,@2112)和数据被存储的路(即,路0)。再次,组(即,组2)不需要被记载,因为它可以从存储器地址的第二多个比特1002中被获得。

[0175] 现在假定在地址1007(即,@2048)上存在第三高速缓存未命中。第二多个比特1002再次是“000”,因此处理器102可以确定其将与地址1007相对应的数据存储在高速缓存1010的组0中。通过处理器特定的逻辑再次选择特定的路,但是假设处理器选择路0(如箭头1011c所示)。结合此高速缓存未命中,由追踪器104a记录的日志数据可以包括存储器地址(即,@2048)和存储数据的路(即,路0)。再次,组(即,组0)不需要被记载,因为它可以从存储器地址的第二多个比特1002中被获得。

[0176] 因为该高速缓存行(0,0)当前对应于地址1005,所以地址1007上的该第三高速缓存未命中引起地址1005从高速缓存1010中被逐出。然而,实施例可以避免记录任何记载该逐出的追踪数据。这是因为逐出可以从追踪中已经存在的数据中被推断出——即,将地址1005上的第一高速缓存未命中到路0中,以及地址1007上的第二高速缓存未命中到路0中。

即使组(即,组0)可能未被明确记载在追踪中,其也可以从这些地址推断出。这样,此追踪数据的重放可以再现该逐出。

[0177] 一些逐出是由于高速缓存未命中以外的事件导致的。例如,CCP可能会引起逐出发生,以便维持不同高速缓存之间的一致性。例如,假设由于CCP事件而从高速缓存1010的高速缓存行(2,0)中逐出地址1006。在这里,可以通过记录逐出的组(即“010”)和路(即“00”)来明确记载该逐出。值得注意的是,被逐出的地址无需被记载,因为它在记载将地址1006带入到高速缓存行(2,0)中的第二高速缓存未命中时已被捕获。因此,在该示例中,可以用仅仅五个比特的日志数据(在任何形式的压缩之前)在(多个)追踪文件104d中完全捕获逐出。

[0178] 即使当在该处理单元处执行的线程与安全包围区又译为指定位址空间(enclave)交互时,一些实施例也能够安全地追踪处理单元的活动。如本领域普通技术人员将理解的,包围区是基于硬件的安全特征,其可以保护敏感信息(例如,密码密钥、证书、生物特征数据等)免受在处理器102处执行的可能甚至最低级别软件的攻击。因此,除了保护敏感信息免受用户模式过程的攻击外,包围区甚至可以保护敏感信息免受内核和/或管理程序的攻击。在许多实现中,包围区对于正在执行的过程来说似乎是被映射到该过程的地址空间中的存储器的(多个)加密部分。例如,这可以通过针对正在执行的过程和包围区使用不同的存储器页表(page table)来实现。当过程与包围区交互时,该过程可以从其自己的映射存储器中读取/写入,并且该包围区可以从其自己的映射存储器和/或该过程的映射存储器中读取/写入。

[0179] 第一包围区感知追踪实施例追踪正在执行的过程,同时避免追踪该过程与之交互的包围区,同时仍使被追踪的过程能够得以完全重放。在这些实施例中,使用本文已经描述的一种或多种机制来追踪/记载正在执行的过程向其地址空间的存储器读取。然而,当上下文切换到包围区时,实施例可以跟踪先前由被追踪的过程读取并且在包围区执行期间由包围区写入到的(多个)任何存储器位置。当切换到包围区之后再次执行被追踪的过程时,(多个)这些存储器位置将被视为未由被追踪的过程记载。这样,如果被追踪的过程再次从(多个)这些存储器位置读取(可能读取由包围区放置在(多个)那些位置中的数据),则这些读取将被记载到追踪中。有效地,这意味着在追踪中捕获对追踪过程可见的包围区的执行的任何副效应,而无需追踪包围区的执行。以这种方式,被追踪的过程可以稍后利用这些副效应而被重放,而实际上并不需要(或者甚至不能)重放包围区的执行。存在几种机制(如前所述)可以被用来跟踪先前由被追踪的过程读取并在包围区执行期间由包围区写入到的(多个)存储器位置,诸如计账比特(例如,标志比特、单元比特、索引比特)、路锁定、CCP数据的使用等。

[0180] 第二包围区感知追踪实施例(例如,基于对它自己的地址空间的访问,诸如读取)追踪正在执行的过程,同时还追踪包围区(例如,基于对它自己的地址空间的访问和/或对被追踪的过程的地址空间的访问)。当内核/管理程序与包围区之间存在必要的信任级别时,可以实现这些实施例。在这些实施例中,与包围区的执行有关的追踪数据可以被记载到分离的追踪数据流中和/或被加密,使得执行重放的任何实体都无法在不访问包围区的分离的追踪数据流和/或在不具有可以被用来解密与包围区的执行有关的追踪数据的(多个)加密密钥的情况下重放包围区。

[0181] 第三包围区感知追踪实施例结合了第一和第二实施例。因此,这些第三实施例可

以记录正在执行的过程的追踪，包括该过程使用包围区的副效应(即第一实施例)，以及包围区本身的追踪(即第二实施例)。这使得缺少必要的特权级别和/或(多个)密码密钥的用户能够重放被追踪的过程的执行，同时使得具有必要的特权级别和/或(多个)密码密钥的用户也能够重放包围区本身的执行。

[0182] 这些包围区追踪实施例中的每一个都适用于包围区以外，并且适用于被追踪实体与在追踪期间需要保护其执行的另一实体(现在被称为受保护实体)交互的任何情形。例如，当追踪与内核模式过程交互的用户模式过程时，可以使用这些实施例中的任何一个——在这里，可以将内核模式过程视为与包围区相同。在另一个示例中，当追踪与管理程序交互的内核模式过程时，可以使用这些实施例中的任何一个——在这里，可以将管理程序视为与包围区大致相同。

[0183] 可能存在以下环境：不现实(例如，由于性能或安全考虑)、不可能(例如，由于缺乏硬件支持)或不希望跟踪先前由被追踪的过程读取的哪个或哪些存储器位置被受保护实体在其执行期间写入。这可以防止使用上述的包围区追踪实施例。然而，在这些情形下也存在用于追踪的技术。

[0184] 第一种技术是将处理器高速缓存视为在上下文从受保护实体切换之后已失效。将处理器高速缓存视为已失效引起被追踪的实体从受保护实体返回后进行读取，从而引起高速缓存未命中——其可以被记载。这些高速缓存未命中将包括受保护实体在被追踪的实体的地址空间中修改并且随后由被追踪的实体读取的任何值。尽管该技术可以生成比上述三个实施例更多的追踪数据，但它确实捕获了被追踪的实体所依赖的受保护实体的执行的效应。在一些实施例中，该第一技术还可以在从受保护实体返回到被追踪的实体时记录一个或多个密钥帧(例如，包括处理器寄存器的快照)。即使追踪数据中缺少连续性(即在受保护实体的执行期间)，(多个)密钥帧也使得被追踪实体的重放能够在从受保护实体返回之后开始。

[0185] 第二种技术是记载与受保护实体从被追踪的实体的地址空间中进行的读取以及由受保护实体执行的到被追踪的实体的地址空间中的写入有关的高速缓存未命中。这允许重放追踪，以再现受保护实体的写入，而无需对产生它们的受保护实体的指令具有访问权。这还对受保护实体读取且被追踪的实体稍后访问的的数据(在被追踪的实体的地址空间中)给出了重放访问权。混合方法是可能的(如果诸如CCP数据之类的足够的簿记信息是可用的)，其可以记载受保护实体的写入(在被追踪的实体的地址空间中)，但不可以记载其读取——如果由于将高速缓存视为无效而将在稍后记载那些读取。

[0186] 在不脱离本发明的精神或基本特性的情况下，本发明可以以其他具体形式来体现。所描述的实施例在所有方面仅被认为是说明性的而非限制性的。因此，本发明的范围由所附权利要求书而不是前述描述来指示。落入权利要求等同含义和范围内的所有改变均应被包含在其范围之内。

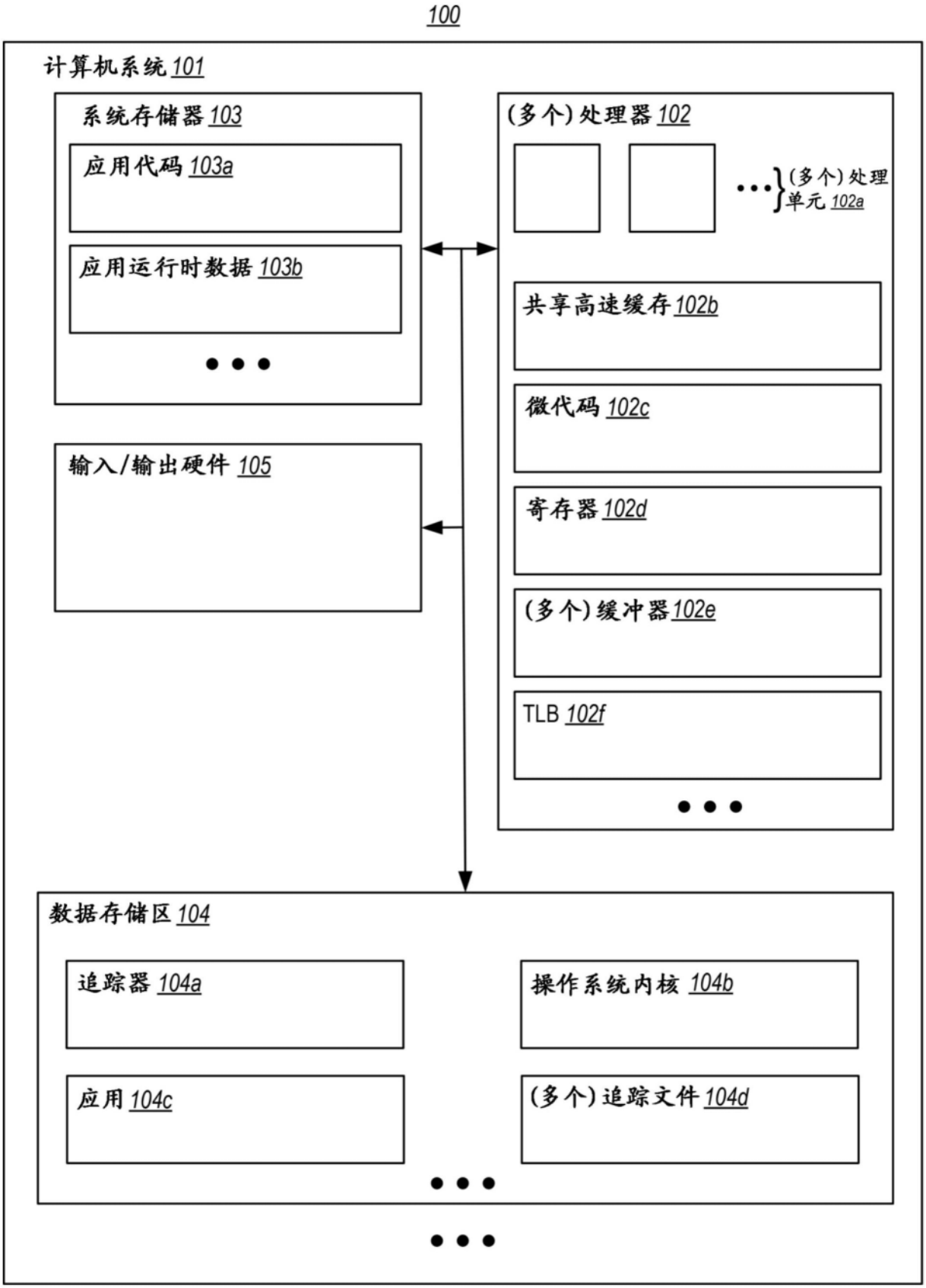

100
计算机系统 101
系统存储器 103
应用代码 103a
应用运行时数据 103b
(多个)处理器 102
(多个)处理单元 102a
共享高速缓存 102b
微代码 102c
寄存器 102d
(多个)缓冲器 102e
TLB 102f
输入/输出硬件 105
数据存储区 104
追踪器 104a
操作系统内核 104b
应用 104c
(多个)追踪文件 104d

图1

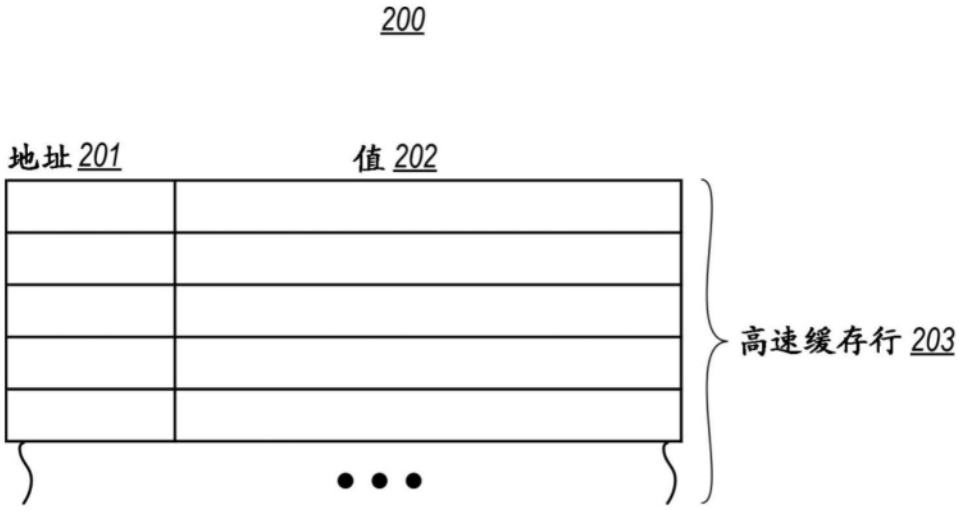


图2

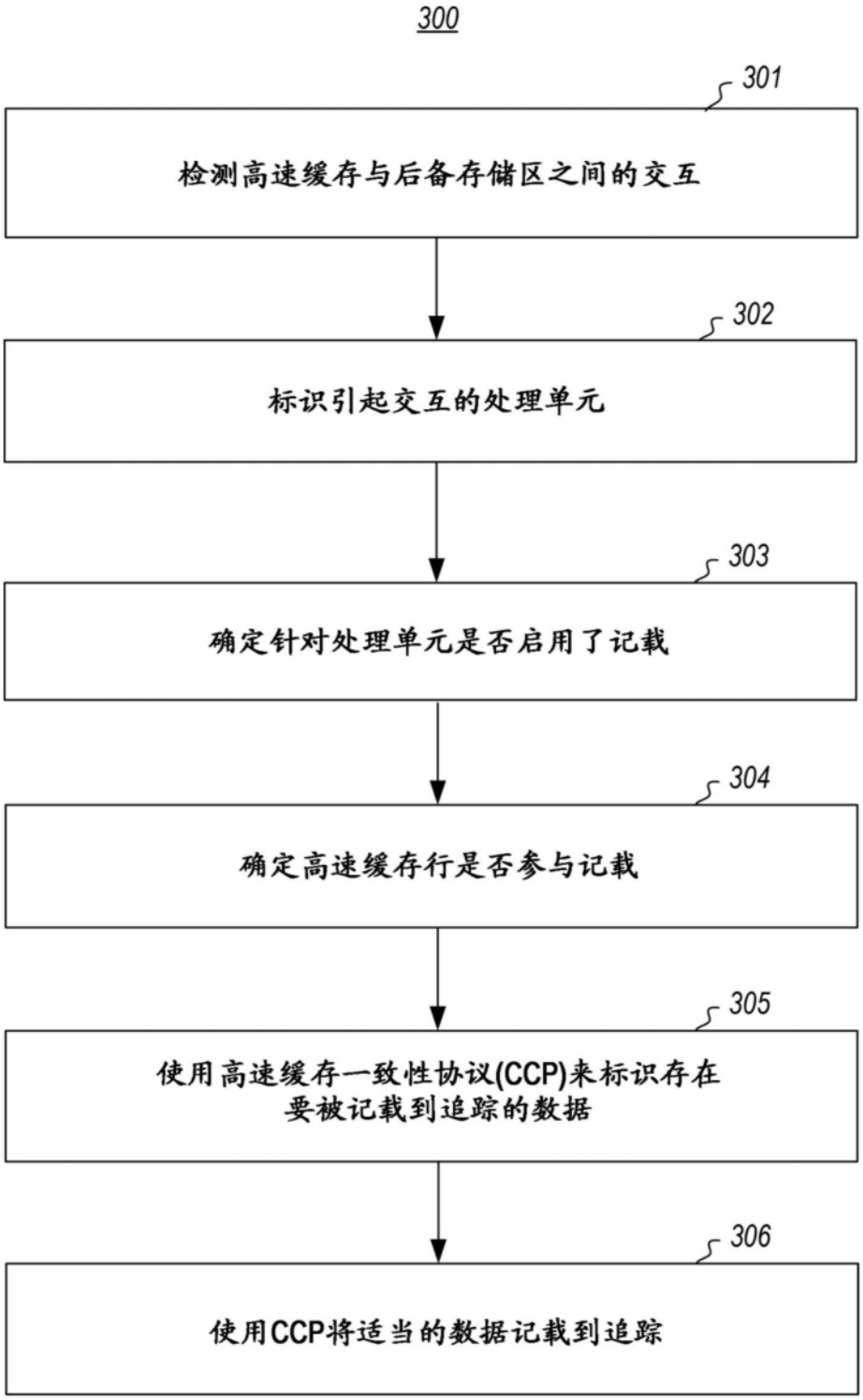

300
301
检测高速缓存与后备存储区之间的交互
302
标识引起交互的处理单元
303
确定针对处理单元是否启用了记载
304
确定高速缓存行是否参与记载
305
使用高速缓存一致性协议(CCP)来标识存在要被记载到追踪的数据
306
使用CCP将适当的数据记载到追踪

图3

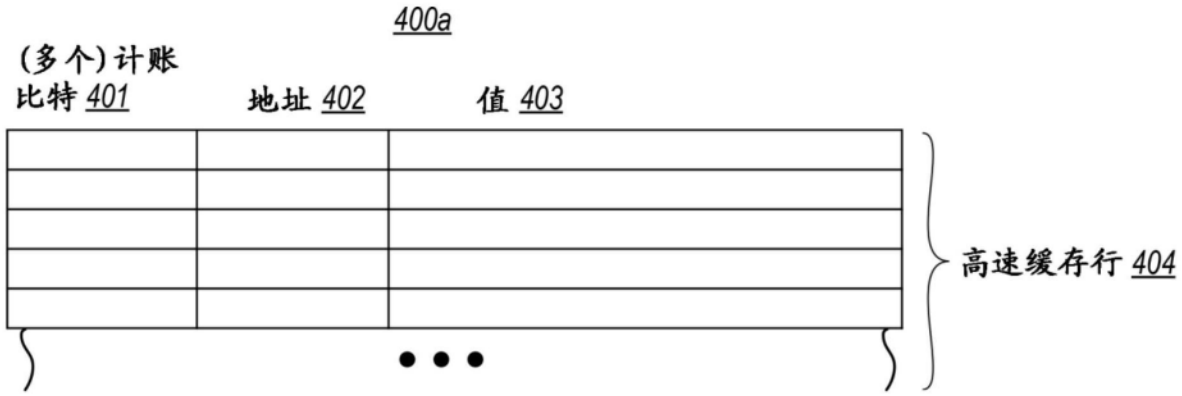

400a
(多个)计账
比特 401
地址 402
值 403
高速缓存行 404

图4A

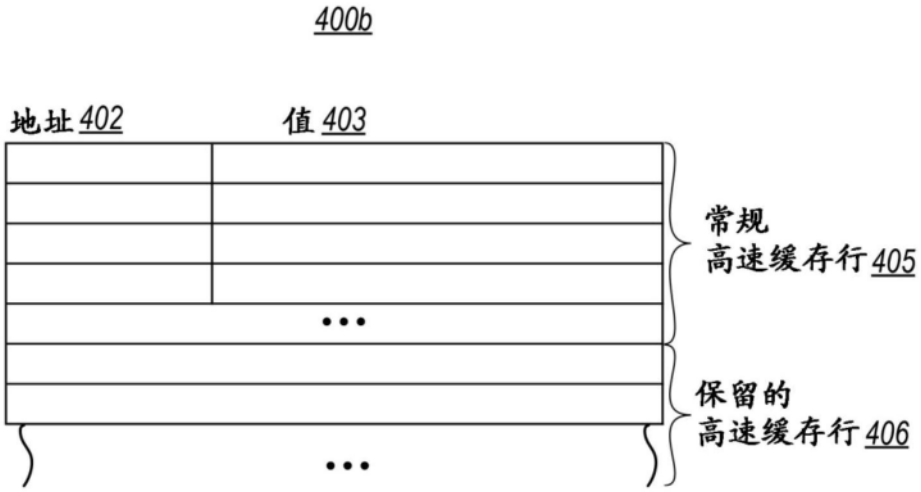

400b
地址 402
值 403
常规
高速缓存行 405
保留的
高速缓存行 406

图4B

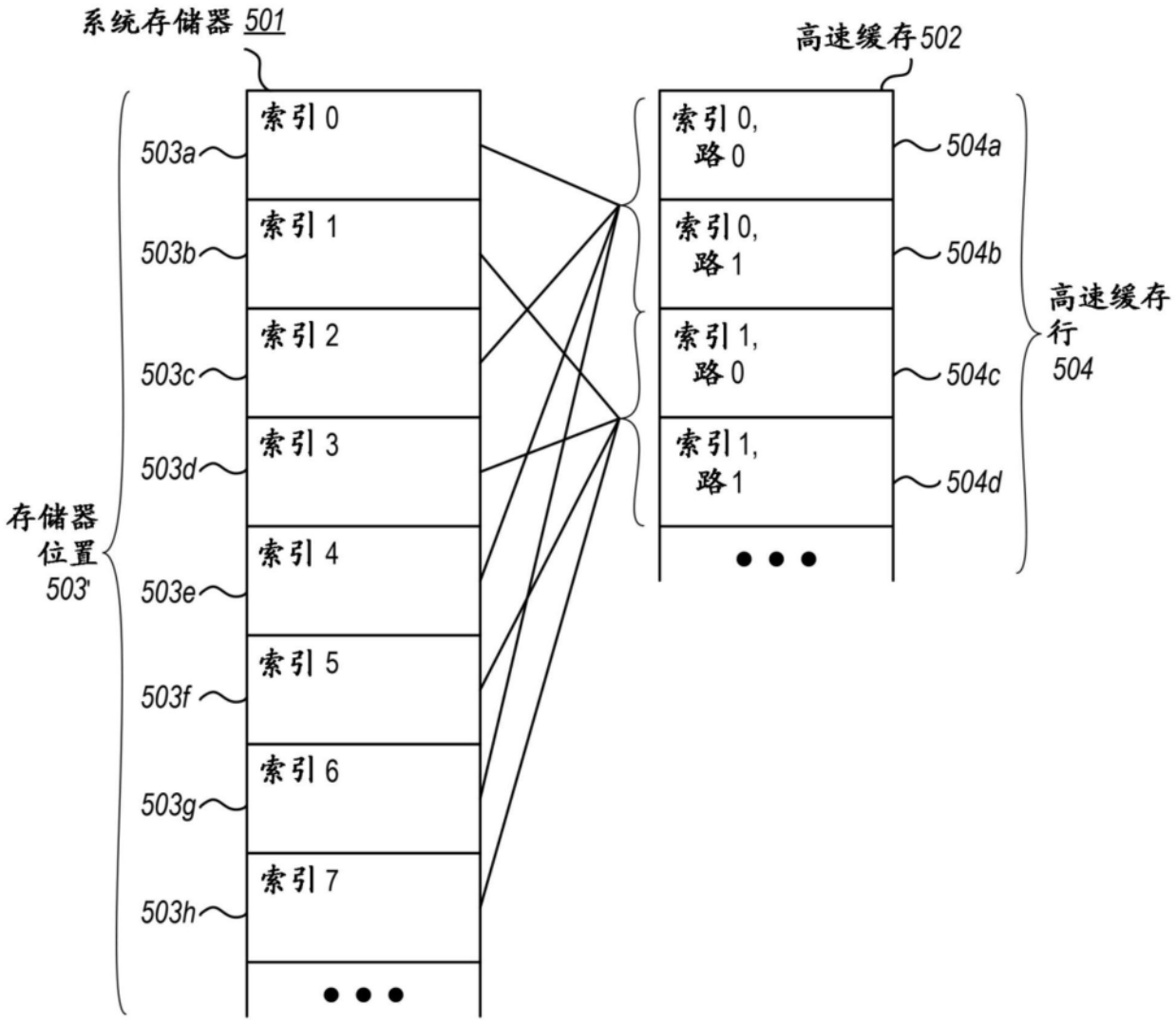

系统存储器 501
高速缓存 502
索引 0
索引 1
索引 2
索引 3
索引 4
索引 5
索引 6
索引 7
503a
503b
503c
503d
503e
503f
503g
503h
存储器位置 503'
索引 0, 路 0
索引 0, 路 1
索引 1, 路 0
索引 1, 路 1
504a
504b
504c
504d
高速缓存行 504

图5

600a

601 ID	602a P0	602b P1	602c P2	602d P3
0	R[1]			
1		R[1]		
2	W[2]			
3		R[2]		
4	W[3]			
5			R[3]	
6		R[3]		
7				R[3]
8				W[4]
9	W[5]			
10			R[5]	
11		R[5]		
12	R[5]			
13		逐出行		

图6A

600b

601 ID	602a P0	602b P1	602c P2	602d P3
0	S			
1	S	S		
2	M	I		
3	S	S		
4	M	I		
5	S		S	
6	S	S	S	
7	S	S	S	S
8	I	I	I	M
9	M			I
10	S		S	
11	S	S	S	
12	S	S	S	
13	-	-	-	-

图6B

600c

601 ID	603 单元比特				604 索引	605 标志
0	1	0	0	0	0	1
1	1	1	0	0	1	1
2	1	0	0	0	0	1
3	1	1	0	0	1	1
4	1	0	0	0	0	1
5	1	0	1	0	2	1
6	1	1	1	0	1	1
7	1	1	1	0	1	1
8	0	0	0	0	-1	0
9	0	0	0	0	-1	0
10	0	0	1	0	2	1
11	0	1	1	0	1	1
12	1	1	1	0	0	1
13	0	0	0	0	-1	0

图6C

600d

601 606

ID	日志数据
0	P0; @; IC[0]; DATA [1]
1	P1; @; IC[1]; R->R; P0->P0, P1
2	P0; @; IC[2]; R->W; P0, P1->P0
3	P1; @; IC[3];W->R; P0->P0, P1
4	P0; @; IC[4]; R->W; P0, P1->P0
5	P2; @; IC[5]; W->R; P0->P0, P2
6	P1; @; IC[6]; R->R; P0, P2->P0, P1, P2
7	
8	
9	
10	P2; @; IC[10]; DATA[5], P2
11	P1; @; IC[11]; R->R; P2->P1, P2
12	P0; @; IC[12]; R->R; P1, P2->P0, P1, P2
13	

图6D

700a

701 ID	702a P0	702b P1	702c P2
0	R[1]		
1		R[1]	
2	W[2]		
3		R[2]	
4			R[2]
5	R[2]		
6		R[2]	
7			R[2]
8	W[3]		
9	R[3]		
10			R[3]

图7A

700b

0	P0; @; IC[0]; DATA [1]
1	P1; @; IC[1]; R->R; P0->P0, P1
2	P0; @; IC[2]; R->W; P0, P1->P0
3	P1; @; IC[3];W->R; P0->P0, P1
4	P2; @; IC[4]; R->R; P0, P1->P0, P1, P2
5	
6	
7	
8	P0; @; IC[8]; R->W; P0, P1, P2->P0
9	
10	P2; @; IC[10]; R->R; P0->P0, P2

图7B

700c

0	P0; @; IC[0]; DATA [1]
1	P1; @; IC[1]; R->R; P0-> P1
2	P0; @; IC[2]; R->W; P1->P0
3	P1; @; IC[3];W->R; P0->P1
4	P2; @; IC[4]; R->R; P1->P2
5	P0; @; IC[5]; R->R; P2->P0
6	P1; @; IC[6]; R->R; P0->P1
7	P2; @; IC[7]; R->R; P1->P2
8	P0; @; IC[8]; R->W; P2->P0
9	
10	P2; @; IC[10]; R->R; P0->P2

图7C

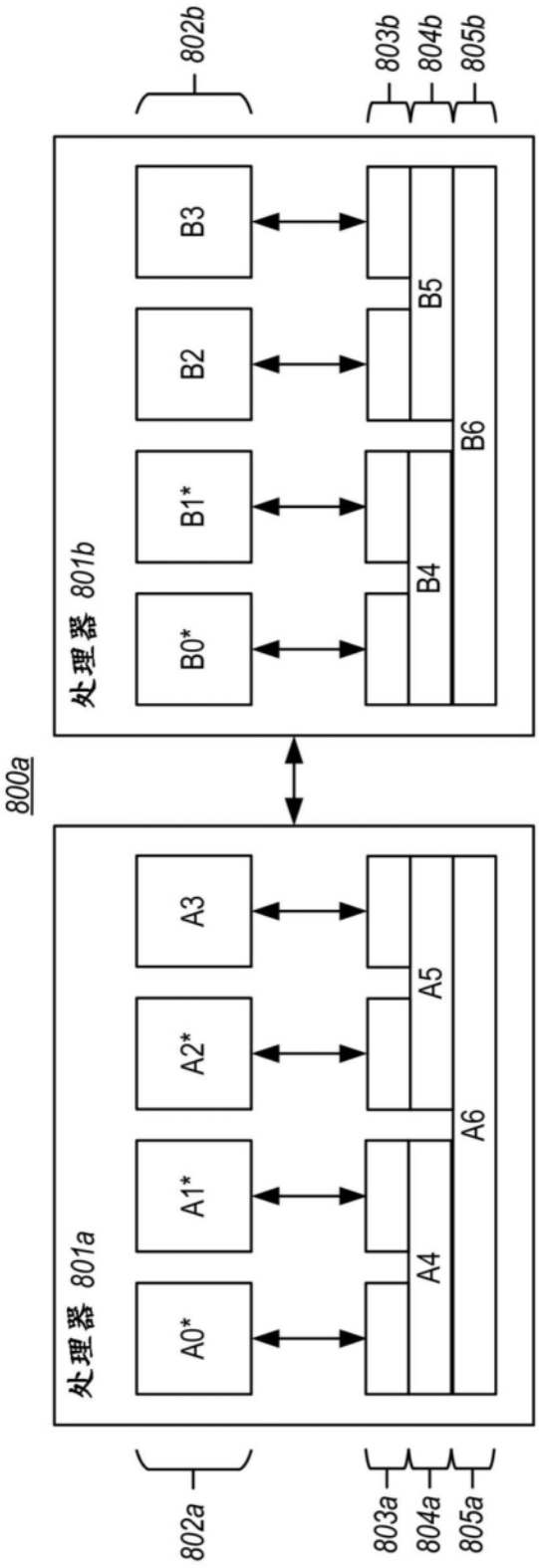

802b
803b
804b
805b
处理器 801b
B3
B2
B1*
B0*
B5
B6
B4
800a
处理器 801a
A3
A2*
A1*
A0*
A5
A6
A4
802a
803a
804a
805a

图8A

800b

ID	A0	A1	A2	A3	B0	B1	B2	B3
0	R[1]							
1					R[1]			
2		R[1]						
3			W[2]					
4			R[2]					
5					R[2]			
6						R[2]		
7					R[2]			
8	W[3]							
9	W[4]							
10		R[4]						
11			R[4]					
12					R[4]			
13						R[4]		
14	R[4]							

图8B

900a

ID	P0	P1
0	W[1]	
1		R[1]
2	R[1]	
3		R[1]
4	W[2]	
5		R[2]

图9A

900b

ID	CCP单元+标志	CCP索引+标志
0		
1	记载DATA	记载DATA
2	记载R->R	记载R->R
3		记载R->R
4	记载R->W	记载R->W
5	记载W->R	记载W->R

图9B

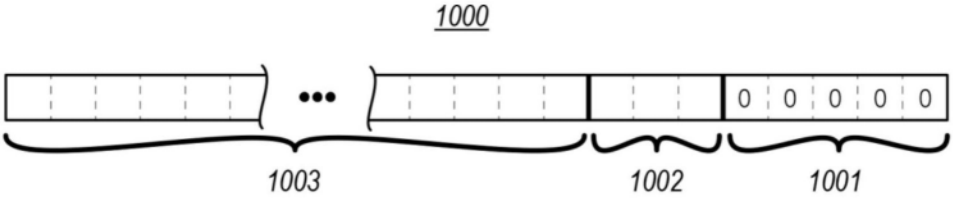


图10A

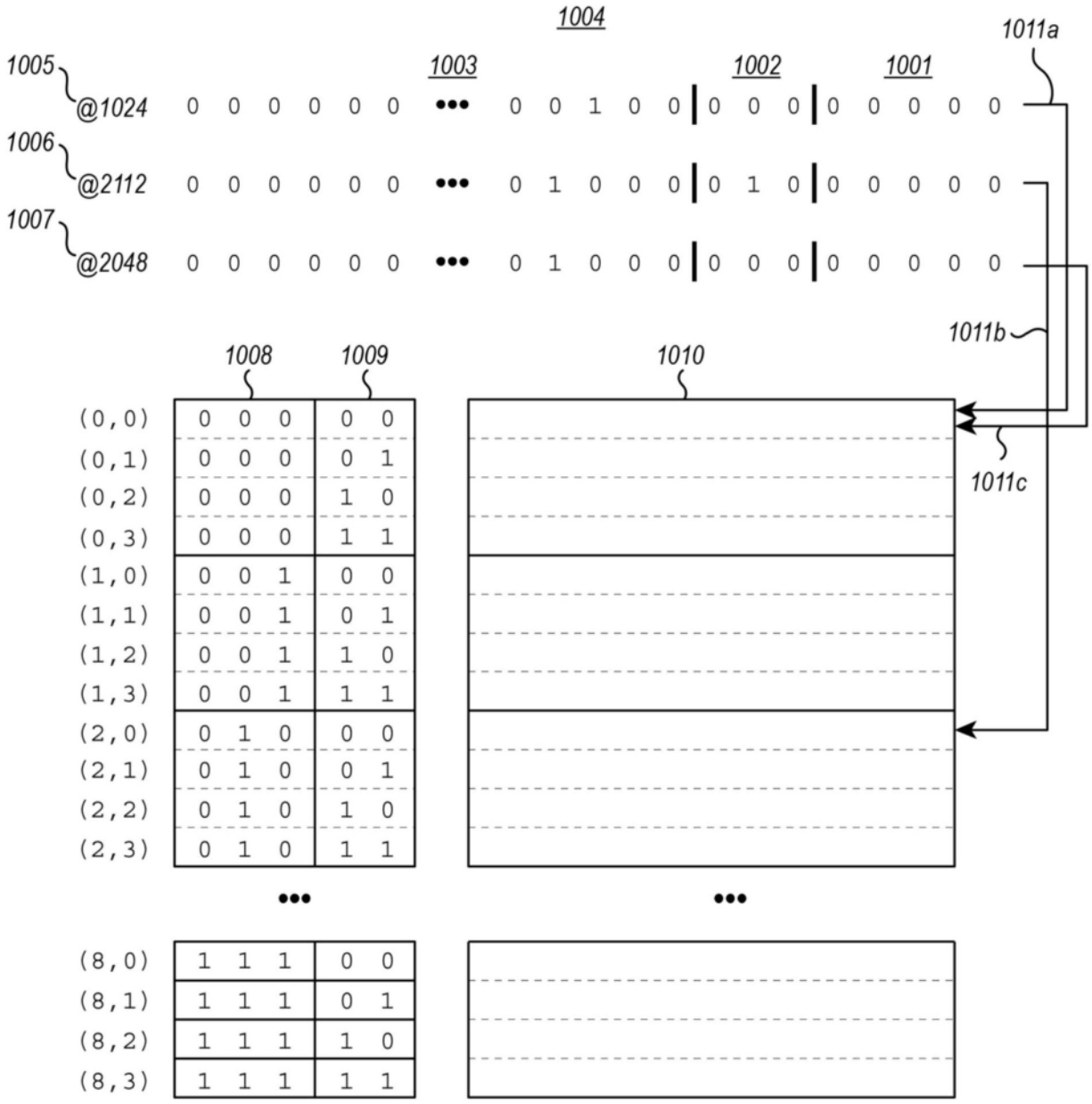

1004
1011a
1005
1003
1002
1001
@1024 0 0 0 0 0 0 ••• 0 0 1 0 0 | 0 0 0 | 0 0 0 0 0
1006
@2112 0 0 0 0 0 0 ••• 0 1 0 0 0 | 0 1 0 | 0 0 0 0 0
1007
@2048 0 0 0 0 0 0 ••• 0 1 0 0 0 | 0 0 0 | 0 0 0 0 0
1011b
1008
1009
1010
(0,0) 0 0 0 | 0 0
(0,1) 0 0 0 | 0 1
(0,2) 0 0 0 | 1 0
(0,3) 0 0 0 | 1 1
(1,0) 0 0 1 | 0 0
(1,1) 0 0 1 | 0 1
(1,2) 0 0 1 | 1 0
(1,3) 0 0 1 | 1 1
(2,0) 0 1 0 | 0 0
(2,1) 0 1 0 | 0 1
(2,2) 0 1 0 | 1 0
(2,3) 0 1 0 | 1 1
1011c
(8,0) 1 1 1 | 0 0
(8,1) 1 1 1 | 0 1
(8,2) 1 1 1 | 1 0
(8,3) 1 1 1 | 1 1

图10B