



(12)发明专利

(10)授权公告号 CN 104657109 B

(45)授权公告日 2017.09.15

(21)申请号 201410815077.4

(22)申请日 2010.12.17

(65)同一申请的已公布的文献号
申请公布号 CN 104657109 A

(43)申请公布日 2015.05.27

(30)优先权数据
12/653,800 2009.12.18 US

(62)分案原申请数据
201010599713.6 2010.12.17

(73)专利权人 英特尔公司
地址 美国加利福尼亚

(72)发明人 J·B·弗莱曼 A·T·福赛思
E·格罗霍夫斯基

(74)专利代理机构 永新专利商标代理有限公司
72002

代理人 刘瑜 王英

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

(56)对比文件
US 6128706 A,2000.10.03,
US 2003/0233495 A1,2003.12.18,
CN 1716186 A,2006.01.04,
US 2009/0113131 A1,2009.04.30,
US 2006/0112306 A1,2006.05.25,

审查员 梁岩

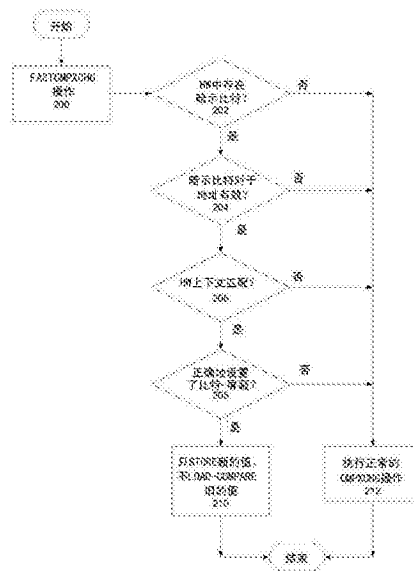
权利要求书3页 说明书8页 附图3页

(54)发明名称

自适应优化的比较-交换操作

(57)摘要

公开了执行快速比较-交换操作的技术。更具体地说,描述了实现快速比较-交换操作以及高速缓存行标记操作的机器可读介质、处理器和系统,所述高速缓存行标记操作使得能够进行所述快速比较-交换操作。



1. 一种用于执行指令的处理器,包括:
 - 一个或多个高速缓存;
 - 一个或多个核心,耦合到所述一个或多个高速缓存,其中每个所述核心包括处理逻辑单元,所述处理逻辑单元包括:
 - 解码逻辑单元,被配置为用于解码快速比较-交换指令;以及
 - 执行逻辑单元,被配置为用于在所述快速比较-交换指令被解码之后执行所述快速比较-交换指令,
 - 其中,所述快速比较-交换指令的执行包括:
 - 确定从存储器地址位置读取的在原子指令序列中使用的数据是否在所述原子指令序列期间被修改;
 - 如果所述数据被修改,则对所述存储器地址位置执行标准的比较-交换指令,所述标准的比较-交换指令包括顺序地执行的加载-比较-存储微操作序列;以及
 - 如果所述数据未被修改,则对所述存储器地址位置仅执行存储微操作。
2. 根据权利要求1所述的处理器,其中:
 - 所述解码逻辑单元进一步被配置为用于解码高速缓存行标记指令;以及
 - 所述执行逻辑单元进一步被配置为用于在所述高速缓存行标记指令被解码之后执行所述高速缓存行标记指令,
 - 其中,所述高速缓存行标记指令的执行包括:在标签识别结构中用微架构暗示比特来为用于存储从所述存储器地址位置处的存储器获取的数据的高速缓存行设置标签。
3. 根据权利要求2所述的处理器,其中,所述处理器还包括:
 - 寄存器,被配置为用于存储所述存储器地址位置。
4. 根据权利要求3所述的处理器,其中,所述执行逻辑单元进一步被配置为用于:
 - 设置用于所述高速缓存行的标签识别结构中的有效比特。
5. 根据权利要求4所述的处理器,其中,所述执行逻辑单元进一步被配置为用于:
 - 设置所述标签识别结构中的比特-屏蔽,所述比特-屏蔽指示所述高速缓存行的哪些比特与所述高速缓存行标记指令相关。
6. 根据权利要求5所述的处理器,其中,所述执行逻辑单元进一步被配置为用于:
 - 设置所述标签识别结构中的硬件上下文识别值,所述硬件上下文识别值指示执行所述高速缓存行标记指令的特定硬件上下文。
7. 根据权利要求6所述的处理器,其中,所述执行逻辑单元进一步被配置为用于:
 - 确定是否设置了所述有效比特;
 - 确定是否在高速缓存行标记比特-屏蔽和快速比较-交换比特屏蔽之间正确地设置了所述比特-屏蔽;以及
 - 确定当前的硬件上下文与所述特定硬件上下文是否匹配;
 - 其中,响应于所有确定被确认,所述执行逻辑单元被配置为用于判定所述数据在所述原子指令序列期间未被修改。
8. 根据权利要求2所述的处理器,其中,所述执行逻辑单元进一步被配置为用于:
 - 响应于确认所述处理器不支持所述高速缓存行标记指令,用无操作指令替换所述高速缓存行标记指令。

9. 一种用于执行指令的系统,包括:
系统存储器;
一个或多个处理器,耦合到所述系统存储器,其中每个所述处理器包括:
一个或多个高速缓存;
一个或多个核心,耦合到所述一个或多个高速缓存,其中每个所述核心包括处理逻辑单元,所述处理逻辑单元包括:
解码逻辑单元,被配置为用于解码快速比较-交换指令;以及
执行逻辑单元,被配置为用于在所述快速比较-交换指令被解码之后执行所述快速比较-交换指令,
其中,所述快速比较-交换指令的执行包括:
确定从存储器地址位置读取的在原子指令序列中使用的数据是否在所述原子指令序列期间被修改;
如果所述数据被修改,则对所述存储器地址位置执行标准的比较-交换指令,所述标准的比较-交换指令包括顺序地执行的加载-比较-存储微操作序列;以及
如果所述数据未被修改,则对所述存储器地址位置仅执行存储微操作。
10. 如权利要求9所述的系统,其中,所述系统存储器被配置为用于存储实现所述快速比较-交换指令的代码。
11. 如权利要求9所述的系统,其中,所述一个或多个处理器还包括:
系统存储器控制器,被配置为用于提供与所述系统存储器进行通信的接口。
12. 根据权利要求9所述的系统,其中:
所述解码逻辑单元进一步被配置为用于解码高速缓存行标记指令;以及
所述执行逻辑单元进一步被配置为用于在所述高速缓存行标记指令被解码之后执行所述高速缓存行标记指令,
其中,所述高速缓存行标记指令的执行包括:在标签识别结构中用微架构暗示比特来为用于存储从所述存储器地址位置处的存储器获取的数据的高速缓存行设置标签。
13. 根据权利要求12所述的系统,其中,所述处理器还包括:
寄存器,被配置为用于存储所述存储器地址位置。
14. 根据权利要求13所述的系统,其中,所述执行逻辑单元进一步被配置为用于:
设置用于所述高速缓存行的标签识别结构中的有效比特。
15. 根据权利要求14所述的系统,其中,所述执行逻辑单元进一步被配置为用于:
设置所述标签识别结构中的比特-屏蔽,所述比特-屏蔽指示所述高速缓存行的哪些比特与所述高速缓存行标记指令相关。
16. 根据权利要求15所述的系统,其中,所述执行逻辑单元进一步被配置为用于:
设置所述标签识别结构中的硬件上下文识别值,所述硬件上下文识别值指示执行所述高速缓存行标记指令的特定硬件上下文。
17. 根据权利要求16所述的系统,其中,所述执行逻辑单元进一步被配置为用于:
确定是否设置了所述有效比特;
确定是否在高速缓存行标记比特-屏蔽和快速比较-交换比特屏蔽之间正确地设置了所述比特-屏蔽;以及

确定当前的硬件上下文与所述特定硬件上下文是否匹配；

其中，响应于所有确定被确认，所述执行逻辑单元被配置为用于判定所述数据在所述原子指令序列期间未被修改。

18. 根据权利要求12所述的系统，其中，所述执行逻辑单元进一步被配置为用于：

响应于确认所述处理器不支持所述高速缓存行标记指令，用无操作指令替换所述高速缓存行标记指令。

自适应优化的比较-交换操作

[0001] 本申请是申请日为2010年12月17日、申请号为201010599713.6的同名专利申请的分案申请。

技术领域

[0002] 本发明涉及在由计算处理器执行的代码中实现的比较-交换操作。

背景技术

[0003] CMPXCHG (比较-交换) 操作通常被用作构成在代码区域(即,指令、操作等)周围的临界区的指令序列中的一个指令。它允许代码区域原子地执行。当满足两个条件时指令集可以被认为是原子的:第一,直到整个指令集完成,没有其他进程可以知道做出的改变,以及第二,如果集合中的任何一个指令失败那么整个指令集失败。当整个指令集失败时,执行指令的计算机系统的状态恢复为指令中任何一个开始之前其所处的状态。在一些实施例中,CMPXCHG指令可以被认为是具有临界区的代码区域的软件定义的包装(wrapper)。换言之,可以将该包装设计为软件代码,但是其不是硬件这样实施意义上的包装。

[0004] CMPXCHG指令通过检查以确保在原子集开始时从存储器获取的值在指令集的执行期间没有被另一进程修改来帮助原子地执行该指令集。例如,如果存储器中特定位置的值被加载到位于原子指令集合的开头的寄存器中,那么在该指令集完成之后可能有将新(即,修改的)值加载回原始存储器位置的请求(该修改的值是该指令集的结果)。

[0005] 可以在原子指令集结束时使用CMPXCHG指令检查从特定存储器位置原始加载的值是否仍然在存储器的该位置中(即,没有被另一进程或线程在原子指令集执行期间的某个时候修改)。如果原始值仍然在那里,那么CMPXCHG指令将新的值加载到存储旧的值的那个特定存储器位置,并且原子指令集成功地完成。如果原始值不在那里,这意味着另一进程或线程在原子指令集执行期间修改了该值,那么CMPXCHG指令不将新的值加载到所述特定存储器位置,而是向系统通知该情况,并且作为通知的结果,可以在代码中实现有条件跳转以跳转到原子指令集的开始处以进行另一成功完成的尝试。

附图说明

[0006] 本发明以示例的方式来描述并且不受附图限制,在附图中,相似的标号指示类似的元件,并且其中:

[0007] 图1说明了用于CLMARK命令的高速缓存行标签的集合的实施例。

[0008] 图2是说明关于执行完整的CMPXCHG微操作序列还是只执行CMPXCHG操作的STORE部分的FASTCMPXCHG指令决策树的实施例的流程图。

[0009] 图3说明了包括能够执行CLMARK和FASTCMPXCHG指令的一个或多个CPU核心的计算机系统的实施例。

具体实施方式

[0010] 描述了实现自适应优化的比较-交换操作的机器可读介质、处理器以及系统的实施例。

[0011] 引入了两个指令:CLMARK (高速缓存行标记)和FASTCMPXCHG (快速比较-交换)。CLMARK指令在标签识别结构中用微架构暗示比特(hint bit)为特定的高速缓存行加标签。当被检验时,暗示比特能够通知处理器中的执行逻辑准备执行两个方法中的FASTCMPXCHG方法。具体地说,从高速缓存读取的在原子指令序列中使用的数据会或不会被篡改。如果数据在原子指令序列期间未被篡改(即,修改),那么FASTCMPXCHG指令退回到标准CMPXCHG(比较-交换)指令,其包括LOAD-COMPARE-STORE(加载-比较-存储)的微操作序列。另一方面,如果通过检验标签识别结构中的暗示比特,执行逻辑可以确定在特定高速缓存行中的数据明确地没有被修改,那么可以执行CMPXCHG的“快速”版本,其中唯一使用的微操作是最后的STORE。因此,在许多实施例中,LOAD和COMPARE微操作可以被丢弃而无需执行,并且这可以加快处理器中的执行效率。

[0012] 在以下说明书和权利要求中引用的本公开技术的“一个实施例”、“实施例”是指结合该实施例而描述的具体特征、结构或特性被包括在所公开技术的至少一个实施例中。因此,在整个说明书不同位置中,短语“在一个实施例中”的多次出现不一定全部都是指同样的实施例。在以下说明书和权利要求中,可以使用术语“包括”和“包含”以及它们的派生词,并且意图将它们用作彼此的同义词。

[0013] 在当今的多线程环境中,在执行中间可以中断正执行原子指令集的线程。然后另一线程可以被给予存储器的控制并且潜在地改变一些或许多存储的值。这仅是原子指令集使用的在存储器位置处的值在新的值有机会被重写到同一存储器位置之前可能已经被修改的原因的一个例子。

[0014] 实际的CMPXCHG指令的使用实质上包括以下基本格式:

[0015] lock CMPXCHG [mem], rdx

[0016] 具体地说,CMPXCHG执行流程导致发生以下操作:

[0017] 1) 将存储在[mem](由mem地址指向的存储器位置)的值加载(LOAD指令)到第一寄存器(“第一”寄存器包括执行逻辑针对嵌入在CMPXCHG中的LOAD微操作所使用的专用寄存器)中。

[0018] 2) 将第一寄存器中的值与eax/rax寄存器(eax还是rax取决于操作数的大小)进行比较(COMPARE微操作)。

[0019] 3) 如果比较表明两个值相等(即,存储器中的值没有改变),那么将rdx寄存器中的值写入(STORE微操作)到[mem]。

[0020] 4) 如果比较表明两个值不同(即,存储器中的值已改变),那么将存储在[mem]中的当前值加载到eax/rax中。

[0021] lock(锁定)前缀使得CMPXCHG指令本身变为原子指令。这是推荐的,因为CMPXCHG指令被分解成上文描述的LOAD-COMPARE-STORE微操作组合。

[0022] 整个CMPXCHG指令成功还是失败基于上文步骤2中的COMPARE指令的结果。用于检查CMPXCHG成功的简单方法是如果它成功那么零标志(ZF)被设置,如果没有成功那么ZF被清零。微操作集合(LOAD-COMPARE-STORE)在CMPXCHG指令本身之前的某个时刻以额外的LOAD指令作为开端,因为在该原子指令集开始时,CMPXCHG指令需要将[mem]处的值初始

LOAD到eax/rax寄存器中。整个原子指令集的例子可以类似于以下代码：

[0023]

try_again:

mov rax, [mem] ;将位置[mem]处的存储器中的值加载到 rax 中

mov rdx, rax ;将 rax 中的值加载到 rdx

插入代码以将 rdx 操控为（潜在地）新的值

lock cmpxchg [mem], rdx ;如果 rax 中的值仍然等于在位置[mem]处的存储器中的值，那么将 rdx 加载到在位置 [mem]处的存储器中

jnz try_again ;如果 cmpxchg 成功，那么 ZF=1，否则 ZF=0，从而跳转，如果未成功，则零操作将导致另一尝试

[0024] 当使用CMPXCHG指令时，使用的LOAD-COMPARE-STORE微操作组合是串行依赖的流。由于该串行流的依赖性，完成CMPXCHG指令所需微操作的数量可能是很大的。

[0025] 如前文所述，CMPXCHG指令通常用作构成在代码区域周围的临界区的指令序列中的一个指令，并且特别用于确定另一进程/线程是否对讨论中的在特定存储器位置处的值进行了修改。如果讨论中的核心内没有其他的进程/线程中断过原子指令集的代码流并且如果没有运行不同代码的其他核心监听讨论中的高速缓存行，那么在该存储器位置处的值将没有改变。因此，如果可以在甚至不用看在存储器位置处的值的情况下预先知道没有同一核心或另一核心中的其他进程/线程修改了在该存储器位置处的存储器内容，那么一般将不需要执行CMPXCHG指令的LOAD和COMPARE部分。相反，如果通过其他的方法，可以确定对在存储器位置处的值没有外部（即，其他进程/线程/核心）的影响，那么CMPXCHG指令唯一需要的部分将是将新的值返回到该存储器位置的最后STORE微操作。

[0026] 所以，在许多实施例中，可以实现两个新的指令以提供标准CMPXCHG微操作组合中通常未必执行的COMPARE-STORE部分。换言之，在多数情况下，新指令将允许仅执行组合的STORE部分。

[0027] 在许多实施例中，这两个指令可以被称为CLMARK指令和FASTCMPXCHG指令。第一新指令是CLMARK。CLMARK指令可以具有以下特定的格式：

[0028] CLMARK8B mem

[0029] Mem字段包括落在一个高速缓存行内的基础存储器地址。8B字段指示考虑从mem地址开始有多少字节用于FASTCMPXCHG指令。在许多实施例中，可以有多个CLMARK指令的版本，其具有的8B字段支持所有2的幂的字节大小，直至CPU（中央处理单元）的高速缓存行大

小。

[0030] CLMARK操作在mem位置引用的目标高速缓存行上设置需要的唯一识别标签,来指示原子序列的预期的所有权。例如,可以在唯一识别标签中使用“有效”比特以表明该特定mem位置的高速缓存行中的数据仍然有效。此外,如果CPU支持每个核心多个硬件上下文,那么唯一识别标签还可以包括硬件上下文ID。唯一识别标签中的另一个额外的可能性是可以包括用于每一个高速缓存行的比特-屏蔽(bit-mask)(或潜在地在存储于CPU内的专门小表中)。针对高速缓存行中的每一个字节,比特-屏蔽可以使用单个比特。该比特-屏蔽可以用来最小化在共享数据结构中的误共享冲突。

[0031] 如果CPU不支持CLMARK指令,那么核心中的执行逻辑可以仅用“无操作”(NOP)指令取代每一个CLMARK。该CLMARK→NOP替换是将CLMARK标签比特认为是暗示比特的原因。此外,实现CLMARK/FASTCMPXCHG指令的系统也可以随时进行CLMARK→NOP替换。关于能够使用CLMARK/FASTCMPXCHG的系统替代地决定使用CMPXCHG的原因,可以有许多例子。例如,内部共享资源已被耗尽、发生了特定事件,以及设置了调试/测试寄存器等等原因。

[0032] 图1说明了用于CLMARK命令的高速缓存行标签集合的实施例。

[0033] 如所陈述的,每一个标签可以包括一个或多个以下结构部件:高速缓存行标签ID 100、有效比特102、HW上下文ID 104,以及比特-屏蔽106。图1中说明的所有标签信息可以作为整体被称为标签识别结构。在许多实施例中,比特-屏蔽基于每个字节屏蔽高速缓存行,其中每个字节具有用于屏蔽的其自己的比特。因此,如果高速缓存行中有N个字节,那么在用于字节0到N-1的比特-屏蔽中存在多个比特。当执行CLMARK指令时,mem基地址和8B大小字段用于设置适合的比特-屏蔽值,其为即将到来的FASTCMPXCHG序列涉及的高速缓存行中的那些字节加标志。

[0034] 在可替代的实施例中,可以不使用每字节比特-屏蔽字段。当不使用每字节比特-屏蔽字段时,单个比特可以用于整个高速缓存行来指示CLMARK/FASTCMPXCHG指令中涉及的高速缓存行。

[0035] FASTCMPXCHG是第二新指令。FASTCMPXCHG指令可以具有以下特定的格式:

[0036] lock FASTCMPXCHG8B [mem], testval, newval

[0037] 以与原始CMPXCHG指令不同的方式给出FASTCMPXCHG指令,以避免遗留代码问题。换言之,如果CMPXCHG指令被更新以像FASTCMPXCHG指令那样操作,则使用CMPXCHG的遗留代码可能不包括正确的格式或可能有意外事件,因此使用两个单独的指令。遗留和新的指令可以或不共享执行单元。但是,考虑到两个指令之间功能复制的量,在遗留的和新的指令间可以高效地共享执行单元。

[0038] FASTCMPXCHG [mem] 字段指向存储器中位置的基本存储器地址,其对应于存储要用于比较和交换目的的值的高速缓存行。与上文中CLMARK8B字段的目地对应地,该8B字段指示考虑从[mem]地址开始有多少字节用于FASTCMPXCHG指令。在许多实施例中,可以有多个FASTCMPXCHG指令的版本,其具有的8B字段支持所有2的幂的字节大小,直至CPU的高速缓存行大小。

[0039] 如上文所详述的,FASTCMPXCHG指令分解为单独的LOAD-COMPARE-STORE μ op(微操作)阶段,但是在LOAD之前可以存在分支。内部分支确定是否顺序地执行全部LOAD-COMPARE-STORE μ op序列,或是否跳过LOAD-COMPARE部分并且实际上只执行STORE。

FASTCMPXCHG指令中的分支基于多个决策来确定要采取哪条路径。因此,FASTCMPXCHG操作可在任一种情况下适于在执行该代码的硬件上工作。

[0040] 图2是说明关于是否执行完整的CMPXCHG微操作序列还是只执行CMPXCHG操作的STORE部分的FASTCMPXCHG指令决策树的实施例的流程图。

[0041] 确定决策树中在哪个方向继续进行的处理逻辑可以包括硬件(例如,执行单元电路)、固件(例如,通用CPU微代码),或硬件和固件二者的组合。转向图2,处理逻辑从接收执行管线中的FASTCMPXCHG操作开始(处理框200)。处理逻辑然后查找包括与FASTCMPXCHG指令相关联的微架构暗示比特的高速缓存行标签(处理框202)。如果硬件能够处理FASTCMPXCHG,则前一个CLMARK可以利用的标签将被呈现给每条高速缓存行。如果硬件不具有该标签,那么不支持CLMARK和FASTCMPXCHG。所以,处理逻辑然后将仅使用[mem]、testval和newval参数,来执行标准CMPXCHG指令以替代代码中列出的FASTCMPXCHG指令(处理框212)。

[0042] 否则,如果存在包含微架构暗示比特的标签,那么处理逻辑进行检查以查看暗示比特对于[mem]地址是否有效(处理框204)。处理逻辑可以通过检查匹配[mem]地址的高速缓存行标签的“有效”比特(如图1中所述)来确定该比特是否有效。CLMARK指令可以将该比特设置为“有效”。如果没有设置“有效”比特,那么该高速缓存行中的数据对于FASTCMPXCHG指令不是有效的数据,并且处理逻辑然后使用[mem]、testval和newval参数执行正常的CMPXCHG指令(处理框212)。如果设置了“有效”比特,那么表明在[mem]地址处的高速缓存行中的数据是有效的,并且处理逻辑然后将确定当前硬件上下文是否与高速缓存行标签中的微架构暗示比特中的硬件上下文ID(即,图1中暗示比特104)匹配(处理框206)。如果硬件不支持多个硬件上下文,那么HW上下文ID 104暗示比特将始终相同并且处理的该部分将始终成功。如果支持多个硬件上下文,那么HW上下文ID104将需要匹配,以使得处理的该部分成功。

[0043] 如果HW上下文ID没有匹配,那么处理逻辑使用[mem]、testval和newval参数执行正常的CMPXCHG指令(处理框212)。另一方面,如果HW上下文ID匹配或HW不支持多个上下文,那么处理逻辑确定是否正确地设置了比特-屏蔽(处理框208)。处理逻辑将CLMARK比特-屏蔽与FASTCMPXCHG比特-屏蔽进行比较。如果这些比特-屏蔽不匹配,那么处理逻辑使用[mem]、testval和newval参数执行正常的CMPXCHG指令(处理框212)。否则,如果比特-屏蔽匹配,那么处理逻辑然后可以继续前进并且执行新的FASTCMPXCHG指令,其使得在不执行CMPXCHG指令的LOAD-COMPARE部分的情况下直接存储(STORE)新的值。

[0044] 如所描述的,如果图2中决策树的任何分叉失败,那么正常的CMPXCHG始终作为退回情况而被执行。这确保代码的向前进行。

[0045] 在以下任何一个条件下,可以针对给定的高速缓存行对微架构暗示比特(图1中详细描述)进行清零和设置为无效:

- [0046] 1) 在高速缓存行上成功地执行了FASTCMPXCHG指令。
- [0047] 2) 从高速缓存回收具有暗示比特的高速缓存行。
- [0048] 3) 以该高速缓存行为目标的另一硬件上下文执行CLMARK。
- [0049] 4) 由任意硬件上下文对该高速缓存行进行写入。
- [0050] 5) 发生中断。

[0051] 6) 在CLMARK发生时与FASTCMPXCHG指令成功完成之前的时间期间发生任何硬件故障。

[0052] 在其他实施例中,当FASTCMPXCHG指令成功执行时,微架构暗示比特可以不被清零。在以下情况下不对暗示比特清零可以更高效:只要指令对相同数据进行操作,后续FASTCMPXCHG操作可以在没有与CLMARK指令相关联的开销的情况下发生。

[0053] 在使用比特-屏蔽的许多实施例中,上文中列出的用于使高速缓存行暗示比特清零和无效的规则可以被相应地修改,以考虑以下高速缓存情形:尽管高速缓存行可能受影响,但是由CLMARK加标志的字节没有受特别地影响。

[0054] 此外,lock前缀还可以用于FASTCMPXCHG指令,以将其变为原子指令。这是推荐的,因为如果通过除去CMPXCHG指令中的LOAD-COMPARE部分未使FASTCMPXCHG指令成功进行,那么要执行完整的LOAD-COMPARE-STORE微操作组合,因此需要锁定,正如原始CMPXCHG指令所需的一样。

[0055] 下面描述了执行CLMARK-FASTCMPXCHG指令的CPU的实施例。在该实施例中,CPU包括能够用需要的微架构暗示比特为高速缓存加标志的一个或多个核心。整个数量的样本代码列出如下:

[0056]

try_again:

clmark [mem] ; 设定[mem]处的高速缓存行的暗示比特

mov rax, [mem] ; 将位置[mem]处的存储器中的值加载到 rax 中

mov rdx, rax ; 将 rax 中的值加载到 rdx

inc rdx ; 修改 rdx

lock fastcmpxchg [mem], rdx ;如果图 2 中关于[mem]处的高速缓存行的暗示比特的处理框 202、204、206 以及 208 都为真,那么将 rdx 存储到[mem]

处的存储器中,否则,执行标准 cmpxchg

jnz try_again ;如果 fastcmpxchg 成功,那么 ZF=1, 否则

ZF=0, 从而跳转, 如果未成功, 则零指令将

导致另一尝试

[0057] 图3说明了包括能够执行CLMARK和FASTCMPXCHG指令的一个或多个CPU核心的计算机系统的实施例。

[0058] 示出了计算机系统300。计算机系统可以是台式机、服务器、工作站、膝上型计算

机、手持设备、电视机顶盒、媒体中心、游戏控制台、集成系统(例如在汽车中),或其他类型的计算机系统。在若干实施例中,计算机系统300包括一个或多个中央处理单元(CPU)。尽管在许多实施例中潜在地有许多CPU,但是为了清楚在图3中示出的实施例中只示出了两个CPU(302和304)。CPU 302和304可以是英特尔®公司的CPU或另外品牌的CPU。每个CPU包括一个或多个核心。在示出的实施例中,CPU 302包括核心A0(306)、核心A1(308)、核心A2(310)和核心A3(312),并且CPU 304包括核心B0(314)、核心B1(316)、核心B2(318)以及核心B3(320)。

[0059] 在其他的实施例中,CPU 302和304的每一个可以具有多于或少于图3中示出的每一个具有的四个核心的多个核心。在许多实施例中,每一个核心(例如核心A0(306))包括内部功能块,例如一个或多个执行单元、收回单元、通用和专用寄存器集合等。如果图3中示出的核心是多线程的或超线程的,那么每一个硬件线程也可以被认为是核心。

[0060] CPU 302和304的每一个还可以分别包括一个或多个高速缓存,例如末级高速缓存(LLC)322和324。在没有示出的许多实施例中,实现了不同于高速缓存322和324的另外的高速缓存,其中在每一个核心中的执行单元和存储器之间存在多级高速缓存。在不同的实施例中,可以以不同的方式分配高速缓存。在不同的实施例中,高速缓存322和324中每一个可以是多个不同大小中的一个。例如,高速缓存322和324每一个可以是8兆字节(MB)高速缓存、16MB高速缓存等。此外,在不同的实施例中,高速缓存可以是直接映射高速缓存、全相联高速缓存、多路组相联高速缓存(multi-way set-associative cache),或具有其他类型映射的高速缓存。每一个高速缓存可以包括在各个CPU中所有核心之间共享的一个大部分,或可以被分为数个单独的功能片(例如,每个核心一片)。每个高速缓存还可以包括在所有核心之间共享的一个部分,和作为每核心单独的功能片的数个其他部分。

[0061] 在许多实施例中,CPU 302和304的每一个包括其自己的系统存储器控制器(分别为326和328),用于提供与系统存储器330和332进行通信的接口。在没有示出的其他实施例中,存储器控制器330和332可以是分立的设备或被集成到计算机系统300中的其他设备内。

[0062] 系统存储器330和332可以包括动态随机存取存储器(DRAM)(例如一种双倍数据速率(DDR)DRAM)、非易失性存储器(例如闪速存储器、相变存储器(PCM)),或另外类型的存储器技术。系统存储器330和332可以是存储分别由CPU 302和304操作的数据和指令的通用存储器。此外,计算机系统300中还可以存在能够对系统存储器进行读取和写入的其他可能的设备,例如具备直接存储器存取(DMA)能力的I/O(输入/输出)设备。

[0063] 将每个CPU与每一个各自的系统存储器耦合起来的链路(即,总线、互连等)可以包括能够传输数据、地址、控制以及时钟信息的一个或多个光、金属或其他线(即,线路)。

[0064] 此外,CPU 302和304可以分别使用P2P接口电路334和336通过点到点(P2P)接口与彼此进行通信。P2P接口可以包括高速双向串行链路、分离的多对单向串行链路,或并行实现的链路等等。除了彼此通信,CPU 302和304还可以通过相同类型的P2P接口与高性能接口复合体338进行通信。具体地,CPU 302可以通过在CPU侧的P2P接口电路340和在复合体338侧的P2P接口电路342与复合体338进行通信,并且CPU 304可以通过在CPU侧的P2P接口电路344和在复合体338侧的P2P接口电路346与复合体338进行通信。

[0065] 高性能接口复合体338可以为需要高数据吞吐量的任何子系统提供接口。例如,高性能图形子系统348可以通过I/O接口350与CPU进行通信,并且高性能通信子系统352可以

通过I/O接口354进行通信。高性能接口复合体338还可以包括用于与使用I/O接口360的I/O集线器复合体358进行通信的I/O接口356。用于计算机系统300中示出的每一个I/O接口的电路可以是相同的或可以是不同的。例如,将高性能图形子系统348耦合到复合体338的I/O接口350可以包括16通道外围组件接口(PCI)-快速协议链路,而将高性能接口复合体338耦合到I/O复合体358的I/O接口356可以使用不同的协议。

[0066] I/O集线器复合体358可以在耦合到一个或多个I/O互连(即,总线)的设备与CPU 302和304之间提供通用通信接口。例如,I/O集线器复合体358可以包括主控制器362和364。每一个主控制器可以提供允许I/O设备通信地耦合到计算机系统300的其余部件的接口。例如,一个I/O集线器复合体可以是通用串行总线(USB)集线器复合体,而另一个可以是遗留的PCI集线器复合体。示出的I/O设备366和370分别耦合到I/O主控制器362和364。在许多实施例中,可以存在耦合到I/O主控制器(例如I/O主控制器362)的大容量存储设备368。大容量存储设备368可以是硬盘驱动器、固态驱动器、相变存储器阵列,或另外形式的大容量存储器。此外,可以存在与其他遗留的总线有接口的一个或多个桥。例如,桥372可以耦合到I/O主控制器接口,并且该桥可以提供到在其上耦合I/O设备374的另一协议互连/总线的协议转换。

[0067] 能够成功地执行CLMARK和FASTCMPXCHG指令的处理逻辑的至少一个实施例可以存在于计算机系统300的每一个核心中。该逻辑由分别位于核心A0(306)、A1(308)、A2(310)和A3(312)中的处理逻辑400、402、404和406表示,以及由分别位于B0(314)、B1(316)、B2(318)和B3(320)中的处理逻辑408、410、412和414表示。此外,在其他实施例中,能够成功执行CLMARK和FASTCMPXCHG指令的处理逻辑可以遍及图3中说明的若干电路、逻辑单元或设备而分布。

[0068] 尽管没有说明,但是使用CPU、总线、存储器等的不同布局的其他计算机系统实现对于实现本发明而言也是完全可接受的。

[0069] 此外,在操作期间,包括CLMARK和FASTCMPXCHG指令的代码在不同时间可以驻留在计算机系统300中的一个或多个位置。例如,实现新的指令的代码416可以驻留(即,被存储在)系统存储器330或332中(代码416A或代码416B)、高速缓存322或324中(代码416C或代码416D)、大容量存储设备368中(代码416E),或计算机系统300内部或外部的其他地方。

[0070] 因此,描述了用于实现自适应优化的比较-交换操作的机器可读介质、处理器以及系统的实施例。参考其具体的示例性实施例描述了这些实施例。对于从本公开获益的人员将显而易见的是,可以在不脱离本文描述的实施例的宽泛精神和范围的情况下,对这些实施例进行各种修改和改变。本说明书和附图应当相应地被认为是说明性的而非限制性的。

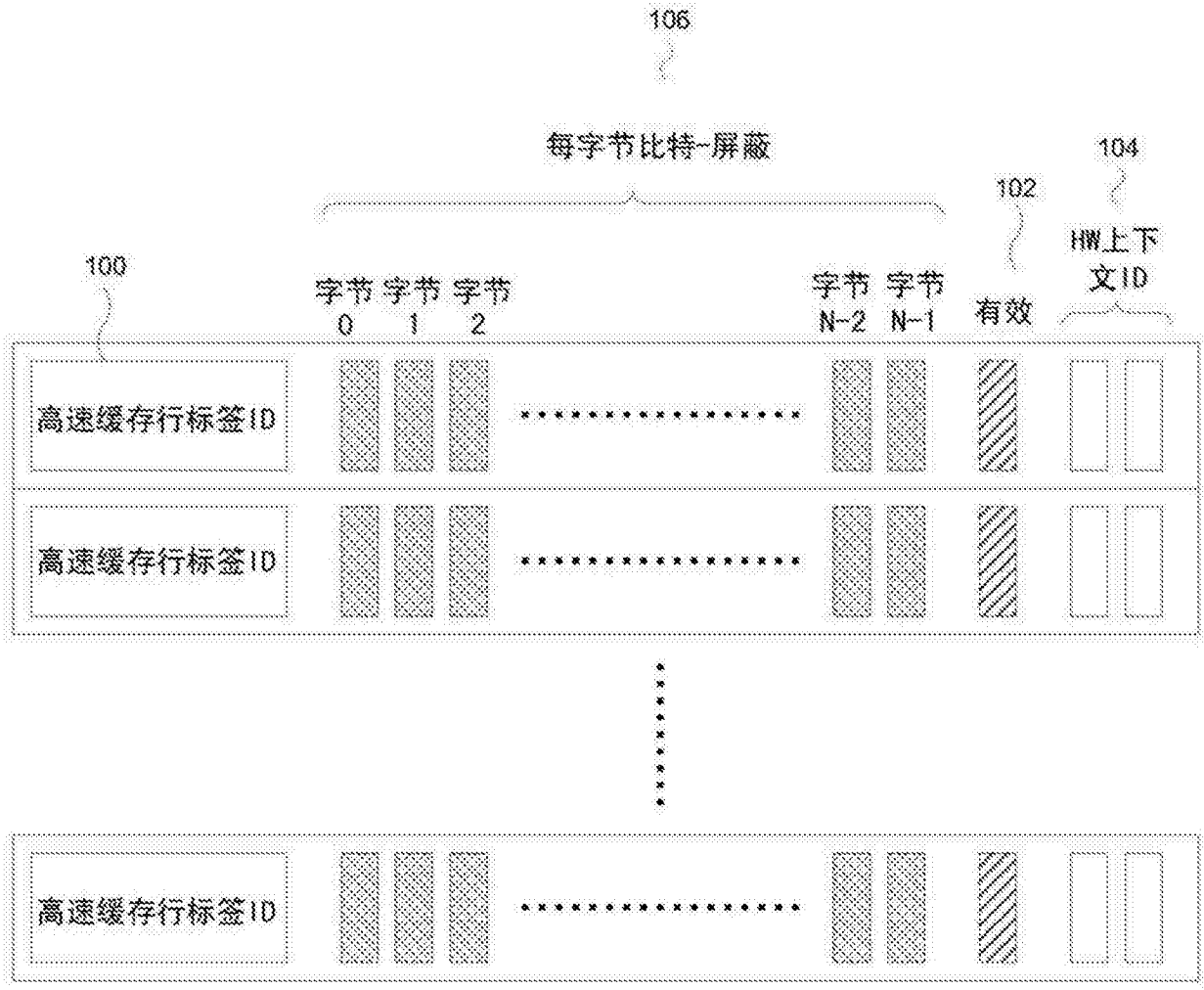


图1

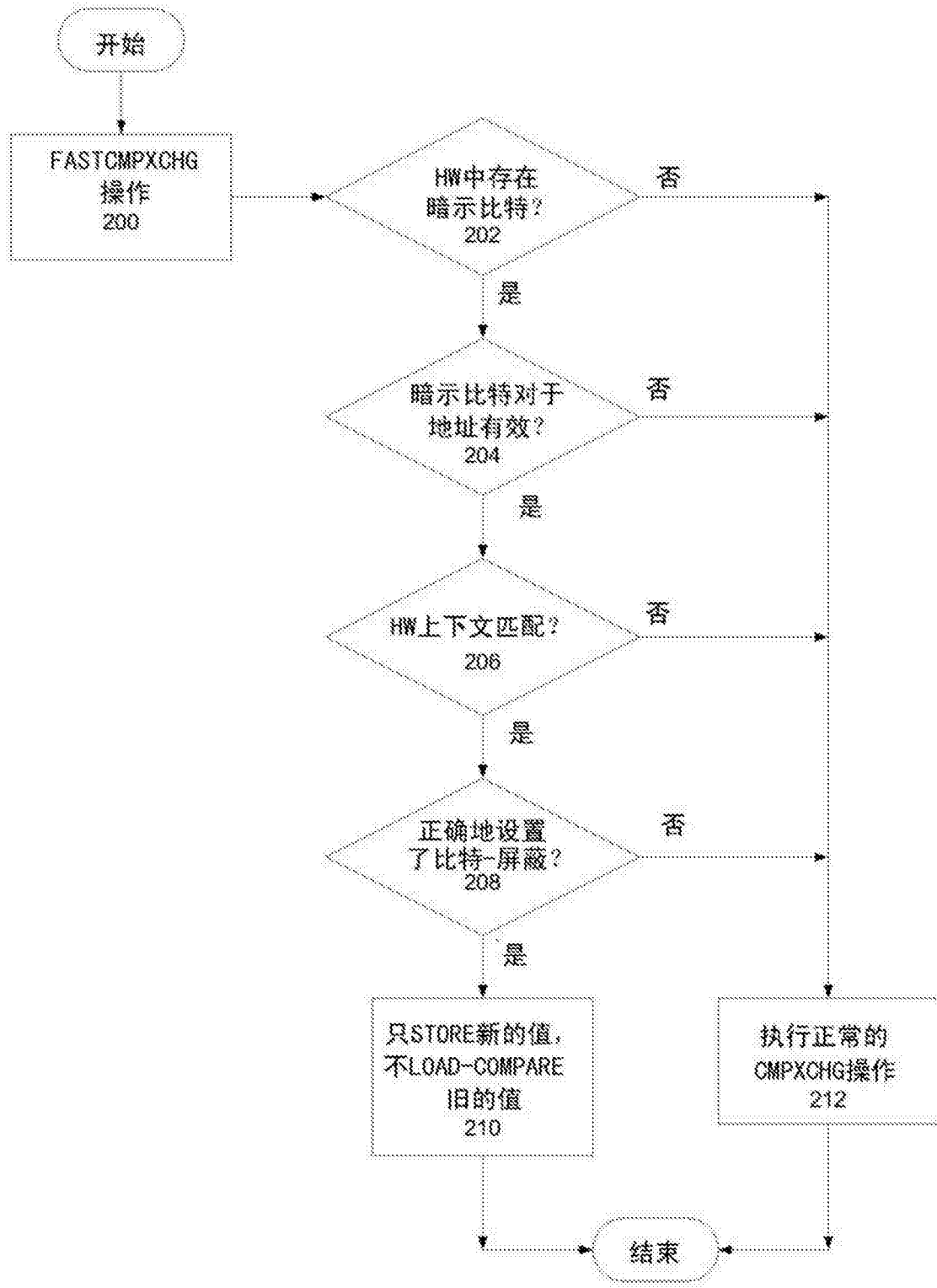


图2

300

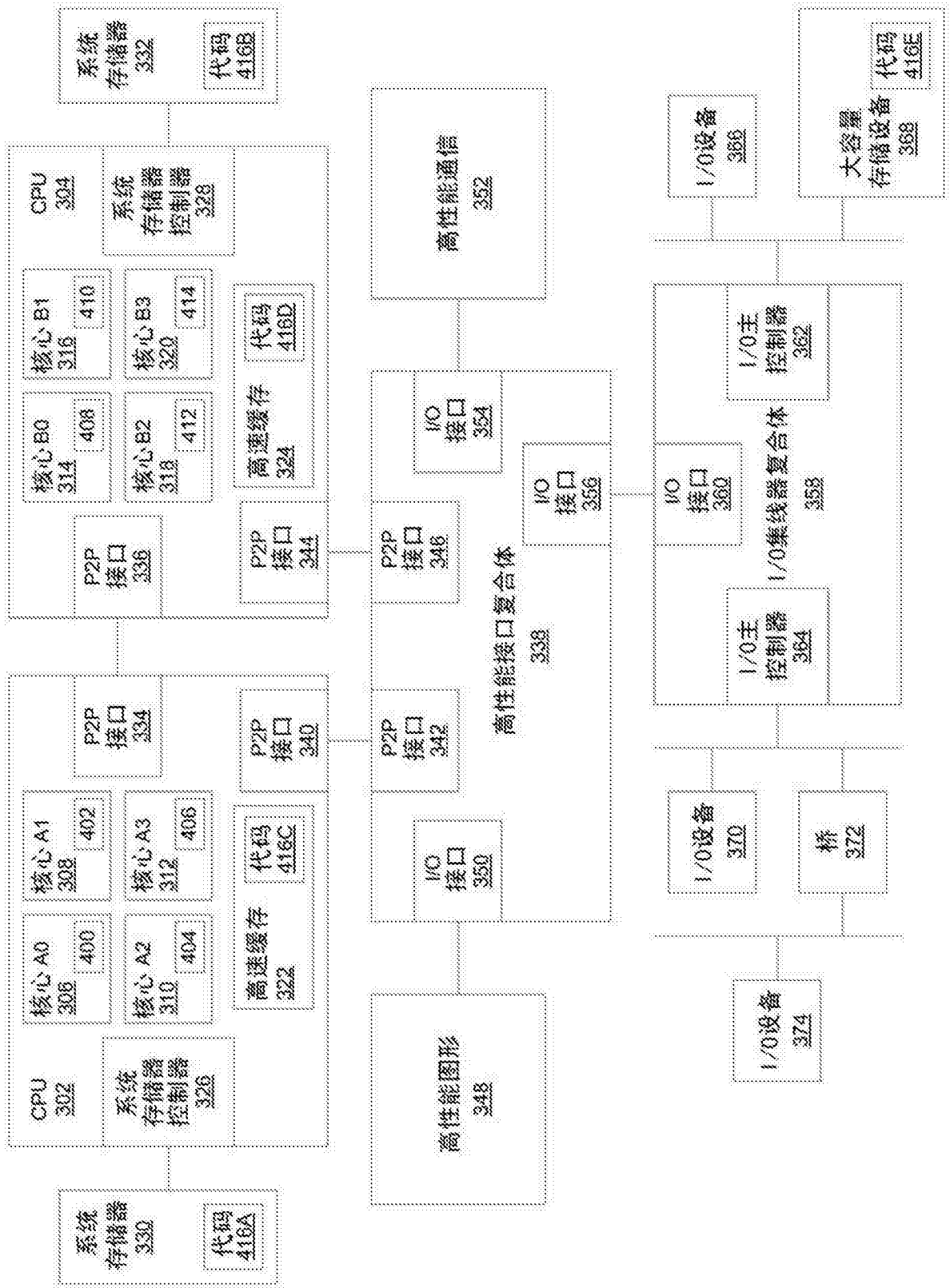


图3