



(12) 发明专利申请

(10) 申请公布号 CN 104603745 A

(43) 申请公布日 2015. 05. 06

(21) 申请号 201380045583. 3

(51) Int. Cl.

(22) 申请日 2013. 06. 12

G06F 9/06(2006. 01)

(30) 优先权数据

G06F 9/30(2006. 01)

13/630, 247 2012. 09. 28 US

G06F 12/00(2006. 01)

(85) PCT国际申请进入国家阶段日

2015. 02. 28

(86) PCT国际申请的申请数据

PCT/US2013/045505 2013. 06. 12

(87) PCT国际申请的公布数据

W02014/051737 EN 2014. 04. 03

(71) 申请人 英特尔公司

地址 美国加利福尼亚州

(72) 发明人 M·普罗特尼科夫 A·纳赖金

C·休斯

(74) 专利代理机构 上海专利商标事务所有限公

司 31100

代理人 姬利永

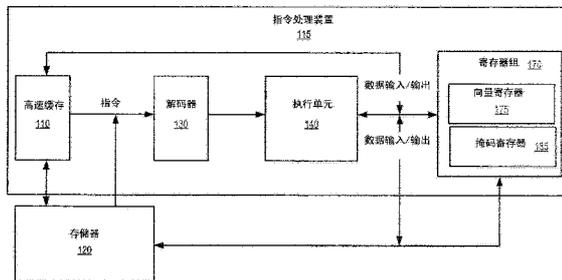
权利要求书2页 说明书15页 附图15页

(54) 发明名称

用于独立数据上递归计算的向量化的读和写掩码更新指令

(57) 摘要

处理器执行掩码更新指令以执行对第一掩码寄存器和第二掩码寄存器的更新。处理器内的寄存器组包括第一掩码寄存器和第二掩码寄存器。处理器包括用于执行掩码更新指令的执行电路。响应于掩码更新指令, 执行电路用于反转第一掩码寄存器中给定数量的掩码位, 并且还用于反转第二掩码寄存器中给定数量的掩码位。



1. 一种装置,包括:
寄存器组,包括第一掩码寄存器和第二掩码寄存器;以及
执行电路,与所述寄存器组耦合,用于执行掩码更新指令,其中响应于所述掩码更新指令,所述执行电路用于反转所述第一掩码寄存器中给定数量的掩码位并且用于反转所述第二掩码寄存器中所述给定数量的掩码位。
2. 如权利要求 1 所述的装置,其中所述给定数量是所述第一掩码寄存器中具有第一位值的掩码位的数量和所述第二掩码寄存器中具有第二位值的掩码位的数量中较小的一个。
3. 如权利要求 1 所述的装置,其中所述第一位值与所述第二位值相同。
4. 如权利要求 1 所述的装置,其中所述第一位值是所述第二位值的反转。
5. 如权利要求 1 所述的装置,其中所述第一掩码寄存器中所述给定数量的掩码位是所述第一掩码寄存器中的低阶掩码位或高阶掩码位。
6. 如权利要求 1 所述的装置,其中所述第二掩码寄存器中所述给定数量的掩码位是所述第二掩码寄存器中的低阶掩码位或高阶掩码位。
7. 如要求 1 所述的装置,进一步包括:
第一向量寄存器,与所述第一掩码寄存器相关联,其中所述第一掩码寄存器的每个掩码位指示用于所述第一向量寄存器中相应数据元素的条件是否被满足;以及
第二向量寄存器,与所述第二掩码寄存器相关联,其中所述第二掩码寄存器的每个掩码位指示用于所述第二向量寄存器中相应数据元素的条件是否被满足。
8. 如权利要求 7 所述的装置,其中当用于给定数据元素的条件被满足时,对于所述给定数据元素,需要进一步计算。
9. 一种方法,包括:
由处理器接收指定第一掩码寄存器和第二掩码寄存器的掩码更新指令;以及
响应于所述掩码更新指令,执行操作,包括:
反转所述第一掩码寄存器中给定数量的掩码位,以及
反转所述第二掩码寄存器中所述给定数量的掩码位。
10. 如权利要求 9 所述的方法,其中所述给定数量是所述第一掩码寄存器中具有第一位值的掩码位的数量和所述第二掩码寄存器中具有第二位值的掩码位的数量中较小的一个。
11. 如权利要求 9 所述的方法,其中所述第一位值与所述第二位值相同。
12. 如权利要求 9 所述的装置,其中所述第一位值是所述第二位值的反转。
13. 如权利要求 9 所述的方法,其中所述第一掩码寄存器中所述给定数量的掩码位是所述第一掩码寄存器中的低阶掩码位或高阶掩码位。
14. 如权利要求 9 所述的方法,其中所述第二掩码寄存器中所述给定数量的掩码位是所述第二掩码寄存器中的低阶掩码位或高阶掩码位。
15. 如权利要求 9 所述的方法,其中所述第一掩码寄存器的每个掩码位指示用于第一向量寄存器中相应数据元素的条件是否被满足,以及
所述第二掩码寄存器的每个掩码位指示用于第二向量寄存器中相应数据元素的条件是否被满足。
16. 如权利要求 15 所述的方法,其中当用于给定数据元素的条件被满足时,对于所述

给定数据元素,需要进一步计算。

17. 一种系统,包括:

存储器,用于存储输入数据数组;

寄存器组,包括用于存储掩码更新指令的操作数的第一掩码寄存器和第二掩码寄存器,以及分别与所述第一掩码寄存器和所述第二掩码寄存器相关联、用于加载向量计算的所述输入数据数组的第一向量寄存器和第二向量寄存器;以及

执行电路,与所述寄存器组耦合,用于执行所述掩码更新指令,其中响应于所述掩码更新指令,所述执行电路用于将所述第一掩码寄存器中给定数量的掩码位从第一位值设置为第二位值并且用于将所述第二掩码寄存器中所述给定数量的掩码位从所述第二位值设置为所述第一位值。

18. 如权利要求 17 所述的系统,其中所述给定数量是所述第一掩码寄存器中具有所述第一位值的掩码位的数量和所述第二掩码寄存器中具有所述第二位值的掩码位的数量中较小的一个。

19. 如权利要求 17 所述的系统,其中所述第一位值与所述第二位值相同。

20. 如权利要求 17 所述的系统,其中所述第一位值是所述第二位值的反转。

21. 如权利要求 17 所述的系统,其中所述第一掩码寄存器中所述给定数量的掩码位是所述第一掩码寄存器中的低阶掩码位或高阶掩码位。

22. 如权利要求 17 所述的系统,其中所述第二掩码寄存器中所述给定数量的掩码位是所述第二掩码寄存器中的低阶掩码位或高阶掩码位。

23. 如权利要求 17 所述的系统,其中所述第一掩码寄存器中所述第一位值的每个掩码位指示用于所述第一向量寄存器中相应数据元素的条件未被满足,并且所述第二掩码寄存器中所述第二位值的每个掩码位指示用于所述第二向量寄存器中相应数据元素的条件被满足。

24. 如权利要求 23 所述的系统,其中当用于给定数据元素的条件被满足时,对于所述给定数据元素,需要进一步计算。

用于独立数据上递归计算的向量化的读和写掩码更新指令

技术领域

[0001] 本公开涉及处理逻辑、微处理器以及相关指令集架构的领域，该指令集架构在被处理器或其他处理逻辑所执行时运行逻辑、数学或其他功能性操作。

背景技术

[0002] 指令集或指令集架构 (ISA) 是计算机架构中与编程有关的部分，并且可包括原生数据类型、指令、寄存器架构、寻址模式、存储器架构、中断和异常处理、以及外部输入和输出 (I/O)。术语指令在本申请中一般表示宏指令——宏指令是被提供给处理器（或指令转换器，该指令转换器（例如利用静态二进制转换、包括动态编译的动态二进制转换）转换、变形、仿真或以其他方式将指令转换成将由处理器处理的一个或多个其他指令）以供执行的指令——作为对比，微指令或微操作（微 ops）是处理器的解码器解码宏指令的结果。

[0003] ISA 与微架构不同，微架构是实现该指令集的处理器器的内部设计。具有不同微架构的处理器可共享共同的指令集。例如，Intel® 酷睿 (Core™) 处理器、以及来自加利福尼亚州桑尼威尔 (Sunnyvale) 的超微半导体有限公司 (Advanced Micro Devices, Inc.) 的诸多处理器执行几乎相同版本的 x86 指令集（在更新的版本中加入了一些扩展），但具有不同的内部设计。例如，可利用公知技术（包括专用物理寄存器、利用寄存器重命名机制的一个或多个动态分配的物理寄存器）在不同微架构中以不同方式实现该 ISA 的同一寄存器架构。

[0004] 许多现代 ISA 支持单指令多数据 (SIMD) 操作。取代仅对一个或两个数据元素进行操作的标量指令，向量指令（也称为紧缩数据指令或 SIMD 指令）可同时或并行地对多个数据元素或多对数据元素进行操作。处理器可具有并行的执行硬件，以响应于该向量指令同时或并行地执行多个操作。SIMD 操作在一个操作中对紧缩在一个向量寄存器或存储器位置之内的多个数据元素进行操作。这些数据元素被称为紧缩数据或向量数据。向量元素中的每一个可表示独立的单条数据（例如像素的颜色，等等），可单独或与其它数据无关地操作该单条数据。

[0005] 在一些情况下，SIMD 操作可以以递归方式对独立向量数据元素进行操作，其中对于不同数据元素，迭代的数量不同。因此，一些数据元素的计算可能完成，而一些其它数据元素仍需要更多迭代。递归计算的一个示例是 WHILE 循环操作。在本示例中，在条件 (X[i]) 为真（满足）时，对 N 元素的数组 X[i] (i = 0, ..., N-1) 进行递归计算。当条件 (X[i]) 变为假时，X[i] 的计算停止。该条件的示例可以是 X[i] > 0。

```
[0006] for(i = 0 ; i < N ; i++) {
```

```
[0007] while(条件 (X[i])) {
```

```
[0008] X[i] = 计算 (X[i]) ; }
```

[0009] 如果对于 X[i] 的不同数据元素，WHILE 循环迭代的数量不同，则以上计算无法被轻易向量化。一种可能的方法是用处理器在不满足条件的那些元素上执行计算，然后丢弃从这些元素导出的结果。然而，该方法具有低效率，因为处理器不仅在那些元素上执行不必要的计算，而且还无法利用由那些元素占据的向量寄存器槽口。

附图说明

[0010] 在附图中的诸个图中通过示例而非限制地示出各个实施例：

[0011] 图 1 是根据一个实施例的包括向量寄存器和掩码寄存器的指令处理装置的框图。

[0012] 图 2 是根据一个实施例的寄存器架构的框图。

[0013] 图 3 示出根据一个实施例的向量操作序列的示例。

[0014] 图 4A 示出根据一个实施例的使处理器在向量寄存器和掩码寄存器上执行操作的指令的伪代码示例。

[0015] 图 4B 示出根据一个实施例的使用图 4A 的指令的代码段示例。

[0016] 图 5A 是根据一个实施例的示出响应于使用掩码更新指令和向量移动指令的代码段而执行的操作的流程图。

[0017] 图 5B 是示出根据一个实施例的响应于掩码更新指令而执行的操作的流程图。

[0018] 图 5C 是示出根据一个实施例的响应于向量移动指令而执行的操作的流程图。

[0019] 图 6 是示出根据一个实施例的使用软件指令转换器将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。

[0020] 图 7A 是根据一个实施例的有序和无序流水线的框图。

[0021] 图 7B 是根据一个实施例的有序和无序核的框图。

[0022] 图 8A-B 是根据一个实施例的更具体的示例性有序核架构的框图。

[0023] 图 9 是根据一个实施例的处理器的框图。

[0024] 图 10 是根据一个实施例的系统的框图。

[0025] 图 11 是根据一个实施例的第二系统的框图。

[0026] 图 12 是根据本发明的实施例的第三系统的框图。

[0027] 图 13 是根据一个实施例的芯片上系统 (SoC) 的框图。

具体实施方式

[0028] 在以下描述中,陈述了多个具体细节。然而,应当理解的是,可不通过这些具体细节来实施本发明的实施例。在其它实例中,未详细示出公知的电路、结构以及技术,以免模糊对本描述的理解。

[0029] 本文所述的实施例提供用于提高独立数据元素上的递归向量计算的效率的指令。这些指令利用一对向量寄存器和一对掩码寄存器来执行递归向量计算,其中第一向量寄存器充当用于累加向量计算结果的累加器,并且第二向量寄存器提供新的数据元素以填充第一向量寄存器的未利用槽口(未使用或已完成的数据元素位置)。掩码寄存器用于指示相应向量寄存器中的哪些数据元素需要进一步的计算。

[0030] 在一个实施例中,第一向量寄存器(即累加器)累加输入数据元素,直到寄存器被完整向量填满。然后,处理器使用非掩码(即,密集)向量操作在这些数据元素上执行计算。在计算之后,累加器中的一些元素(对于这些元素,完成了计算)可以被发送回存储器或其它存储位置,而其它元素(对于这些元素,未完成计算)可以被保持在累加器中用于附加数量的迭代。累加器中已完成计算的数据元素位置可以由还需要相同递归计算的新数据元素所利用。

[0031] 本文描述两个指令 RWMASKUPDATE 和 SPARSEMOV。这些指令在许多情况下提高向量化的效率。例如,在一种情况下,输入数据元素可能来自一个或多个稀疏向量数据集,每个稀疏向量数据集不具有足以填满整个累加器(即,第一向量寄存器)的元素。此外,来自不同数据集的输入数据元素在计算中可能需要不同数量的迭代。因此,由于无需更多计算的那些数据元素,在累加器中剩余未利用的槽口。本文所述的指令允许这些未利用的槽口被有用的元素填充,从而实现在完整向量上的递归计算。如在以下进一步细节中所述的,SPARSEMOV 指令是将有用的数据元素(即需要计算的数据元素)从第二向量寄存器移动到累加器中的向量移动指令。RWMASKUPDATE 指令更新读掩码寄存器(与第二向量寄存器相关联)和写掩码寄存器(与累加器相关联)以标识在这两个向量寄存器中的有用数据元素的位置。

[0032] 与 SPARSEMOV 相结合地使用 RWMASKUPDATE 可以降低递归计算中所需指令的总数,并且简化上溢和下溢情形,在上溢和下溢情形中第二向量寄存器中有用数据元素(即,源数据元素)的数量不与第一向量寄存器中未利用槽口(即,目标位置)的数量相匹配。经更新的读和写掩码用于控制两个向量寄存器之间的数据移动;具体而言,写掩码位 0 用于标识累加器中的目标位置;并且读掩码位 1 用于标识第二向量寄存器中的源数据元素。使用反转的写掩码位来标识目标位置简化稀疏和递归计算的向量化中的数据累加。

[0033] 图 1 是指令处理装置 115 的实施例的框图,该指令处理装置具有执行单元 140,该执行单元包括可操作于执行指令(包括 RWMASKUPDATE 和 SPARSEMOV 指令)的电路。在一些实施例中,指令处理装置 115 可以是处理器、多核处理器的处理器核、或者电子系统中的处理元件。

[0034] 解码器 130 接收高级机器指令或宏指令形式的传入指令,并且解码所述指令以生成低级微操作、微代码进入点、微指令或其他低级指令或控制信号,它们反映了原始的高级指令和/或从原始的高级指令导出。低级指令或控制信号可通过低级(例如,电路级或硬件级)操作来实现高级指令的操作。可使用各种不同的机制来实现解码器 130。合适机制的示例包括但不限于,微代码、查找表、硬件实现、可编程逻辑阵列(PLA)、用于实现本领域已知的解码器的其他机制等。

[0035] 解码器 130 可接收针对高速缓存 110、存储器 120 或其他源的传入指令。经解码的指令被发送到执行单元 140。执行单元 140 可从解码器 130 接收一个或多个微操作、微代码进入点、微指令、其它指令或其它控制信号,它们反映了所接收的指令或者是从所接收的指令导出的。执行单元 140 从寄存器组 170、高速缓存 110 和/或存储器 120 接收数据输入并向它们生成数据输出。

[0036] 在一个实施例中,寄存器组 170 包括架构寄存器,架构寄存器也被称为寄存器。短语架构寄存器、寄存器组、以及寄存器在本文中用于表示对软件和/或编程器可见(例如,软件可见的)和/或由宏指令指定来标识操作数的寄存器,除非另外予以规定或清楚明显可知。这些寄存器不同于给定微架构中的其他非架构式寄存器(例如,临时寄存器、重排序缓冲器、引退寄存器等)。

[0037] 为了避免混淆描述,已示出和描述了相对简单的指令处理装置 115。应当理解,其他实施例可具有超过一个执行单元。例如,装置 115 可包括多个不同类型的执行单元,诸如例如算术单元、算术逻辑单元(ALU)、整数单元、浮点单元等。指令处理装置或处理器的再

其他实施例可具有多个核、逻辑处理器或执行引擎。稍后将参考图 7-13 提供指令处理装置 115 的多个实施例。

[0038] 根据一个实施例,寄存器组 170 包括一组向量寄存器 175 和一组掩码寄存器 185,两者存储 RWMASKUPDATE 和 SPARSEMOV 指令的操作数。每个向量寄存器 175 可以是 512 位、256 位、或 128 位宽,或者可以使用不同的向量宽度。每个掩码寄存器 185 包含多个掩码位,每个掩码位对应于向量寄存器 175 之一的一个数据元素。由于每个掩码位用于掩码向量寄存器的数据元素,64 位掩码寄存器可以用于掩码 512 位寄存器的六十四 8 位数据元素。对于具有不同宽度(例如 256 位或 128 位)的向量寄存器和不同尺寸(例如 16 位、32 位或 64 位)数据元素,可以结合向量操作,使用不同数量的掩码位。

[0039] 图 2 示出了支持本文描述的指令的底层寄存器架构 200 的实施例。寄存器架构 200 基于 Intel® 酷睿(Core™)处理器,该处理器实现包括 x86、MMX™、流 SIMD 扩展(SSE)、SSE2、SSE3、SSE4.1 和 SSE4.2 指令的指令集,以及 SIMD 扩展的附加集,该附加集被称为高级向量扩展(AVX)(AVX1、和 AVX2)。然而,应理解,也可使用支持不同寄存器长度、不同寄存器类型和/或不同数量的寄存器的不同的寄存器架构。

[0040] 在所示的实施例中,存在 512 位宽的 32 个向量寄存器 210;这些寄存器被称为 zmm0 至 zmm31。低位 16 个 zmm 寄存器的低阶 256 位覆盖在寄存器 ymm0-16 上。低位 16 个 zmm 寄存器的低阶 128 位(ymm 寄存器的低阶 128 位)覆盖在寄存器 xmm0-15 上。在所示实施例中,存在 8 个掩码寄存器 220(k0 到 k7),每个掩码寄存器的长度为 64 位。在一替代实施例中,掩码寄存器 220 的宽度是 16 位。

[0041] 在所示实施例中,寄存器架构 200 进一步包括 16 个 64 位通用(GP)寄存器 230。在一实施例中,上述通用寄存器沿循现有 x86 寻址模式被使用以对存储器操作数寻址。该实施例还示出 RFLAGS 寄存器 260、RIP 寄存器 270 和 MXCSR 寄存器 280。

[0042] 该实施例还示出了标量浮点(FP)堆栈寄存器组(x87 堆栈)240,在其上面重叠了 MMX 紧缩整数平坦寄存器组 250。在所示出的实施例中,x87 堆栈是用于使用 x87 指令集扩展来对 32/64/80 位浮点数据执行标量浮点运算的八元素堆栈;而 MMX 寄存器用于对 64 位紧缩整数数据执行操作,以及为在 MMX 和 xmm 寄存器之间执行的一些操作保存操作数。

[0043] 本发明的替代实施例可以使用较宽的或较窄的寄存器。另外,本发明的替代实施例可以使用更多、更少或不同的寄存器组和寄存器。

[0044] 图 3 是示出由处理器(例如,指令处理装置 115)执行以有效地向量化独立数据元素上的计算的操作示例的示图。为了简化说明,该示例中的每个向量寄存器被示为仅具有八个数据元素。替代实施例可以在向量寄存器中具有不同数量的数据元素。向量寄存器可以是 128 位、256 位、或 512 位宽(例如,图 2 的 xmm、ymm 或 zmm 寄存器),或者可以使用不同宽度。由于每个向量寄存器中有八个数据元素,结合每个向量寄存器,仅使用八个掩码位。

[0045] 在该示例中,向量寄存器 V1 用作累加器,并且向量寄存器 V2 用于向 V1 提供新的数据元素。掩码寄存器 K1(写掩码)和 K2(读掩码)分别用于掩码 V1 和 V2 中的数据元素。在该示例中,掩码位 0 指示从计算中掩码相应的数据元素(即,进一步的计算是不必要的),并且掩码位 1 指示相应数据元素需要进一步的计算。在替代实施例中,掩码位值的意义可以颠倒;例如,掩码位 1 可以用于指示相应数据元素不需要进一步的计算,并且掩码位 0 可以用于指示相应数据元素需要进一步的计算。

[0046] 最初,假定累加器 V1 存储两组数据作为输入向量 :A 和 B,每一个向量可以是稀疏数组的一部分。 A_j 和 B_j 的下标 j 指示数据元素所经过的迭代数量 ;例如, A_0 是任何迭代之前的元素 A,且 A_1 是第一次迭代 310 之后的元素 A。为了简化说明,来自相同迭代中相同数据集的不同数据元素被示为具有相同的标识符 ;例如,输入向量的位置 0 处的 A_0 和位置 2 处的 A_0 是两个不同元素并可以具有相同或不同的值,并且输入向量的位置 1 处的 B_0 和位置 3 处的 B_0 是两个不同元素并可以具有相同或不同的值。掩码寄存器 K1 中掩码位的初始值为全 1,指示 V1 中的初始输入向量是完整向量并且 V1 的每个元素都可以参与向量计算的第一次迭代 310。

[0047] 在该示例中,每个迭代表示执行递归向量计算的 WHILE 循环的迭代。在第一次迭代 310 之后,累加器 V1 包括一组 A_1 和 B_1 ,其中下标指示这些元素已经完成第一次迭代。假定 A 元素仅需要一次 WHILE 循环的迭代,并且 B 元素需要两次迭代。因此,在一次 WHILE 循环的迭代之后, A 元素的计算完成,而 B 元素需要再一次迭代。此时,对于每个 A 元素的条件为假(因为它们不满足进一步计算的条件),并且对于每个 B 元素的条件为真(因为它们满足进一步计算的条件)。因此,对于与 A_1 对应的那些掩码位, K1 中的掩码位值被设置为 0,而对于与 B_1 对应的那些掩码位, K1 中的掩码位值被设置为 1。

[0048] 在一个实施例中,掩码位 0 指示相应元素位置中的结果会在整个向量寄存器(在本情形中, V1) 上的向量操作之后被丢弃。在替代实施例中,掩码位 0 指示不会进行相应元素位置的计算,并因此,该元素位置未被利用。在任一情况下,将 A_1 保持在累加器 V1 中是向量资源的浪费,并且会降低向量计算的效率。因此,根据本发明的一个实施例,第二向量寄存器 V2 用于向 V1 提供新的数据元素以填充因为 A_1 剩余的未利用槽口(即,数据元素位置)。数据元素 A_1 可被保存到存储器、高速缓存或其它数据储存器中。

[0049] 在图 3 的示例中,向量寄存器 V2 存储数据集 C 的元素,数据集 C 可以是另一稀疏向量数组的一部分。V2 中由“*”标记的位置表示“不关心”,这意味着它们并未包含针对递归向量计算目的的有效数据元素。假定 C 的每个数据元素需要通过三次 WHILE 循环的迭代。作为 C 的元素的替代或者除其之外, V2 可以提供需要通过一次或多次 WHILE 循环迭代(以及因此的进一步计算)的新数据元素 A 和 / 或 B(例如 A_0 、 B_0 和 / 或 B_1)。V2 中需要进一步计算的这些数据元素被称为“源数据元素”。V2 中的这些源数据元素可以填充 V1 中因为 A_1 剩余的未利用槽口(被称为“目标数据元素”)。为了描述简便, V1 和 / 或 V2 中需要进一步计算的数据元素可被称为“有用数据元素”。因此,执行合并操作 320 以合并 V1 和 V2 中的有用数据元素,使得 V2 中的源数据元素被移动到 V1 中由目标数据元素占据的位置,并且递归计算可以前进到使用 V1 中附加的有用数据元素的第二次迭代 330。

[0050] 在这种合并操作中可能发生三种情况 :上溢、下溢和精确匹配。

[0051] 精确匹配指示在 V2 中存在与 V1 中剩余的未利用槽口数量相同的有用数据元素。因此,在精确匹配中,将 V2 中所有源数据元素移动到(即,替换)V1 中剩余的未利用槽口中。结果, V1 具有完整向量以开始下一次迭代,并且 K1 被更新为包含全 1。在 V2 中没有剩余其它源数据元素,并且 K2 被更新为包含全 0。

[0052] 合并操作 320 示出上溢情况,其中新数据元素 (C_0) 的数量大于 K1 中零值掩码位的数量(即 A_1 的数量)。因此,并非 V2 中所有新数据元素都可以移动到 V1 中。在该示例中, V2 的位置 7 处所圈出的 C_0 被剩余在 V2 中,而位置 2、4 和 6 处的其它 C_0 被移动到 V1 中。

在该实施例中, V2 的低阶元素被移动到 V1 中; 在替代实施例中, V2 的高阶元素被移动到 V1 中。合并操作 320 还更新 K1 和 K2 中的相应掩码位。

[0053] 在合并操作 320 之后, V1 包含完整的八元素向量以开始第二次迭代 330, 并且 V2 仅具有在位置 7 处剩余的一个 C_0 。此时 (在合并操作 320 之后), 相应掩码寄存器 K1 包含全 1, 并且 K2 仅包含在位置 7 处具有值 1 的一个掩码位。

[0054] 在第二次迭代 330 之后, 累加器 V1 包含 B_2 和 C_1 的组合。由于 B 元素的计算在该迭代之后完成, 这些 B_2 可以被存储到存储器、高速缓存或其它数据存储器中。因此, 对于每个 B 元素的条件为假 (因为它们不满足进一步计算的条件), 并且对于每个 C 元素的条件为真 (因为它们满足进一步计算的条件)。因此, 对于与 B_2 对应的那些掩码位, K1 中的掩码位值被设置为 0, 而对于与 C_1 对应的那些掩码位, K1 中的掩码位值被设置为 1。

[0055] 由 B2 剩余的未利用槽口可以由 V2 中遗留的源数据元素填充; 在该情形下, 是 V2 的位置 7 处的 C_0 。然而, 由于存在比 B2 数量少的 C_0 , 在随后的合并操作 340 中发生下溢。在如图 3 所示的下溢情况下, V1 中的低阶 B_2 被 C_0 替换; 在替代实施例中, V1 中的高阶 B_2 可以被 C_0 替换。合并操作 340 还更新 K1 和 K2 中的相应掩码位。

[0056] 在合并操作 340 之后, 累加器 V1 未被完全填充, 并且 V2 没有可以移动到 V1 中的任何其它有用的数据元素。此时 (在合并操作 340 之后), 掩码寄存器 K1 在与 C 元素对应的位置处包含 1, 并且 K2 包含全 0。V2 可以加载要移动到 V1 中的附加的有用数据元素, 并且合并操作 320 和 / 或 340 可以重复, 直到所有的有用数据元素被处理并且 V2 中没有剩余其它源数据元素。此时, V1 可以通过多个附加迭代, 直到 V1 中的所有元素达到所要求数量的迭代。

[0057] 应该理解, 掩码位值 0 和 1 的意义可以与图 3 示例所示的意义相反; 例如, 掩码位值 0 可以用于表示条件被满足, 而掩码位值 1 可以用于表示条件未被满足。在一些实施例中, K1 掩码位值的意义可以与 K2 掩码位值的意义相反; 例如, K1 掩码位值 1 可以用于表示条件未被满足, 而 K2 掩码位值 1 可以用于表示条件被满足。

[0058] 因此, 对于相同的情况, 在图 3 的示例中可以使用不同的掩码位值, 只要每个掩码寄存器中每个掩码位的意义被一致地定义以允许一致的解釋。

[0059] 根据本发明的一个实施例, 响应于包括 RWMASKUPDATE 和 SPARSEMOV 指令的向量指令, 结合图 3 所描述的操作由处理器 (例如指令处理装置 115) 执行。SPARSEMOV 指令可以用于将源数据元素从向量寄存器 V2 移动到向量寄存器 V1 中, 替换 V1 中不满足条件的目标元素 (例如, 不需要更多计算的元素)。RWMASKUPDATE 指令可以用于更新掩码寄存器 K1 和 K2 以由此分别标识 V1 和 V2 中满足条件的数据元素 (例如, 需要更多计算的元素) 的位置。在一个实施例中, RWMASKUPDATE 具有两个操作数 K1 和 K2, 并且 SPARSEMOV 具有四个操作数 K1、V1、K2 和 V2。在替代实施例中, RWMASKUPDATE 和 / 或 SPARSEMOV 的操作数中的一些可以是隐含的。

[0060] 图 4A 示出根据一个实施例的 RWMASKUPDATE 和 SPARSEMOV 指令的伪代码 401 和 402 的示例。在伪代码 401 和 402 中, KL 表示向量长度, 向量长度是每个向量寄存器 (例如, V1 和 V2 中的每一个) 中总的的数据元素数量。如果 zmm 寄存器用作具有 8 位数据元素的累加器, 则 $KL = 512/8 = 64$ 。伪代码 401 描述 RWMASKUPDATE 指令, 而伪代码 402 描述 SPARSEMOV 指令。应该注意, 处理器可以使用与伪代码 401 和 402 所示出的不同的操作或逻

辑来实现 RWMASKUPDATE 和 SPARSEMOV 指令。

[0061] RWMASKUPDATE 和 SPARSEMOV 指令分别更新掩码寄存器和在向量寄存器之间移动数据元素。可以执行附加指令以利用这些指令的结果来更高效地执行递归向量计算。图 4B 示出根据一个实施例的使用 RWMASKUPDATE 和 SPARSEMOV 指令的代码段 400 的示例。当由处理器执行时,代码段 400 使处理器在数组 X 的独立数据元素上执行递归向量计算。数组 X 可以存储在存储器、高速缓存或其它数据存储位置中。代码段 400 包括初始化部分 410、初始合并部分 420、后续合并部分 430、计算部分 440 和余数部分 450。以下参照图 5A 的流程图描述在部分 410-450 的每一个中的操作,图 5A 示出由处理器(例如如图 1 的指令处理装置 115)执行的方法 500 的实施例。

[0062] 在初始化部分 410,掩码寄存器 K1 和 K2 都被初始化为 0,指示在其相应的向量寄存器 V1 和 V2 中没有有用数据元素。术语“有用数据元素”表示需要计算的数据元素。在初始合并部分 420 开始迭代,其中首先检查 K2 以确定 V2 中是否剩余任何有用数据元素(框 531)。如果 V2 中不存在有用数据,从数组 X 将输入数据元素加载到 V2 中(框 532),并且相应地设置 K2 中的其相应掩码位。

[0063] 后续合并部分 430 应对其中 V2 包含有用数据元素的情况。有用数据元素可能由于先前上溢而剩余在 V2 中,或者可以在框 532 被加载到 V2 中。响应于 SPARSEMOV 指令 431,根据 K1 和 K2 中的掩码位,V2 中的这些有用数据元素可被移动到 V1 中(框 533)。

[0064] 响应于 RWMASKUPDATE 指令 433,在框 533 中的移动之后,更新掩码寄存器 K1 和 K2 以分别标识 V1 和 V2 中有用数据元素的当前位置(框 534)。

[0065] 在后续合并部分 430,执行第二 SPARSEMOV 指令 432 以存储数组 X 中从 V2 移动到 V1 的数据元素的索引(位置),使得计算结果被存储回数组 X 中其原始位置。

[0066] 计算部分 440 应对完整向量的向量计算(如相应掩码为全 1 所指示的;即,当 IsFullMask(K1) 为真时)。如果 V1 不具有完整的有用数据元素向量(框 535)并且存在未被加载到 V1 中的输入数据元素(框 538),这指示附加的输入数据元素可以通过 V2 被加载到 V1 中(框 532-534)。如果,V1 不具有完整向量并且没有其它输入数据元素要加载到 V1 中(框 538),这指示操作前进到余数部分 450,其中计算 V1 中的剩余数据元素直到计算完成并将结果存储回数组 X(框 539)。

[0067] 如果 V1 具有完整的有用数据元素向量(框 535),可以对 V1 执行向量计算(框 536)。如果 V1 中的任何数据元素不需要更多计算,则更新掩码寄存器 K1。继续向量计算直到 V1 中的一个或多个数据元素不需要更多计算(如 K1 中相应的零值掩码位所指示的);此时,将这些数据元素存储回数组 X(框 537)。在所示实施例中,可以使用 SCATTER 指令存储数据元素,并且可以使用函数 knot(K1) 来标识 K1 中的零值掩码位。除了 RWMASKUPDATE 和 SPARSEMOV 指令之外,可以由替代指令序列仿真在代码段 400 中使用的具体指令和函数,诸如 SCATTER、knot、IsFullMask 等等。

[0068] 重复框 531-537 的操作直到没有更多的输入数据元素要通过 V2 被加载到 V1 中(框 538);即,当数组 X 中的所有输入数据元素被加载到 V2 中并且 V2 中的所有有用数据元素被移动到 V1 中时。余数部分 450 在此时开始。此时,V1 可能不具有完整的有用数据元素向量,但是 V1 中的那些数据元素需要进一步计算。继续向量计算直到 V1 中的所有剩余数据元素达到所需数量的迭代(框 539)。此时,可以将 V1 中的计算结果存储回数组 X 中

(例如,使用 SCATTER 指令)(框 539)。

[0069] 图 5B 是根据一个实施例的用于执行 RWMASKUPDATE 指令的方法 510 的流程框图。方法 510 以处理器(例如,图 1 的指令处理装置 115)接收指定第一掩码寄存器和第二掩码寄存器的掩码更新指令开始(框 511)。处理器解码掩码更新指令(框 512)。响应于经解码的掩码更新指令,处理器执行操作,包括:反转第一掩码寄存器中给定数量的掩码位;例如,通过将这些掩码位从第一位值(例如 0)设置为第二位值(例如 1)(框 513);以及反转第二掩码寄存器中给定数量的掩码位;例如,通过将这些掩码位从第二位值(例如 1)设置成第一位值(例如 0)(框 514)。给定数量是第一掩码寄存器中具有第一位值的掩码位数量和第二掩码寄存器中具有第二位值的掩码位数量中较小的一个。在替代实施例中,第一位值可以是 1 并且第二位值可以是 0。

[0070] 图 5C 是根据一个实施例的用于执行 SPARSEMOV 指令的方法 520 的流程框图。方法 520 以处理器(例如,图 1 的指令处理装置 115)接收指定第一掩码寄存器、第二掩码寄存器、第一向量寄存器和第二向量寄存器的向量移动指令开始(框 521)。处理器解码向量移动操作(框 522)。响应于经解码的向量移动指令并且基于第一和第二掩码寄存器中的掩码位值,处理器用第二向量寄存器中给定数量的源数据元素替换第一向量寄存器中给定数量的目标数据元素(框 523)。在一个实施例中,每个源数据元素对应于第二掩码寄存器中具有第二位值(例如 1)的掩码位,并且其中每个目标数据元素对应于第一掩码寄存器中具有第一位值(例如 0)的掩码位。在替代实施例中,第一位值可以是 1 并且第二位值可以是 0。给定数量是第一掩码寄存器中具有第一位值的掩码位数量和第二掩码寄存器中具有第二位值的掩码位数量中较小的一个。

[0071] 在各实施例中,图 5A-C 的方法可由通用处理器、专用处理器(例如,图形处理器或数字信号处理器)、或另一种类型的数字逻辑设备或指令处理装置执行。在一些实施例中,图 5A-C 的方法可由图 1 的指令处理装置 115、或类似的处理器、装置或系统(诸如图 7-13 中所示的实施例)执行。而且,图 1 的指令处理装置 115 以及图 7-13 中所示的处理器、装置或系统可执行与图 5A-C 的方法的实施例相同、类似或不同的操作和方法的实施例。

[0072] 在一些实施例中,图 1 的指令处理装置 115 可结合指令转换器操作,该指令转换器将来自源指令集的指令转换到目标指令集。例如,指令转换器可以变换(例如使用静态二进制变换、包括动态编译的动态二进制变换)、变形、仿真或以其它方式将指令转换成将由核来处理的一个或多个其它指令。指令转换器可以用软件、硬件、固件、或其组合实现。指令转换器可以在处理器上、在处理器外、或者部分在处理器上且部分在处理器外。

[0073] 图 6 是对比根据本发明的实施例的软件指令转换器的使用的框图。在所示的实施例中,指令转换器是软件指令转换器,但作为替代,该指令转换器可以用软件、固件、硬件或其各种组合来实现。图 6 示出可以使用 x86 编译器 604 来编译利用高级语言 602 的程序,以生成可以由具有至少一个 x86 指令集核的处理器 616 原生执行的 x86 二进制代码 606。具有至少一个 x86 指令集核的处理器 616 表示任何处理器,这些处理器能通过兼容地执行或以其它方式处理以下内容来执行与具有至少一个 x86 指令集核的英特尔处理器基本相同的功能:1) 英特尔 x86 指令集核的指令集的本质部分,或 2) 目标为在具有至少一个 x86 指令集核的英特尔处理器上运行的应用或其它程序的目标代码版本,以便取得与具有至少一个 x86 指令集核的英特尔处理器基本相同的结果。x86 编译器 604 表示用于生成 x86 二进

制代码 606 (例如, 目标代码) 的编译器, 该二进制代码可通过或不通过附加的链接处理在具有至少一个 x86 指令集核的处理器 616 上执行。类似地, 图 6 示出可以使用替代的指令集编译器 608 来编译利用高级语言 602 的程序, 以生成可以由不具有至少一个 x86 指令集核的处理器 614 (例如具有执行加利福尼亚州桑尼维尔市的 MIPS 技术公司的 MIPS 指令集、和 / 或执行加利福尼亚州桑尼维尔市的 ARM 控股公司的 ARM 指令集的核的处理器) 原生执行的替代指令集二进制代码 610。指令转换器 612 被用来将 x86 二进制代码 606 转换成可以由不具有 x86 指令集核的处理器 614 原生执行的代码。该转换后的代码不大可能与替代性指令集二进制代码 610 相同, 因为能够这样做的指令转换器难以制造; 然而, 转换后的代码将完成一般操作并由来自替代指令集的指令构成。因此, 指令转换器 612 通过仿真、模拟或任何其它过程来表示允许不具有 x86 指令集处理器或核的处理器或其它电子设备执行 x86 二进制代码 606 的软件、固件、硬件或其组合。

[0074] 示例性核架构

[0075] 有序和无序核框图

[0076] 图 7A 是示出根据本发明的各实施例的示例性有序流水线和示例性的寄存器重命名的无序发布 / 执行流水线的框图。图 7B 是示出根据本发明的各实施例的要包括在处理器中的有序架构核的示例性实施例和示例性的寄存器重命名的无序发布 / 执行架构核的框图。图 7A 和 7B 中的实线框示出了有序流水线和有序核, 而可选增加的虚线框示出了寄存器重命名的、无序发布 / 执行流水线和核。给定有序方面是无序方面的子集的情况下, 将描述无序方面。

[0077] 在图 7A 中, 处理器流水线 700 包括取出级 702、长度解码级 704、解码级 706、分配级 708、重命名级 710、调度 (也称为分派或发布) 级 712、寄存器读取 / 存储器读取级 714、执行级 716、写回 / 存储器写入级 718、异常处理级 722 和提交级 724。

[0078] 图 7B 示出了包括耦合到执行引擎单元 750 的前端单元 730 的处理器核 790, 且执行引擎单元和前端单元两者都耦合到存储器单元 770。核 790 可以是精简指令集计算 (RISC) 核、复杂指令集计算 (CISC) 核、超长指令字 (VLIW) 核或混合或替代核类型。作为又一选项, 核 790 可以是专用核, 诸如例如网络或通信核、压缩引擎、协处理器核、通用计算图形处理器单元 (GPGPU) 核、或图形核等等。

[0079] 前端单元 730 包括耦合到指令高速缓存单元 734 的分支预测单元 732, 该指令高速缓存单元耦合到指令转换后备缓冲器 (TLB) 736, 该指令转换后备缓冲器耦合到指令取出单元 738, 指令取出单元耦合到解码单元 740。解码单元 740 (或解码器) 可解码指令, 并生成从原始指令解码出的、或以其它方式反映原始指令的、或从原始指令导出的一个或多个微操作、微代码进入点、微指令、其它指令、或其它控制信号作为输出。解码单元 740 可使用各种不同的机制来实现。合适的机制的示例包括但不限于查找表、硬件实现、可编程逻辑阵列 (PLA)、微代码只读存储器 (ROM) 等。在一个实施例中, 核 790 包括 (例如, 在解码单元 740 中或否则在前端单元 730 内的) 用于存储某些宏指令的微代码的微代码 ROM 或其它介质。解码单元 740 耦合至执行引擎单元 750 中的重命名 / 分配器单元 752。

[0080] 执行引擎单元 750 包括重命名 / 分配器单元 752, 该重命名 / 分配器单元耦合至引退单元 754 和一个或多个调度器单元 756 的集合。调度器单元 756 表示任何数目的不同调度器, 包括预留站、中央指令窗等。调度器单元 756 耦合到物理寄存器组单元 758。每个物

理寄存器组单元 758 表示一个或多个物理寄存器组,其中不同的物理寄存器组存储一种或多种不同的数据类型,诸如标量整数、标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点、状态(例如,作为要执行的下一指令的地址的指令指针)等。在一个实施例中,物理寄存器组单元 758 包括向量寄存器单元、写掩码寄存器单元和标量寄存器单元。这些寄存器单元可以提供架构向量寄存器、向量掩码寄存器、和通用寄存器。物理寄存器组单元 758 与引退单元 754 重叠以示出可以用来实现寄存器重命名和无序执行的各种方式(例如,使用重新排序缓冲器和引退寄存器组;使用将来的文件、历史缓冲器和引退寄存器组;使用寄存器映射和寄存器池等等)。引退单元 754 和物理寄存器组单元 758 耦合到执行群集 760。执行群集 760 包括一个或多个执行单元 762 的集合和一个或多个存储器访问单元 764 的集合。执行单元 762 可以对各种类型的数据(例如,标量浮点、紧缩整数、紧缩浮点、向量整型、向量浮点)执行各种操作(例如,移位、加法、减法、乘法)。尽管一些实施例可以包括专用于特定功能或功能集合的多个执行单元,但其它实施例可包括全部执行所有功能的仅一个执行单元或多个执行单元。调度器单元 756、物理寄存器组单元 758、执行群集 760 被示出为可能是复数个,因为某些实施例为某些数据/操作类型创建了诸个单独流水线(例如,均具有各自调度器单元、物理寄存器组单元和/或执行群集的标量整数流水线、标量浮点/紧缩整数/紧缩浮点/向量整数/向量浮点流水线、和/或存储器访问流水线,以及在单独的存储器访问流水线的情况下特定实施例被实现为仅仅该流水线的执行群集具有存储器访问单元 764)。还应当理解,在使用分开的流水线的情况下,这些流水线中的一个或多个可以为无序发布/执行,并且其余流水线可以为有序发布/执行。

[0081] 存储器访问单元 764 的集合耦合到存储器单元 770,该存储器单元包括耦合到数据高速缓存单元 774 的数据 TLB 单元 772,其中数据高速缓存单元耦合到二级(L2)高速缓存单元 776。在一个示例性实施例中,存储器访问单元 764 可包括加载单元、存储地址单元和存储数据单元,其中的每一个均耦合至存储器单元 770 中的数据 TLB 单元 772。指令高速缓存单元 734 还耦合到存储器单元 770 中的第二级(L2)高速缓存单元 776。L2 高速缓存单元 776 耦合到一个或多个其它级的高速缓存,并最终耦合到主存储器。

[0082] 作为示例,示例性寄存器重命名的、无序发布/执行核架构可以如下实现流水线 700:1) 指令取出 738 执行取出和长度解码级 702 和 704;2) 解码单元 740 执行解码级 706;3) 重命名/分配器单元 752 执行分配级 708 和重命名级 710;4) 调度器单元 756 执行调度级 712;5) 物理寄存器组单元 758 和存储器单元 770 执行寄存器读取/存储器读取级 714;执行群集 760 执行执行级 716;6) 存储器单元 770 和物理寄存器组单元 758 执行写回/存储器写入级 718;7) 各单元可牵涉到异常处理级 722;以及 8) 引退单元 754 和物理寄存器组单元 758 执行提交级 724。

[0083] 核 790 可支持一个或多个指令集(例如,x86 指令集(具有与较新版本一起添加的一些扩展);加利福尼亚州桑尼维尔市的 MIPS 技术公司的 MIPS 指令集;加利福尼亚州桑尼维尔市的 ARM 控股的 ARM 指令集(具有诸如 NEON 等可选附加扩展)),其中包括本文中描述的各指令。在一个实施例中,核 790 包括用于支持紧缩数据指令集扩展(例如 SSE、AVX1、AVX2 等等)的逻辑,由此允许许多多媒体应用所使用的操作利用紧缩数据来执行。

[0084] 应当理解,核可支持多线程化(执行两个或更多个并行的操作或线程的集合),并且可以按各种方式来完成该多线程化,此各种方式包括时分多线程化、同步多线程化(其

中单个物理核为物理核正在同步多线程化的各线程中的每一个线程提供逻辑核)、或其组合(例如,时分取出和解码以及此后诸如用Intel®超线程化技术来同步多线程化)。

[0085] 尽管在无序执行的上下文中描述了寄存器重命名,但应当理解,可以在有序架构中使用寄存器重命名。尽管所示出的处理器的实施例还包括分开的指令和数据高速缓存单元 734/774 以及共享 L2 高速缓存单元 776,但替代实施例可以具有用于指令和数据两者的单个内部高速缓存,诸如例如一级(L1)内部高速缓存或多个级别的内部高速缓存。在一些实施例中,该系统可包括内部高速缓存和在核和/或处理器外部的的外部高速缓存的组合。或者,所有高速缓存都可以在核和/或处理器的外部。

[0086] 具体的示例性有序核架构

[0087] 图 8A-B 示出了更具体的示例性有序核架构的框图,该核将是芯片中的若干逻辑块之一(包括相同类型和/或不同类型的其它核)。根据应用,这些逻辑块通过高带宽的互连网络(例如,环形网络)与一些固定的功能逻辑、存储器 I/O 接口和其它必要的 I/O 逻辑通信。

[0088] 图 8A 是根据本发明的各实施例的单个处理器核以及它与管芯上互连网络 802 的连接及其二级(L2)高速缓存的本地子集 804 的框图。在一个实施例中,指令解码器 800 支持具有紧缩数据指令集扩展的 x86 指令集。L1 高速缓存 806 允许对进入标量和向量单元中的高速缓存存储器的低等待时间访问。尽管在一个实施例中(为了简化设计),标量单元 808 和向量单元 810 使用分开的寄存器集合(分别为标量寄存器 812 和向量寄存器 814),并且在这些寄存器之间转移的数据被写入到存储器并随后从一级(L1)高速缓存 806 读回,但是本发明的替代实施例可以使用不同的方法(例如使用单个寄存器集合或包括允许数据在这两个寄存器组之间传输而无需被写入和读回的通信路径)。

[0089] L2 高速缓存的本地子集 804 是全局 L2 高速缓存的一部分,该全局 L2 高速缓存被划分成多个分开的本地子集,即每个处理器核一个本地子集。每个处理器核具有到其自己的 L2 高速缓存 804 的本地子集的直接访问路径。被处理器核读出的数据被存储在其 L2 高速缓存子集 804 中,并且可以与其它处理器核访问其自己的本地 L2 高速缓存子集并行地被快速访问。被处理器核写入的数据被存储在其自己的 L2 高速缓存子集 804 中,并在必要的情况下从其它子集清除。环形网络确保共享数据的一致性。环形网络是双向的,以允许诸如处理器核、L2 高速缓存和其它逻辑块之类的代理在芯片内彼此通信。每个环形数据路径为每个方向 1012 位宽。

[0090] 图 8B 是根据本发明的各实施例的图 8A 中的处理器核的一部分的展开图。图 8B 包括 L1 高速缓存 804 的 L1 数据高速缓存 806A 部分,以及关于向量单元 810 和向量寄存器 814 的更多细节。具体地说,向量单元 810 是 16 宽向量处理单元(VPU)(见 16 宽 ALU 828),该单元执行整型、单精度浮点以及双精度浮点指令中的一个或多个。该 VPU 通过混合单元 820 支持对寄存器输入的混合、通过数值转换单元 822A-B 支持数值转换、并通过复制单元 824 支持对存储器输入的复制。写掩码寄存器 826 允许断言所得的向量写入。

[0091] 具有集成存储器控制器和图形器件的处理器

[0092] 图 9 是根据本发明的各实施例可能具有多于一个核、可能具有集成存储器控制器、以及可能具有集成图形器件的处理器 900 的框图。图 9 中的实线框示出具有单个核 902A、系统代理 910、一个或多个总线控制器单元 916 的集合的处理器 900,而虚线框的可

选附加示出具有多个核 902A-N、系统代理单元 910 中的一个或多个集成存储器控制器单元 914 的集合以及专用逻辑 908 的替代处理器 900。

[0093] 因此,处理器 900 的不同实现可包括:1)CPU,其中专用逻辑 908 是集成图形和/或科学(吞吐量)逻辑(其可包括一个或多个核),并且核 902A-N 是一个或多个通用核(例如,通用的有序核、通用的无序核、这两者的组合);2)协处理器,其中核 902A-N 是旨在主要用于图形和/或科学(吞吐量)的多个专用核;以及3)协处理器,其中核 902A-N 是多个通用有序核。因此,处理器 900 可以是通用处理器、协处理器或专用处理器,诸如例如网络或通信处理器、压缩引擎、图形处理器、GPGPU(通用图形处理单元)、高吞吐量的集成众核(MIC)协处理器(包括30个或更多核)、或嵌入式处理器等。该处理器可以被实现在一个或多个芯片上。处理器 900 可以是一个或多个衬底的一部分,和/或可以使用诸如例如 BiCMOS、CMOS 或 NMOS 等的多个加工技术中的任何一个技术将该处理器实现在一个或多个衬底上。

[0094] 存储器层次结构包括在各核内的一个或多个级别的高速缓存、一个或多个共享高速缓存单元 906 的集合、以及耦合至集成存储器控制器单元 914 的集合的外部存储器(未示出)。该共享高速缓存单元 906 的集合可以包括一个或多个中间级高速缓存,诸如二级(L2)、三级(L3)、四级(L4)或其它级别的高速缓存、未级高速缓存(LLC)、和/或其组合。尽管在一个实施例中,基于环的互连单元 912 将集成图形逻辑 908、共享高速缓存单元 906 的集合以及系统代理单元 910/集成存储器控制器单元 914 互连,但替代实施例可使用任何数量的公知技术来将这些单元互连。在一个实施例中,可以维护一个或多个高速缓存单元 906 和核 902-A-N 之间的一致性(coherency)。

[0095] 在一些实施例中,核 902A-N 中的一个或多个核能够多线程化。系统代理 910 包括协调和操作核 902A-N 的那些组件。系统代理单元 910 可包括例如功率控制单元(PCU)和显示单元。PCU 可以是或包括用于调整核 902A-N 和集成图形逻辑 908 的功率状态所需的逻辑和组件。显示单元用于驱动一个或多个外部连接的显示器。

[0096] 核 902A-N 在架构指令集方面可以是同构的或异构的;即,这些核 902A-N 中的两个或更多个核可能能够执行相同的指令集,而其它核可能能够执行该指令集的仅仅子集或不同的指令集。

[0097] 示例性计算机架构

[0098] 图 10-13 是示例性计算机架构的框图。本领域已知的对膝上型设备、台式机、手持 PC、个人数字助理、工程工作站、服务器、网络设备、网络集线器、交换机、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的其它系统设计和配置也是合适的。一般地,能够包含本文中所公开的处理器和/或其它执行逻辑的多个系统和电子设备一般都是合适的。

[0099] 现在参考图 10,所示出的是根据本发明一个实施例的系统 1000 的框图。系统 1000 可以包括一个或多个处理器 1010、1015,这些处理器耦合到控制器中枢 1020。在一个实施例中,控制器中枢 1020 包括图形存储器控制器中枢(GMCH)1090 和输入/输出中枢(IOH)1050(其可以在分开的芯片上);GMCH 1090 包括存储器和图形控制器,存储器 1040 和协处理器 1045 耦合到该存储器和图形控制器;IOH 1050 将输入/输出(I/O)设备 1060 耦合到 GMCH 1090。或者,存储器和图形控制器中的一个或两者被集成在处理器内(如本文

中所描述的), 存储器 1040 和协处理器 1045 直接耦合到处理器 1010 以及控制器中枢 1020, 该控制器中枢与 IOH 1050 处于单个芯片中。

[0100] 附加处理器 1015 的任选性质用虚线表示在图 10 中。每一处理器 1010、1015 可包括本文中描述的处理核中的一个或多个, 并且可以是处理器 900 的某一版本。

[0101] 存储器 1040 可以是例如动态随机存取存储器 (DRAM)、相变存储器 (PCM) 或这两者的组合。对于至少一个实施例, 控制器中枢 1020 经由诸如前端总线 (FSB) 之类的多分支总线、诸如快速通道互连 (QPI) 之类的点对点接口、或者类似的连接 1095 与处理器 1010、1015 进行通信。

[0102] 在一个实施例中, 协处理器 1045 是专用处理器, 诸如例如高吞吐量 MIC 处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、或嵌入式处理器等等。在一个实施例中, 控制器中枢 1020 可以包括集成图形加速器。

[0103] 在物理资源 1010、1015 之间会存在包括架构、微架构、热、和功耗特征等的一系列品质度量方面的各种差异。

[0104] 在一个实施例中, 处理器 1010 执行控制一般类型的数据处理操作的指令。协处理器指令可嵌入在这些指令中。处理器 1010 将这些协处理器指令识别为应当由附连的协处理器 1045 执行的类型。因此, 处理器 1010 在协处理器总线或者其它互连上将协处理器指令 (或者表示协处理器指令的控制信号) 发布到协处理器 1045。协处理器 1045 接受并执行所接收的协处理器指令。

[0105] 现在参考图 11, 所示为根据本发明的一实施例的更具体的第一示例性系统 1100 的框图。如图 11 所示, 多处理器系统 1100 是点对点互连系统, 并包括经由点对点互连 1150 耦合的第一处理器 1170 和第二处理器 1180。处理器 1170 和 1180 中的每一个都可以是处理器 900 的某一版本。在本发明的一个实施例中, 处理器 1170 和 1180 分别是处理器 1010 和 1015, 而协处理器 1138 是协处理器 1045。在另一实施例中, 处理器 1170 和 1180 分别是处理器 1010 和协处理器 1045。

[0106] 处理器 1170 和 1180 被示为分别包括集成存储器控制器 (IMC) 单元 1172 和 1182。处理器 1170 还包括作为其总线控制器单元的一部分的点对点 (P-P) 接口 1176 和 1178; 类似地, 第二处理器 1180 包括点对点接口 1186 和 1188。处理器 1170、1180 可以使用点对点 (P-P) 接口电路 1178、1188 经由 P-P 接口 1150 来交换信息。如图 11 所示, IMC 1172 和 1182 将各处理器耦合至相应的存储器, 即存储器 1132 和存储器 1134, 这些存储器可以是本地附连至相应的处理器的主存储器的部分。

[0107] 处理器 1170、1180 可各自经由使用点对点接口电路 1176、1194、1186、1198 的各个 P-P 接口 1152、1154 与芯片组 1190 交换信息。芯片组 1190 可以可选地经由高性能接口 1139 与协处理器 1138 交换信息。在一个实施例中, 协处理器 1138 是专用处理器, 诸如例如高吞吐量 MIC 处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、或嵌入式处理器等等。

[0108] 共享高速缓存 (未示出) 可以被包括在任一处理器之内, 或被包括在两个处理器外部但仍经由 P-P 互连与这些处理器连接, 从而如果将某处理器置于低功率模式时, 可将任一处理器或两个处理器的本地高速缓存信息存储在该共享高速缓存中。

[0109] 芯片组 1190 可经由接口 1196 耦合至第一总线 1116。在一个实施例中, 第一总线 1116 可以是外围组件互连 (PCI) 总线, 或诸如 PCI Express 总线或其它第三代 I/O 互连总

线之类的总线,但本发明的范围并不受此限制。

[0110] 如图 11 所示,各种 I/O 设备 1114 可以连同总线桥 1118 耦合到第一总线 1116,该总线桥将第一总线 1116 耦合至第二总线 1120。在一个实施例中,诸如协处理器、高吞吐量 MIC 处理器、GPGPU 的处理器、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器的一个或多个附加处理器 1115 耦合到第一总线 1116。在一个实施例中,第二总线 1120 可以是低引脚计数(LPC)总线。各种设备可以被耦合至第二总线 1120,在一个实施例中这些设备包括例如键盘/鼠标 1122、通信设备 1127 以及诸如可包括指令/代码和数据 1130 的盘驱动器或其它大容量存储设备的存储单元 1128。此外,音频 I/O 1124 可以被耦合至第二总线 1120。注意,其它架构是可能的。例如,代替图 11 的点对点架构,系统可以实现多分支总线或其它这类架构。

[0111] 现在参考图 12,所示为根据本发明的实施例的更具体的第二示例性系统 1200 的框图。图 11 和图 12 中的相同部件用相同附图标记表示,并从图 12 中省去了图 11 中的某些方面,以避免使图 12 的其它方面变得模糊。

[0112] 图 12 示出处理器 1170、1180 可分别包括集成存储器和 I/O 控制逻辑("CL") 1172 和 1182。因此,CL 1172、1182 包括集成存储器控制器单元并包括 I/O 控制逻辑。图 12 不仅示出存储器 1132、1134 耦合至 CL 1172、1182,而且还示出 I/O 设备 1214 也耦合至控制逻辑 1172、1182。传统 I/O 设备 1215 被耦合至芯片组 1190。

[0113] 现在参照图 13,所示出的是根据本发明一个实施例的 SoC 1300 的框图。在图 9 中,相似的部件具有同样的附图标记。另外,虚线框是更先进的 SoC 的可选特征。在图 13 中,互连单元 1302 被耦合至:应用处理器 1310,该应用处理器包括一个或多个核 202A-N 的集合以及共享高速缓存单元 906;系统代理单元 910;总线控制器单元 916;集成存储器控制器单元 914;一组或一个或多个协处理器 1320,其可包括集成图形逻辑、图像处理、音频处理器和视频处理器;静态随机存取存储器(SRAM)单元 1330;直接存储器存取(DMA)单元 1332;以及用于耦合至一个或多个外部显示器的显示单元 1340。在一个实施例中,协处理器 1320 包括专用处理器,诸如例如网络或通信处理器、压缩引擎、GPGPU、高吞吐量 MIC 处理器、或嵌入式处理器等等。

[0114] 本文公开的机制的各实施例可以被实现在硬件、软件、固件或这些实现方法的组合中。本发明的实施例可实现为在可编程系统上执行的计算机程序或程序代码,该可编程系统包括至少一个处理器、存储系统(包括易失性和非易失性存储器和/或存储元件)、至少一个输入设备以及至少一个输出设备。

[0115] 可将程序代码(诸如图 11 中示出的代码 1130)应用于输入指令,以执行本文描述的各项功能并生成输出信息。可以按已知方式将输出信息应用于一个或多个输出设备。为了本申请的目的,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何系统。

[0116] 程序代码可以用高级程序化语言或面向对象的编程语言来实现,以便与处理系统通信。在需要时,也可用汇编语言或机器语言来实现程序代码。事实上,本文中描述的机制不限于任何特定编程语言的范围。在任一情形下,该语言可以是编译语言或解释语言。

[0117] 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表示性指令来实现,指令表示处理器中的各种逻辑,指令在被机器读取时使得该机器制作用于执行本

文所述的技术的逻辑。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,并被提供给多个客户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。

[0118] 这样的机器可读存储介质可以包括但不限于通过机器或设备制造或形成的物品的非瞬态的有形安排,其包括存储介质,诸如:硬盘;任何其它类型的盘,包括软盘、光盘、紧致盘只读存储器(CD-ROM)、紧致盘可重写(CD-RW)以及磁光盘;半导体器件,例如只读存储器(ROM)、诸如动态随机存取存储器(DRAM)和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);相变存储器(PCM);磁卡或光卡;或适于存储电子指令的任何其它类型的介质。

[0119] 因此,本发明的各实施例还包括非瞬态的有形机器可读介质,该介质包含指令或包含设计数据,诸如硬件描述语言(HDL),它定义本文中描述的结构、电路、装置、处理器和/或系统特征。这些实施例也被称为程序产品。

[0120] 虽然已经描述并在附图中示出了特定示例实施例,但可以理解,这些实施例仅仅是对本宽泛发明的说明而非限制,并且本发明不限于所示出和所描述的特定结构和配置,因为本领域技术人员在研究了本公开文本之后可以料知到多种其它修改方式。在诸如本申请这样的技术领域,因为发展很快且未来的进步难以预见,所以本公开的诸个实施例可通过受益于技术进步而容易地获得配置和细节上的改动,而不背离本公开的原理和所附权利要求书的范围。

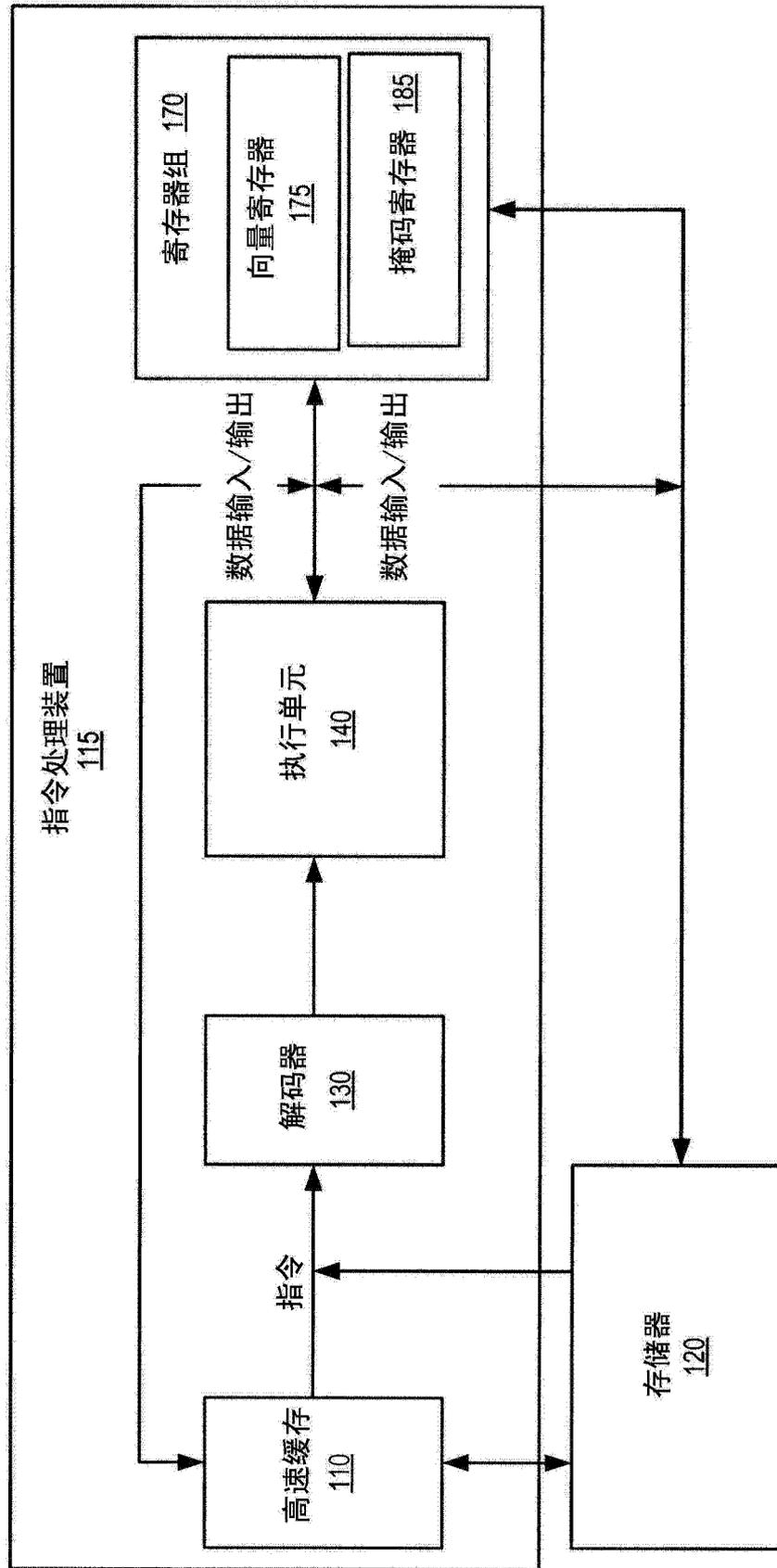


图 1

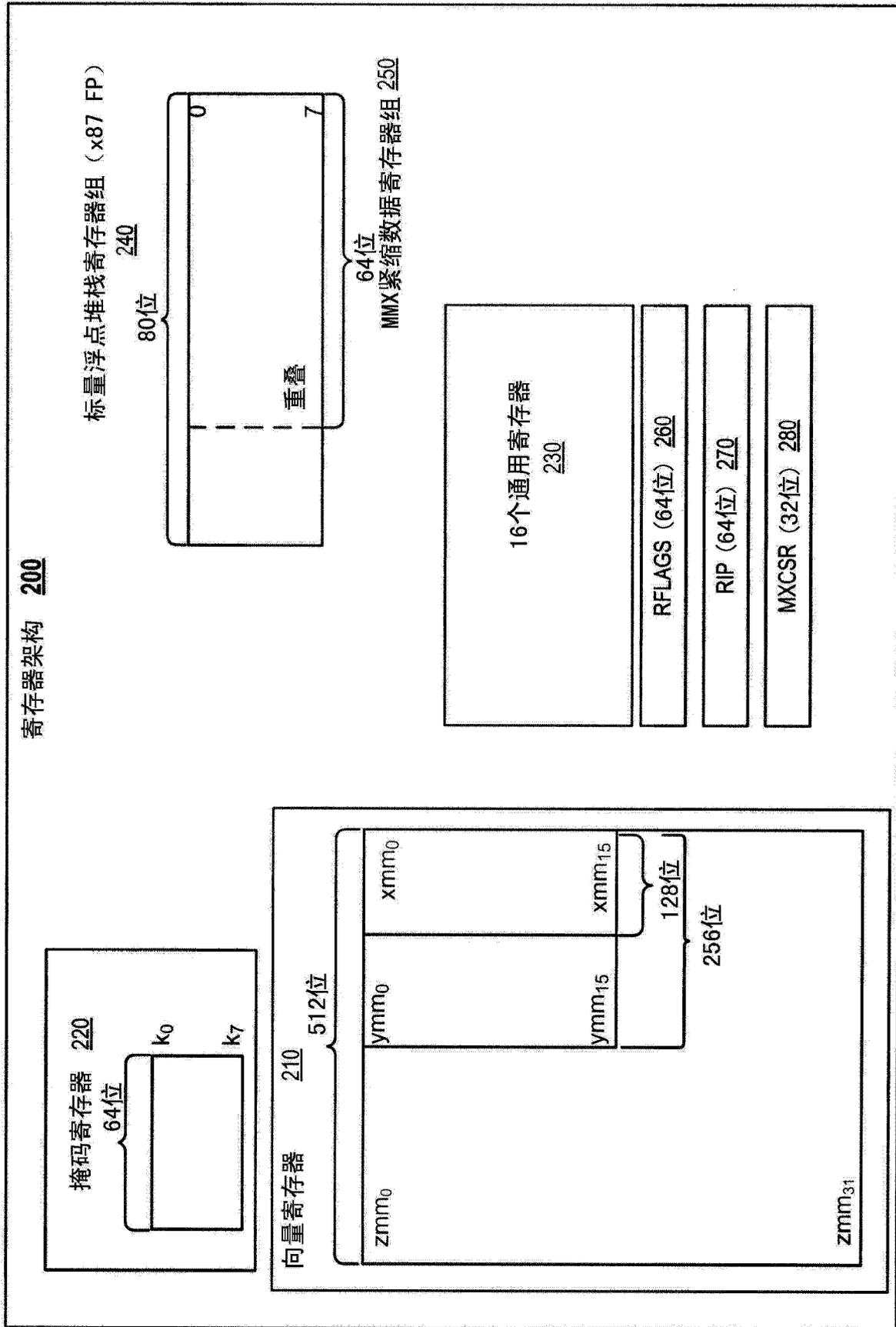


图 2

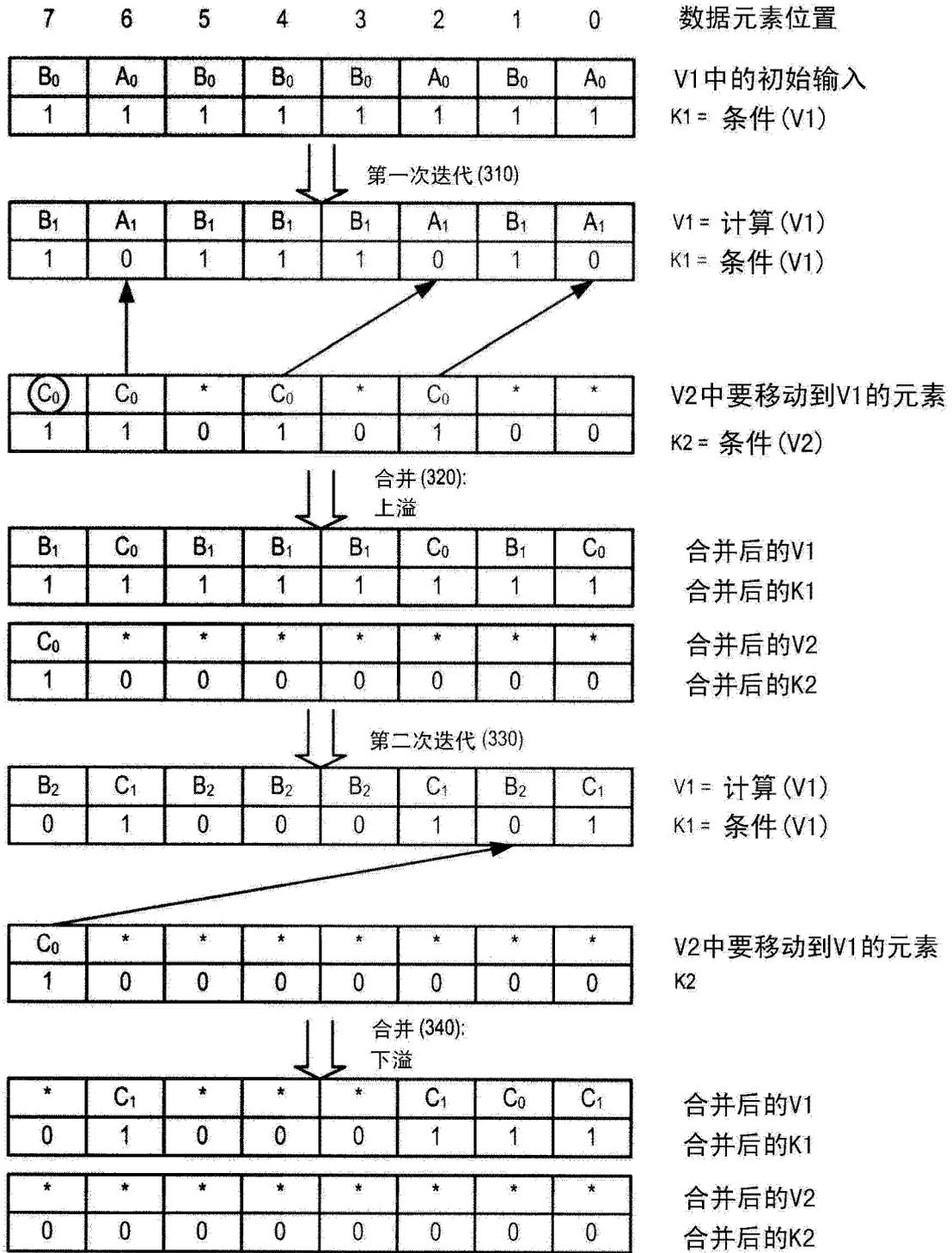


图 3

```
401
rwmaskupdate(K1,K2)
for (i = 0, k = 0; i < KL; i++){
    if (!K1[i]){
        for (; k < KL; k++){
            if (K2[k]){
                K2[k] = 0;
                K1[i] = 1;
                k++;
                break;
            }
        }
    }
}

402
sparsemov(K1,V1,K2,V2)
for (i = 0, k = 0; i < KL; i++){
    if (!K1[i]){
        for (; k < KL; k++){
            if (K2[k]){
                V1[i] = V2[k];
                k++;
                break;
            }
        }
    }
}
```

图 4A

400
↙

```

i = 0; // 初始化循环计数器
v_index = -1:-2:...:-KL+1:-KL // 初始索引向量
v_KL = KL:KL:...:KL:KL // 对于索引向量的递增
K1 = 0; // 累加器最初为空
K2 = 0; // 还没有上溢
do{
  if (K2 == 0){ // 如果没有由于先前上溢剩余的元素
    V2 = vector_load(X[i+KL-1:i]); // 加载X数组的新的KL个元素
    K2 = condition(V2); // 为新元素生成读掩码
    i += KL; // 递增循环计数器
    v_index += v_KL; // 递增索引向量
  } // 否则以读掩码K2继续
431 sparsemov(K1, V1, K2, V2); // 向V1添加新的元素...
432 sparsemov(K1, V3, K2, v_index); // 并且向V3添加其索引
433 rwmaskupdate(K1,K2); // 更新读和写掩码
} while((i < N) || (K2 != 0)); // 当V2中没有更多输入流或新数...
// 据时继续
// 开始余数计算
K3 = K1; // 存储用于最终分散的余数掩码
do{
  V1{K1} = computation(V1); // 在掩码K1下计算
  K1 = condition(V1); // 计算后检查条件
}while(K1 != 0) // 检查是否还剩余需要处理的
scatter(K3,V1,V3,X); // 用索引V3从V1向X数组分散剩余...
// 元素

```

图 4B

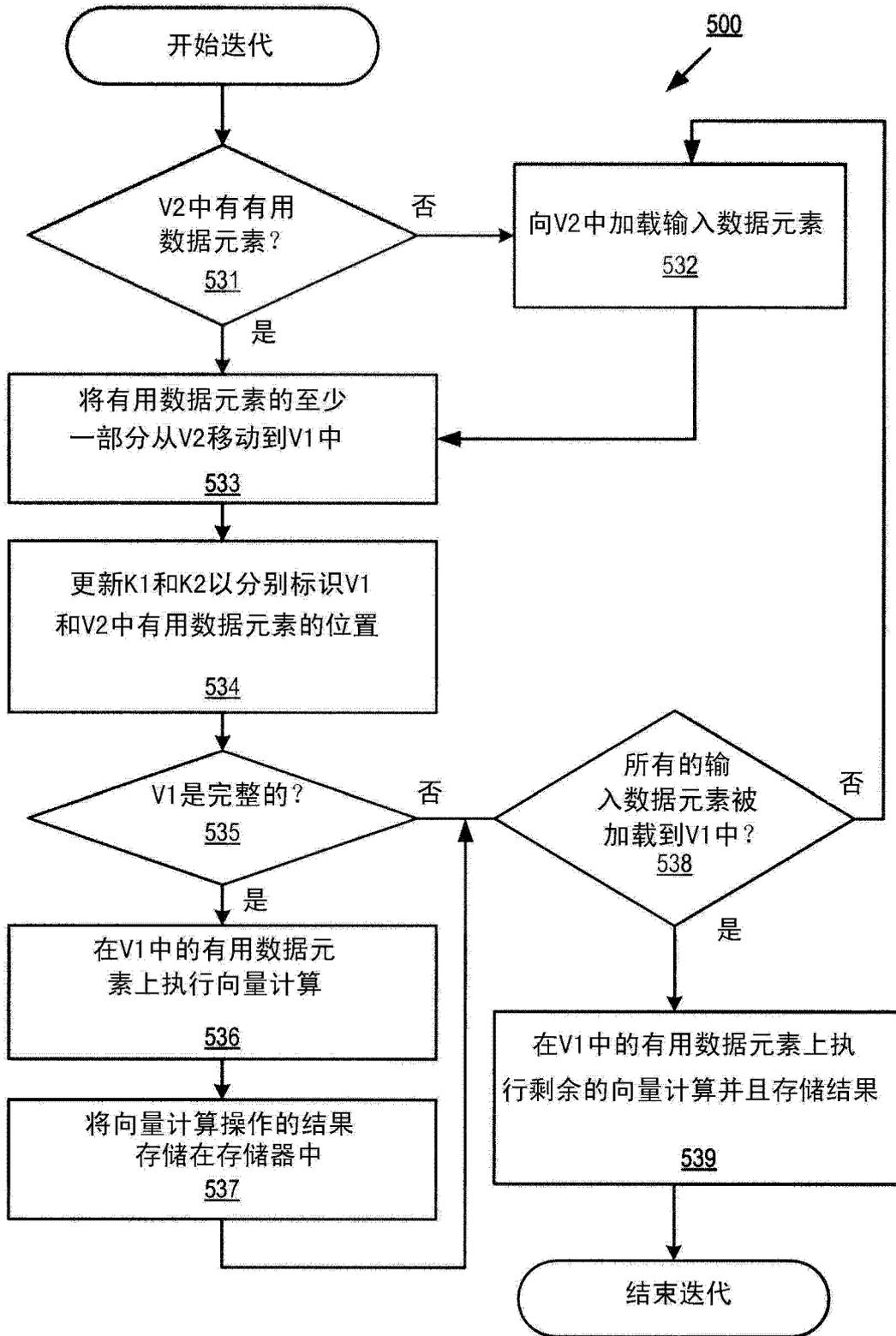


图 5A

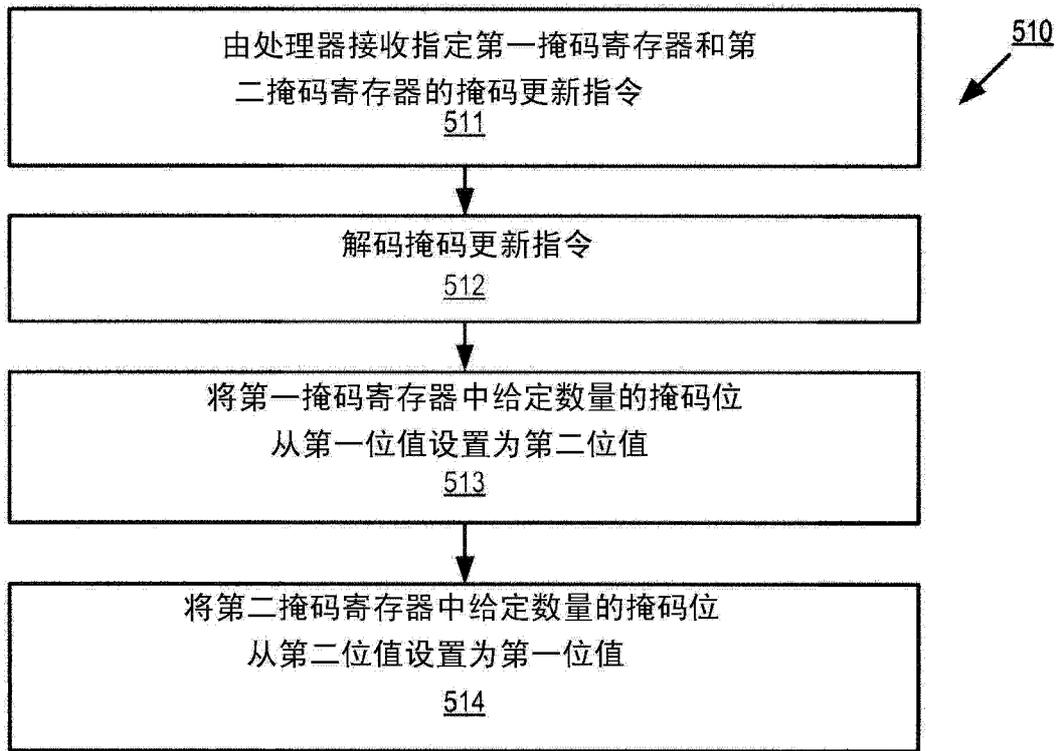


图 5B

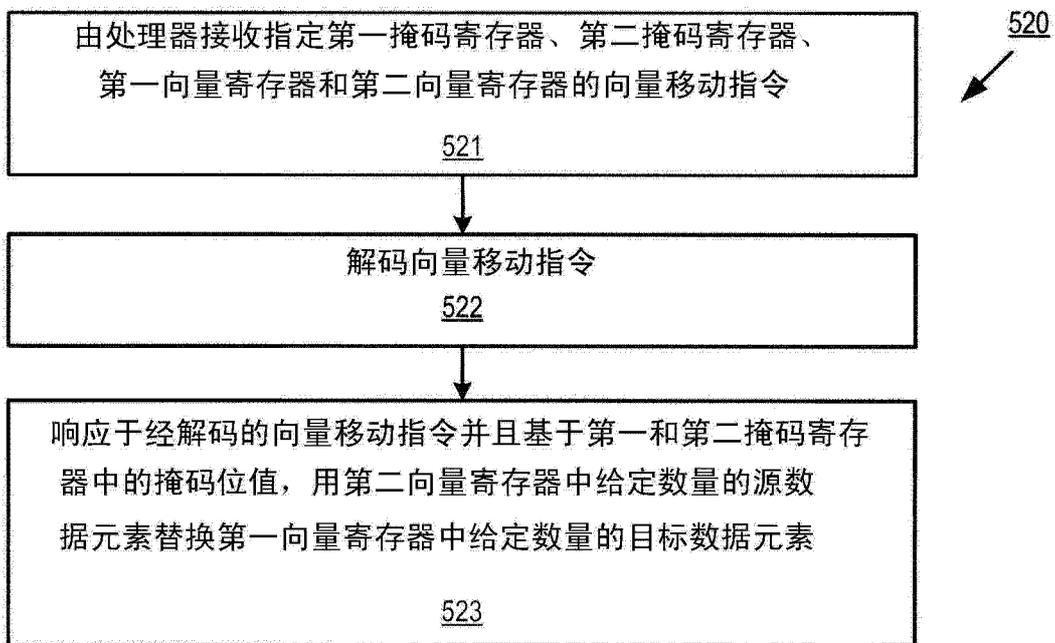


图 5C

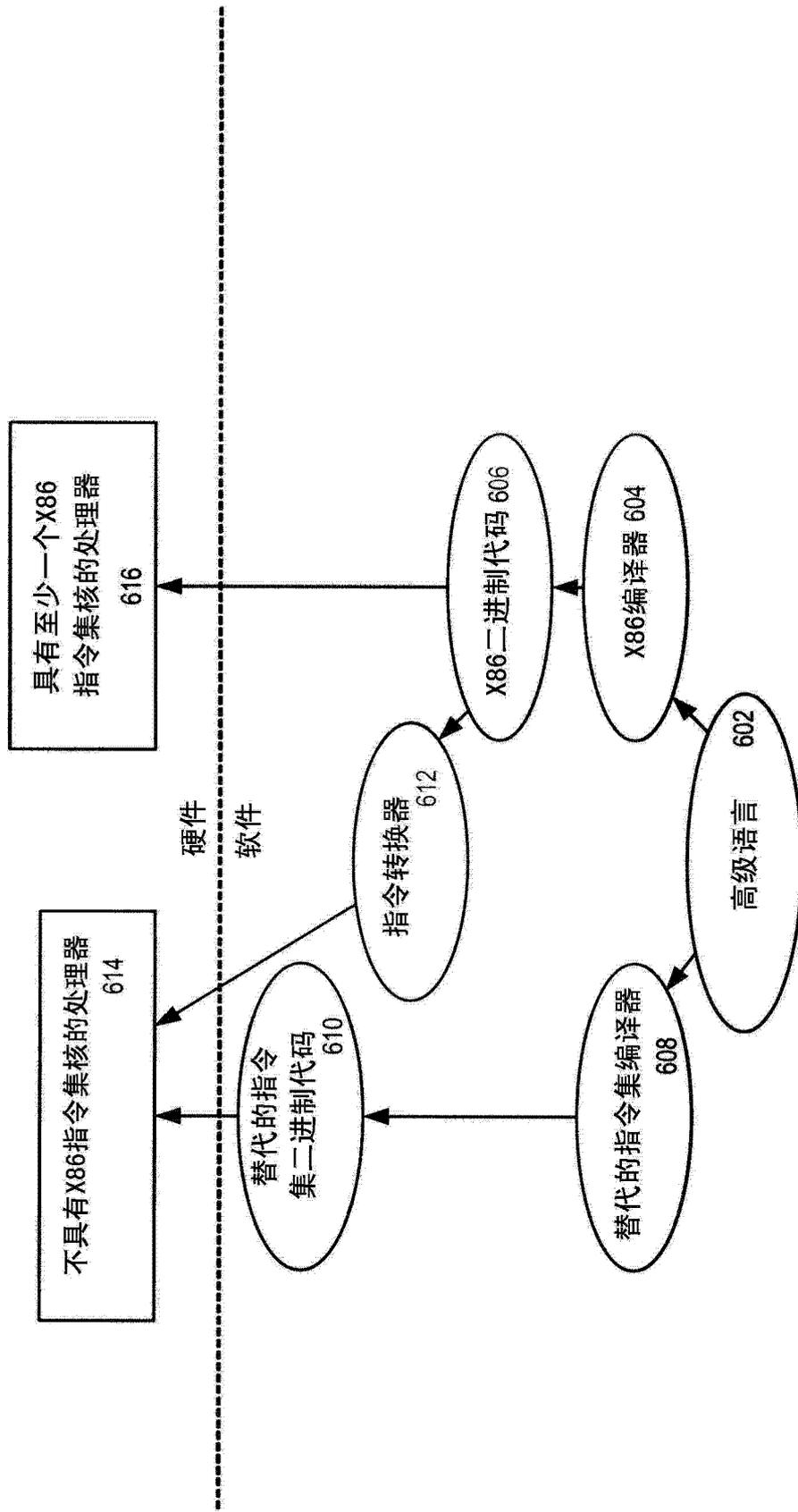
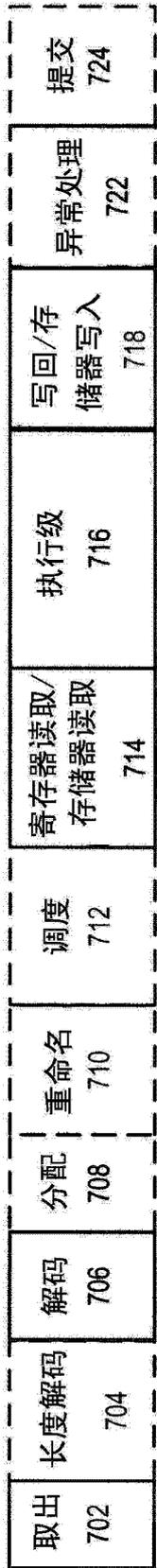
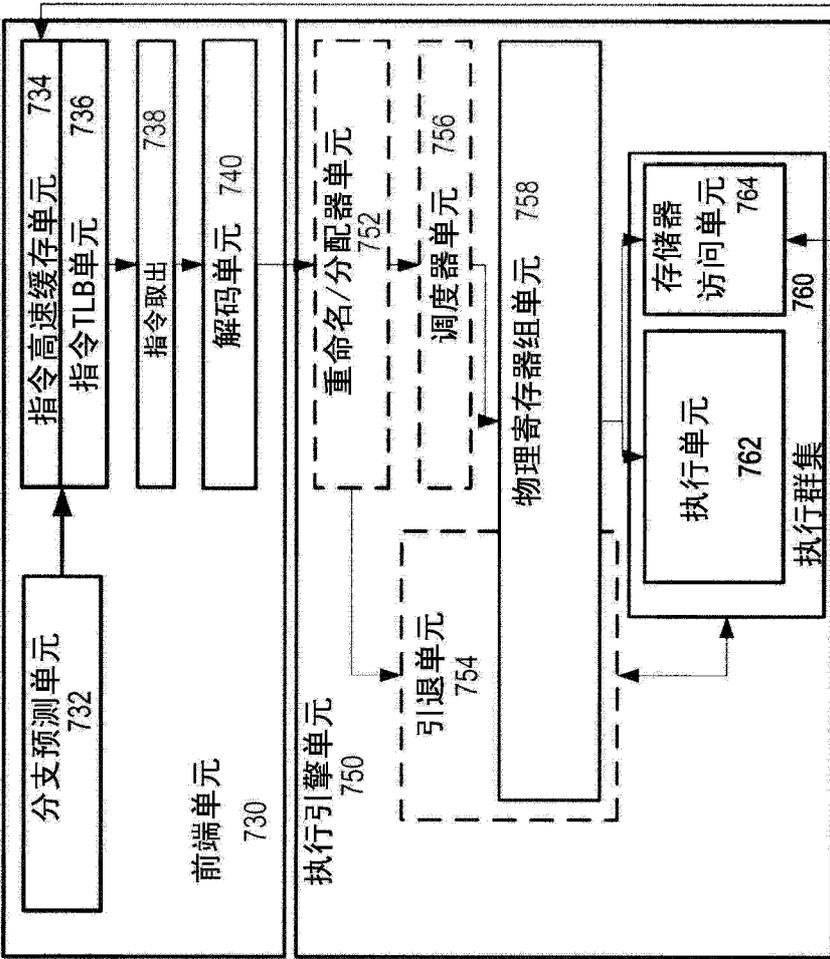


图 6



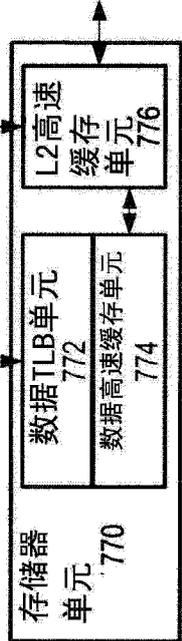
7A

图



7B

图



流水线 700

核 790

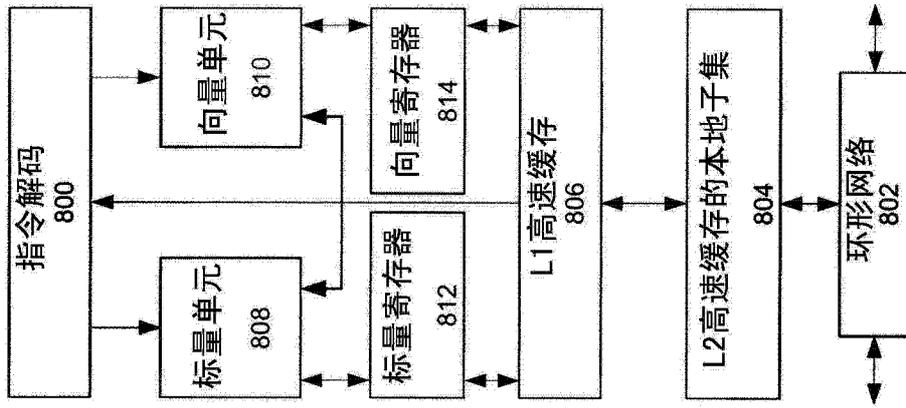


图 8A

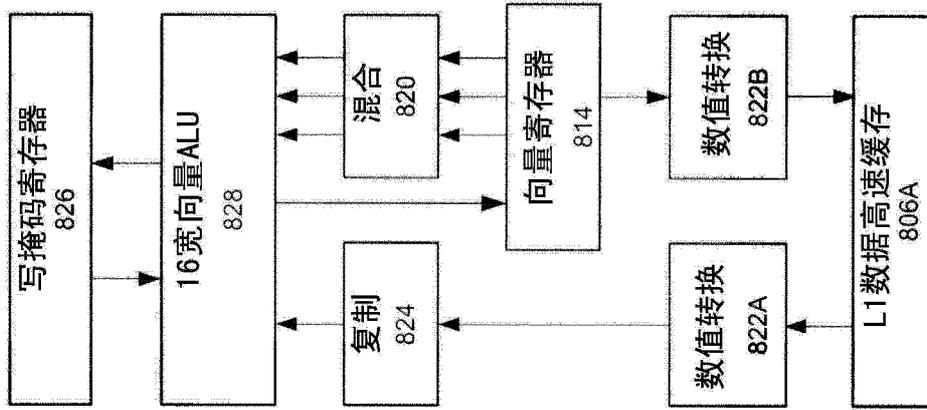


图 8B

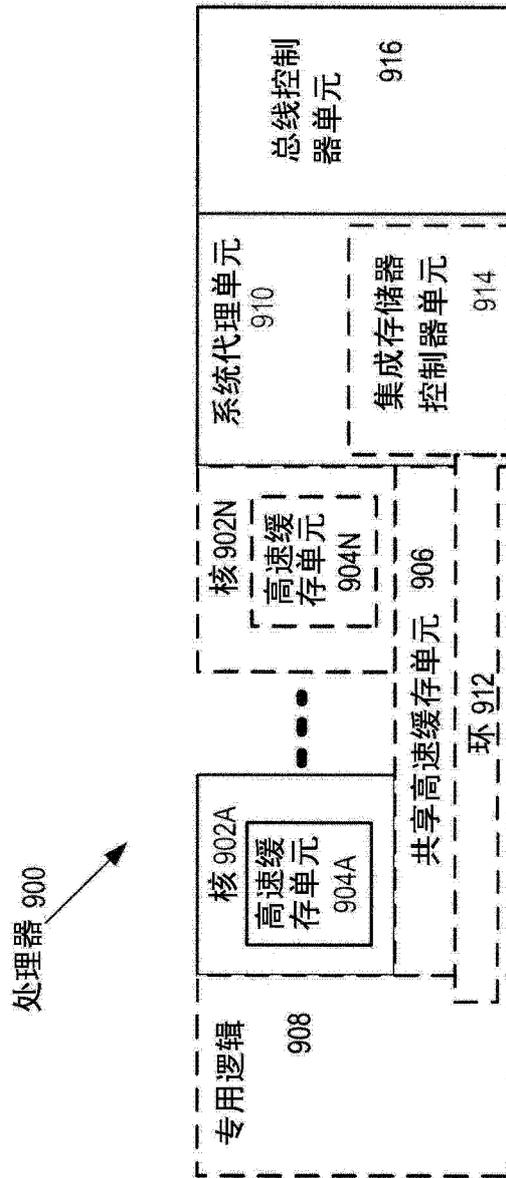


图 9

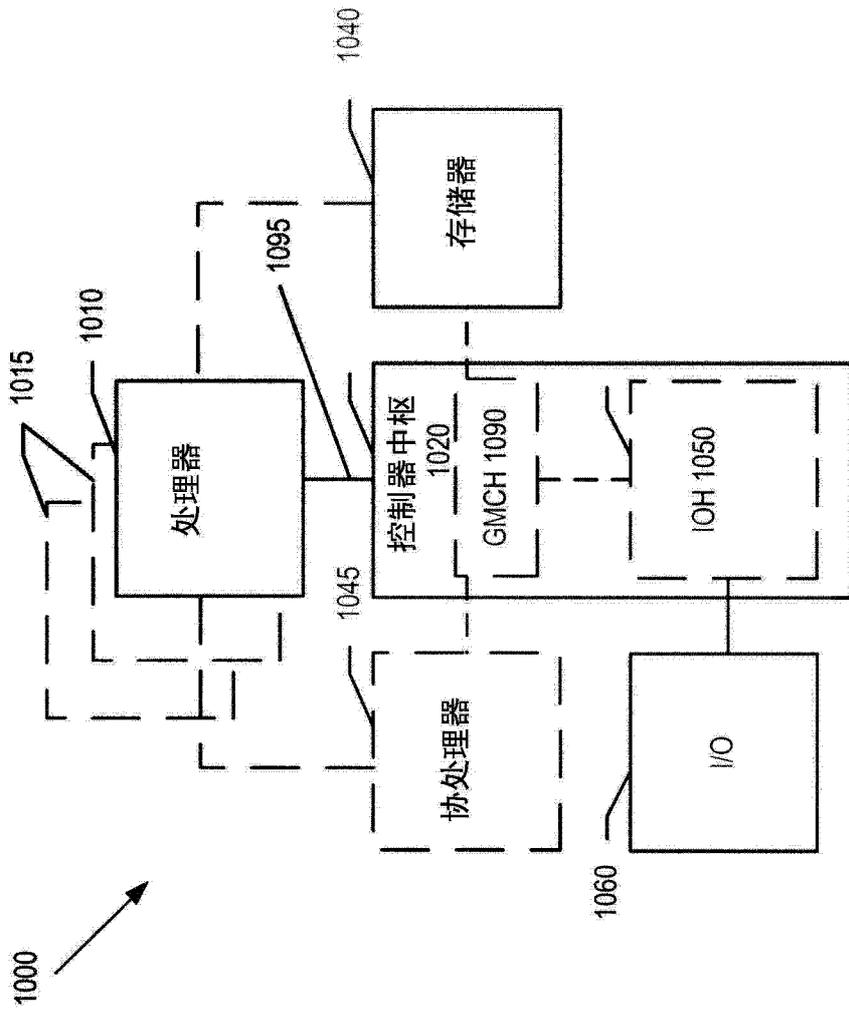


图 10

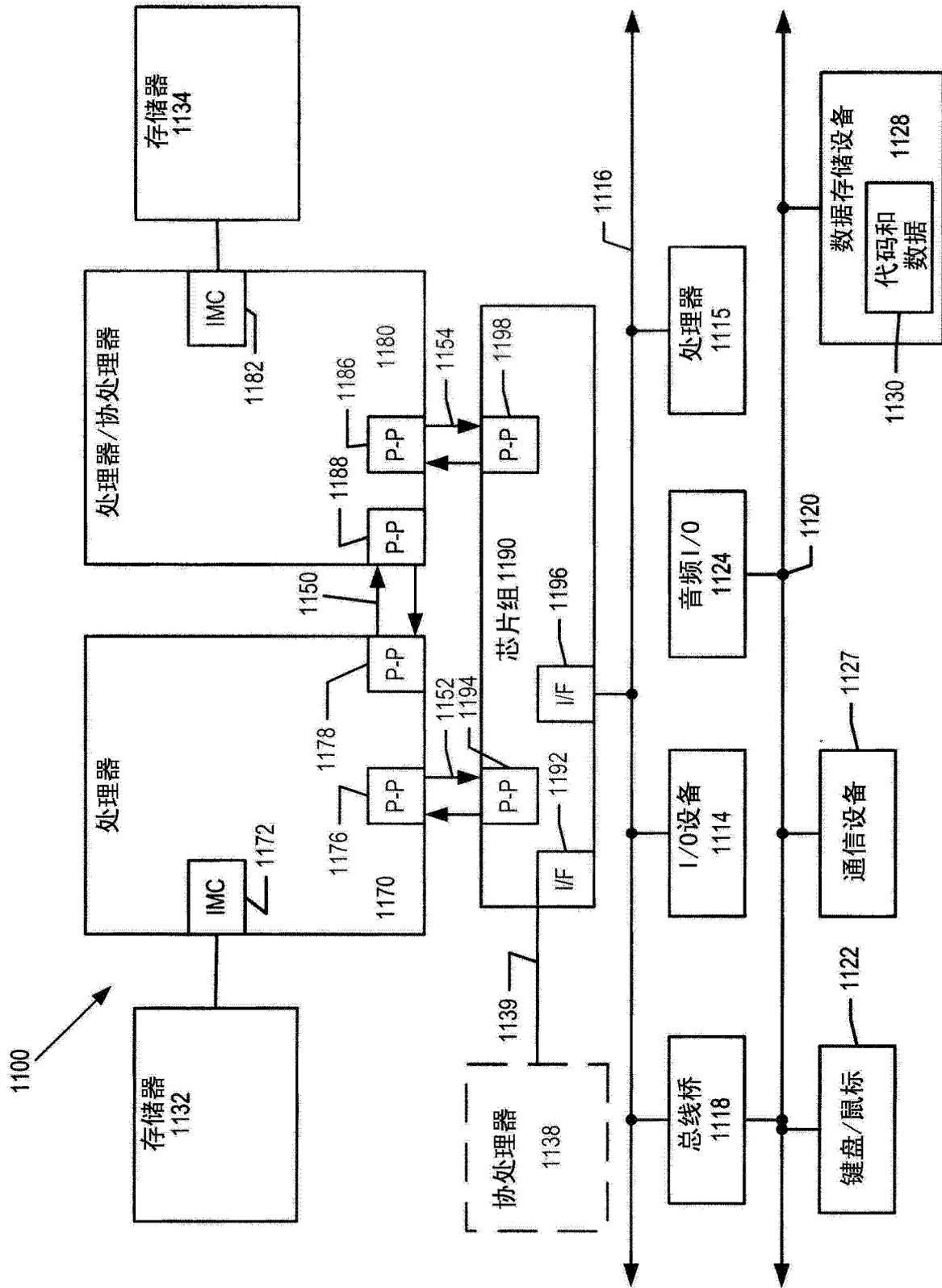


图 11

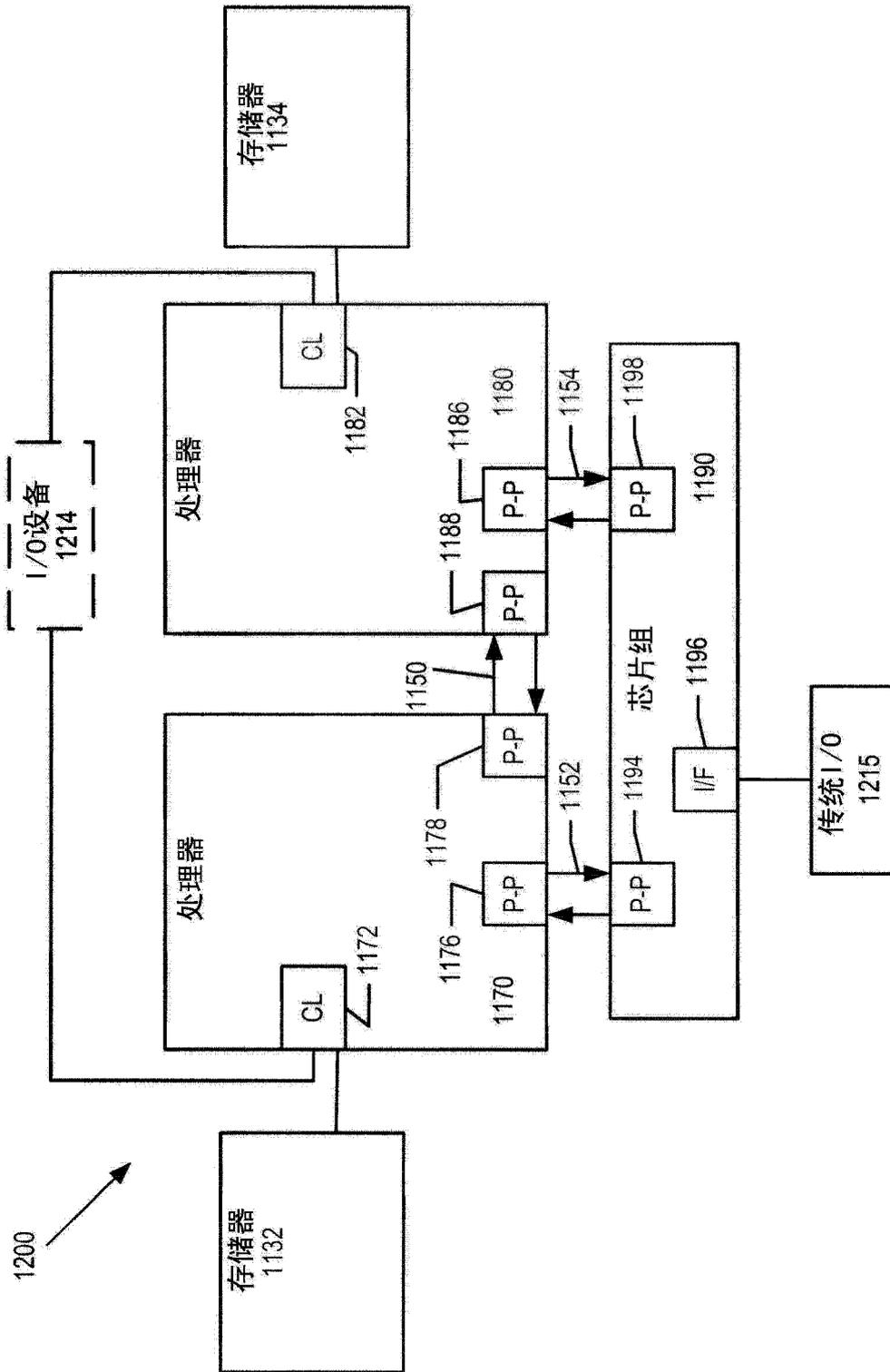


图 12

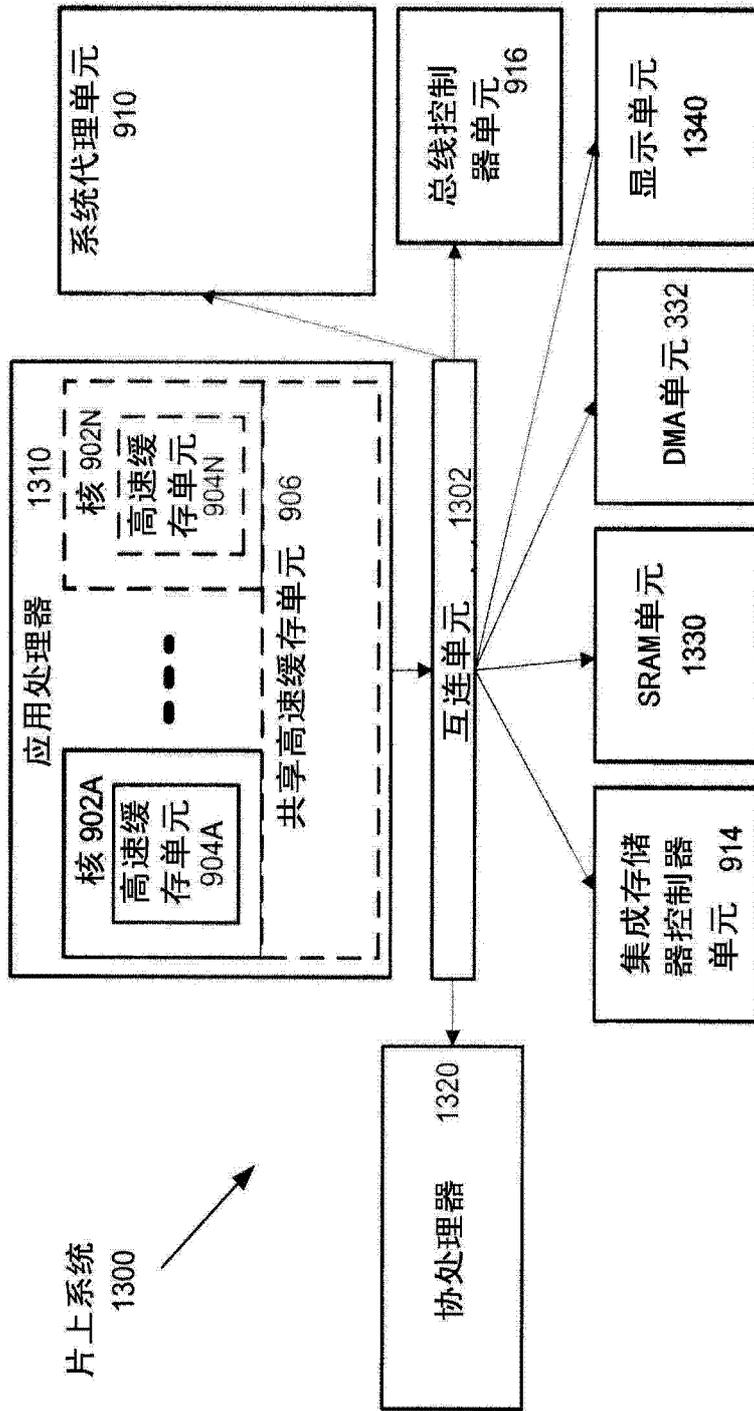


图 13