(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 110214317 B
(45)授权公告日 2023.05.02

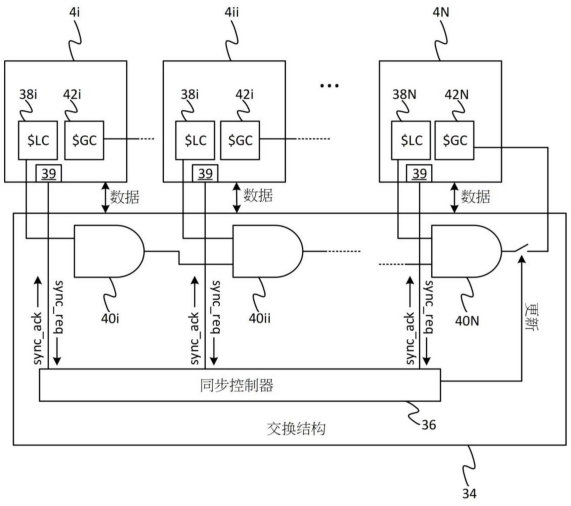
(21)申请号 201880004197.2
(22)申请日 2018.10.11
(65)同一申请的已公布的文献号
申请公布号 CN 110214317 A
(43)申请公布日 2019.09.06
(30)优先权数据
1717291.7 2017.10.20 GB
(85)PCT国际申请进入国家阶段日
2019.05.06
(86)PCT国际申请的申请数据
PCT/EP2018/077674 2018.10.11
(87)PCT国际申请的公布数据
WO2019/076714 EN 2019.04.25
(73)专利权人 图核有限公司
地址 英国布里斯托
(72)发明人 西蒙·克里斯蒂安·诺尔斯
艾伦·格雷汉姆·亚历山大
(74)专利代理机构 深圳鹰翅知识产权代理有限公司 44658
专利代理师 周婧 黄幸兒
(51)Int.Cl.
G06F 15/173(2006.01)
G06F 15/80(2006.01)
G06F 9/30(2006.01)
G06F 9/38(2006.01)
G06F 9/46(2006.01)
G06F 9/52(2006.01)
(56)对比文件
US 2005021567 A1,2005.01.27
US 2011219208 A1,2011.09.08
US 2017277567 A1,2017.09.28
审查员 何相甫
权利要求书3页 说明书26页 附图12页

(54)发明名称

多瓦片处理布置中的同步

(57)摘要

一种处理系统,包括多个瓦片和瓦片之间的互连。互连用于根据批量同步并行方案在一些或所有瓦片的组之间通信,由此组中的每个瓦片执行瓦片上计算阶段,接着是瓦片间交换阶段,不进行交换阶段,直至组中的所有瓦片已经完成计算阶段。组中的每个瓦片具有本地退出状态,直至计算阶段完成。指令集包括同步指令,用于由每个瓦片在完成其计算阶段时执行,以将同步请求以信号发给互连中的逻辑。响应于从组中的所有瓦片接收到同步请求,逻辑释放下一个交换阶段,并使组中的所有瓦片的聚合状态变得可用。



CN 110214317 B

1.一种处理系统,其包括瓦片布置和用于在瓦片之间通信的互连,每个瓦片包括其自己的处理单元和存储器,其中:

每个瓦片的处理单元包括用于执行机器代码指令的执行单元,每个机器代码指令是瓦片的指令集中预定义指令类型集的实例,所述指令集中的每个指令类型由相应操作码和用于取得零个或多个操作数的零个或多个操作数字段定义;

互连可操作为根据批量同步并行方案进行一些或所有瓦片的组之间的通信,由此所述组中的每个瓦片执行瓦片上计算阶段,接着是瓦片间交换阶段,不进行交换阶段,直至所述组中的所有瓦片已经完成计算阶段,其中组中的每个瓦片在完成计算阶段时具有本地退出状态;

指令集包括同步指令,用于由组中的每个瓦片在完成其计算阶段时执行,其中同步指令的执行使执行单元向互连中的硬件的逻辑发送同步请求;和

互连中的逻辑配置为将本地退出状态聚合到全局退出状态中,并且响应于组中的所有瓦片完成计算阶段,如通过从组中的所有瓦片接收到同步请求所指示,将全局退出状态存储在组中的每个瓦片上的全局退出状态寄存器中,从而使在组中的每个瓦片上运行的代码的一部分可访问全局退出状态。

2.根据权利要求1所述的处理系统,其中每个瓦片上的执行单元配置为响应于执行同步指令而暂停指令发布;并且,其中互连中的逻辑配置为响应于从组中的所有瓦片接收到同步请求,向组中的每个瓦片发回同步确认信号,从而恢复指令发布。

3.根据权利要求1或2所述的处理系统,其中每个本地退出状态和全局退出状态是单一位。

4.根据权利要求3所述的处理系统,其中聚合由本地退出状态的布林AND或本地退出状态的布林OR组成。

5.根据权利要求1或2所述的处理系统,其中聚合退出状态包括表示三进制值的至少两个位,所述三进制值指示本地退出状态是全是真的、全是假的,还是混合的。

6.根据权利要求1或2所述的处理系统,其中所述组中的每个瓦片包括布置为表示瓦片的本地退出状态的本地退出状态寄存器。

7.根据权利要求1所述的处理系统,其中组中的每个瓦片包括:

多个上下文寄存器集,每个上下文寄存器集各自布置为存储多个线程的相应一个的程序状态;和

调度器,其布置为调度执行交错时隙的重复顺序中的多个时隙中的每个中的多个工作者线程中的相应一个,每个工作者线程的程序状态存储在所述上下文寄存器集的相应一个中;

其中根据所述批量同步并行方案,不进行交换阶段,直至组中的所有瓦片上的所有工作者线程已经完成计算阶段;

其中每个瓦片上的本地退出状态是瓦片上的每个工作者线程所输出的单独退出状态的聚合;和

其中代码的所述一部分包括瓦片上的多个线程中的至少一个。

8.根据权利要求7所述的处理系统,其中组中的每个瓦片包括硬件逻辑,所述硬件逻辑配置为执行将单独退出状态聚合到本地退出状态中。

9.根据权利要求8所述的处理系统，其中指令集包括用于包括在每个工作者线程中的退出指令，执行单元配置为输出相应工作者线程的单独退出状态，并响应于退出指令的操作码而终止相应工作者线程。

10.根据权利要求7、8或9所述的处理系统，其中每个单独退出状态和本地退出状态是单一位，并且单独退出状态的聚合由单独退出状态的布林AND或单独退出状态的布林OR组成。

11.根据权利要求7、8或9所述的处理系统，其中本地退出状态包括表示三进制值的至少两个位，所述三进制值指示单独退出状态是全是真的、全是假的，还是混合的。

12.根据权利要求7、8或9所述的处理系统，其中交换阶段布置为由独立于工作者线程的监督者线程执行，并且所述至少一个线程包括监督者线程。

13.根据权利要求12所述的处理系统，其中

每个瓦片上的执行单元配置为响应于执行同步指令而暂停指令发布；并且，其中互连中的逻辑配置为响应于从组中的所有瓦片接收到同步请求，向组中的每个瓦片发回同步确认信号，从而恢复指令发布；和

指令发布的暂停包括至少暂停来自监督者线程的指令的发布，等待同步确认。

14.根据权利要求12所述的处理系统，其中每个瓦片上的上下文寄存器集包括：多个工作者上下文寄存器集，其布置为表示所述多个工作者线程中的相应一个的程序状态，以及附加监督者上下文寄存器集，其包括布置为表示监督者线程的程序状态的附加寄存器集。

15.根据权利要求14所述的处理系统，其中：

监督者线程布置为通过在所述时隙中的每个中运行而开始；

指令集还包括放弃指令，并且执行单元配置为响应于放弃指令的操作码，将其中执行放弃指令的时隙放弃给相应工作者线程；和

退出指令使其中执行退出指令的相应时隙被交回监督者线程，使得监督者线程恢复在相应时隙中运行。

16.根据权利要求1、2、7、8或9所述的处理系统，其用代码编程；其中代码的所述一部分配置为一旦有效就使用全局退出状态来执行取决于全局退出状态的分支决策。

17.根据权利要求1、2、7、8或9所述的处理系统，其被编程为执行机器智能算法，其中图形中的每个节点具有一个或多个相应输入边缘和一个或多个输出边缘，至少一些节点的输入边缘是至少一些其他节点的输出边缘，每个节点包括使其输出边缘与其输入边缘相关的相应函数，每个相应函数由一个或多个相应参数参数化，并且每个相应参数具有相关的误差，使得图形随着一些或所有参数中的误差减少而收敛于解；

其中每个瓦片对包括图形中的节点的子集的相应子图进行建模，并且每个本地退出状态用于指示相应子图中的节点的一个或多个参数中的误差是否满足到预定条件。

18.根据权利要求1所述的处理系统，其中所述组至少部分地由同步指令的操作数选择。

19.根据权利要求18所述的处理系统，其中同步指令的操作数选择仅包括同一芯片上的瓦片还是所述组中的不同芯片上的瓦片。

20.根据权利要求18或19所述的处理系统，其中同步指令的操作数从不同的分级层次中选择所述组。

21.根据权利要求1、2、7、8、9、18或19所述的处理系统，其中指令集还包括弃权指令，其使在其上执行弃权指令的瓦片被选择退出所述组。

22.一种操作包括瓦片布置和用于在瓦片之间通信的互连的处理系统的方法，每个瓦片包括其自己的处理单元和存储器，其中每个瓦片的处理单元包括用于执行机器代码指令的执行单元，每个机器代码指令是瓦片的指令集中预定义指令类型集的实例，所述指令集中的每个指令类型由相应操作码和用于取得零个或多个操作数的零个或多个操作数字段定义；所述方法包括：

根据批量同步并行方案通过互连进行一些或所有瓦片的组之间的通信，由此所述组中的每个瓦片执行瓦片上计算阶段，接着是瓦片间交换阶段，不进行交换阶段，直至所述组中的所有瓦片已经完成计算阶段，其中组中的每个瓦片在完成计算阶段时具有本地退出状态；

其中指令集包括同步指令，用于由组中的每个瓦片在完成其计算阶段时执行，其中同步指令的执行使执行单元向互连中的硬件逻辑发送同步请求；和

所述方法包括：响应于组中的所有瓦片完成计算阶段，如通过从组中的所有瓦片接收到同步请求所指示，触发互连中的逻辑将本地退出状态聚合到全局退出状态中，并将全局退出状态存储在组中的每个瓦片上的全局退出状态寄存器中，从而使在组中的每个瓦片上运行的代码的一部分可访问全局退出状态。

23.一种包括代码的计算机程序产品，其实现在计算机可读存储上，并配置为在权利要求1、2、7、8、9、18或19所述的处理系统上执行，所述代码包括用于在组中的每个瓦片上执行的一部分，包括每个部分中的同步指令的实例。

多瓦片处理布置中的同步

技术领域

[0001] 本公开涉及在多瓦片处理布置中同步多个不同瓦片的工作负载，每个瓦片包括其自己的处理单元和存储器。具体地，本公开涉及批量同步并行(BSP)通信方案，其中瓦片组中的每个瓦片必须在组中的任何瓦片可以继续到交换阶段之前完成计算阶段。

背景技术

[0002] 多线程处理器是能够彼此并行执行多个程序线程的处理器。处理可以包括对多个不同线程是共同的一些硬件(例如共同指令存储器、数据存储器和/或执行单元)；但是为了支持多线程，处理器还包括一些特定于每个线程的专用硬件。

[0003] 专用硬件至少包括可以一次执行的多个线程中的每个的相应上下文寄存器文件。当谈到多线程处理器时，“上下文”指的是彼此并行执行的线程的相应一个的程序状态(例如程序计数器值、状态和当前操作数值)。上下文寄存器文件指的是用于表示相应线程的这种程序状态的相应寄存器集合。寄存器文件中的寄存器不同于通用存储器，因为寄存器地址固定为指令字中的位(bit)，而存储器地址可以通过执行指令来计算。给定上下文的寄存器通常包括用于相应线程的相应程序计数器，以及相应操作数寄存器集，用于临时保持在由该线程执行的计算期间作用于相应线程并由相应线程输出的数据。每个上下文还可以具有相应的状态寄存器，用于存储相应线程的状态(例如它是暂停还是正在运行)。因此，每个当前运行的线程都具有自己独立的程序计数器，并可选地具有操作数寄存器和状态寄存器。

[0004] 多线程的一种可能形式是并行(parallelism)。即是说，除了多个上下文之外，还提供了多个执行流水线：即，用于要并行执行的每个指令流的独立执行流水线。但是，这需要在硬件方面进行大量重复。

[0005] 因此，另一种形式的多线程处理器采用并发(concurrency)而不是并行，其中线程共享共同的执行流水线(或至少流水线的共同部分)，并且不同的线程通过该相同、共享的执行流水线交错。由于增加了隐藏流水线延迟(latency)的机会，与没有并发或并行相比，多线程处理器的性能仍然可以改善。此外，这种方法不需要具有多个执行流水线的完全并行处理器所需那么多的专用于每个线程的额外硬件，因此不会需要如此多的额外硅。

[0006] 一种形式的并行可以通过包括在同一芯片(即相同管芯)上的多个瓦片的布置的处理器来实现，其中每个瓦片包括其自己独立的相应处理单元和存储器(包括程序存储器和数据存储器)。因此，程序代码的独立部分可以在不同的瓦片上并行运行。这些瓦片通过芯片上互连连接在一起，这使得在不同瓦片上运行的代码能够在瓦片之间进行传送(communicate)。在一些情况下，每个瓦片上的处理单元本身可以在瓦片上运行多个并发线程，每个瓦片具有其自己的相应上下文集和如上所述的相应流水线，从而支持通过同一流水线的在同一瓦片上的多个线程的交错。

[0007] 通常，不同瓦片上运行的程序的各部分之间可以存在依赖性。因此，需要一种技术来防止一个瓦片上的代码段在它所依赖于的数据被另一瓦片上的另一段代码使其变得可

用之前运行。存在许多用于实现此目的的可能方案，但是本文中感兴趣的方案被称为“批量同步并行”(BSP)。根据BSP，每个瓦片以交替周期执行计算阶段和交换阶段。在计算阶段期间，每个瓦片在瓦片上本地执行一个或多个计算任务，但是不与任何其他瓦片传送其计算的任何结果。在交换阶段，允许每个瓦片将来自先前计算阶段的计算的一个或多个结果交换到组中的一个或多个其他瓦片，和/或从组中的一个或多个其他瓦片交换，但是还未继续到下一个计算阶段。此外，根据BSP原理，屏障同步被置于从计算阶段过渡到交换阶段、或者从交换阶段过渡到计算阶段、或两者的接合点。也就是说：(a)在允许组中的任何瓦片继续到下一个交换阶段之前，需要所有瓦片都完成其相应的计算阶段，或者(b)在允许组中的任何瓦片继续到下一个计算阶段之前，需要组中的所有瓦片都完成其相应的交换阶段，或者(c)两者。在一些情况下，可以允许执行计算的瓦片与其他系统资源(例如网卡或存储盘)进行传送，前提是不涉及与组中其他瓦片的传送。

[0008] 在机器智能中可以找到多线程和/或多瓦片并行处理的示例性使用。如机器智能领域的技术人员所熟悉的，机器智能算法是基于对“知识模型”执行迭代更新，该模型可以由多个互连节点的图形表示。每个节点表示其输入的函数。一些节点接收对图形的输入，一些接收来自一个或多个其他节点的输入，而一些节点的输出形成其他节点的输入，一些节点的输出提供图形的输出(在一些情况下，给定节点甚至可能具有以下全部：对图形的输入、来自图形的输出和对其他节点的连接)。此外，由一个或多个相应参数(例如权重)对每个节点的函数进行参数化。在学习级期间，目标是基于经验输入数据集，找到各个参数的值，使得图形作为整体针对可能输入范围而生成期望的输出。用于这样做的各种算法在本领域中是已知的，例如基于随机梯度下降(stochastic gradient descent)的反向传播算法(back propagation algorithm)。在基于输入数据的多次迭代中，逐渐调整参数以减少它们的误差，并且因此图形收敛于解(solution)。在随后的阶段，学习的模型然后可以用于对在给定的指定输入集的情况下对输出进行预测，或者在给定的指定输出集的情况下对输入(原因)进行推断。

[0009] 每个节点的实现将涉及数据的处理，并且图形的互连对应于要在节点之间交换的数据。通常，每个节点的至少一些处理可以独立于图形中的一些或所有其他节点来执行，因此大的图形显露出巨大的并发和/或并行的机会。

发明内容

[0010] 以下描述了具有一种架构的处理器的部件，该架构已被开发用于解决机器智能应用中涉及的计算中出现的问题。本文所述的处理器可用作工作加速器，也就是说，它从在主计算机上运行的应用接收工作负载，该工作负载通常采用要处理的非常大的数据组的形式(例如机器智能算法用来学习知识模型的大型经验数据组，或者使用先前学习的知识模型从中执行预测或推断的数据)。本文所示的架构的目的是高效地处理这些非常大量的数据。处理器架构是为处理机器智能中涉及的工作负载而开发的。尽管如此，显而易见的是，所公开的架构也可以适用于共享相似特性的其他工作负载。

[0011] 当通过多个瓦片执行程序的不同部分时，可能需要执行屏障同步以将多个瓦片带到共同执行点。可能需要在所有瓦片已经完成计算阶段时确定程序整体上的状态，例如，确定应否向主机报告异常，或作出分支决策，以确定分支到程序的下一部分还是继续迭代当

前部分。例如，如果瓦片组中的每个在执行机器智能图形的相应子图的计算，可能期望确定子图的节点是否全部已经满足了某个条件，指示图形收敛于解。使用现有技术来作出这样确定，需要使用通用指令编程的一些步骤。

[0012] 本文认识到，期望将处理器的指令集定制为大规模多线程应用，例如机器学习。根据本公开，这通过以下方式实现：仅在组中所有瓦片已经完成当前BSP计算阶段后，才提供用于验证瓦片组的结果的专用机器代码指令，从而提供同步瓦片的能力，同时以更小的延迟和更低的代码密度确定多个线程的整体结果。

[0013] 根据本文公开的一个方面，提供了一种处理系统，其包括瓦片布置和用于在瓦片之间通信的互连，其中：

[0014] 每个瓦片包括用于执行机器代码指令的执行单元，每个机器代码指令是处理器的指令集中预定义指令类型集的实例，所述指令集中的每个指令类型由相应操作码和用于取得零个或多个操作数的零个或多个操作数字段定义；

[0015] 互连可操作为根据批量同步并行方案进行一些或所有瓦片的组之间的通信，由此所述组中的每个瓦片执行瓦片上计算阶段，接着是瓦片间交换阶段，不进行交换阶段，直至所述组中的所有瓦片已经完成计算阶段，其中组中的每个瓦片在完成计算阶段时具有本地退出状态；

[0016] 指令集包括同步指令，用于由组中的每个瓦片在完成其计算阶段时执行，其中同步指令的执行使执行单元向互连中的硬件逻辑发送同步请求；和

[0017] 互连中的逻辑配置为将本地退出状态聚合到全局退出状态中，并且响应于组中的所有瓦片完成计算阶段，如通过从组中的所有瓦片接收到同步请求所指示，将全局退出状态存储在组中的每个瓦片上的全局退出状态寄存器中，从而使在组中的每个瓦片上运行的代码的一部分可访问全局退出状态。

[0018] 根据本文公开的另一方面，提供了一种处理系统，其包括瓦片布置和用于在瓦片之间通信的互连，其中：

[0019] 每个瓦片包括用于执行机器代码指令的相应执行单元，每个机器代码指令是处理器的指令集中预定义指令类型集的实例，所述指令集中的每个指令类型由相应操作码和用于取得零个或多个操作数的零个或多个操作数字段定义；

[0020] 互连包括采用专用硬件逻辑形式的同步逻辑，用于协调所述瓦片的组中的一些或全部；

[0021] 指令集包括同步指令，其中每个相应瓦片上的相应执行单元配置为使得如果同步指令的实例通过相应执行单元执行，则响应于同步指令的操作码，使同步请求的实例被从相应瓦片发送到互连中的同步逻辑，并暂停相应瓦片上的指令发布，等待从同步逻辑接收同步确认；

[0022] 每个瓦片具有本地退出状态寄存器，用于存储瓦片的本地退出状态，直至相应计算阶段完成；

[0023] 同步逻辑配置为将所述组中的瓦片的本地退出状态聚合成全局退出状态；和

[0024] 同步逻辑还被配置为：响应于从所述组中的所有瓦片接收到同步请求的实例，将同步确认发回组中的每个瓦片，从而允许指令发布恢复，并将全局退出状态存储在组中的每个瓦片上的全局退出状态寄存器中，从而使在组中的每个瓦片上运行的代码的一部分可

访问全局退出状态。

[0025] 在实施例中，每个瓦片上的执行单元可被配置为响应于执行同步指令而暂停指令发布；并且，互连中的逻辑可被配置为响应于从组中的所有瓦片接收到同步请求，向组中的每个瓦片发回同步确认信号，从而恢复指令发布。

[0026] 在实施例中，每个本地退出状态和全局退出状态可能是单一位。

[0027] 在实施例中，聚合可以由本地退出状态的布林AND或本地退出状态的布林OR组成。

[0028] 在替代实施例中，聚合退出状态可以包括表示三进制值的至少两个位，所述三进制值指示本地退出状态是全是真的、全是假的，还是混合的。

[0029] 在实施例中，所述组中的每个瓦片可包括布置为表示瓦片的本地退出状态的本地退出状态寄存器。

[0030] 在实施例中，组中的每个瓦片可包括：

[0031] 多个上下文寄存器集，每个上下文寄存器集各自布置为存储多个线程的相应一个的程序状态；和

[0032] 调度器，其布置为调度执行交错时隙的重复顺序中的多个时隙中的每个中的多个工作者线程中的相应一个，每个工作者线程的程序状态存储在所述上下文寄存器集的相应一个中；

[0033] 其中根据所述批量同步并行方案，不进行交换阶段，直至组中的所有瓦片上的所有工作者线程已经完成计算阶段；

[0034] 其中每个瓦片上的本地退出状态可以是瓦片上的每个工作者线程所输出的单独退出状态的聚合；和

[0035] 其中代码的所述一部分可包括瓦片上的多个线程中的至少一个。

[0036] 在实施例中，组中的每个瓦片可包括硬件逻辑，所述硬件逻辑配置为执行所述将单独退出状态聚合到本地退出状态中。

[0037] 在实施例中，指令集可包括用于包括在每个工作者线程中的退出指令，并且执行单元可被配置为输出相应工作者线程的单独退出状态，并响应于退出指令的操作码而终止相应工作者线程。

[0038] 在实施例中，每个单独退出状态和本地退出状态可能是单一位，并且单独退出状态的聚合可由单独退出状态的布林AND或单独退出状态的布林OR组成。

[0039] 在实施例中，本地退出状态可以包括表示三进制值的至少两个位，所述三进制值指示单独退出状态是全是真的、全是假的，还是混合的。

[0040] 在实施例中，交换阶段可布置为由独立于工作者线程的监督者线程执行，并且所述至少一个线程可包括监督者线程。

[0041] 在实施例中，指令发布的暂停可包括至少暂停来自监督者线程的指令的发布，等待同步确认。

[0042] 在实施例中，每个瓦片上的上下文寄存器集可包括：多个工作者上下文寄存器集，其布置为表示所述多个工作者线程中的相应一个的程序状态，以及附加监督者上下文寄存器集，其包括布置为表示监督者线程的程序状态的附加寄存器集。

[0043] 在实施例中：

[0044] 监督者线程可布置为通过在所述时隙中的每个中运行而开始；

[0045] 指令集还可包括放弃指令，并且执行单元配置为响应于放弃指令的操作码，将其中执行放弃指令的时隙放弃给相应工作者线程；和

[0046] 退出指令可使其中执行退出指令的相应时隙被交回监督者线程，使得监督者线程恢复在相应时隙中运行。

[0047] 在实施例中，代码的所述一部分可被配置为一旦有效就使用全局退出状态来执行取决于全局退出状态的分支决策。

[0048] 在实施例中，处理系统可被编程为执行机器智能算法，其中图形中的每个节点具有一个或多个相应输入边缘和一个或多个输出边缘，至少一些节点的输入边缘是至少一些其他节点的输出边缘，每个节点包括使其输出边缘与其输入边缘相关的相应函数，每个相应函数由一个或多个相应参数参数化，并且每个相应参数具有相关的误差，使得图形随着一些或所有参数中的误差减少而收敛于解；其中每个瓦片可对包括图形中的节点的子集的相应子图进行建模，并且每个本地退出状态可用于指示相应子图中的节点的一个或多个参数中的误差是否满足到预定条件。

[0049] 在实施例中，所述组可以至少部分地由同步指令的操作数选择。

[0050] 在实施例中，同步指令的操作数可选择仅包括同一芯片上的瓦片还是所述组中的不同芯片上的瓦片。

[0051] 在实施例中，同步指令的操作数可从不同的分级层次中选择所述组。

[0052] 在实施例中，指令集还可包括弃权指令，其使在其上执行弃权指令的瓦片被选择退出所述组。

[0053] 根据本文公开的另一方面，提供一种操作包括瓦片布置和用于在瓦片之间通信的互连的处理系统的方法，其中每个瓦片包括用于执行机器代码指令的执行单元，每个机器代码指令是处理器的指令集中预定义指令类型集的实例，所述指令集中的每个指令类型由相应操作码和用于取得零个或多个操作数的零个或多个操作数字段定义；所述方法包括：

[0054] 根据批量同步并行方案通过互连进行一些或所有瓦片的组之间的通信，由此所述组中的每个瓦片执行瓦片上计算阶段，接着是瓦片间交换阶段，不进行交换阶段，直至所述组中的所有瓦片已经完成计算阶段，其中组中的每个瓦片在完成计算阶段时具有本地退出状态；

[0055] 其中指令集包括同步指令，用于由组中的每个瓦片在完成其计算阶段时执行，其中同步指令的执行使执行单元向互连中的硬件逻辑发送同步请求；和

[0056] 所述方法包括：响应于组中的所有瓦片完成计算阶段，如通过从组中的所有瓦片接收到同步请求所指示，触发互连中的逻辑将本地退出状态聚合到全局退出状态中，并将全局退出状态存储在组中的每个瓦片上的全局退出状态寄存器中，从而使在组中的每个瓦片上运行的代码的一部分可访问全局退出状态。

[0057] 根据本文公开的另一方面，提供了一种包括代码的计算机程序产品，其实现在计算机可读存储上，并配置为在本文公开的任何实施例所述的处理系统上执行，所述代码包括用于在组中的每个瓦片上执行的一部分，包括每个部分中的同步指令的实例。

附图说明

[0058] 为了帮助理解本公开，并示出如何实施实施例，通过示例的方式参考附图，其中：

[0059] 图1是多线程处理单元的示意框图，

[0060] 图2是多个线程上下文的示意框图，

[0061] 图3示意性地示出了交错执行时隙的方案，

[0062] 图4示意性地示出了监督者线程和多个工作者线程，

[0063] 图5是用于聚合多个线程的退出状态的逻辑的示意图，

[0064] 图6示意性示出了同一瓦片上的工作者线程之间的同步，

[0065] 图7是包括多个瓦片的处理器芯片的示意框图，

[0066] 图8是批量同步并行(BSP)计算模型的示意图，

[0067] 图9是BSP模型的另一个示意图，

[0068] 图10是多线程处理单元之间的BSP的示意图，

[0069] 图11是互连系统的示意框图，

[0070] 图12是多个互连处理器芯片的系统的示意图，

[0071] 图13是多层BSP方案的示意图，

[0072] 图14是多个处理器芯片的系统的另一示意图，

[0073] 图15是机器智能算法中使用的图形的示意图，和

[0074] 图16示出了用于在芯片之间同步的示例性布线。

具体实施方式

[0075] 下文描述了一种处理器架构，所述处理器架构在其指令集中包括专用指令，用于执行屏障同步，同时将横跨多个瓦片的多个线程的聚合退出状态聚合成退出状态寄存器中的单个聚合状态，其中该聚合退出状态寄存器存在于每个瓦片中，并包含每个已被聚合的瓦片的相同结果。然而，首先参照图1至图4，描述可以并入这点的示例性处理器。

[0076] 图1示出了根据本公开实施例的处理器模块4的示例。例如，处理器模块4可以是同一芯片上类似处理器瓦片的阵列中的一个瓦片，或者可以实现为在其自己的芯片上的独立处理器。处理器模块4包括采取桶形线程处理单元的形式的多线程处理单元10，以及本地存储器11(即，在多瓦片阵列的情况下在同一瓦片上，或者在单一处理器芯片的情况下在同一芯片上)。桶形线程处理单元是一种多线程处理单元，其中流水线的执行时间被划分为重复顺序的交错时隙，每个都可以由给定的线程拥有。这将在稍后更详细地讨论。存储器11包括指令存储器12和数据存储器22(其可以在不同的可寻址存储器单元中或同一可寻址存储器单元的不同区域中实现)。指令存储器12存储将由处理单元10执行的机器代码，而数据存储器22存储由执行的代码操作的数据和由执行的代码输出的数据(例如，作为这种操作的结果)。

[0077] 存储器12存储程序的各种不同线程，每个线程包括用于执行特定任务的相应顺序的指令。注意，本文所指的指令指机器代码指令，即处理器指令集的基本指令之一的实例，由单个操作码和零个或更多个操作数组成。

[0078] 本文描述的程序包括多个工作者线程和监督者子程序，监督者子程序可被构造为一个或多个监督者线程。这些将在稍后更详细地讨论。在实施例中，一些或所有工作者线程中的每一个采用相应“小代码”(codelet)的形式。小代码是一种特殊类型的线程，有时也称为″原子″线程。它具有从线程开头(从启动时起)执行所需的所有输入信息，即，它在启动后

不会从程序的任何其他部分或从存储器获取任何输入。此外，程序的其他任何部分都不会使用线程的任何输出(结果)，直到它终止(完成)。除非遇到误差，否则保证完成。注意，有些文献也将小代码定义为无状态，即如果它运行两次，则无法从第一次运行中继承任何信息，但这里不采用这个附加定义。还要注意，并非所有工作者线程都需要是小代码(原子)，在实施例中，一些或所有工作者可能能够彼此进行传送。

[0079] 在处理单元10内，来自指令存储器12的多个不同的线程可以通过单个执行流水线13交错(尽管在整个程序的任何给定的时间点，通常只有存储在指令存储器中的总线程的子集可以交错)。多线程处理单元10包括：多个上下文寄存器文件26，每个布置为表示要并发执行的线程中不同的相应一个的状态(上下文)；共享执行流水线13，其对并发执行的线程是共同的；和调度器24，用于以交错方式(优选地以轮循(round robin)方式)调度并发线程以便通过共享流水线执行。处理单元10连接到对多个线程是共同的共享指令存储器12，以及对多个线程也是共同的共享数据存储器22。

[0080] 执行流水线13包括获取级14，解码级16和执行级18，所述执行级18包括执行单元，所述执行单元可以执行算术和逻辑运算、地址计算、加载和存储操作，以及其他操作，如指令集架构所定义。每个上下文寄存器文件26包括用于表示相应线程的程序状态的相应寄存器集。

[0081] 在图2中示意性地示出了构成每个上下文寄存器文件26的寄存器的示例。每个上下文寄存器文件26包括相应的一个或多个控制寄存器28，至少包括相应线程的程序计数器(PC)(用于跟踪线程当前正在执行的指令地址)，并且在实施例中还有一个或多个状态寄存器(SR)的集，其记录相应线程的当前状态(例如它当前正在运行还是暂停，例如因为遇到了误差而暂停)。每个上下文寄存器文件26还包括相应的操作数寄存器32的(OP)集，用于临时保存由相应线程执行的指令的操作数，即，在相应线程的指令的操作码在执行时定义的操作上操作或由其产生的值。应当理解，每个上下文寄存器文件26可以可选地包括相应的一个或多个其他类型的寄存器(未示出)。还要注意，虽然术语“寄存器文件”有时用于表示共同地址空间中的寄存器组，但这不一定是本公开中的情况，并且每个硬件上下文26(表示每个上下文的寄存器组26中的每一个)可以更一般地包括一个或多个这样的寄存器文件。

[0082] 如稍后将更详细讨论的，所公开的布置对于M个(在示出的示例中M＝3，但这不是限制性的)可以并发执行的线程中的每一个具有一个工作者上下文寄存器文件CX0...CX(M-1)，以及一个额外的监督者上下文寄存器文件CXS。工作者上下文寄存器文件被保留用于存储工作者线程的上下文，并且监督者上下文寄存器文件被保留用于存储监督者线程的上下文。注意，在实施例中，监督者上下文是特殊的，因为它与每个工作者具有不同数量的寄存器。每个工作者上下文优选地彼此具有相同数量的状态寄存器和操作数寄存器。在实施例中，监督者上下文可以比每个工作者具有更少的操作数寄存器。工作者上下文可能具有而监督者没有的操作数寄存器的示例包括：浮点寄存器、累加寄存器和/或专用权重寄存器(用于保持神经网络的权重)。在实施例中，监督者还可以具有不同数量的状态寄存器。此外，在实施例中，处理器模块4的指令集架构可以配置为使得工作者线程和监督者线程执行一些不同类型的指令，但是也共享一些指令类型。

[0083] 连接获取级14以在调度器24的控制下从指令存储器12中获取要执行的指令。调度器24配置为控制获取级14在重复顺序的时隙中依次从并发执行线程集中的每一个获取指

令，从而将流水线13的资源划分为多个时间上交错的时隙，如将在稍后更详细地讨论。例如，调度方案可以是轮循或加权轮循。以这种方式操作的处理器的另一个术语是桶形线程处理器。

[0084] 在一些实施例中，调度器24可以访问每个线程的状态寄存器SR之一，其指示线程是否被暂停，以便调度器24实际上控制获取级14仅获取当前有效的那些线程的指令。在实施例中，优选地每个时隙(和相应的上下文寄存器文件)总由一个或另一个线程拥有，即每个时隙总是由某个线程占用，并且每个时隙总是包括在调度器24的顺序中；尽管占用任何给定时隙的线程可能恰好在该时间被暂停，在这种情况下，当顺序来到该时隙时，相应线程的指令获取被传递。替代地，不排除例如在替代、较不优选的实现中，一些时隙可以暂时空出，并从调度顺序中排除。在参考执行单元可操作以交错的时隙数量或类似的情况下，这指的是执行单元能够并发执行的时隙的最大数量，即执行单元的硬件支持的并发时隙数。

[0085] 获取级14可以访问每个上下文的程序计数器(PC)。对于每个相应的线程，获取级14从指程序存储器12中的下一个地址获取该线程的下一个指令，如程序计数器所指示。程序计数器递增每个执行周期，除非由分支指令分支。然后，获取级14将所获取的指令传递到解码级16以进行解码，然后解码级16将所解码的指令的指示连同指令中所指定的任何操作数寄存器32的解码地址一起传递给执行单元18，从而执行指令。执行单元18可以访问操作数寄存器32和控制寄存器28，它可以将其用于基于解码的寄存器地址执行指令，例如在算术指令的情况下(例如通过加、乘、减或除两个操作数寄存器中的值，并将结果输出到相应线程的另一个操作数寄存器中)。或者，如果指令定义了存储器访问(加载或存储)，则执行单元18的加载/存储逻辑根据指令将来自数据存储器的值加载到相应线程的操作数寄存器中，或者将来自相应线程的操作数寄存器的值存储到数据存储器22中。或者，如果指令定义了分支或状态改变，则执行单元相应地改变其中一个状态寄存器SR或程序计数器PC中的值。注意，当执行单元18正在执行一个线程的指令时，解码级16可以解码来自交错顺序中的下一个时隙的线程的指令；和/或当一个指令被解码级16解码时，来自这之后的下一个时隙的线程的指令可以被获取级14获取(尽管一般地本公开的范围不限于一个时隙一个指令，例如，在替代的情况下，可以在每个时隙从给定线程发出两个或更多个指令的批次)。因此，根据已知的桶形线程处理技术，交错有利地隐藏了流水线13中的延迟。

[0086] 图3示出了由调度器24实现的交错方案的示例。这里根据轮循方案交错并发线程，由此在方案的每个轮次内，轮次被分为时隙顺序S0、S1、S2...，每个时隙用于执行相应的线程。通常，每个时隙是一个处理器周期长，并且不同的时隙是均匀的大小，尽管在所有可能的实施例中不一定如此，例如加权轮循方案也是可能的，由此一些线程在每个执行轮次中比其他线程获得更多循环。通常，桶形线程可以采用偶数轮循或加权的轮循调度，其中在后一种情况下，加权可以是固定或自适应的。

[0087] 然后，不管每次执行轮次的顺序，重复这种模式，每一轮次包括每个时隙的相应实例。因此，请注意，本文所指的时隙指顺序中的重复分配位置，而不是时隙在顺序的给定重复中的特定实例。换句话说，调度器24将流水线13的执行周期分配到多个时间交错(时分多路复用)的执行信道中，其中每个包括重复的时隙顺序中的相应时隙的重现。在所示的实施例中，有四个时隙，但这仅用于说明目的，而且其他数量也是可能的。例如，在一个优选实施例中，实际上有六个时隙。

[0088] 然后,不管轮循方案被划分成多少个时隙,根据本公开,处理单元10包括比时隙数量多一个的上下文寄存器文件26,即它支持比它能够进行桶形线程的交错时隙的数量多一个的上下文。

[0089] 这在图2中以示例方式示出:如果存在四个时隙S0...S3,如图3所示,则存在五个上下文寄存器文件,在此标记为CX0、CX1、CX2、CX3和CXS。也就是说,尽管桶形线程方案中只有四个执行时隙S0...S3,因此可并发执行仅四个线程,本文公开了添加第五个上下文寄存器文件CXS,包括第五程序计数器(PC)、第五集操作数寄存器32,以及在实施例中还有第五集的一个或多个状态寄存器(SR)。尽管注意到如上所述,在实施例中,监督者上下文可以与其他CX0...3不同,并且监督者线程可以支持用于操作执行流水线13的不同指令集。

[0090] 首四个上下文CX0...CX3中的每个用于表示当前分配给四个执行时隙S0...S3之一的多个"工作者线程"的相应一个的状态,用于执行程序员所期望的任何特定于应用的计算任务(再次注意,这可能只是存储在指令存储器12中的程序的工作者线程总数的子集)。然而,第五个上下文CXS被保留用于特殊函数,以表示"监督者线程"(SV)的状态,其角色是至少在以下意义上协调工作者线程的执行:在整个程序中哪一点在时隙S0、S1、S2...中的哪一个中分配要执行哪个工作者线程W。可选地,监督者线程可以具有其他"监视者"或协调职责。例如,监督者线程可以负责执行屏障同步以确保特定的执行顺序。例如,在一个或多个第二线程依赖于由在同一处理器模块4上运行的一个或多个第一线程输出的数据的情况下,监督者可以执行屏障同步以确保第二线程都不会开始,直至第一线程已经完成。和/或,监督者可以执行屏障同步以确保处理器模块4上的一个或多个线程不会开始,直至某个外部数据源(例如另一个瓦片或处理器芯片)已经完成了使该数据变得可用所需的处理。监督者线程还可以用于执行与多个工作者线程相关的其他功能。例如,监督者线程可以负责向处理器模块4外部发送数据(以接收要由一个或多个线程作用的外部数据,和/或发送由一个或多个工作者线程输出的数据)。通常,监督者线程可以用于提供程序员期望的任何类型的监视或协调函数。例如,作为另一个例子,监督者可以监视瓦片本地存储器12与诸如存储盘或网卡的较宽系统(在阵列6外部)中的一个或多个资源之间的传输。

[0091] 当然要注意,四个时隙仅是示例,并且通常在其他实施例中,可以存在其他数量,使得如果每个轮次存在最多M个时隙0...M-1,则处理器模块4包括M+1个上下文CX...CX(M-1)&CXS,即每个可以在任何给定时间交错的工作者线程一个,以及为监督者提供额外的上下文。例如,在一个示例性实现中,存在六个时隙和七个上下文。

[0092] 参考图4,在交错时隙的方案中,监督者线程SV本身没有其自己的时隙。工作者线程的时隙分配也不是灵活定义的。相反,每个时隙都有其自己专用的上下文寄存器文件(CX0...CXM-1),用于存储工作者上下文,其在将时隙分配给工作者时由工作者使用,但不在将时隙分配将监督者时使用。当给定的时隙被分配给监督者时,该时隙使用监督者的上下文寄存器文件CXS。注意监督者总是可以访问其自己的上下文,而没有工作者能够占用监督者上下文寄存器文件CXS。

[0093] 监督者线程SV具有在任何和所有时隙S0...S3(或更一般地S0...SM-1)中运行的能力。调度器24被配置成,当程序作为整体开始时,通过将监督者线程分配给所有时隙而开始,即,监督者SV开始在所有时隙S0...S3中运行。但是,向监督者线程提供了一种机制,用于在某个后续点(直接或者在执行一个或多个监督者任务之后)将其正在运行的每个时隙

暂时放弃给相应的一个工作者线程，例如图4所示的示例中的初始工作者W0...W3。这是通过监督者线程执行放弃指令来实现，在本文中通过示例的方式称为“RUN”(运行)。在实施例中，该指令采用两个操作数：指令存储器12中的工作者线程的地址和数据存储器22中针对该工作者线程的一些数据的地址：

[0094] RUN task_addr,data_addr

[0095] 工作者线程是可以彼此并发运行的代码的一部分，每个表示要执行的一个或多个相应计算任务。数据地址可以指定工作者线程要作用的一些数据。替代地，放弃指令可以只采用指定工作者线程地址的单个操作数，并且数据地址可以被包括在工作者线程的代码中；或者，在另一个示例中，单个操作数可以指向指定工作者线程和数据的地址的数据结构。如上所述，在实施例中，至少一些工作者可以采用小代码的形式，即，可并发执行的代码的原子单元。替代地或附加地，一些工作者不一定是小代码，并且可能能够彼此进行传送。

[0096] 放弃指令(“RUN”)作用于调度器24，从而将当前时隙(该指令本身在该时隙被执行)放弃给由操作数指定的工作者线程。注意，在放弃指令中隐含的是，在其中执行该指令的时隙正被放弃(在机器代码指令的上下文中隐含指不需要操作数来指定这点——从操作码本身隐含地理解)。因此，交出的时隙是其中监督者执行放弃指令的时隙。或者，换句话说，监督者正在它所交出的相同空间中执行。监督者说“在这个位置运行这段代码”，然后从该点起，重复的时隙由相关的工作者线程(临时)拥有。

[0097] 监督者线程SV在一个或多个其他时隙中的每一个中执行类似的操作，以将其一些或全部时隙分配给工作者线程W0...W3中的不同的相应的一些(从指令存储器12中较大组W0...wj中选择)。一旦它为了最后时隙这样做，监督者就会被暂停(后来当工作者W将其一中一个时隙交还时，将从它停下来的地方继续)。

[0098] 因此，监督者线程SV能够将不同的工作者线程分配给不同交错执行时隙S0...S3，每个工作者线程执行一个或多个任务。当监督者线程确定是时候运行工作者线程时，它使用放弃指令(“RUN”)将该工作者分配给其中执行RUN指令的时隙。

[0099] 在一些实施例中，指令集还包括运行指令的变体RUNALL(“运行全部”)。这个指令用于一起启动多于一个工作者的集，全部都执行相同的代码。在实施例中，这在处理单元的时隙S0...S3(或更一般地S0...S(M-1))的每一个中启动工作者。

[0100] 此外，在一些实施例中，RUN和/或RUNALL指令在被执行时还自动地将一些状态从一个或多个监督者状态寄存器CXS(SR)复制到由RUN或RUNALL启动的工作者线程的相应的一个或多个状态寄存器中。例如，复制的状态可以包括一个或多个模式，例如浮点舍入模式(例如舍入到最近或舍入到零)，和/或溢出模式(例如饱和或使用表示无穷大的独立值)。然后，复制的状态或模式控制讨论中的工作者根据复制的状态或模式操作。在实施例中，工作者稍后可以在其自己的状态寄存器中重写它(但是不能改变监督者的状态)。在进一步的替代或附加实施例中，工作者可以选择从监督者的一个或多个状态寄存器中读取一些状态(并且稍后也可以改变它们自己的状态)。例如，再次，这可以是采用来自监督者状态寄存器的模式，例如浮点模式或舍入模式。然而，在实施例中，监督者不能读取工作者的上下文寄存器CX0...中的任何一个。

[0101] 一旦启动，当前分配的工作者线程W0...W3中的每一个继续执行在由相应的放弃指令指定的代码中定义的一个或多个计算任务。在此结束时，相应的工作者线程然后将其

中它正在运行的时隙交给监督者线程。这是通过执行退出指令(“EXIT”(退出))来实现的。

[0102] EXIT指令采取至少一个操作数并且优选地仅采取单个操作数,exit_state(例如二进制值),用于程序员期望的任何目的,以在结束时指示相应小代码的状态(例如以指示是否达到某个条件):

[0103] EXIT exit_state

[0104] EXIT指令作用于调度器24,使得将它在其中执行的时隙归还给监督者线程。然后,监督者线程可以执行一个或多个后续监督者任务(例如,屏障同步和/或与诸如其他瓦片的外部资源交换数据),和/或继续执行另一个放弃指令以将新的工作者线程(W4等)分配给讨论中的时隙。因此,再次注意,指令存储器12中的线程总数可以大于桶形线程处理单元10在任何一个时刻可以交错的数量。监督者线程SV的作用是调度来自指令存储器12的工作者线程W0...Wj中的哪一个在整个程序的哪个阶段分配到调度器24的轮循调度中的交错时隙S0...SM中的哪一个。

[0105] 此外,EXIT指令还有一个特殊函数,即,使EXIT指令的操作数中指定的退出状态与通过相同处理器模块4(例如同一瓦片)的相同流水线13运行的多个其他工作者线程的退出状态(通过专用硬件逻辑)自动聚合。因此,在用于终止工作者线程的指令中包括额外的隐式设施。

[0106] 用于实现此目的的示例性电路如图5所示。在该示例中,单独线程的退出状态和聚合的退出状态均采用单个位的形式,即0或1。处理器模块4包括寄存器38,用于存储该处理器模块4的聚合退出状态。该寄存器在本文中可称为“本地共识”寄存器$LC(与当中处理器模块4包括为类似处理器瓦片阵列之一的全局共识相反,将在稍后更详细地讨论)。在实施例中,这种本地共识寄存器$LC 38是监督者上下文寄存器文件CXS中的监督者状态寄存器之一。用于执行聚合的逻辑包括AND(和)门37,其被布置为执行(A)EXIT指令的操作数中指定的退出状态和(B)本地共识寄存器($LC)38中的当前值的逻辑AND,以及将结果(Q)输出回到本地共识寄存器$LC 38中作为本地聚合的新值。

[0107] 在程序中的适当同步点,存储在本地共识寄存器($LC)38中的值最初被重置成为1。即,在此点之后退出的任何线程将对本地聚合退出状态$LC做出贡献,直到下次重置为止。如果两个输入(A,B)都是1,则AND门37的输出(Q)是1,但是如果输入(A,B)的任何一个是0,则输出Q变为0。每次执行EXIT指令时,其退出状态与之前已经过去(自上次重置后)的那些聚合。因此,透过图5中所示的布置,逻辑保持自上次重置本地共识寄存器($LC)38以来已经通过EXIT指令终止的任何工作者线程的退出状态的运行聚合。在此示例中,运行聚合指的是目前为止所有线程是否已退出了真(true):来自任何工作者线程的任何退出状态为0将意味着寄存器38中的聚合被锁存为0,直到下一次重置。在实施例中,监督者SV可以通过从本地共识寄存器($LC)38获得当前值来随时读取运行聚合(它不需要等待瓦片上同步来这样做)。

[0108] 本地共识寄存器($LC)38中的聚合的重置可以由监督者SV使用一个或多个通用指令对本地共识寄存器($LC)38的寄存器地址执行PUT(放置)来执行,在这示例中对寄存器38放置1的值。替代地,不排除可以通过自动机制执行重置,例如通过执行本文稍后所述的SYNC指令来触发。

[0109] 使用用于形成布林(Boolean)AND的功能的任何适合的电子组件组合在执行级18

的执行单元中的专用硬件电路中实现聚合电路37,在这种情况下是AND门。专用电路或硬件指具有硬连线函数的电路,而不是使用通用代码软件中编程。通过执行特殊EXIT指令触发本地退出状态的更新,该特殊EXIT指令是处理器模块4的指令集中的基本机器代码指令之一,具有聚合退出状态的固有功能。此外,本地聚合存储在控制寄存器38中,意味着专用存储器(在实施例中是单个存储位),其值可以由在流水线上运行的代码访问,但是不能由加载-存储单元(LSU)用来存储任何通用数据。相反,保持在控制寄存器中的数据的函数是固定的,在这种情况下是固定到存储本地聚合退出状态的函数。优选地,本地共识寄存器($LC)38形成处理器模块4上(例如瓦片上)的控制寄存器之一,其值可以由监督者通过执行GET(取得)指令访问和通过执行PUT指令来设置。

[0110] 注意,图5中所示的电路只是一个例子。等效电路将用OR(或)门替换AND门37,并反转软件中退出状态0和1的解释,即0→真,1→假(其中寄存器38在每个同步点被重置为0而不是1)。等效地,如果AND门被替换为OR门,但退出状态的解释和重置值未被反转,那么$LC中的聚合状态将记录有没有任何(而不是全部)工作者状态以状态1退出。在其他实施例中,退出状态不需要是单个位。例如,每个单独工作者的退出状态可以是单个位,但是聚合的退出状态$LC可以包括表示三态的两个位:所有工作者以状态1退出,所有工作者以状态0退出,或者工作者的退出状态是混合的。作为用于实现这点的逻辑的示例,编码三元值的两个位之一可以是单独退出状态的布林AND(或OR),并且三元值的另一个位可以是单独退出状态的布林OR。然后,可以随着这两个位的XOR形成表示工作的退出状态混合的第三编码情况。

[0111] 退出状态可用于表示程序员希望的任何内容,但一个特别设想示例是使用退出状态1来指示相应的工作者线程已经以“成功”或“真”状态退出,而退出状态0指示相应工作者线程以“不成功”或“假”状态退出(或者,如果聚合电路37执行OR而不是AND,并且寄存器$LC38最初被重置为0,则反之亦然)。例如,考虑一个应用,其中每个工作者线程执行具有相关条件的计算,例如指示机器智能算法的图形中的相应节点的一个或多个参数的误差是否落在根据预定度量的可接受水平内的条件。在这种情况下,可以使用一个逻辑级别的单独退出状态(例如1)可以用于指示条件被满足(例如,节点的一个或多个参数中的误差在根据某个度量的可接受级别内);而相反逻辑级别的单独退出状态(例如0)可以用于指示条件未被满足(例如,误差不在根据讨论中的度量的可接受级别内)。条件可以例如是放置在单个参数或每个参数上的误差阈值,或者可以是与工作者线程执行的相应计算相关联的多个参数的更复杂的函数。

[0112] 作为另一个更复杂的示例,工作者的单独退出状态和聚合的退出状态可以包括两个或更多个位,例如,可以用于表示工作者线程的结果的置信度。例如,每个单独工作者线程的退出状态可以表示对相应工作者线程的结果的置信度的概率性度量,并且聚合逻辑37可以替换为较复杂的电路,用于执行硬件中的单独置信度的概率性聚合。

[0113] 然后,不管程序员给退出状态赋予什么意义,监督者线程可以从本地共识寄存器($LC)38获得聚合值,以确定自上次重置以来(例如在最后的同步点)退出的所有工作者线程的聚合退出状态,例如以确定所有工人是否以成功或真的状态退出。然后,根据该聚合值,监督者线程可以根据程序员的设计做出决定。程序员可以选择地使用他或她希望的本地聚合退出状态。例如,监督者线程可以查询本地聚合的退出状态,以便确定由工作者线程

的某个子集构成的程序的特定部分是否已按预期或期望完成。如果不是(例如,至少一个工作者线程以不成功或假的状态退出),则它可以向主处理器报告,或者可以执行包括相同工作者线程的程序部分的另一次迭代;但如果是的话(例如,所有工作者线程都以成功或真的状态退出),它可以转而分支到包括一个或多个新工作者的程序的另一部分。

[0114] 优选地,监督者线程不应访问本地共识寄存器($LC)38中的值,直至所有讨论中的工作者线程已经退出,使得其中存储的值表示所有期望的线程的正确、最新的聚合状态。等待这点可以通过由监督者线程执行的屏障同步来强制执行,以等待所有当前运行的本地工作者线程(即,在同一处理器模块4上的那些,通过同一流水线13运行)退出。也就是说,监督者线程重置本地共识寄存器($LC)38,启动多个工作者线程,然后启动本地屏障同步(局部于处理器模块4,局部于一个瓦片),从而等待所有剩余的工作者线程在监督者被允许继续到从本地共识寄存器($LC)38获得聚合退出状态之前退出。

[0115] 参考图6,在实施例中,在处理器的指令集中提供SYNC(同步)指令。SYNC指令具有使监督者线程SV等待直到所有当前正在执行的工作者W已通过EXIT指令退出的效果。在实施例中,SYNC指令采取操作数(在实施例中是其唯一的操作数)的模式,该模式指定SYNC是否仅与那些与监督者在同一处理器模块4(例如同一瓦片)上本地运行的工作者线程相关联地仅本地作用,其中SYNC在该处理器模块4的部分上执行(即,只有通过同一桶形线程处理单元10的同一流水线13的线程);或者它是否跨多个瓦片或甚至跨多个芯片应用。

[0116] SYNC mode//mode∈{tile,chip,zone_1,zone_2}

[0117] 这将在后面更详细地讨论,但是出于图6的目的,将假设本地SYNC(“SYNC瓦片”,即单个瓦片内的同步)。

[0118] 工作者不需要被识别为SYNC指令的操作数,因为隐含的是然后使监督者SV自动等待,直到桶形线程处理单元10的时隙S0、S1....都没有被工作者占用。如图6所示,一旦当前批次的工作者WLn中的每一个都由监督者启动,监督者便会执行SYNC指令。如果监督者SV启动桶形线程处理单元10的所有时隙S0...3中(在所示示例中的全部四个,但这仅是一个示例性实现)的工作者W,则一旦当前的批次工作者线程WLn中的第一个离开了,SYNC便由监督者执行,从而将对至少一个时隙的控制交还给监督者SV。否则,如果工作者没有占用所有时隙,则只需在启动当前批次的WLn的最后一个线程后立即执行SYNC。无论哪种方式,SYNC使得监督者SV等待当前批次的全部其他工作者WLn-1在监督者可以继续之前执行EXIT。只有在此之后,监督者才执行GET指令以获得本地共识寄存器($LC)38的内容。一旦执行了SYNC,这种监督者线程的等待就会施加在硬件中。即,响应于SYNC指令的操作码,执行级18的执行单元(EXU)中的逻辑使得获取级14和调度器24暂停发出监督者线程的指令,直至所有余下的工作者线程都执行了EXIT指令。在获得本地共识寄存器($LC)38的值(可选地在其间具有一些其他监督者代码)之后的某个点刻,监督者执行PUT指令以重置本地共识寄存器($LC)38(在所示示例中重置为1)。

[0119] 还如图6所示,SYNC指令还可以用于在工作者线程的不同相互依赖层WL1、WL2、WL3...之间放置同步屏障,其中每个连续层中的一个或多个线程依赖于由其前一层中的一个或多个工作者线程输出的数据。由监督者线程执行的本地SYNC确保下一层WL+1中的工作者线程都不会执行,直至紧接在前的层WLn中的所有线程已经退出(通过执行EXIT指令)。

[0120] 如上所述,在实施例中,处理器模块4可以实现为形成多瓦片处理器的互连瓦片阵

列之一，其中每个瓦片可以如上面关于图1至图6所述那样配置。

[0121] 这在图7中进一步示出，图7示出了单芯片处理器2，即单个管芯，包括多个处理器瓦片4的阵列6和连接瓦片4之间的芯片上互连34。芯片2可以单独在其自己的单芯片集成电路封装上实现，或者作为封装在同一IC封装中的多个管芯之一实现。芯片上互连在本文也可以称为“交换结构”34，因为它使得瓦片4能够彼此交换数据。每个瓦片4包括桶形线程处理单元10和存储器11的相应实例，每个如上面关于图1至6所述那样布置。例如，作为说明，芯片2可以包括数百个，甚至超过一千个瓦片4的数量级。为了完整起见，还要注意，本文提到的“阵列”不一定意味着瓦片4的任何特定数量的维度或物理布局。

[0122] 在实施例中，每个芯片2还包括一个或多个外部链路8，使得芯片2连接到不同芯片上的一个或多个其他外部处理器(例如，同一芯片2的一个或多个其他实例)。这些外部链路8可以包括以下任何一个或多个：用于将芯片2连接到主处理器的一个或多个芯片到主机链路、用于在同一IC封装或卡上或在不同卡上的芯片2的一个或多个实例连接在一起的一个或多个芯片到芯片链路。在一个示例性布置中，芯片2从主处理器(未示出)接收采取要由芯片2处理的输入数据的形式的工作，该主处理器通过芯片到主机链路之一连接到芯片。芯片2的多个实例可以通过芯片到芯片链路一起连接到卡中。因此，主机可以访问被构造为可能布置在互连卡上的单芯片处理器2或多个单芯片处理器2的计算机，这取决于主机应用所需的工作负载。

[0123] 互连34配置为使阵列6中不同的处理器瓦片4能够与另一个芯片上2进行传送。然而，除了在相同瓦片4上的线程之间可能存在依赖性之外，在阵列6中不同的瓦片4运行的程序的部分之间也可能存在依赖性。因此，需要一种技术来防止一个瓦片4上的代码段在它所依赖于的数据被另一瓦片4上的另一段代码使其变得可用之前运行。

[0124] 在实施例中，这可以通过实现批量同步并行(BSP)交换方案来实现，如图8和9中示意性所示。

[0125] 根据BSP的一个版本，每个瓦片4在交换周期中执行计算阶段52和交换阶段50，其通过瓦片之间的屏障同步30彼此分离。在所示的情况下，在每个计算阶段52和随后的交换阶段50之间放置屏障同步。在计算阶段52期间，每个瓦片4在瓦片上本地执行一个或多个计算任务，但是不与任何其他瓦片4传送这些计算的任何结果。在交换阶段50期间，每个瓦片4被允许将来自先前计算阶段的计算的一个或多个结果向和/或从组中的一个或个其他瓦片交换，但是不会执行任何新计算，直至它已经从其他瓦片4接收到其任务依赖于的任何数据。除了在前面的计算阶段计算的数据之外，它也不向任何其他瓦片发送任何数据。不排除可以在交换阶段执行诸如内部控制相关操作的其他操作。在实施例中，交换阶段50不包括任何非时间确定性计算，但在交换阶段50期间可以可选地允许少量时间确定性计算。还要注意，在计算阶段52期间可以允许执行计算的瓦片4与正被同步的瓦片4阵列外部的其他外部系统资源通信，例如网卡、磁盘驱动器或现场可编程门阵列(FPGA)，只要这不及与被同步的组内的其他瓦片4的传送。瓦片组外部的传送可以可选地利用BSP机制，但是可选地可以不利用BSP，而是可以使用其自身的一些其他同步机制。

[0126] 此外，根据BSP原理，屏障同步30被置于从计算阶段52过渡到交换阶段50的接合点、或者从交换阶段50过渡到计算阶段52的接合点或两者。也就是说：(a)在允许组中的任何瓦片继续到下一个交换阶段50之前，需要所有瓦片4都完成它们各自的计算阶段52，或者

(b)在允许组中的任何瓦片继续到下一个计算阶段52之前,需要组中的所有瓦片4都完成其相应的交换阶段50,或者(c)这两个条件都被强制执行。在所有三种变体中,单独处理器在阶段之间交替,并且整个组装同步。然后,这个交换和计算阶段的顺序可以重复多次重复。在BSP术语中,交换阶段和计算阶段的每次重复有时被称为"超步"(superstep)(尽管注意在文献中这个术语并不总是一致地使用:有时每个单独交换阶段和计算阶段单独地被称为超步,而其他地方,如本文采用的术语中,交换和计算阶段一起被称为超步)。

[0127] 还要注意,不排除在同一芯片2或不同芯片上的多个不同独立的瓦片4组可以各自形成彼此异步的独立的相应BSP组,其中BSP周期的计算、同步和交换是仅在给定的组内施加,但每个组独立于其他组执行此操作。即,多瓦片阵列6可以包括多个内部同步组,每个与其他这样组独立地和异步地操作(稍后更详细地讨论)。在一些实施例中,存在同步和交换的分层分组,这将在下文更详细地讨论。

[0128] 图9示出了在阵列6中的一些或所有瓦片的组4i、4ii、4iii中实现的BSP原理,在这种情况下实施:(a)从计算阶段52到交换阶段50的屏障同步(见上文)。注意,在这种布置中,允许一些瓦片4开始计算52,而其他一些仍在交换。

[0129] 根据本文公开的实施例,可以通过将附加、特殊、专用的功能结合到执行屏障同步的机器代码指令(即SYNC指令)中来促进这种类型的BSP。

[0130] 在实施例中,SYNC函数在通过瓦片间模式(例如芯片上模式)限定为操作数时采用此功能:SYNC芯片。

[0131] 这在图10中示意性地示出。在每个瓦片4包括多线程处理单元10的情况下,每个瓦片的计算阶段52实际上可以包括由同一瓦片4上多个工作者线程W执行的任务(而且,给定瓦片4的给定计算阶段52可以包括工作者线程的一个或多个层WL,其在多个层的情况下可以使用SYNC指令通过内部屏障同步分离,其中本地瓦片上模式作为操作数)。一旦给定瓦片4上的监督者线程SV已经在当前BSP超步中启动了最后的工作者线程,则该瓦片4上的监督者执行SYNC指令,其中瓦片间模式设置为操作数:SYNC芯片。如果监督者要在其相应处理单元10的所有时隙中启动(RUN)工作者线程,一旦将不再需要运行当前BSP超步中的任何更多工作者的第一时隙交还给监督者,便尽快执行"SYNC芯片"。例如,这可能发生在第一线程退出最后层WL之后,或者如果只有一个层,则发生在第一工作者线程退出之后。否则,如果不是所有的时隙都用于运行当前BSP超步的工作者,一旦启动了需要在当前BSP超步中运行的最后工作者,便尽快执行"SYNC芯片"。这可能在一旦运行了最后层的所有工作者便发生,或者如果只有一层,则一旦运行了所有工作者线程便发生。

[0132] 执行级18的执行单元(EXU)被配置成:响应于SYNC指令的操作码,当由芯片上(瓦片间)操作数限定时,使得其中"SYNC芯片"被执行的监督者线程被暂停,直到阵列6中的所有瓦片4完成了工作者的运行。这可用于实现下一个BSP超步的屏障。即,在芯片2上的所有瓦片4已经通过屏障之后,跨瓦片程序作为整体可以前进到下一个交换阶段50。

[0133] 图11给出了说明根据本文公开的实施例的由"SYNC芯片"触发的逻辑的示意图。

[0134] 一旦监督者启动(RUN)了它企图在当前计算周期52中启动的所有线程,它就会以芯片上、瓦片间的操作数执行SYNC指令:SYNC芯片。这触发了在瓦片4上的专用同步逻辑39中以及在硬件互连34中实行的同步控制器36中触发以下功能。瓦片上同步逻辑39和互连34中的同步控制器36二者的这种功能在专用硬件电路中实现,使得一旦执行SYNC芯片,其余

的功能继续进行而无需执行另外的指令。

[0135] 首先,瓦片上同步逻辑39使得讨论中的瓦片4上的监督者的指令发出自动暂停(使得获取级14和调度器24暂停发出监督者的指令)。一旦本地瓦片4上的所有余下的工作者线程都已经执行退出(EXIT),则同步逻辑39自动向互连34中的同步控制器36发送同步请求“sync_req”。然后,本地瓦片4继续等待,并且暂停监督者指令发出。类似的过程也在阵列6中的每个其他瓦片4上实现(每个瓦片包括其自己的同步逻辑39的实例)。因此,在某个点处,一旦当前计算阶段52中的所有最终工作者已经在阵列6中的所有瓦片4上退出(EXIT),则同步控制器36将已从阵列6中的所有瓦片4接收到相应的同步请求(sync_req)。只有这样,响应于从同一芯片2上的阵列6中的每个瓦片4接收到sync_req,同步控制器36将同步确认信号“sync_ack”发送回到每个瓦片4上的同步逻辑39。直到此时,每个瓦片4都已暂停其监督者指令发出,等待同步确认信号(sync_ack)。在接收到sync_ack信号后,瓦片4中的同步逻辑39自动取消对该瓦片4上的相应监督者线程的监督者指令发出的暂停。然后,监督者可以在随后的交换阶段50中自由地继续通过互连34与其他瓦片4交换数据。

[0136] 优选地,通过互连34中将每个瓦片4连接到同步控制器36的一个或多个专用同步线,分别向同步控制器和从同步控制器发送和接收sync_req和sync_ack信号。

[0137] 此外,根据本文公开的实施例,SYNC指令中包括附加功能。即,至少当以瓦片间模式(例如,SYNC芯片)执行时,SYNC指令还使用每个同步的瓦片4的本地退出状态$LC自动聚合在互连34中的另一专用硬件40中。在所示的实施例中,该逻辑采用多输入AND门(阵列6中的每个瓦片4一个输入)的形式,例如,由双输入AND门40i、40ii.....的串形成,如图11通过示例所示。该瓦片间聚合逻辑40从阵列中的每个瓦片4接收本地退出状态寄存器(本地共识寄存器)$LC 38中的值(在实施例中每个都是单个位),并且将它们聚合成单个值,例如所有本地聚合退出状态的AND。因此,逻辑形成跨越阵列6中的所有瓦片4上的所有线程的全局聚合的退出状态。

[0138] 每个瓦片4包括全局共识寄存器($GC)42的相应实例,其被布置成接收和存储来自互连34中的全局聚合逻辑40的全局退出状态。在实施例中,这是监督者上下文寄存器文件CXS的另一个状态寄存器。响应于从阵列6中的所有瓦片4接收到同步请求(sync_req),同步控制器36使得聚合逻辑40的输出(例如AND的输出)存储在每个瓦片4上的全局共识寄存器($GC)42中(应当理解,图11所示的“开关”是功能的示意图,并且实际上可以通过任何合适的数字逻辑来实现更新)。一旦恢复了监督者指令发出,该寄存器$GC 42可由相应瓦片4上的监督者线程SV访问。在实施例中,将全局共识寄存器$GC实现为控制寄存器文件中的控制寄存器,使得监督者线程可以通过GET指令获得全局共识寄存器($GC)42中的值。注意,同步逻辑36等待,直至在更新任何全局共识寄存器($GC)42中的值之前从所有瓦片4接收到sync_req,否则使得不正确的值可以由未完成其计算阶段52部分并因此仍在运行的瓦片上的监督者线程访问。

[0139] 全局聚合的退出状态$GC使程序能够确定在多个不同瓦片4上运行的程序的各部分的整体结果,而不必单独检查每个单独的瓦片上的每个单独的工作者线程的状态。它可以用于程序员期望的任何目的。例如,在图11所示的示例中,其中全局聚合是布林AND,这意味着任何输入为0都会导致聚合为0,但如果所有输入为1,则聚合为1。即,如果1用于表示真或成功的结果,这意味着如果任何瓦片4的任何本地退出状态是假的或不成功的,则全局聚

合状态也将是假的或表示不成功的结果。例如,这可以用于确定在所有瓦片上运行的代码的部分是否都满足预定条件。因此,程序可以查询单个寄存器(在实施例中是单个位)来询问“有没有出现问题?有或没有?”或“图形中的所有节点是否都达到了可接收误差级别?是或否?”,而不需要检查每个单独瓦片上的单独工作者线程的单独状态(再次,在实施例中,除了通过退出状态寄存器38和42之外,监督者实际上不能查询工作者的状态)。换句话说,EXIT和SYNC指令每个将多个单独的退出状态减少成单个组合状态。

[0140] 在一个示例性用例中,一个或多个瓦片上的监督者可以向主处理器报告全局聚合是否指示了假的或不成功的结果。作为另一示例,程序可以根据全局退出状态执行分支决策。例如,程序检查全局聚合退出状态$GC,并根据此确定是否继续循环或是否在其他地方分支。如果全局退出状态$GC仍然是假的或不成功的,则程序继续迭代程序的相同的第一部分,但是,一旦全局退出状态$GC是真的或成功的,程序则分支到程序的不同的第二部分。分支决策可以在每个监督者线程中单独实现,或者由担任主管(master)角色并指示其他瓦片上的其他从属(slave)监督者的监督者之一实现(主管角色在软件中配置)。

[0141] 注意,图11中所示的聚合逻辑40仅是一个示例。在另一个等效示例中,AND可以用OR替换,并且可以反转0和1的解释(0→真,1→假)。等效地,如果AND门被替换为OR门,但是退出状态的解释和重置值未被反转,那么$GC中的聚合状态将记录有没有任何(而不是全部)瓦片以本地聚合状态1退出。在另一个示例中,全局退出状态$GC可以包括表示三元状态的两个位:所有瓦片的本地聚合退出状态$LC为状态1,所有瓦片的本地聚合退出状态$LC为状态0,或者瓦片的本地聚合退出状态$LC是混合的。作为另一个更复杂的示例,瓦片4的本地退出状态和全局聚合退出状态可以各自包括两个或更多个位,其可以用于例如表示瓦片4的结果的置信度。例如,每个单独的瓦片的本地聚合退出状态$LC可以表示相应瓦片4的结果中的置信度的统计概率测量,并且全局聚合逻辑40可以用更复杂的电路替代,用于执行硬件中的单独置信水平的统计聚合。

[0142] 如前所述,在实施例中,芯片2的多个实例可以连接在一起以形成跨越多个芯片2的甚至更大的瓦片4阵列。这在图12中说明。一些或全部芯片2可以在同一IC封装上实现,或者一些或全部芯片2可以在不同的IC封装上实现。芯片2通过外部互连72(通过图7中所示的外部链路8)连接在一起。除了提供用于在不同芯片上的瓦片4之间交换数据的管道之外,外部交换外围72还提供硬件支持,用于在不同芯片2上的瓦片4之间执行屏障同步,并且聚合不同芯片2上的瓦片4的本地退出状态。

[0143] 在实施例中,SYNC指令可以采用其模式操作数至少一个另外的可能值来指定外部(即芯片间)同步:SYNC zone_n,其中zone_n表示外部同步区域。外部互连72包括与关于图11描述的硬件逻辑类似的硬件逻辑,但是在外部芯片间的规模上。当使用在其操作数中指定的两个或更多个芯片2的外部同步区域执行SYNC指令时,这使得外部互连72中的逻辑以与关于内部互连34所描述的方式类似的方式操作,但是跨越指定的同步区域中的多个不同芯片2上的瓦片4操作。

[0144] 即,响应于外部SYNC,暂停监督者指令发出,直至外部同步区域中的所有芯片2上的所有瓦片4已完成其计算阶段52并提交了同步请求。此外,外部互连72中的逻辑聚合跨越讨论中的区域中的多个芯片2的所有这些瓦片4的本地退出状态。一旦外部同步区域中的所有瓦片4都发出了同步请求,外部互连72便向瓦片4发信号传回同步确认,并将跨芯片全局

聚合退出状态存储到讨论中的的所有瓦片4的全局共识寄存器($GC)42中。响应于同步确认,区域中的所有芯片2上的瓦片4恢复监督者的指令发布。

[0145] 在实施例中,互连72的功能可以在芯片2中实现,即逻辑可以分布在芯片2之间,使得仅需要芯片之间的有线连接(图11和12是示意性的)。

[0146] 所提及的同步区域内的所有瓦片4都被编程为通过其相应的SYNC指令的模式操作数指示相同的同步区域。在实施例中,外部互连72外周中的同步逻辑配置为使得如果不是编程误差或其他误差(例如存储器奇偶误差)的情况,则一些或所有瓦片4不会接收到确认,因此系统将在下一个外部屏障停止,从而允许管理外部CPU(例如主机)进行干预以进行调试系统恢复。在其他实施例中,在同步区域不匹配的情况下引发误差。然而,优选地,编译器配置为确保相同区域的瓦片全部在相关时间都指示相同正确的同步区域。

[0147] 图13示出了涉及内部(芯片上)和外部(芯片间)同步的示例性BSP程序流程。如图所示,优选地保持内部交换50(同一芯片2上的瓦片4之间的数据的交换)与外部交换50′(不同芯片2上的瓦片4之间数据的交换)分开。其中一个原因是,就延迟和负载平衡复杂性而言,跨多个芯片的全局交换(其由全局同步划分)将比仅仅芯片上同步和交换更“昂贵”。另一个可能原因是,通过内部(芯片上)互连34的数据交换可以是时间确定性的,而在实施例中,通过外部互连72的数据交换可以是非时间确定性的。在这种情况下,分开内部交换和外部交换可能是有用的,这样外部同步和交换过程不会“污染”内部同步和交换。

[0148] 因此,为了实现这种分开,在实施例中,程序布置为执行一系列同步、交换阶段和计算阶段,按照以下顺序:(i)第一计算阶段,然后(ii)内部屏障同步30,然后(iii)内部交换阶段50,然后(iv)外部屏障同步80,然后(v)外部交换阶段50’。参见图13中的芯片2II。在内部交换阶段50之后,施加外部屏障80,使得程序仅在内部交换50之后继续到外部交换50’。还要注意,如参照图12中的芯片2I所示,可选地,可以在内部交换(iii)和外部屏障(iv)之间包括计算阶段。整个顺序由程序执行(例如由编译器生成成这样),并且内部同步和交换不延伸到另一芯片2上的任何瓦片或其他实体。可以在一系列整体迭代中重复序列(i)-(v)(iii和iv之间具有上述可选的计算阶段)。每次迭代,在外部同步和交换之前,可能存在内部计算、同步和交换(i)-(iii)的多个实例。

[0149] 注意,在外部交换50期间,通信不仅限于外部:一些瓦片可以仅执行内部交换,一些可以仅执行外部交换,并且一些可以执行两者的混合。还要注意,如图13所示,在任何给定的BSP超步中具有空计算阶段52或空交换阶段50通常是可能的。

[0150] 在一些实施例中,也如图13所示,一些瓦片4可以在计算阶段期间执行本地输入/输出,例如它们可以与主机交换数据。

[0151] 如图14所示,在实施例中,SYNC指令的模式可用于指定多个不同的可能外部同步区域中的一个,例如zone_1或zone_2。在实施例中,这些对应于不同的层级。也就是说,每个较高层级92(例如区域2)包含至少一个较低层级的两个或更多个区域91A、91B。在实施例中,仅存在两个层级,但不排除更多数量的嵌套级别。如果SYNC指令的操作数设置为外部同步区域(SYNC zone_1)的较低层级,则相对于仅在执行SYNC的瓦片的相同较低层外部同步区域中的芯片2上的瓦片4执行上述同步和聚合操作。另一方面,如果SYNC指令的操作数设置为外部同步区域(SYNC zone_2)的较高层级,则相对于在与执行SYNC的瓦片相同的较高层外部同步区域中的所有芯片2上的所有瓦片自动执行上述同步和聚合操作。在实施例中,

同步区域最高层级包含所有芯片,即它用于执行全局同步。当使用多个较低级别区域时,可以在每个区域内的芯片2上的瓦片4组内部施加BSP,但是每个区域可以相对于彼此异步操作,直至执行全局同步。

[0152] 注意在其他实施例中,可以由SYNC指令的模式指定的同步区域本质上不限于分层。通常,可以为SYNC指令提供与任何类型的分组相对应的模式。例如,模式可以能够仅从非分层组中进行选择,或者从分层组和一个或多个非分层组的混合中进行选择(其中至少一个组不完全嵌套在另一个组中)。这有利地使程序员或编译器在最小代码密度下灵活地在相互异步的内部同步组中的不同布局之间进行选择。

[0153] 在图16中示出了用于在所选同步组91和92之间实现同步的示例性机制。如图所示,外部互连72中的外部同步逻辑76包括与每个相应芯片2相关联的相应同步块95。每个同步块95包括相应的门逻辑和相应的同步聚合器。门逻辑包括硬件电路,其以菊花链拓扑将芯片2连接在一起以实现同步和退出状态聚合,并且根据以下所述传播同步和退出状态信息。同步聚合器包括配置成根据以下所述聚合同步请求(sync_req)和退出状态的硬件电路。

[0154] 与每个芯片2相关联的相应同步块95连接到其相应的芯片2,使得它可以检测由该芯片2引发的同步请求(Sync_req)和该芯片2的退出状态,并且使得它可以将同步确认(Sync_ack)和全局退出状态送回相应的芯片2。与每个芯片2相关联的相应同步块95还通过包括四条同步线96的线束的外部同步接口连接到至少另一个芯片2的同步块95,其细节将在后面详细讨论。这可以是芯片到芯片链路8之一的一部分。在不同卡上的芯片2之间的链路的情况下,接口8可以例如包括PCI接口,并且可以通过重新使用PCI接口的四条线来实现四条同步线96。一些芯片的同步块95连接到两个相邻芯片2的同步块,每个连接通过四条同步线96的相应实例。这样,芯片2可以通过它们的同步块95以一个或多个菊花链连接。这使得同步请求、同步确认、退出状态的运行聚合和全局退出状态能够在链中上和下传播。

[0155] 在操作中,对于每个同步组91和92,与该组中的一个芯片2相关联的同步块95被设置为用于同步和退出状态聚合目的的主管,该组中其余的同步块则是用于这种目的的从属。每个从属同步块95配置有它为了每个同步组91和92传播同步请求、同步确认和退出状态所需的方向(例如,左或右)(即,朝向主管的方向)。在实施例中,这些设置可由软件配置,例如在初始配置阶段中配置,之后该配置在整个系统的后续操作中一直保持不变。例如,这可以由主处理器配置。替代地,不排除配置可以是硬连线的。无论是哪种方法,不同的同步组91和92可以具有不同的主管,并且通常可以使给定的芯片2(或者是其同步块95)成为一个组的主管,而不是其中它是成员的另一个组的主管,或者成为多个组的主管。

[0156] 例如,作为说明,考虑图16的示例性场景。例如,芯片2IV的同步块95被设置为给定同步组91A的主管。现在,考虑芯片2链中的第一芯片2I,其通过它们的同步块95和线96最终连接到芯片2IV。当第一芯片2I上的当前计算阶段的所有工作者线程都执行了EXIT指令,而且所有(参与)瓦片4上的监督者都执行了指定同步组91A的SYNC指令,则第一芯片2I发信号通知其相应的相关同步块95其同步准备度。芯片2I还向其相应的同步块95输出其芯片级聚合退出状态(相应芯片2I上的所有参与瓦片上的所有现有工作者的聚合)。作为响应,第一芯片2I的同步块95将同步请求(Sync_req)传播到链中的下一个芯片2II的同步块95。它还将第一芯片2I的退出状态传播到该下一个芯片2II的同步块95。该第二芯片2II的同步块95

等待，直至其自己的(参与)瓦片4的监督者全部都执行了指定同步组91A的SYNC指令，导致第二芯片2II发信号通知其同步准备度。然后，第二芯片的同步块95才会将同步请求传播给链中的下一个(第三)芯片2III的同步块95，并且还传播第一芯片2I与第二芯片2II的退出状态的运行聚合。如果第二芯片2II在第一芯片2I之前已经准备好同步，则在将同步请求传播到第三芯片2III的同步块95之前，第二芯片2II的同步块95将等待第一芯片2I以信号发出同步请求。第三芯片2III的同步块95以类似的方式运行，这次聚合来自第二芯片2II的运行聚合退出状态以获得下一个运行聚合以向前传递等。这继续朝向主管同步块，在这个示例中是芯片2IV的主管同步块。

[0157] 然后，主管的同步块95基于其接收的运行聚合和其自己的芯片2IV的退出状态来确定所有退出状态的全局聚合。它将此全局聚合连同同步确认(Sync_ack)沿着链传播回到所有芯片2。

[0158] 如果主管是沿着链的一部分，而不是如上例中那样位于一端，那么同步和退出状态信息在主管的任一侧的相反方向上传播，两侧朝向主管。在这样情况下，一旦接收到来自两侧的同步请求，主管才会发出同步确认和全局退出状态。例如，考虑芯片2III是组92的主管的情况。此外，在实施例中，一些芯片2的同步块95可以连接到三个或更多个其他芯片2的同步块，从而创建朝向主管的链的多个分支。然后，每个链如上述运行，并且一旦接收到来自所有链的同步请求，主管才会发出同步确认和全局退出状态。和/或，一个或多个芯片2可以连接到外部资源，例如主处理器、网卡、存储装置或FPGA。

[0159] 在实施例中，以下述方式实现同步信息和退出状态信息的信号发送。每对芯片2之间的四条同步线96的线束包括两对线：第一对96_0和第二对96_1。每对包括同步请求线的实例和同步确认线的实例。为了以信号发出值0的运行聚合退出状态，发送芯片2的同步块95在以信号发出同步请求(sync_req)时使用第一线对96_0的同步请求线，或者，为了以信号发出值1的运行聚合，同步块95在以信号发出同步请求时使用第二线对96_1的同步请求线。为了以信号发出值0的全局聚合退出状态，发送芯片2的同步块95在以信号发出同步确认(sync_ack)时使用第一线对96_0的同步确认线，或者，为了以信号发出值1的全局聚合，同步块95在以信号发出同步确认时使用第二线对96_1的同步请求线。

[0160] 注意，以上仅是用于传播同步信息和退出状态信息的机制。实际数据(内容)由另一个信道发送，例如稍后参考图16所讨论。此外，应当理解，这仅是一个示例性实现，并且一旦给出了本文公开的功能的说明，技术人员将能够构建用于实现所公开的同步和聚合功能的其他电路。例如，同步逻辑(图18中的95)可以使用通过互连34、72承载的分组作为专用布线的替代。例如，sync_req和/或sync_ack可以各自以一个或多个分组的形式发送。

[0161] 下面概述了SYNC指令在不同可能模式下的功能。

[0162] SYNC瓦片(执行本地、瓦片上屏障同步)

[0163] • 监督者运行模式从执行转换到等待工作者退出

[0164] • 暂停监督者线程的指令发布，直到所有工作者线程都非活动

[0165] • 当所有工作者线程都是非活动时，聚合的工作者退出状态通过本地共识寄存器($LC)38可用。

[0166] SYNC芯片(执行内部、芯片上屏障同步)

[0167] • 监督者运行模式从执行转换到等待工作者退出

[0168] • 暂停监督者线程的指令发布，直到所有工作者线程都非活动

[0169] • 当所有工作者线程都是非活动时：

[0170] -聚合的本地工作者退出状态通过本地共识寄存器($LC)38可用

[0171] -以信号将内部同步参与发给交换结构34

[0172] -监督者维持非活动，直至瓦片4从交换结构34接收到内部同步确认

[0173] -系统范围的退出状态在全局共识寄存器($GC)42中更新。

[0174] SYNC zone_n(执行跨区域n的外部屏障同步)

[0175] • 监督者运行模式从执行转换到等待工作者退出

[0176] • 暂停监督者线程的指令发布，直到所有工作者线程都非活动

[0177] • 当所有工作者线程都是非活动时：

[0178] -聚合的工作者退出状态通过本地共识寄存器($LC)38可用

[0179] -以信号将外部同步参与发给外部系统，例如上述外部互连72中的同步逻辑

[0180] -监督者维持非活动，直至瓦片4从外部系统72接收到外部同步确认

[0181] -系统范围的退出状态在全局共识寄存器($GC)42中更新。

[0182] 如前所述，并非所有瓦片4都必须参与同步。在实施例中，如所讨论的，参与瓦片的组可以由同步指令的模式操作数设置。但是，这仅允许选择预定义的瓦片的组。在此认识到，还期望能够逐个瓦片地选择同步参与。因此，在实施例中，提供了替代或附加的机制，用于选择哪些单独的瓦片4参与屏障同步。

[0183] 特别地，这是通过在处理器指令集中提供附加类型的指令来实现的，其由一个或一些瓦片4代替SYNC指令执行。该指令可以称为“弃权”指令或“SANS”(开始自动非参与同步)指令。在实施例中，SANS被保留由监督者线程使用。在实施例中，它采用单个立即操作数：

[0184] SANS n_barriers

[0185] SANS指令的行为是使得执行它的瓦片放弃当前的屏障同步，但不会阻延等待指定同步组中所有瓦片进行SYNC的其他瓦片。实际上它说“在没有我的情况下继续”。当执行SANS指令时，SANS指令的操作码触发执行级18的执行单元中的逻辑向内部和/或外部同步控制器36和76(取决于模式)发送同步请求信号(Sync_req)的实例。在实施例中，由SANS生成的同步请求适用于包含执行SANS的瓦片4的任何同步组91和92。即，不管该本地芯片中的瓦片4接下来使用什么同步组(它们必须同意该同步组)，来自执行了SANS的那些的sync_req将总是有效的。

[0186] 因此，从同步控制器逻辑36和76和同步组中的其他瓦片4的角度来看，执行SANS指令的瓦片4看起来与执行SYNC指令的瓦片4完全相同，并且不会阻延同步屏障和从同步逻辑36和76发送同步确认信号(Sync_ack)。即，执行SANS而不是SYNC的瓦片4不会阻延或停止涉及其中讨论中的瓦片是成员的任何同步组的任何其他瓦片4。由SANS执行的任何握手对所有同步组91和92都有效。

[0187] 然而，与SYNC指令不同，SANS指令不会导致监督者指令发出暂停等待来自同步逻辑36和76的同步确认信号(Sync_ack)。相反，相应的瓦片可以继续不受在执行SYNC指令的其他瓦片4之间进行的当前屏障同步所禁止。因此，通过模仿同步但不等待，SANS指令允许其瓦片4继续处理一个或多个任务，同时仍允许其他瓦片4同步。

[0188] 操作数n_barriers指定“已发布”同步的数量，即瓦片不会参与未来同步点(屏障)的数量。替代地，不排除在其他实施例中SANS指令不采用该操作数，而是SANS指令的每次执行仅引起一次性弃权。

[0189] 通过SANS指令，某些瓦片4可以负责执行BSP操作调度的直接范围之外的任务。例如，可能期望在芯片2内分配少量的瓦片4以启动(和处理)到主机存储器和/或从主机存储器的数据传输，同时大多数瓦片4被主要计算任务占用。在这些情况下，那些不直接参与主要计算的瓦片4可以使用自动非参与同步特征(SANS)声明它们与同步机制有效地断开连接了一段时间。当使用该特征时，瓦片4不需要主动(即通过执行SYNC指令)发信号通知其对同步的准备度(对于任何同步区域)，并且在实施例中对聚合的退出状态作出零贡献。

[0190] SANS指令开始或延伸一段时间，在该时段期间执行它的瓦片4将放弃主动参与瓦片间同步(或者，如果它们也参与同步，则放弃参与与其他外部资源的同步)。在此时段期间，该瓦片4可以在所有区域内自动发信号通知其对同步的准备度，并且在在实施例中还对全局聚合共识$GC作出零贡献。该时间段可以表示为无符号立即操作数(n_barriers)，其指示该瓦片4将自动发信号通知多少个附加的未来同步点。在执行SANS时，由其操作数指定的值n_barriers被放置在相应的瓦片4的倒计时寄存器$ANS_DCOUNT中。这是一段架构状态，用于跟踪应该作出多少个附加的未来sync_req。如果自动非参与同步机制当前处于非活动状态，则立即执行第一准备度断言(同步请求sync_req)。一旦完成了先前的同步，将在背景发生后续断言(即，同步确认sync_ack的断言之后)。如果自动非参与同步机制当前处理活动状态，则倒计时计数器寄存器$ANS_DCOUNT将以自动方式更新，使得没有同步确认信号会不被考虑。自动非参与同步机制在专用硬件逻辑中实现，优选地在每个瓦片4中实现，尽管在其他实施例中，不排除它可以对于瓦片组或所有瓦片集中实现。

[0191] 关于退出状态行为，实际上取决于实现存在许多可能性。在实施例中，为了获得全局聚合的退出状态，同步逻辑36和76仅聚合来自执行SYNC指令的指定同步组中的那些瓦片4的本地退出状态，而不是那些执行SANS指令的(弃权瓦片)。替代地，通过聚合来自执行SYNC的同步组中和执行SANS的所有瓦片4(参与和弃权瓦片4)的本地退出状态，获得全局聚合的退出状态。在后一种情况下，由弃权瓦片4输出以用于全局聚合的本地退出状态输出可以是在执行SANS时该瓦片的工作者的实际本地聚合退出状态，正如SYNC指令一样(参见本地共识寄存器$LC 38的描述)。替代地，由弃权瓦片4输出的本地“退出状态”可以是默认值，例如在退出状态是二进制的实施例中是真值(例如逻辑1)。在任何假的本地退出状态导致全局退出状态是假的实施例中，这防止了弃权瓦片4干扰全局退出状态。

[0192] 关于全局退出状态的返回，存在两种可能性，不论弃权瓦片是否提交本地退出状态以用于生成全局聚合，并且不论该值是实际值还是默认值。也就是说，在一个实现中，由互连34和72中的同步逻辑36和76产生的全局聚合退出状态仅存储在执行SYNC指令的参与瓦片4(而不是执行SANS指令的弃权瓦片4)的全局共识寄存器$GC 42中。在实施例中，将默认值存储在执行SANS的瓦片4(弃权瓦片)的全局共识寄存器$GX 42中。例如，在二进制全局退出状态的情况下，该默认值可以是真的，如逻辑1。然而，在替代的实现中，由同步逻辑36和76产生的实际全局聚合存储在执行SYNC指令的参与瓦片4和执行SANS指令的弃权瓦片4两者的全局共识寄存器$GC 42中。因此，组中的所有瓦片仍然可以访问全局聚合的退出状态。

[0193] 图15示出了本文所公开的处理器架构的示例性应用,即,对机器智能的应用。

[0194] 如机器智能领域的技术人员所熟悉的,机器智能从学习级开始,其中机器智能算法学习知识模型。该模型包括互连节点(即顶点)102和边缘(即链路)104的图形。图形中的每个节点102具有一个或多个输入边缘和一个或多个输出边缘。一些节点102的一些输入边缘是一些其他节点的输出边缘,从而将节点连接在一起以形成图形。此外,一个或多个节点102的一个或多个输入边缘作为整体形成对图形的输入,而一个或多个节点102的一个或多个输出边缘作为整体形成图形的输出。有时,给定的节点甚至可能具有以下所有这些:对图形的输入、图形的输出和对其他节点的连接。每个边缘104传送值或更常传送张量(n维矩阵),这些值分别形成在其输入边缘和输出边缘上提供给节点102和从节点102提供的输入和输出。

[0195] 每个节点102表示在其输入边缘上接收的一个或多个输入的函数,该函数的结果是在输出边缘上提供的输出。由一个或多个相应参数(有时称为权重,尽管它们不需要一定是乘法权重)对每个函数进行参数化。通常,由不同节点102表示的函数可以是不同形式的函数和/或可以通过不同参数来参数化。

[0196] 此外,每个节点的函数的一个或多个参数中的每一个的特征在于相应的误差值。此外,相应的条件可以与每个节点102的参数中的误差相关联。对于表示由单个参数参数化的函数的节点102,条件可以是简单阈值,即如果误差在指定的阈值内,则满足条件,但如果误差超出阈值,则不满足条件。对于由多于一个相应参数参数化的节点102,对于该节点102达到了可接受的误差水平的条件可能更复杂。例如,仅当该节点102的每个参数都落入相应阈值内时,才可以满足条件。作为另一示例,可以组合相同节点102的不同参数的误差来定义组合度量,并且在组合度量的值落入指定阈值的条件下可以满足条件,但是如果组成度量值超出阈值,则不满足条件(或者反之亦然,取决于度量的定义)。无论条件如何,这都给出了节点的参数中的误差是否低于某一可接受程度或水平的度量。通常,可以使用任何合适的度量。条件或度量对于所有节点可以是相同的,或者对于不同的相应节点可以是不同。

[0197] 在学习级中,算法接收经验数据,即,表示对图形的输入的不同可能组合的多个数据点。随着接收到越来越多的经验,算法基于经验数据逐渐调整图形中各节点102的参数,以试图最小化参数中的误差。目标是找到参数的值,使得图形的输出尽可能接近给定输入的期望输出。由于图形作为整体倾向于这样状态,因此图形会被形容为收敛。在适当程度的收敛之后,图形可以用于执行预测或推断,即预测某些给定输入的结果或推断某些给定输出的原因。

[0198] 学习级可以采用多种不同的形式。例如,在监督方法中,输入经验数据采用训练数据的形式,即与已知输出相应的输入。对于每个数据点,算法可以调整参数,使得输出更接近地匹配给定输入的已知输出。在随后的预测级中,图形然后可以用于将输入查询映射到近似预测输出(或者如果进行推断,则反之亦然)。其他方法也是可能的。例如,在无监督的方法中,不存在每个输入数据的参考结果的概念,而是改为将机器智能算法留下来在输出数据中识别其自己的结构。或者在强化方法中,算法针对输入经验数据中的每个数据点试验至少一个可能的输出,并且被告知该输出是正还是负(以及潜在地它是正还是负的程度),例如赢或输、奖励或惩罚等等。在许多试验中,算法可以逐渐调整图形的参数,以便能够预测将导致正结果的输入。用于学习图形的各种方法和算法对于机器学习领域的技术人

员来说是已知的。

[0199] 根据本文所公开的技术的示例性应用,每个工作者线程被编程为执行与机器智能图形中相应的单独一个节点102相关联的计算。在这种情况下,节点102之间的至少一些边缘104对应于线程之间的数据交换,并且一些可以涉及瓦片之间的交换。此外,程序员使用工作者线程的单独退出状态来表示相应的节点102是否满足到其对于该节点的参数的收敛的相应条件,即,参数中的误差是否落入误差空间的可接受水平或区域内。例如,这是实施例的一个示例性使用,其中每个单独的退出状态是单独的位,并且聚合的退出状态是单独退出状态的AND(或等效地,如果0被认为是正的,则是OR);或者,其中聚合的退出状态是三元值,表示各个退出状态都是真的,都是假的,还是混合的。因此,通过检查退出状态寄存器38中的单个寄存器值,程序可以确定图形作为整体或图形的至少一个子区域已经收敛到可接受的程度。

[0200] 作为其另一变体,可以使用其中聚合采用单独置信度值的统计聚合的形式的实施例。在这种情况下,每个单独的退出状态表示由相应线程表示的节点的参数已达到可接受的误差程度的置信度(例如作为百分比)。然后,可以使用聚合的退出状态来确定对于图形或图形的子域是否已经收敛到可接受程度的总体置信度。

[0201] 在多瓦片布置6的情况下,每个瓦片运行图形的子图。每个子图包括包含一个或多个监督者线程的监督者子程序,以及工作者线程集,其中一些或所有工作者可以采用小代码的形式。

[0202] 在这样的应用中,或者实际上任何基于图形的应用中,每个工作者线程用于表示图形中的相应节点,每个工作者所包含的“小代码”可以被定义为在一个顶点的持久状态和输入和/或输出上操作的软件过程,其中小代码:

[0203] • 由执行“运行”指令的监督者线程在一个工作者线程寄存器上下文上启动以在一个桶形时隙中运行;

[0204] • 在没有与其他小代码或监督者通信的情况下(除了当小代码退出时,返回监督者的情况)运行至完成;

[0205] • 能够通过“运行”指令提供的存储器指针访问顶点的持久状态,并以能够访问对该桶形时隙是专用的存储器中的非持久工作区域;和

[0206] • 执行“EXIT”作为其最后的指令,由此其使用的桶形时隙被发回给监督者,并且由退出指令指定的退出状态与对监督者可见的瓦片的本地退出状态聚合。

[0207] 更新图形(或子图形)意味着以与边缘定义的因果关系一致的任何顺序更新每个组成顶点一次。更新顶点意味着在顶点状态上运行小代码。小代码是用于顶点的更新过程,一个小代码通常与很多顶点相关联。监督者每个顶点执行一个RUN指令,每个这种指令指定顶点状态地址和小代码地址。

[0208] 应当理解,仅通过示例方式描述了上述实施例。

[0209] 例如,退出状态聚合机制的适用性不限于上述架构,其中为监督者线程提供独立的上下文,或者其中监督者线程在某时隙中运行然后将其时隙放弃给工作者。在另一种布置中,例如,监督者可以在其自己专用的时隙中运行。

[0210] 另外,除非另外明确说明,否则术语“监督者”和“工作者”不暗示具体责任,并且特别地本身并不一定限制于上述方案,其中监督者线程将其时隙放弃给工作者等等。通常,工

作者线程可以指分配了某些计算任务的任何线程。监督者可以代表任何类型的监视或协调线程，负责以下行动：将工作者分配给桶形时隙，和/或执行多个线程之间的屏障同步，和/或根据多于一个线程的结果执行任何控制流操作(例如分支)。

[0211] 在参考交错时隙顺序等的情况下，这并不一定意味着所指的顺序构成所有可能或可用的时隙。例如，讨论中的的顺序可以是所有可能的时隙或仅是当前活动的时隙。不一定排除可能存在当前未包括在调度顺序中的其他潜在时隙。

[0212] 本文所用的术语瓦片不一定限于任何特定的拓扑图等，并且通常可以指处理资源的任何模块化单元，包括处理单元10和相应的存储器11，其在相似模块的阵列中，通常至少一些在相同的芯片(即同一管芯)上。

[0213] 此外，本公开的范围不受时间确定性内部互连或非时间确定性外部互连所限制。本文公开的同步和聚合机制还可用于完全不同的时间确定性布置，或完全不同的非时间确定性布置。

[0214] 此外，除非明确指出，否则本文参考在瓦片组或多个瓦片等之间执行同步或聚合时，不一定必须参考系统中的所有瓦片或芯片上的所有瓦片。例如，SYNC和EXIT指令可以配置为仅对于给定芯片上的瓦片4的特定子集和/或仅给定系统中的芯片2的子集执行同步和聚合。而给定芯片上的一些其他瓦片4和/或给定系统中的一些其他芯片可能不涉及在给定的BSP组中，甚至可以用于与由手头上的组正在执行的计算无关的一些完全独立的任务集。

[0215] 此外，虽然上文描述了SYNC指令的某些模式，但是本公开范围更一般地不限于这些模式。例如，上面给出的模式列表不一定是详尽无遗的。或者，在其他实施例中，SYNC指令可以具有更少模式，例如，SYNC不一定支持外部同步的不同层级，或者不需要区分芯片上和芯片间同步(即，在瓦片间模式中，总是相对于所有瓦片作用，而不论是芯片上还是芯片外)。在另外替代实施例中，SYNC指令完全不需要作为操作数的模式。例如，在实施例中，可以为不同级别的同步和退出状态聚合提供不同版本的同步指令(不同的操作码)(例如用于瓦片上同步和瓦片间、芯片上同步的不同的SYNC指令)。或者，在实施例中，专用SYNC指令可以仅提供以用于瓦片间同步(如有需要，让线程之间的瓦片上同步在通用软件中执行)。

[0216] 此外，同步区域不限于是分层的(即，一个嵌套在另一个中)，并且在其他实施例中，可选择的同步区域可以由一个或多个非分层组组成或包括一个或多个非分层组(该组的所有瓦片不嵌套在单个其他可选组内)。

[0217] 此外，在实施例中，上述同步方案不排除多瓦片处理器以外的外部资源(例如诸如主处理器的CPU处理器)，甚至不是处理器的一个或多个部件(例如一个或多个网卡、存储装置和/或FPGA)的参与。例如，一些瓦片可以选择与外部系统进行数据传输，其中这些传输形成该瓦片的计算负担。在这种情况下，传输应在下一个屏障之前完成。在一些情况下，瓦片的退出状态可以取决于与外部资源通信的结果，并且该资源可以间接地影响退出状态。替代地或附加地，除了多瓦片处理器之外的资源(例如主机或一个或多个FPGA)可以并入同步网络本身中。也就是说，这种/这些额外的资源需要诸如Sync_req的同步信号，从而满足屏障同步和让瓦片继续下一个交换阶段。此外，在实施例中，聚合的全局退出状态可以在聚合中包括外部资源的退出状态，例如来自FPGA的。

[0218] 一旦给出了本文的公开内容，所公开的技术的其他应用和变体对本领域技术人员而言可变得显而易见。本公开的范围不由所描述的实施例来限定，而是仅由所附权利要求

来限制。

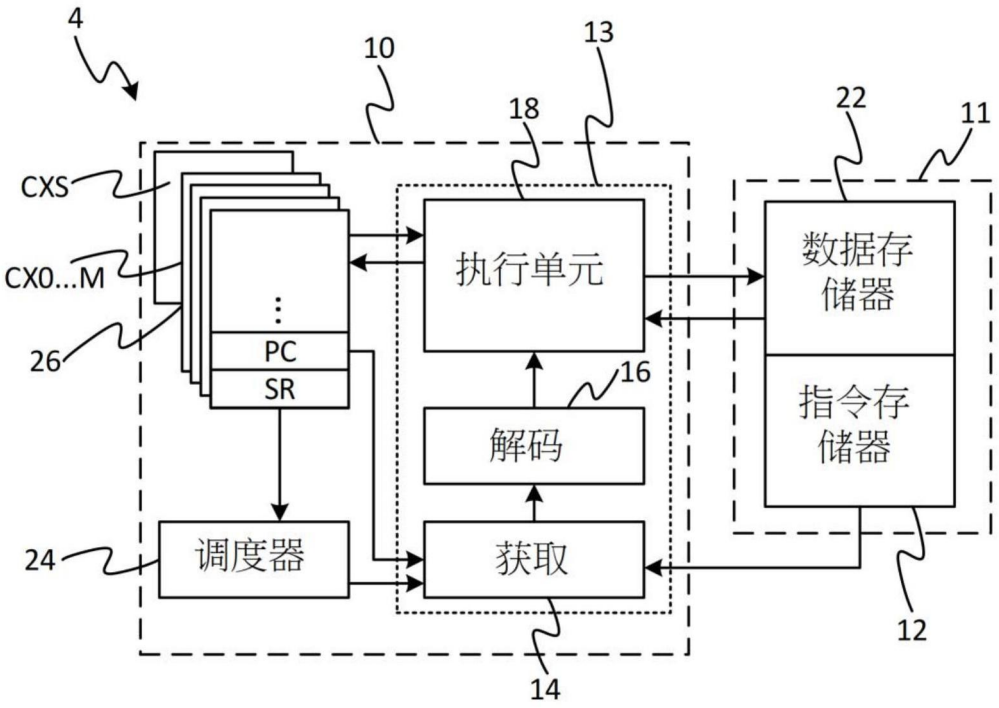

4
10
13
18
22
11
CXS
CX0...M
26
PC
SR
执行单元
数据存储器
指令存储器
16
解码
24
调度器
获取
12
14

图1

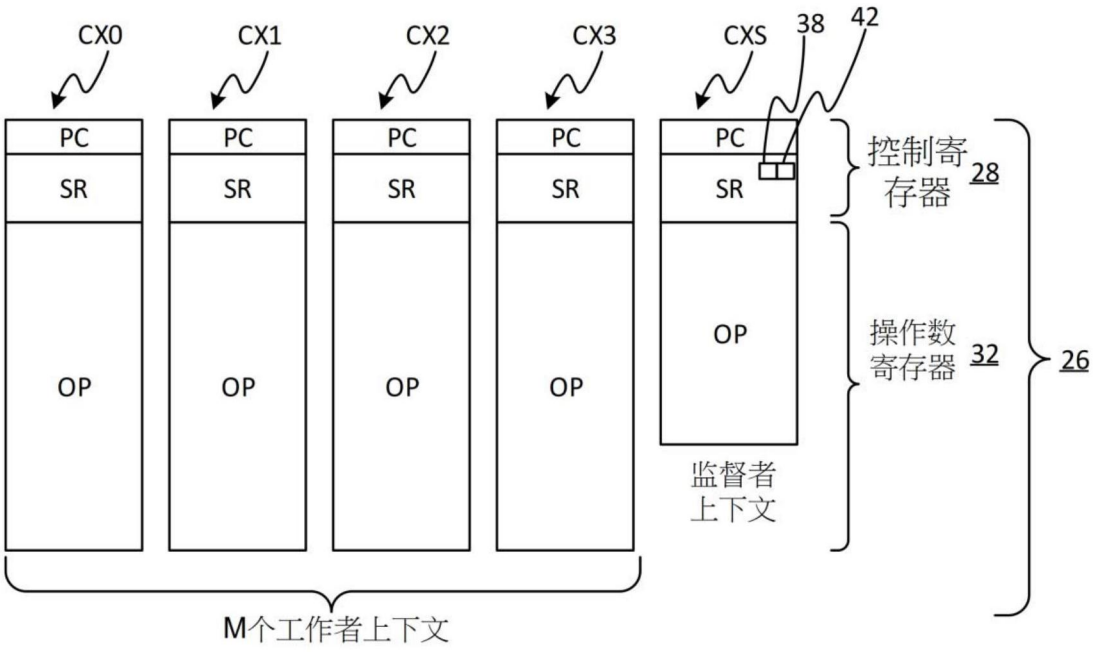

CX0
CX1
CX2
CX3
CXS
38
42
PC
SR
OP
控制寄存器 28
操作数寄存器 32
26
监督者上下文
M个工作者上下文

图2

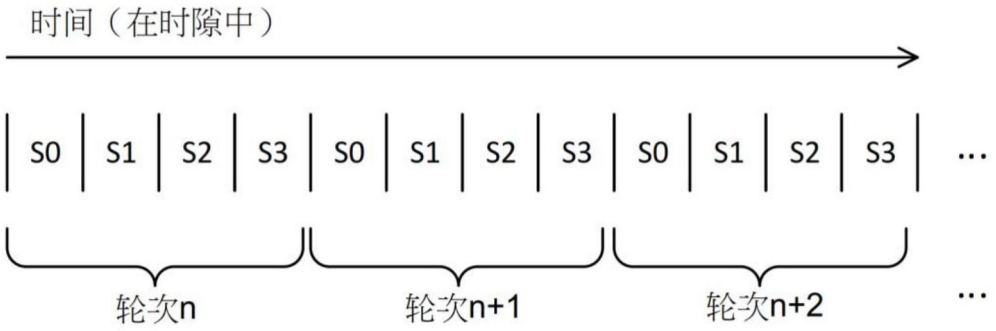

时间（在时隙中）
S0
S1
S2
S3
S0
S1
S2
S3
S0
S1
S2
S3
…
轮次n
轮次n+1
轮次n+2
…

图3

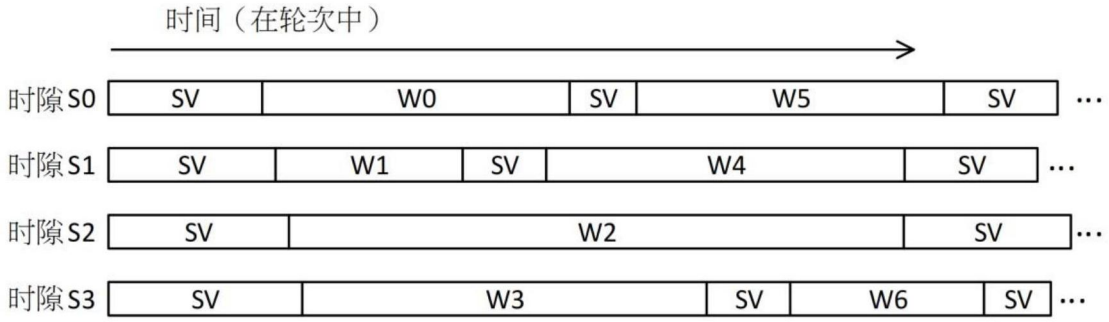

时间（在轮次中）
时隙S0
SV
W0
SV
W5
SV
…
时隙S1
SV
W1
SV
W4
SV
…
时隙S2
SV
W2
SV
…
时隙S3
SV
W3
SV
W6
SV
…

图4

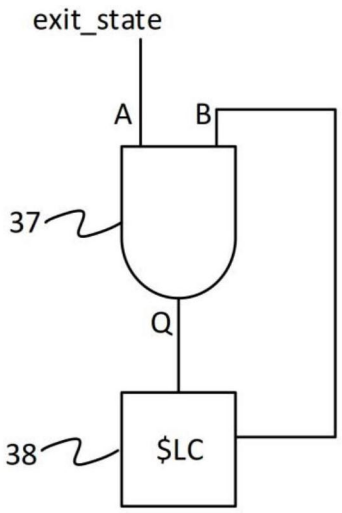

exit_state
A
B
37
Q
38
$LC

图5

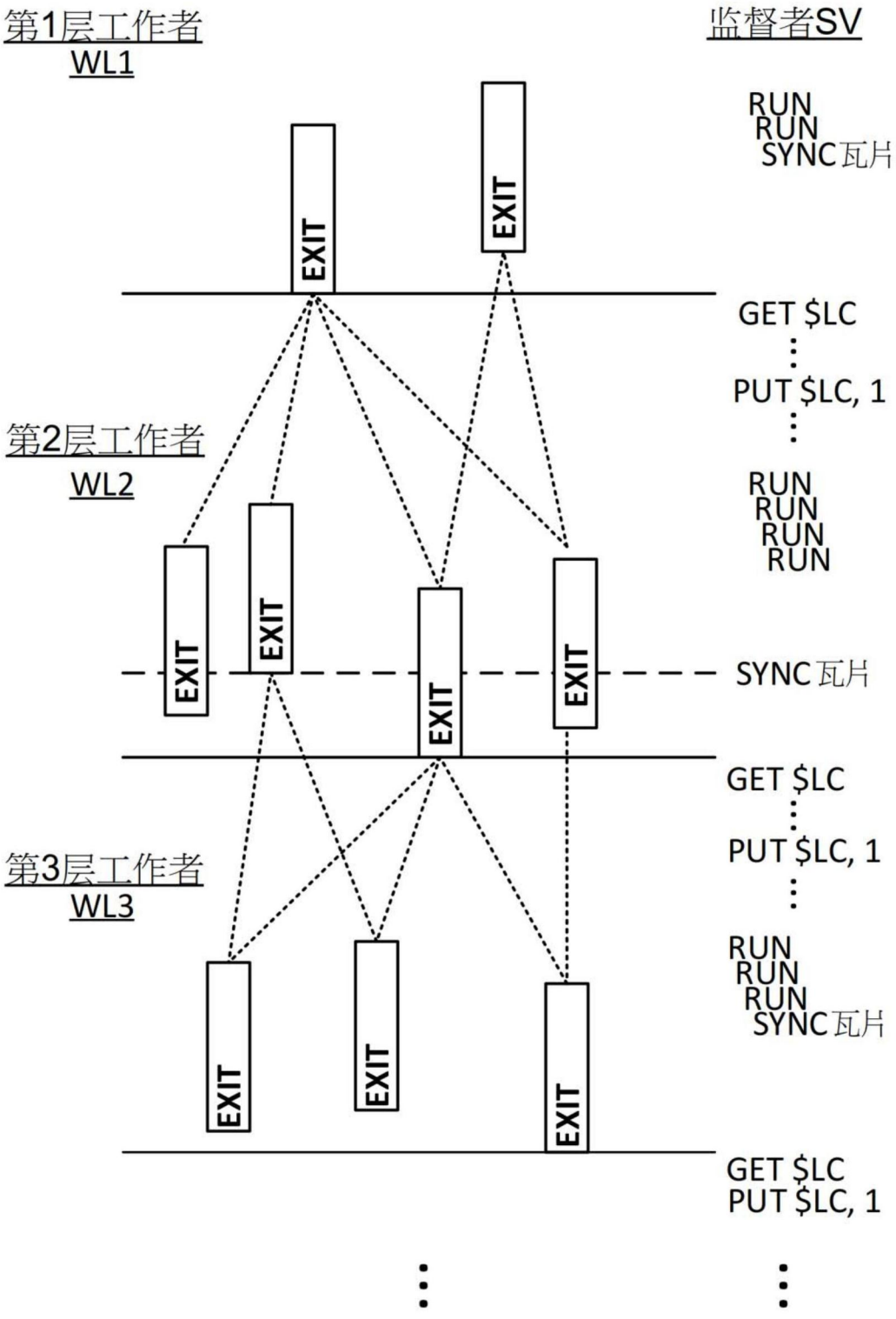

第1层工作者
WL1
监督者SV
RUN
RUN
SYNC瓦片
EXIT
EXIT
GET $LC
PUT $LC, 1
第2层工作者
WL2
RUN
RUN
RUN
RUN
EXIT
EXIT
EXIT
EXIT
SYNC瓦片
GET $LC
PUT $LC, 1
第3层工作者
WL3
RUN
RUN
RUN
SYNC瓦片
EXIT
EXIT
EXIT
GET $LC
PUT $LC, 1

图6

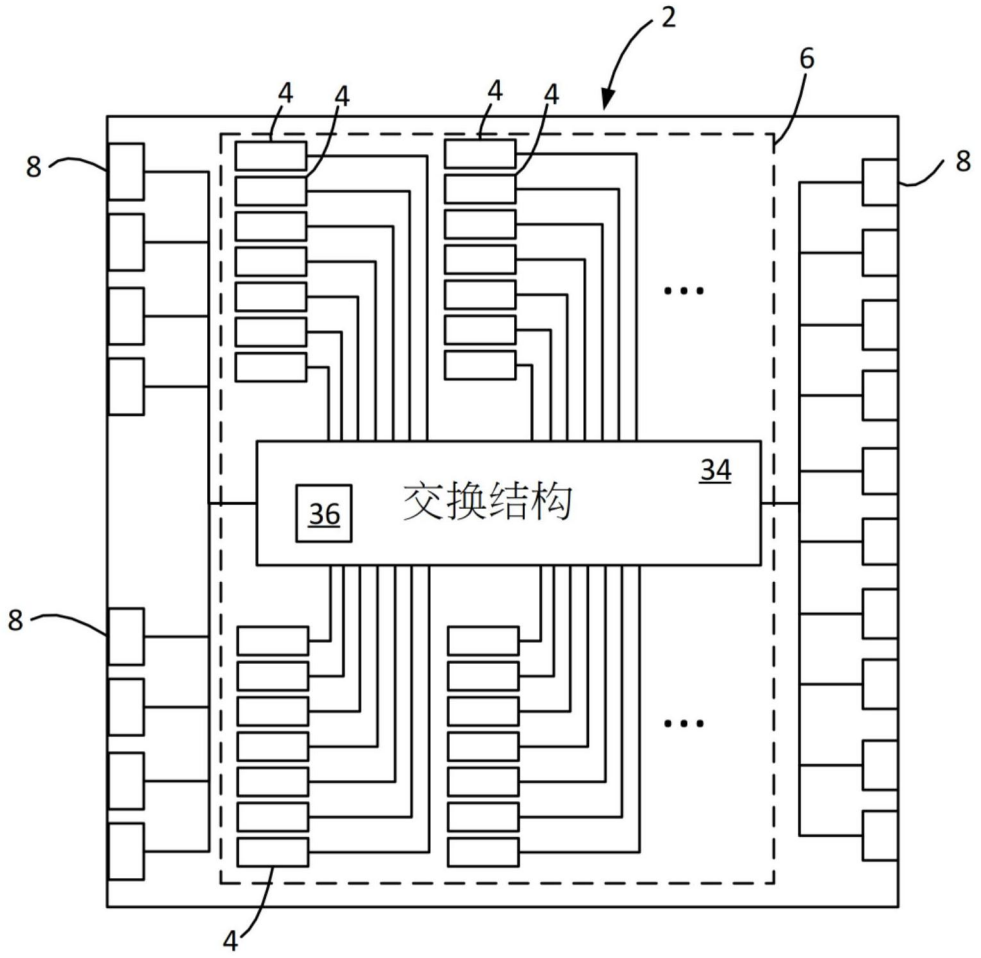


图7

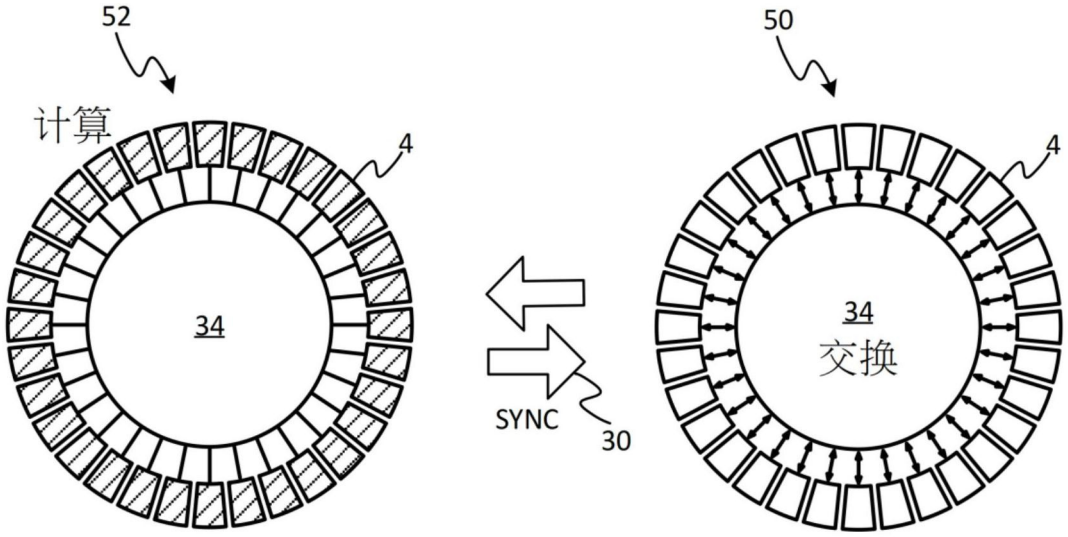


图8

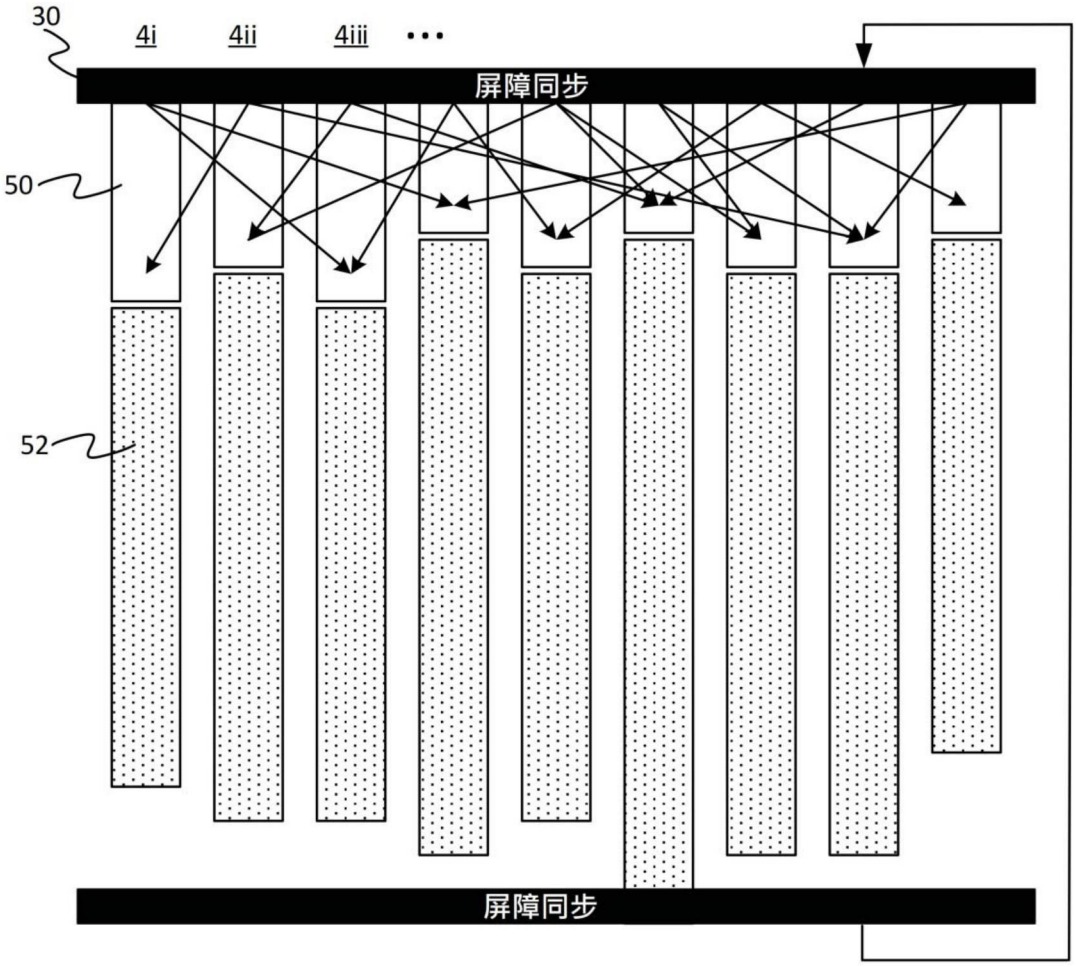


图9

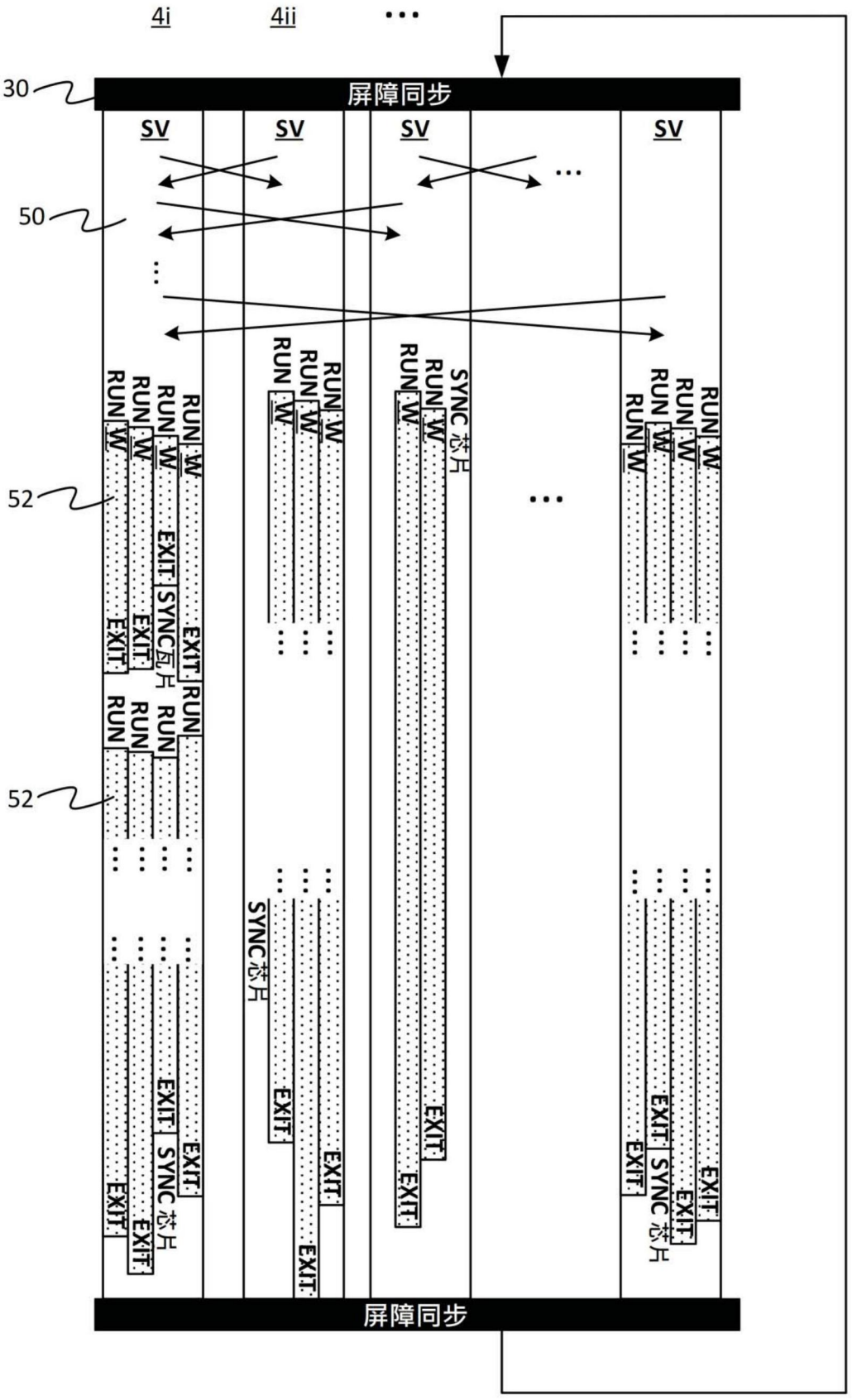


图10

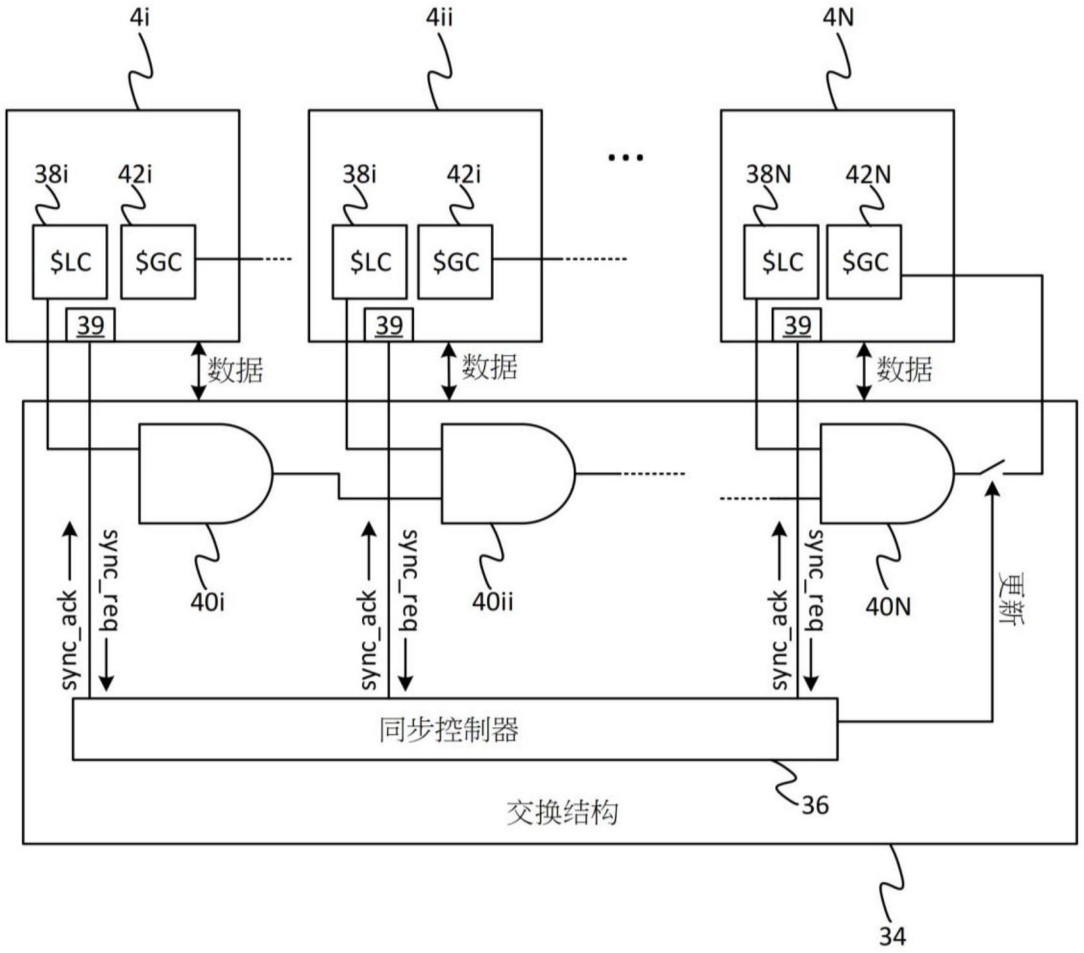

4i
4ii
4N
38i
42i
38i
42i
38N
42N
$LC
$GC
$LC
$GC
$LC
$GC
39
39
39
数据
数据
数据
sync_ack
sync_req
sync_ack
sync_req
sync_ack
sync_req
40i
40ii
40N
更新
同步控制器
36
交换结构
34

图11

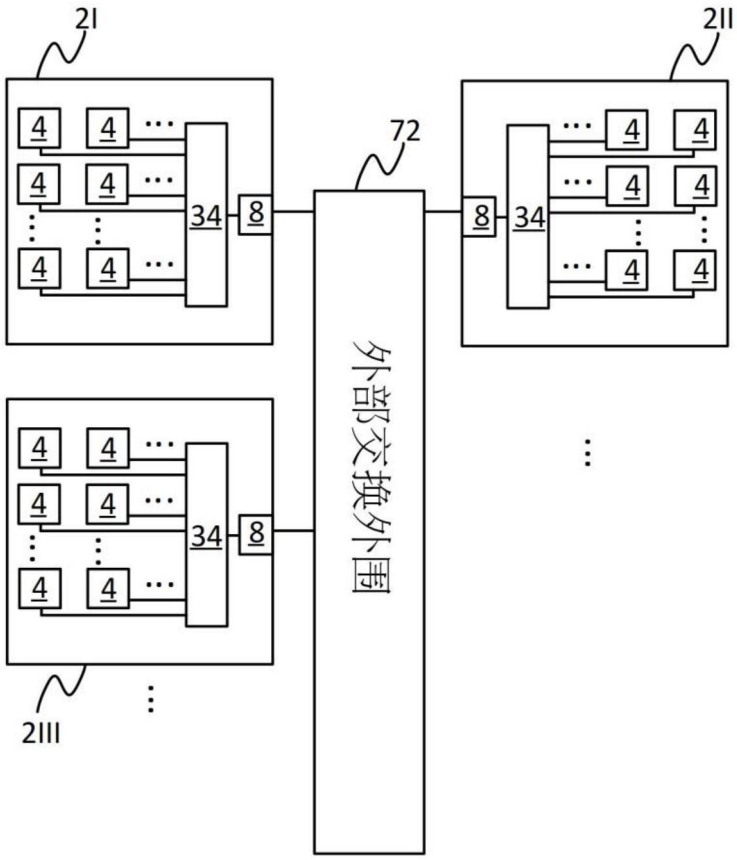


图12

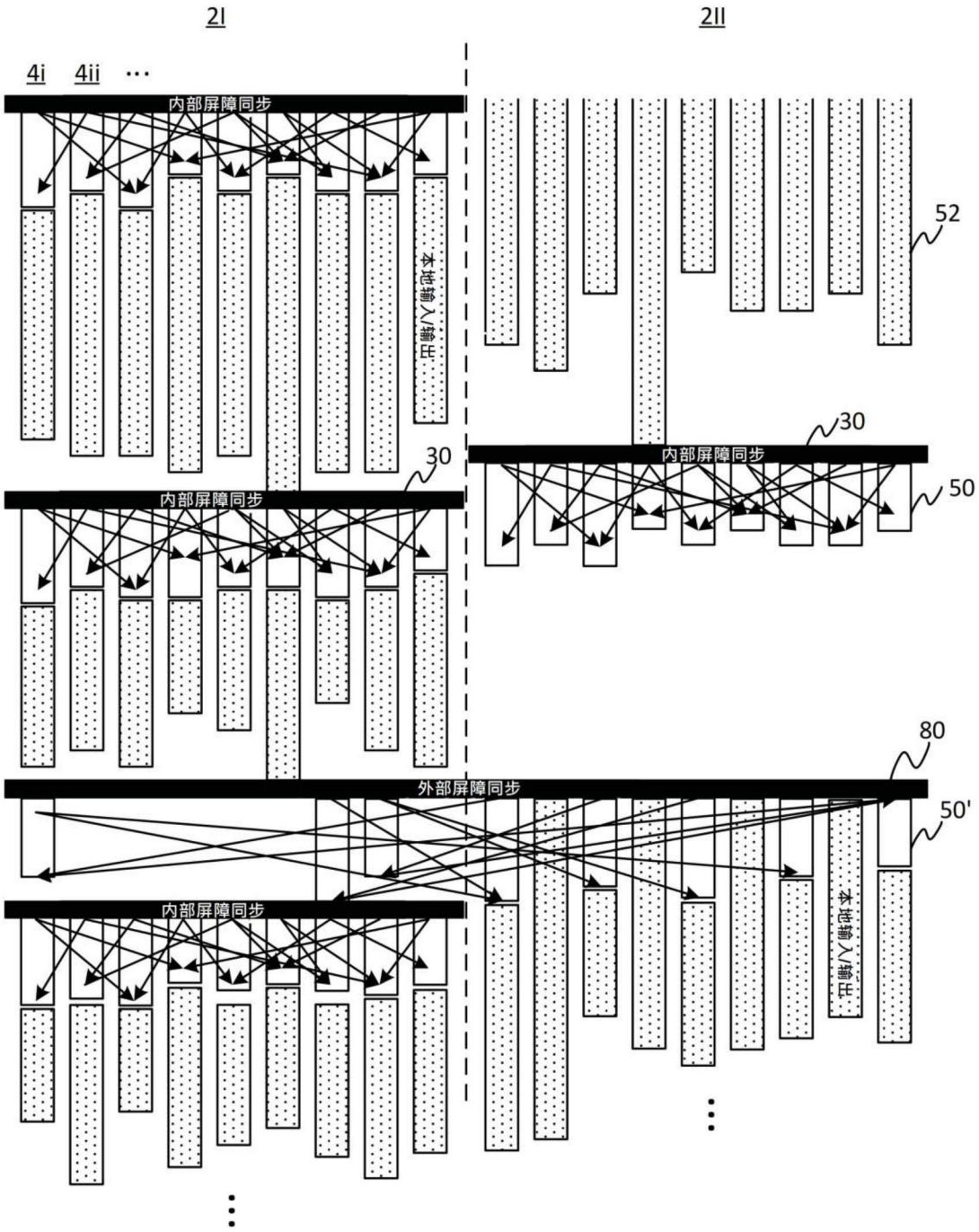

2I
2II
4i
4ii
…
内部屏障同步
本地输入/输出
52
30
内部屏障同步
50
30
内部屏障同步
80
外部屏障同步
50'
内部屏障同步
本地输入/输出

图13

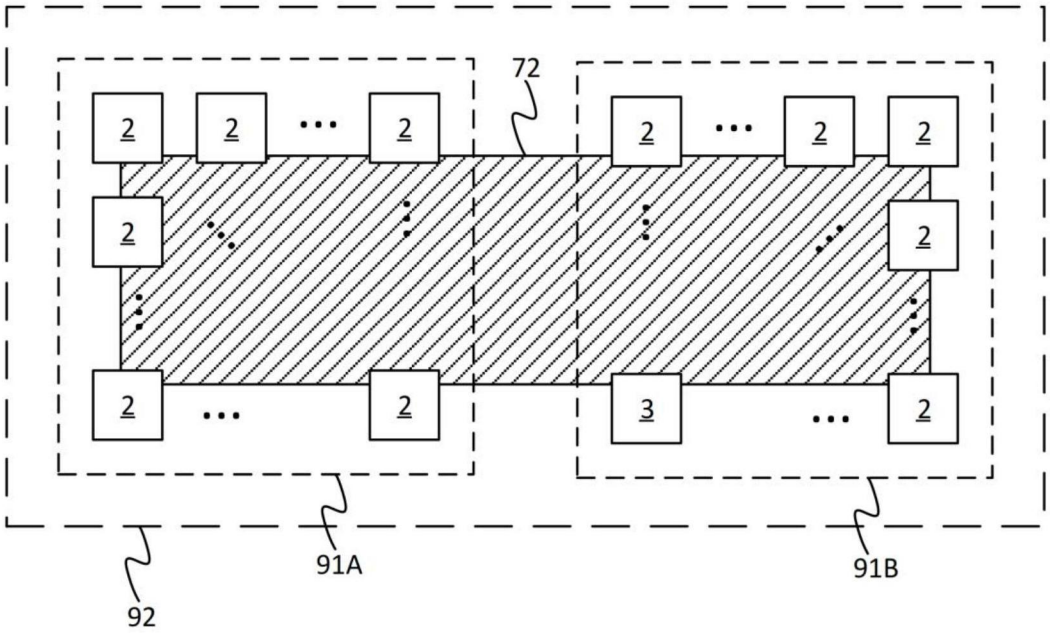


图14

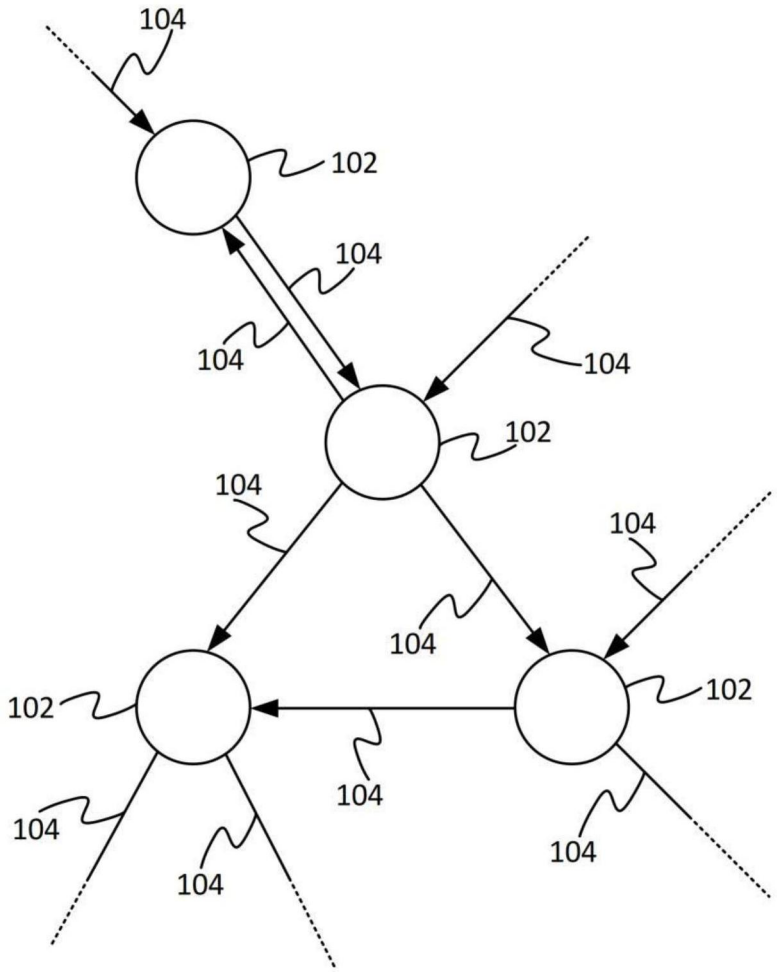


图15

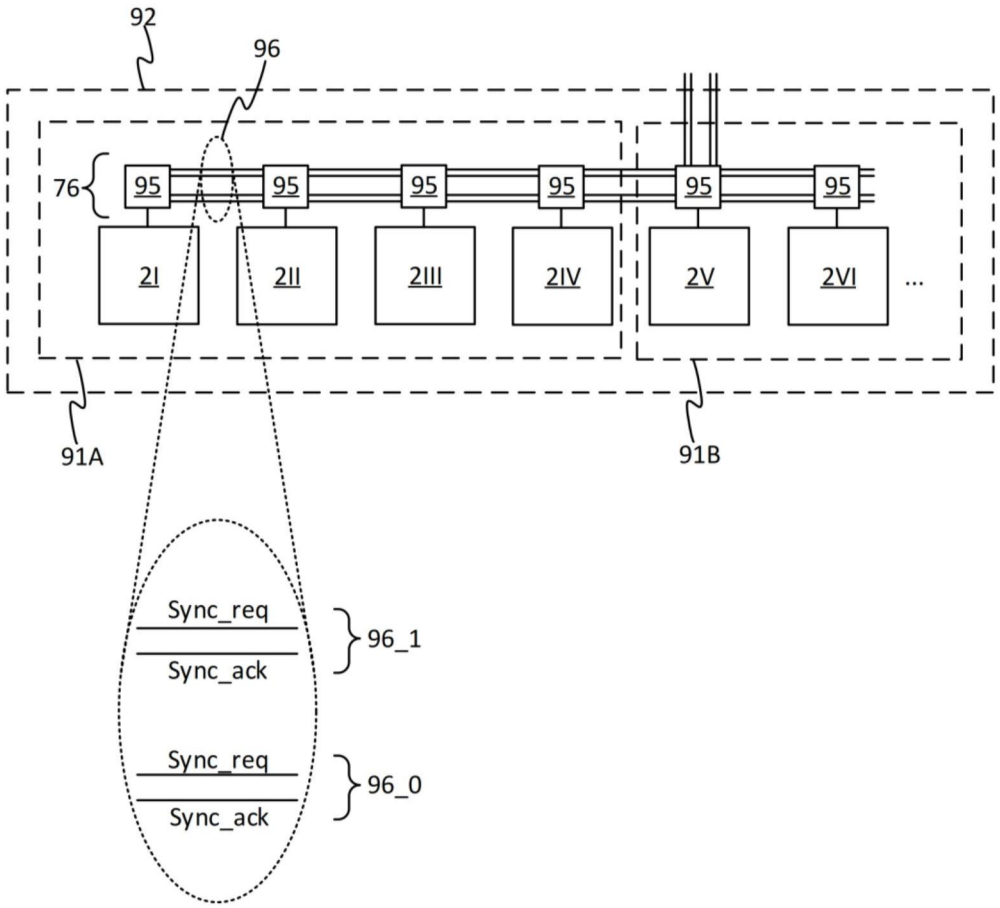

92
96
76
95
95
95
95
95
95
2I
2II
2III
2IV
2V
2VI
...
91A
91B
Sync_req
Sync_ack
96_1
Sync_req
Sync_ack
96_0

图16