(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101231585 B
(45)授权公告日 2010.12.01

(21)申请号 200810007026.3
(22)申请日 2008.01.25
(30)优先权数据
11/627,892 2007.01.26 US
(73)专利权人 辉达公司
地址 美国加利福尼亚州
(72)发明人 约翰·R·尼科尔斯
亨利·P·莫尔顿 拉尔斯·S·尼兰
伊恩·A·巴克 理查德·C·约翰逊
罗伯特·S·格兰维尔
贾扬特·B·科尔希
(74)专利代理机构 北京市磐华律师事务所 11336
代理人 董巍 顾珊
(51)Int.Cl.
G06F 9/38(2006.01)
G06F 9/44(2006.01)
G06F 9/46(2006.01)
审查员 张坦

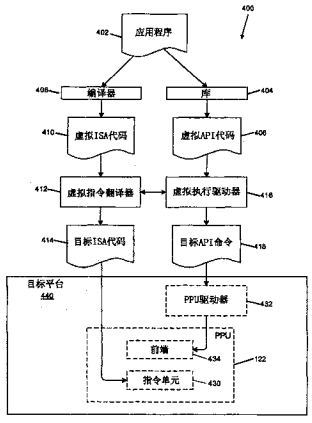
权利要求书 3 页 说明书 25 页 附图 10 页

(54)发明名称

定义用于目标平台结构的并行处理操作方法

(57)摘要

一种虚拟结构和指令集支持明确的并行线程计算。所述虚拟结构定义：虚拟处理器，其支持具有不同虚拟线程之间的不同级别的数据共享和协调（例如，同步）的多个虚拟线程的并发执行；以及虚拟执行驱动器，其控制所述虚拟处理器。所述虚拟处理器的虚拟指令集结构用来定义虚拟线程的行为，且包含与并行线程行为（例如，数据共享和同步）有关的指令。通过使用所述虚拟平台，编程人员能够开发出其中虚拟线程并发地执行以处理数据的应用程序；虚拟翻译器和驱动器以对于所述编程人员透明的方式来调适所述应用程序代码以适应其所执行于的特定硬件。



CN 101231585 B

1. 一种定义并行处理操作的方法，所述方法包括：

提供第一程序代码，所述第一程序代码定义要针对协作的虚拟线程的阵列中的多个虚拟线程中的每一个执行的操作序列；

将所述第一程序代码编译成虚拟线程程序，所述虚拟线程程序定义要针对所述多个虚拟线程中的代表性虚拟线程执行的每个线程指令序列，所述每个线程指令序列包含至少一个指令，所述指令定义所述代表性虚拟线程与所述多个虚拟线程中的一个或一个以上其它虚拟线程之间的协作行为，其中所述代表性虚拟线程与所述多个虚拟线程中的每一个一样执行相同的处理任务；以及

存储所述虚拟线程程序。

2. 根据权利要求 1 所述的方法，其进一步包括：

将所述存储的虚拟线程程序翻译成符合目标平台结构的指令序列。

3. 根据权利要求 1 所述的方法，其进一步包括：

提供第二程序代码，所述第二程序代码定义协作的虚拟线程的阵列，所述阵列适合处理输入数据集以产生输出数据集，其中所述阵列中的每个虚拟线程并发地执行所述虚拟线程程序；

将所述第二程序代码转换成虚拟函数库中的函数调用序列，所述库包含初始化并致使执行协作的虚拟线程的阵列的虚拟函数；以及

存储所述函数调用序列。

4. 根据权利要求 3 所述的方法，其进一步包括：

将所述存储的虚拟线程程序和所述函数调用序列翻译成可在目标平台结构上执行的程序代码，所述可执行的程序代码定义执行所述协作的虚拟线程的阵列的一个或一个以上平台线程。

5. 根据权利要求 4 所述的方法，其进一步包括：

在符合所述目标平台结构的计算机系统上执行所述可执行的程序代码，因而产生所述输出数据集；以及

将所述输出数据集存储在存储媒体中。

6. 根据权利要求 1 所述的方法，其中所述每个线程指令序列包含用以在所述每个线程指令序列中的特定点处暂停对所述代表性虚拟线程的操作的执行，直到所述其它虚拟线程中的一个或一个以上到达所述特定点时为止的指令。

7. 根据权利要求 1 所述的方法，其中所述每个线程指令序列包含用以使所述代表性虚拟线程将数据存储在所述其它虚拟线程中的一个或一个以上可存取的共享存储器中的指令。

8. 根据权利要求 1 所述的方法，其中所述每个线程指令序列包含用以使所述代表性虚拟线程自动读取和更新存储在所述其它虚拟线程中的一个或一个以上可存取的共享存储器中的数据的指令。

9. 根据权利要求 1 所述的方法，其中所述虚拟线程程序包含变量定义语句，所述变量定义语句定义多个虚拟状态空间之一中的变量，其中所述多个虚拟状态空间中的不同状态空间对应于所述虚拟线程之间的数据共享的不同模式。

10. 根据权利要求 9 所述的方法，其中所述数据共享的模式包含每个线程不共享模式

和全局共享模式。

11. 根据权利要求 9 所述的方法，其中所述数据共享的模式包含每个线程不共享模式、虚拟线程的一个阵列内的共享模式以及全局共享模式。

12. 根据权利要求 9 所述的方法，其中所述数据共享的模式包含每个线程不共享模式、虚拟线程的一个阵列内的共享模式、虚拟线程的多个阵列之间的共享模式以及全局共享模式。

13. 一种产生执行于目标平台结构上的程序代码的方法，所述方法包括：

提供输入程序代码，其包含第一部分，所述第一部分定义要对虚拟线程的阵列中的多个虚拟线程中的每一者执行的适于处理输入数据集以产生输出数据集的操作序列，

所述输入程序代码进一步包含第二部分，所述第二部分定义所述虚拟线程的阵列的维度；

将所述输入程序代码的所述第一部分编译成虚拟线程程序，所述虚拟线程程序定义要针对所述多个虚拟线程中的代表性虚拟线程执行的每个线程指令序列，所述每个线程指令序列包含至少一个指令，所述指令定义所述代表性虚拟线程与所述多个虚拟线程中的一个或一个以上其它虚拟线程之间的协作行为，其中所述代表性虚拟线程与所述多个虚拟线程中的每一个一样执行相同的处理任务；

将所述输入程序代码的所述第二部分转换成对虚拟函数库的函数调用序列，所述库包含虚拟函数，所述虚拟函数初始化和致使执行所述协作的虚拟线程的阵列；

将所述虚拟线程程序和所述函数调用序列翻译成可在所述目标平台结构上执行的程序代码，所述可执行的程序代码定义执行所述协作的虚拟线程的阵列的一个或一个以上真实线程；

在符合所述目标平台结构的计算机系统上执行所述可执行的程序代码，因而产生所述输出数据集；以及

将所述输出数据集存储在存储媒体中。

14. 根据权利要求 13 所述的方法，其中所述输入程序代码的所述第二部分包含定义所述虚拟线程的阵列的两个或两个以上维度的程序代码。

15. 根据权利要求 14 所述的方法，其中所述输入程序代码的所述第二部分进一步包含：函数调用，其定义虚拟线程阵列的栅格的一个或一个以上维度，其中所述栅格中的每个阵列均将被执行。

16. 根据权利要求 13 所述的方法，其中所述目标平台结构包含主处理器和协处理器，且其中所述翻译的动作包含：

将所述虚拟线程程序翻译成可由所述协处理器上定义的多个线程并行执行的程序代码；以及

将所述函数调用序列翻译成对所述协处理器的驱动器程序的调用序列，其中所述驱动器程序在所述主处理器上执行。

17. 根据权利要求 13 所述的方法，其中所述目标平台结构包含中央处理单元 (CPU)，且其中所述翻译的动作包含：

将所述虚拟线程程序和所述函数调用序列的至少一部分翻译成目标程序代码，所述目标程序代码使用少于所述虚拟线程数目的数目的 CPU 线程执行所述虚拟线程阵列。

18. 一种产生执行于目标平台结构上的程序代码的方法，所述方法包括：

获得虚拟线程程序，所述虚拟线程程序定义要对虚拟线程阵列中的多个虚拟线程中的代表性虚拟线程执行的每个线程指令序列，所述阵列适合于处理输入数据集以产生输出数据集，其中所述代表性虚拟线程与所述多个虚拟线程中的每一个一样执行相同的处理任务；

所述每个线程指令序列包含至少一个指令，所述指令定义所述代表性虚拟线程与所述多个虚拟线程中的一个或一个以上其它虚拟线程之间的协作行为；

获得定义所述虚拟线程阵列的维度的额外程序代码；

将所述虚拟线程程序和所述额外程序代码翻译成可在所述目标平台结构上执行的程序代码，所述可执行的程序代码定义执行所述虚拟线程阵列的一个或一个以上平台线程；

在符合所述目标平台结构的计算机系统上执行所述可执行的程序代码，因而产生所述输出数据集并将所述输出数据集存储在存储器中。

19. 根据权利要求 18 所述的方法，其中所述获得所述虚拟线程程序的动作包含：

接收用高级编程语言编写的 CTA 源程序代码；以及

编译所述 CTA 源程序代码以产生所述虚拟线程程序。

20. 根据权利要求 18 所述的方法，其中所述获得所述虚拟线程程序的动作包含：

从存储媒体中读取所述虚拟线程程序。

21. 根据权利要求 18 所述的方法，其中所述获得所述虚拟线程程序的动作包含：

经由网络从远程计算机系统接收所述虚拟线程程序。

定义用于目标平台结构的并行处理操作方法

技术领域

[0001] 本发明大体上涉及并行处理，且确切地说涉及用于并行线程计算的虚拟结构和指令集。

[0002] 背景技术

[0003] 在并行处理中，多个处理单元（例如，多个处理器芯片或单个芯片内的多个处理核心）同时操作以处理数据。可使用此种系统来解决便于分解成多个部分的问题。一个实例是图像过滤，其中根据输入图像的某一数目的像素来计算输出图像的每个像素。每个输出像素的计算一般独立于其它所有输出像素，因此不同的处理单元可并行计算不同的输出像素。其它许多类型的问题也适于并行分解。总的来说，N 向 (N-way) 并行执行可将此种问题的解决方案加速大约 N 倍。

[0004] 另一类问题在并行的执行线程可彼此协调的情况下则适于并行处理。一个实例是快速傅里叶变换 (FFT)，其是一种递归算法：其中在每个级，对前一级的输出执行计算，以便产生新的值，所述新的值用于对下一级的输入，如此继续直到到达输出级为止。单个执行线程可执行多个级，只要所述线程能可靠地获得来自前一级的输出数据即可。如果要在多个线程之间划分任务，那么必须提供某种协调机制，以便（例如）线程不会试图读取尚未写入的输入数据。（在 2005 年 12 月 15 日申请的共同转让、共同待决的第 11/303,780 号美国专利申请案中描述了这个问题的一个解决方案）。

[0005] 然而，并行处理系统的编程可能较为困难。通常要求编程人员知道可用的处理单元的数目及其能力（指令集、数据寄存器数目、互连件等），以便建立处理单元可实际执行的代码。虽然机器特定的编译器可在这个方面提供相当大的帮助，但仍有必要在每次将代码移植 (port) 到不同处理器时重新编译代码。

[0006] 此外，并行处理结构的各种方面正在迅速发展。举例来说，正在陆续开发新的平台结构、指令集和编程模型。随着并行结构的各种方面（例如，编程模型或指令集）从一代改变成下一代，必须也相应地改变应用程序、软件库、编译器和其它软件及工具。这种不稳定性可能会为并行处理代码的开发和维护增加相当大的额外开销。

[0007] 当需要线程之间协调时，并行编程变得更加困难。编程人员必须确定在特定处理器或计算机系统中有可用于支持（或仿真）线程间通信的机制，且必须编写利用可用机制的代码。由于不同的计算机系统上的可用和 / 或最优机制一般不同，所以这种类型的并行代码一般不可移植；必须针对其运行于的每种硬件平台重编写并行代码。

[0008] 此外，除了为处理器提供可执行代码，编程人员还必须为“主”处理器提供控制代码，所述控制代码协调各种处理单元的操作，例如向每个处理单元指示要执行哪个程序和处理哪些输入数据。此控制代码通常专用于特定的主处理器和处理器间的通信协议，且如果要替换不同的主处理器则通常必须重编写所述控制代码。

[0009] 编译和重新编译并行处理代码中的困难会使用户不愿随着计算技术的发展而升级其系统。因此，将需要使已编译的并行处理代码脱离特定的硬件平台，并提供一种作为并行应用程序和工具的目标的稳定的并行处理结构和指令集。

发明内容

[0010] 本发明的实施例提供一种用于并行线程计算的虚拟结构和虚拟指令集。所述虚拟并行结构定义：一种虚拟处理器，其支持具有不同虚拟线程之间的不同级别的数据共享和协调（例如，同步）的多个虚拟线程的并发执行；以及一种虚拟执行驱动器，其控制所述虚拟处理器。虚拟处理器的虚拟指令集结构用来定义虚拟线程的行为，且包含与并行线程行为（例如，数据共享和同步）有关的指令。通过使用虚拟并行平台，编程人员能够开发出其中虚拟线程并发执行以处理数据的应用程序。应用程序可用高度可移植的中间形式存储和分布，所述形式例如为瞄准虚拟并行平台的程序代码。在安装时间或运行时间，硬件特定的虚拟指令翻译器和执行驱动器使中间形式的应用程序代码适应于其执行于的特定硬件。因此，应用程序更加可移植且更容易开发，因为开发过程独立于特定处理硬件。

[0011] 根据本发明的一个方面，一种用于定义并行处理操作的方法包含提供第一程序代码，所述第一程序代码定义要针对协作的虚拟线程的阵列中的若干虚拟线程中的每一者执行的操作序列。将第一程序代码编译成虚拟线程程序，所述虚拟线程程序定义要针对阵列中的代表性虚拟线程执行的每线程指令的序列，且所述每线程指令的序列包含至少一个指令，所述指令定义代表性虚拟线程与所述阵列中的一个或一个以上其它虚拟线程之间的协作行为。存储虚拟线程程序（例如，存储在存储器中或盘上），且可随后将所述虚拟线程程序翻译成符合目标平台结构的指令序列。

[0012] 此外，也可提供第二程序代码以定义协作的虚拟线程的阵列，所述阵列适合于处理输入数据集以产生输出数据集，其中阵列中的每个虚拟线程并发执行虚拟线程程序。有利地将第二程序代码转换成虚拟函数库中的函数调用序列，其中所述库包含初始化和致使执行协作的虚拟线程的阵列的虚拟函数。也可存储这个函数调用序列。接着，可将所存储的虚拟线程程序和函数调用序列翻译成可在目标平台结构上执行的程序代码，其中所述可执行的程序代码定义一个或一个以上执行所述协作的虚拟线程的阵列的平台线程。可在符合目标平台结构的计算机系统上执行可执行的程序代码，因而产生输出数据集，可将所述输出数据集存储在存储媒体（例如，计算机存储器、盘等）中。

[0013] 如已提到的，虚拟线程程序代码中的每线程指令的序列有利地包含至少一个指令，所述指令定义代表性虚拟线程与所述阵列中的一个或一个以上其它虚拟线程之间的协作行为。举例来说，所述每线程指令的序列可包含：在序列中的特定点处暂停执行代表性虚拟线程的操作直到例如其它虚拟线程中的一者或一者以上达到所述特定点为止的时间的指令，使代表性虚拟线程将数据存储在其它虚拟线程中的一者或一者以上可存取的共享存储器中的指令，使代表性虚拟线程自动读取和更新存储在其它虚拟线程中的一者或一者以上可存取的共享存储器中的数据的指令，等等。

[0014] 所述虚拟线程程序也可包含变量定义语句，所述语句在若干虚拟状态空间之一中定义变量，其中不同的虚拟状态空间对应于虚拟线程之间的不同的数据共享模式。在一个实施例中，至少支持每线程不共享模式和全局共享模式。在其它实施例中，也可支持额外模式，例如虚拟线程的一个阵列内的共享模式和／或多个虚拟线程阵列之间的共享模式。

[0015] 根据本发明的另一方面，一种用于操作目标处理器的方法包含提供输入程序代码。所述输入程序代码包含第一部分，所述第一部分定义要对虚拟线程阵列中的若干虚拟

线程中的每一者执行的操作序列，所述阵列适合于处理输入数据集以产生输出数据集，且所述输入程序代码还包含第二部分，所述第二部分定义虚拟线程的阵列的维度。将所述输入程序代码的第一部分编译成虚拟线程程序，所述虚拟线程程序定义要对阵列中的代表性虚拟线程执行的每线程指令的序列。所述每线程指令的序列包含至少一个指令，所述指令定义所述代表性虚拟线程与所述阵列中的一个或一个以上其它虚拟线程之间的协作行为。将所述输入程序代码的第二部分转换成对虚拟函数库的函数调用序列，其中所述库包含初始化和致使执行协作的虚拟线程的阵列的虚拟函数。将所述虚拟线程程序和函数调用序列翻译成可在目标平台结构上执行的程序代码，其中所述可执行的程序代码定义执行协作的虚拟线程的阵列的一个或一个以上真实线程。在符合目标平台结构的计算机系统上执行所述可执行程序代码，因而产生输出数据集，可将所述输出数据集存储在存储媒体中。

[0016] 在一些实施例中，可在两个或两个以上维度上定义虚拟线程的阵列。此外，所述输入程序代码的第二部分还可包含定义虚拟线程阵列的栅格的一个或一个以上维度的函数调用，其中栅格中的每个阵列均将被执行。

[0017] 可使用任何目标平台结构。在一些实施例中，目标平台结构包含主处理器和协处理器。在翻译期间，可将虚拟线程程序翻译成可由定义于协处理器上的若干线程并行执行的程序代码，同时将函数调用序列翻译成对在主处理器上执行的用于协处理器的驱动程序的调用序列。在其它实施例中，目标平台结构包含中央处理单元 (CPU)。在翻译期间，将虚拟线程程序和函数调用序列的至少一部分翻译成目标程序代码，所述目标程序代码使用少于虚拟线程数目的数目的 CPU 线程来执行虚拟线程阵列。

[0018] 根据本发明的又一实施例，一种用于操作目标处理器的方法包含获得虚拟线程程序，所述虚拟线程程序定义要针对虚拟线程阵列中的若干虚拟线程中的代表性虚拟线程执行的每线程指令序列，所述虚拟线程阵列适合于处理输入数据集以产生输出数据集。所述每线程指令序列包含至少一个指令，所述指令定义所述代表性虚拟线程与所述阵列中的一个或一个以上其它虚拟线程之间的协作行为。也获得定义虚拟线程阵列的维度的额外程序代码。将所述虚拟线程程序和额外的程序代码翻译成可在目标平台结构上执行的程序代码，其中所述可执行的程序代码定义执行所述虚拟线程阵列的一个或一个以上平台线程。在符合目标平台结构的计算机系统上执行可执行的程序代码，因而产生输出数据集且将所述输出数据集存储在存储器中。

[0019] 在一些实施例中，可通过接收用高级编程语言编写的源程序代码并编译所述源程序代码以产生虚拟线程程序来获得虚拟线程程序。或者，可从存储媒体读取虚拟线程程序或经由网络从远程计算机系统接收虚拟线程程序。应了解，所读取或接收的虚拟线程代码可能之前已经从高级语言编译，或者直接建立为符合虚拟指令集结构的代码。

[0020] 以下详细描述连同附图一起将提供对本发明的性质和优点的更好了解。

附图说明

[0021] 图 1 是根据本发明实施例的计算机系统的方框图。

[0022] 图 2A 和 2B 说明本发明实施例中使用的编程模型中的栅格、线程阵列和线程之间的关系。

[0023] 图 3 是根据本发明实施例的虚拟结构的方框图。

[0024] 图 4 是根据本发明实施例使用虚拟结构来操作目标处理器的概念模型。

[0025] 图 5 是列举由根据本发明实施例的虚拟指令集结构 (ISA) 定义的特殊变量的表格。

[0026] 图 6 是列举在根据本发明实施例的虚拟 ISA 中支持的变量类型的表格。

[0027] 图 7 是列举在根据本发明实施例的虚拟 ISA 中支持的虚拟状态空间的表格。

[0028] 图 8A-8H 是列举在根据本发明实施例的虚拟 ISA 中定义的虚拟指令的表格。

[0029] 图 9 是根据本发明实施例的用于使用虚拟指令翻译器的过程的流程图。

[0030] 图 10 是列举根据本发明实施例的在虚拟库中可供虚拟执行驱动器使用的函数的表格。

具体实施方式

[0031] 本发明实施例提供一种用于并行线程计算的虚拟结构和指令集。所述虚拟结构提供支持在不同线程之间具有多层数据共享和协调（例如，同步）的多个线程的并发执行的处理器模型，以及控制所述模型处理器的虚拟执行驱动器。用于定义处理线程的行为的虚拟指令集包含与并行线程行为有关的指令，例如允许某些线程间的数据共享的指令和要求不同线程在程序内的由编程人员指定的某些点处变得同步的指令。使用虚拟平台，编程人员可开发在其中执行并发、协作的线程以处理数据的应用程序。硬件特定的虚拟指令翻译器和执行驱动器使应用代码适应其执行于的特定硬件。因此，应用程序更具移植性并更易于开发，因为开发过程独立于特定处理硬件。

[0032] 1. 系统概述

[0033] 图 1 是根据本发明实施例的计算机系统 100 的方框图。计算机系统 100 包含中央处理单元 (CPU)102，和包含存储器桥 105 的经由总线路径通信的系统存储器 104。存储器桥 105（其可以是（例如）Northbridge 芯片）经由总线或其它通信路径 106（例如，HyperTransport 链路）连接到 I/O（输入 / 输出）桥 107。I/O 桥 107（其可以是（例如）Southbridge 芯片）从一个或一个以上用户输入装置 108（例如，键盘、鼠标）接收用户输入，并将输入经由路径 106 和存储器桥 105 转发到 CPU 102。并行处理子系统 112 经由总线或其它通信路径 113（例如，PCI Express 或加速图形端口链路）耦合到存储器桥 105；在一个实施例中，并行处理子系统 112 是图形子系统，其将像素递送到显示器装置 110（例如，常规的基于 CRT 或 LCD 的监视器）。系统盘 114 也连接到 I/O 桥 107。切换器 116 提供 I/O 桥 107 与例如网络适配器 118 和各种内插卡 120 及 121 等其它组件之间的连接。其它组件（未明确展示）包含 USB 或其它端口连接、CD 驱动器、DVD 驱动器等，也可连接到 I/O 桥 107。使图 1 中的各个组件互连的通信路径可使用任何适宜的协议来实施，例如 PCI（外围组件互连）、PCI Express(PCI-E)、AGP（加速图形端口）、HyperTransport，或任何其它总线或点对点通信协议，且不同装置之间的连接可使用此项 技术中已知的不同协议。

[0034] 并行处理子系统 112 包含并行处理单元 (PPU)122 和并行处理 (PP) 存储器 124，其可（例如）使用例如可编程处理器、专用集成电路 (ASIC) 和存储器装置等的一个或一个以上集成电路装置来实施。PPU 122 有利地实施包含一个或一个以上处理核心的高度并行处理器，所述处理核心的每一者能够并发执行大量（例如，数百个）线程。PPU 122 可经编程以执行一大批计算，包含线性和非线性数据变换、视频和 / 或音频数据的过滤、建模（例

如，应用物理定律来确定对象的位置、速度和其它属性）、图像渲染等。PPU 122 可将数据从系统存储器 104 和 / 或 PP 存储器 124 转移到内部存储器中，处理所述数据，并将结果数据写回到系统存储器 104 和 / 或 PP 存储器 124，在其中此类数据可由其它系统组件（包含，例如，CPU 102）存取。在一些实施例中，PPU 122 是图形处理器，其还可经配置以执行与以下操作有关的各种任务 ：依据由 CPU 102 和 / 或系统存储器 104 经由存储器桥 105 和总线 113 供应的图形数据产生像素数据、与 PP 存储器 124（其可用作图形存储器，包含（例如）常规帧缓冲器）交互以存储和更新像素数据、将像素数据递送到显示器装置 110 等。在一些实施例中，PP 子系统 112 可包含一个作为图形处理器操作的 PPU 122，和另一用于进行通用计算的 PPU 122。PPU 可相同或不同，且每一 PPU 可具有其自身的专用 PP 存储器装置。

[0035] CPU 102 作为系统 100 的主处理器操作，其控制和协调其它系统组件的操作。明确地说，CPU 102 发布控制 PPU 122 的操作的命令。在一些实施例中，CPU 102 将针对 PPU122 的命令流写入到命令缓冲器，所述命令缓冲器可处于系统存储器 104、PP 存储器 124 或 CPU 102 和 PPU 122 两者均可存取的另一存储位置中。PPU 122 从命令缓冲器读取命令流，并与 CPU 102 的操作异步地执行命令。

[0036] 将了解，本文展示的系统是说明性的，且可能作出变化和修改。可视需要修改连接拓扑，包含桥的数目和配置。举例来说，在一些实施例中，系统存储器 104 直接而不通过桥连接到 CPU 102，且其它装置经由存储器桥 105 和 CPU 102 与系统存储器 104 通信。在其它替代拓扑中，PP 子系统 112 连接到 I/O 桥 107 而不连接到存储器桥 105。在另外其它实施例中，I/O 桥 107 和存储器桥 105 可能集成到单一芯片中。本文展示的特定组件是任选的 ；举例来说，可能支持任何数目的内插卡或外围装置。在一些实施例中，排除切换器 116，且网络适配器 118 和内插卡 120、121 直接连接到 I/O 桥 107。

[0037] 还可改变 PPU 122 到系统 100 的其余部分的连接。在一些实施例中，PP 系统 112 被实施为可插入到系统 100 的扩充插槽中的内插卡。在其它实施例中，PPU 可与总线桥（例如，存储器桥 105 或 I/O 桥 107）一起集成在单一芯片上。在另外其它实施例中，PPU 122 的一些或所有元件可与 CPU 102 集成。

[0038] PPU 可具备任何量的本地 PP 存储器，包含无本地存储器，且可以任何组合使用本地存储器与系统存储器。举例来说，在统一存储器结构 (UMA) 实施例中，PPU 122 可以是图形处理器 ；在此类实施例中，提供极少或不提供专门的图形存储器，且 PPU 122 将专门地或几乎专门地使用系统存储器。在 UMA 实施例中，PPU 可集成到桥芯片中或提供成离散芯片，其中高速链路（例如，PCI-E）将 PPU 连接到桥芯片和系统存储器。

[0039] 还将了解，可（例如）通过在单一内插卡上包含多个 PPU，通过将多个内插卡连接到路径 113，和 / 或通过将一个或一个以上 PPU 直接连接到系统母板，而将任何数目的 PPU 包含在系统中。多个 PPU 可并行操作，从而以高于利用单一 PPU 可能实现的处理量处理数据。

[0040] 所属领域的技术人员还将了解，CPU 和 PPU 可集成到单一装置中，且 CPU 和 PPU 可共享例如指令逻辑、缓冲器、高速缓冲存储器、存储器、处理引擎等各种资源 ；或者可针对并行处理和其它操作提供单独的资源。因此，本文描述为与 PPU 相关联的任何或所有电路和 / 或功能性也可在适当装备的 CPU 中实施并由所述适当装备的 CPU 执行。

[0041] 并入有 PPU 的系统可以多种配置和形状因素进行实施，包含台式计算机、膝上型

计算机，或手提个人计算机、服务器、工作站、游戏机、嵌入式系统等。

[0042] 所属领域的技术人员还将了解，本发明的一个优点是相对于特定计算硬件的独立性增加。因此，将了解，本发明实施例可使用任何计算机系统（包含不提供 PPU 的系统）来实践。

[0043] 2. 虚拟编程模型概述

[0044] 在本发明实施例中，需要使用 PPU 122 或计算系统的其它处理器来使用线程阵列执行通用计算。如本文所使用，“线程阵列”是由对输入数据集并发执行相同程序以产生输出数据集的若干 (n_0) 线程组成的群组。线程阵列中的每一线程被分派有唯一线程识别符（“线程 ID”），其可由线程在其执行期间存取。可定义为一维或多维数值（例如，0 到 n_0-1）的线程 ID 控制线程的处理行为的各个方面。举例来说，线程 ID 可用于确定线程将处理输入数据集的哪一部分，和 / 或确定线程将产生或写入输出数据集的哪一部分。

[0045] 在一些实施例中，线程阵列是“协作的”线程阵列或 CTA。与其它类型的线程阵列一样，CTA 是对输入数据集并发执行相同程序（本文称为“CTA 程序”）以产生输出数据集的多个线程的群组。在 CTA 中，线程可通过以依赖于线程 ID 的方式彼此共享数据 而进行协作。举例来说，在 CTA 中，数据可由一个线程产生并由另一线程消耗。在一些实施例中，可在将共享数据的点处将同步指令插入到 CTA 程序代码中，以确保在消耗数据的线程试图访问数据之前已由产生数据的线程实际产生数据。CTA 的线程之间的数据共享（如果存在的话）的程度由 CTA 程序确定；因此，将了解，在使用 CTA 的特定应用中，CTA 的线程可能或可能不实际上彼此共享数据，这取决于 CTA 程序，且本文中同义地使用术语“CTA”和“线程阵列”。

[0046] 在一些实施例中，CTA 中的线程与同一 CTA 中的其它线程共享输入数据和 / 或中间结果。举例来说，CTA 程序可能包含用于计算共享存储器中的特定数据将被写入到的地址的指令，其中所述地址是线程 ID 的函数。每一线程使用其自身的线程 ID 来计算所述函数，并写入到相应位置。有利地定义地址函数使得不同线程写入到不同位置；只要函数是确定性的，就可预测任何线程写入到的位置。CTA 程序还可包含用于计算共享存储器中的将从中读取数据的地址的指令，其中所述地址是线程 ID 的函数。通过定义适宜的函数和提供同步技术，数据可以可预测方式由 CTA 的一个线程写入到共享存储器中的给定位置，并由同一 CTA 的不同线程从所述位置进行读取。因此，可支持线程之间的任何所需的数据共享模式，且 CTA 中的任何线程可与同一 CTA 中的任何其它线程共享数据。

[0047] 有利地使用 CTA（或其它类型的线程阵列）来执行便于数据并行分解 (data-parallel decomposition) 的计算。如本文所使用，“数据并行分解”包含其中通过对输入数据并行执行同一算法多次以产生输出数据来对计算问题求解的任何情形；举例来说，数据并行分解的一个普通实例涉及将同一处理算法应用于输入数据集的不同部分以便产生输出数据集的不同部分。适于数据并行分解的问题的实例包含矩阵代数、任何数目的维度的线性和 / 或非线性变换（例如，快速傅里叶变换），和包含任何数目的维度的卷积滤波、多个维度的可分离滤波等各种滤波算法。在 CTA 程序中指定待应用于输入数据集的每一部分的处理算法，且 CTA 中的每一线程对输入数据集的一个部分执行同一 CTA 程序。CTA 程序可使用广范围的数学和逻辑运算来实施算法，且所述程序可包含有条件或分支执行路径以及直接和 / 或间接存储器存取。

[0048] 上文引用的第 11/303,780 号申请案中进一步详细描述了 CTA 及其执行。

[0049] 在一些情形中，定义相关 CTA（或更一般来说，线程阵列）的“栅格”也是有用的。如本文所使用，CTA 的“栅格”是若干 (n_1) 个 CTA 的集合，其中所有 CTA 为相同大小（即，线程数目）并执行相同的 CTA 程序。栅格内的 n_1 个 CTA 有利地彼此独立，意味着栅格中任何 CTA 的执行不受栅格中的任何其它 CTA 的执行的影响。如将了解，此 特征提供在可用的处理核心之间分配 CTA 的显著灵活性。

[0050] 为了区分栅格内的不同 CTA，有利地向栅格的每一 CTA 分派“CTA 识别符”（或 CTAID）。与线程 ID 一样，任何唯一识别符（包含但不限于数值识别符）均可用作 CTA ID。在一个实施例中，CTA ID 只是从 0 到 n_1-1 的连续（一维）指数值。在其它实施例中，可使用多维指数标定方案。CTA ID 对于 CTA 的所有线程是共同的，且栅格内给定 CTA 的线程可使用其 CTA ID 并结合其线程 ID 来确定（例如）用于读取输入数据的源位置和 / 或用于写入输出数据的目的地位置。以此方式，同一栅格的不同 CTA 中的线程可对同一数据集并发操作，但在一些实施例中，不支持栅格中的不同 CTA 之间的数据共享。

[0051] 定义 CTA 栅格（例如）在需要使用多个 CTA 来对单一大问题的不同部分求解的情况下可能是有用的。举例来说，可能需要执行滤波算法以产生高分辨率电视 (HDTV) 图像。如此项技术中已知，HDTV 图像可能包含 2 百万个以上的像素。如果每一线程产生一个像素，那么待执行的线程数目将超过单一 CTA 中可处理的线程数目（假定为使用常规技术构造的具有合理尺寸和成本的处理平台）。

[0052] 可通过在多个 CTA 之间划分图像使每一 CTA 产生输出像素的不同部分（例如，16x16 小片）来管理这种大处理任务。所有 CTA 执行相同程序，且线程使用 CTAID 与线程 ID 的组合来确定用于读取输入数据和写入输出数据的位置，使得每一 CTA 对输入数据集的正确部分操作并将其输出数据集部分写入到正确位置。

[0053] 应注意，与 CTA 内的线程（其可共享数据）不同，栅格内的 CTA 有利地不彼此共享数据或以另外的方式彼此依赖。也就是说，可循序（以任一次序）或并发执行同一栅格的两个 CTA，且仍产生相同结果。因此，处理平台（例如，图 1 的系统 100）可通过首先执行一个 CTA 接着执行下一 CTA 等等直到已执行栅格的所有 CTA 为止，来执行 CTA 栅格并获得结果。或者，如果有足够的资源可用，处理平台可通过并行执行多个 CTA 来执行相同栅格并获得相同结果。

[0054] 在一些实例中，可能需要定义 CTA 的多个 (n_2) 栅格，其中每一栅格执行数据处理程序或任务的不同部分。举例来说，数据处理任务可能划分为若干个“求解步骤”，其中通过执行 CTA 栅格来执行每一求解步骤。作为另一实例，数据处理任务可能包含对一连串输入数据集（例如，连续的视频数据帧）执行相同或类似的操作；可针对每一输入数据集执行 CTA 栅格。虚拟编程模型有利地支持至少这三个等级的工作定义（即，线程、CTA 和 CTA 栅格）；视需要还可支持额外的等级。

[0055] 将了解，用于对特定问题求解的 CTA 的大小（线程的数目 n_0）、栅格的大小 (CTA 的数目 n_1) 和栅格的数目 (n_2) 将取决于问题的参数和定义问题分解的编程人员或自动代理的偏好。因此，在一些实施例中，CTA 的大小、栅格的大小和栅格的数目有利地由编程人员定义。

[0056] 受益于 CTA 方法的问题通常特征在于，存在可被并行处理的大量数据元素。在一

些实例中，数据元素是输出元素，其每一者是通过对输入数据集的不同（可能重叠）部分执行相同算法而产生的。在其它实例中，数据元素可以是输入元素，其每一者将使用相同算法予以处理。

[0057] 此类问题可始终分解为至少两个等级并映射到上文描述的线程、CTA 和栅格上。举例来说，每一栅格可能表示复杂数据处理任务中的一个求解步骤的结果。每一栅格有利地划分为若干“区块”，其每一者可作为单一 CTA 被处理。每一区块有利地含有多个“元素”，即待求解的问题的基本部分（例如，单一输入数据点或单一输出数据点）。在 CTA 内，每一线程处理一个或一个以上元素。

[0058] 图 2A 和 2B 说明在本发明实施例中使用的虚拟编程模型中栅格、CTA 与线程之间的关系。图 2A 展示若干栅格 200，每一栅格由 CTA 202 的二维 (2-D) 阵列组成。（本文中，相似对象的多个实例用表示所述对象的参考标号和（需要时）表示实例的括弧标号来指示。）如图 2B 中针对 202(0,0) 所示，每一 CTA 202 包含线程 (Θ)204 的 2-D 阵列。对于每一栅格 200 的每一 CTA 202 中的每一线程 204，可定义 $I = [i_g, i_c, i_t]$ 的形式的唯一识别符，其中栅格识别符 i_g 唯一地识别栅格，CTA ID i_c 唯一地识别栅格内的 CTA，且线程 ID i_t 唯一地识别 CTA 内的线程。在此实施例中，可由一维栅格识别符 i_g、二维 CTA 识别符 i_c 和二维线程识别符 i_t 来构成识别符 I。在其它实施例中，唯一识别符 I 是整数的三元组，其中 $0 \leqslant i_g < n_2$；$0 \leqslant i_c < n_1$；且 $0 \leqslant i_t < n_0$。在另外其它实施例中，栅格、CTA 和线程识别符中的任一者或全部可能表达为一维整数、2D 坐标对、3D 三元组等。唯一线程识别符 I 可用于（例如）确定包含针对整个栅格或多个栅格的输入数据集的阵列内的输入数据的源位置，和 / 或确定用于在包含针对整个栅格或多个栅格的输出数据集的阵列内存储输出数据的目标位置。

[0059] 举例来说，在 HDTV 图像的情况下，每一线程 204 可能对应于输出图像的像素。CTA202 的大小（线程 204 的数目）是问题分解中的选择事项，其仅受对单一 CTA 202 中的最大线程数目的约束的限制（其反映处理器资源的有限性）。栅格 200 可对应于 HDTV 数据的整个帧，或者多个栅格可映射到单一帧。

[0060] 在一些实施例中，问题分解是统一的，意味着所有栅格 200 具有相同数目和配置的 CTA 202，且所有 CTA 202 具有相同数目和配置的线程 204。在其它实施例中，分解可能不统一。举例来说，不同栅格可能包含不同数目的 CTA，且不同 CTA（相同栅格或不同栅格中）可能包含不同数目的线程。

[0061] 如上定义的 CTA 可包含数打或甚至数百个并发线程。上面将执行 CTA 的并行处理系统可能或可能不支持如此大量的并发线程。在一个方面，本发明通过允许编程人员在不考虑实际硬件能力的情况下使用 CTA 和 CTA 栅格的模型来定义处理任务，使编程人员脱离于此类硬件限制。举例来说，编程人员可编写代码（“CTA 程序”），其定义待由 CTA 的单一代表性线程执行的处理任务；将 CTA 定义为若干此类线程，每一线程具有唯一识别符；并将栅格定义为若干 CTA，每一 CTA 具有唯一识别符。如下文所描述，此类代码被自动翻译为可在特定平台上执行的代码。举例来说，如果将 CTA 定义为包含若干 (n_0) 个并发线程但目标平台仅支持一个线程，那么翻译器可定义执行分派到所有 n_0 个线程的任务的一个实际线程。如果目标平台支持一个以上但少于 n_0 个并发线程，那么可视需要在所述数目的可用线程之间划分任务。

[0062] 因此，CTA 和栅格的编程模型应理解为虚拟模型，即与任何特定物理实现脱离的作为对编程人员的概念上的辅助的模型。可在对并行处理具有不同程度的硬件支持的多种目标平台中实现 CTA 和栅格的虚拟模型。明确地说，本文所使用的术语“CTA 线程”是指离散处理任务（可能与一个或一个以上其它处理任务协作）的虚拟模型，且应了解，CTA 线程可能或可能不一对一地映射到目标平台上的线程。

[0063] III. 虚拟结构

[0064] 根据本发明的一个方面，定义用于执行 CTA 和 CTA 栅格的虚拟并行结构。所述虚拟并行结构是支持执行能够在所需时间实现彼此的协作行为（例如，共享数据和同步）的大量并发 CTA 线程的并行处理器和相关联的存储器空间的表示。此虚拟并行结构可映射到多种实际处理器和 / 或处理系统上，包含（例如）图 1 的系统 100 的 PPU 122。虚拟结构有利地定义支持不同等级的数据共享和存取类型的若干虚拟存储器空间，以及识别可由虚拟处理器执行的所有功能的虚拟指令集结构 (ISA)。虚拟结构还有利地（例如）通过定义和启动 CTA 或 CTA 栅格来定义可用于控制 CTA 执行的虚拟执行驱动器。

[0065] 图 3 是根据本发明实施例的虚拟结构 300 的方框图。虚拟结构 300 包含带有虚拟核心 308 的虚拟处理器 302，所述虚拟核心 308 经配置以并行执行大量 CTA 线程。虚拟结构 300 还包含可由虚拟处理器 302 存取的全局存储器 304，和供应控制虚拟处理器 302 的操作的命令的虚拟驱动器 320。虚拟驱动器 320 也可存取全局存储器 304。

[0066] 虚拟处理器 302 包含接收和解译来自虚拟驱动器 320 的命令的前端 306，以及能够并发执行单一 CTA 的所有 n_0 个线程的执行核心 308。虚拟核心 308 包含大量（n_0 个或 n_0 个以上）虚拟处理引擎 310；在一个实施例中，每一虚拟处理引擎 310 执行一个 CTA 线程。虚拟处理引擎 310 并发执行其各自 CTA 线程，但不一定并行执行。在一个实施例中，虚拟结构 300 指定虚拟处理引擎 310 的数目 T（例如，384、500、768 等）；此数目设定 CTA 中的线程数目 n_0 的上限。应了解，虚拟结构 300 的实现可包含少于指定数目 T 的物理处理引擎，且单一处理引擎可执行若干 CTA 线程，作为单一“真实”（即，平台支持的）线程或作为多个并发的真实线程。

[0067] 虚拟处理器 302 还包含虚拟指令单元 312，其保持向虚拟处理引擎 310 供应针对其各自 CTA 线程的指令；所述指令由作为虚拟结构 300 的一部分的虚拟 ISA 定义。下文描述用于并行线程计算的虚拟 ISA 的实例。指令单元 312 在向虚拟处理引擎 310 供应指令的过程中管理 CTA 线程同步和 CTA 线程行为的其它协作性方面。

[0068] 执行核心 308 提供具有不同等级的可存取性的内部数据存储。特殊寄存器 311 可由虚拟处理引擎 310 读取但不可由其写入，且用于存储定义图 2 的问题分解模型内每一 CTA 线程的“位置”的参数。在一个实施例中，特殊寄存器 311 对于每 CTA 线程（或每虚拟处理引擎 310）包含一个存储线程 ID 的寄存器；每一线程 ID 寄存器仅可由虚拟处理引擎 310 中的个别虚拟处理引擎 310 存取。特殊寄存器 311 还可包含可由所有 CTA 线程（或由所有虚拟处理引擎 310）读取的额外寄存器，其存储 CTA 识别符、CTA 维度、CTA 所属的栅格的维度，和 CTA 所属的栅格的识别符。在初始化期间响应于经由前端 306 从虚拟驱动器 320 接收的命令而写入特殊寄存器 311，且特殊寄存器 311 在 CTA 执行期间不改变。

[0069] 局部虚拟寄存器 314 由每一 CTA 线程用作临时空间 (scratch space)；分配每一寄存器专用于一个 CTA 线程（或一个虚拟处理引擎 310），且局部寄存器 314 的任一者中的数

据仅可由其被分配到的 CTA 线程存取。共享存储器 316 可由所有 CTA 线程（单一 CTA 内）存取；共享存储器 316 中的任何位置可由同一 CTA 内的任何 CTA 线程存取（或可由虚拟核心 308 内的任何虚拟处理引擎 310 存取）。参数存储器 318 存储可由任何 CTA 线程（或任何虚拟处理引擎 310）读取但不可由其写入的运行时间参数（常数）。在一个实施例中，虚拟驱动器 320 在引导虚拟处理器 302 开始执行使用这些参数的 CTA 之前将参数提供到参数存储器 318。任何 CTA 内的任何 CTA 线程（或虚拟核心 308 内的任何虚拟处理引擎 310）均可通过存储器接口 322 存取全局存储器 304。

[0070] 在虚拟结构 300 中，虚拟处理器 302 在虚拟驱动器 320 的控制下作为协处理器操作。虚拟结构规范有利地包含虚拟应用程序接口 (API)，其识别由虚拟驱动器 320 认可的函数调用和每一函数调用预期产生的行为。下文描述用于并行线程计算的虚拟 API 的实例函数调用。

[0071] 虚拟结构 300 可在多种硬件平台上实现。在一个实施例中，虚拟结构 300 在图 1 的系统 100 中实现，其中 PPU 122 实施虚拟处理器 302，且在 CPU 102 上执行的 PPU 驱动器程序实施虚拟驱动器 320。全局存储器 304 可在系统存储器 104 和 / 或 PP 存储器 124 中实施。

[0072] 在一个实施例中，PPU 122 包含一个或一个以上处理核心，其使用单指令多数据 (SIMD) 和多线程技术来支持来自单一指令单元（实施虚拟指令单元 312）的大量（例如，384 或 768 个）线程的并发执行。每一核心包含 P 个（例如，8、16 等）并行处理引擎 302 的阵列，其经配置以从指令单元接收并执行 SIMD 指令，从而允许并行处理具有多达 P 个线程的群组。核心为多线程的，其中每一处理引擎能够（例如）通过维持与每一线程相关联的当前状态信息使得处理引擎可从一个线程快速切换到另一线程，而并发执行多达某一数目 G（例如，24 个）的线程群组。因此，核心并发执行各有 P 个线程的 G 个 SIMD 群组，总计为 P*G 个并发线程。在此实现过程中，只要 $P*G \geq n_0$，（虚拟）CTA 线程与在真实 PPU 122 上执行的并发线程之间就可存在一对一对应。

[0073] 特殊寄存器 311 可通过以下操作而实施在 PPU 122 中：向每一处理核心提供 P*G 条目寄存器堆，其中每一条目能够存储一线程 ID；以及提供一组全局可读取的寄存器来存储 CTA ID、栅格 ID 和 CTA 及栅格维度。或者，可使用其它存储位置来实施特殊寄存器 311。

[0074] 局部寄存器 314 可实施在 PPU 122 中，作为在物理上或逻辑上划分为 P 个巷道 (lane) 的局部寄存器堆，其中每一巷道具有某一数目的条目（其中每一条目可能存储（例如）32 位字）。一个巷道被分派到 P 个处理引擎中的每一者，且不同巷道中的相应条目可用执行相同程序以有助于 SIMD 执行的不同线程的数据填充。巷道的不同部分可分配到 G 个并发线程群组中的不同线程群组，使得局部寄存器堆中的给定条目仅可由特定线程存取。在一个实施例中，局部寄存器堆内的某些条目保留用于存储线程识别符，从而实施特殊寄存器 311 中的一者。

[0075] 共享存储器 316 可实施在 PPU 122 中作为共享寄存器堆或共享芯片上高速缓冲存储器，其具有允许任何处理引擎对共享存储器中的任何位置进行读取或写入的互连。参数存储器 318 可实施在 PPU 122 中作为实施共享存储器 316 的同一共享寄存器堆或共享高速缓冲存储器内的指定区域，或作为处理引擎具有只读 (read-only) 存取权的单独共享寄存器堆或芯片上高速缓冲存储器。在一个实施例中，实施参数存储器的区域还用于存储 CTA

ID 和栅格 ID 以及 CTA 和栅格维度，从而实施特殊寄存器 311 的若干部分。

[0076] 在一个实施例中，在图 1 的 CPU 102 上执行的 PPU 驱动器程序通过将命令写入到存储器（例如，系统存储器 104）中的推播缓冲器 (pushbuffer)（未明确展示）来响应虚拟 API 函数调用，PPU 122 从所述推播缓冲器处读取所述命令。所述命令有利地与状态参数相关联，所述状态参数例如 CTA 中线程的数目、待使用 CTA 处理的输入数据集在全局存储器中的位置、待执行的 CTA 程序在全局存储器中的位置，和输出数据将被写入到的全局存储器中的位置。响应于命令和状态参数，PPU 122 将状态参数载入到其核心中的一者中，接着开始启动线程直到已启动 CTA 参数中指定的数目的线程为止。在一个实施例中，PPU 122 包含控制逻辑，其在启动线程时将线程 ID 依次分派到线程；可将线程 ID 存储（例如）在局部寄存器堆内或专用于此目的的特殊寄存器中的指定位置。

[0077] 在替代实施例中，在单线程处理核心中（例如，在一些 CPU 中）实现虚拟结构 300，所述单线程处理核心使用少于 n_0 个实际线程来执行所有 CTA 线程；虚拟编程模型使得与不同 CTA 线程相关联的处理任务可（例如）通过针对一个 CTA 线程接着针对下一 CTA 线程等等执行任务（或其一部分）而组合到单一线程中。可利用向量执行、SIMD 执行和 / 或机器中可用的任何其它形式的并行性来并行执行与多个 CTA 线程相关联的处理任务，或并行执行与同一 CTA 线程相关联的多个处理任务。因此，可使用单一线程、n_0 个线程或任何其它数目的线程来实现 CTA。如下文所描述，虚拟指令翻译器有利地将编写为目标虚拟结构 300 的代码翻译为特定针对目标平台的指令。

[0078] 将了解，本文描述的虚拟结构是说明性的且可能作出变化和修改。举例来说，在一个替代实施例中，每一虚拟处理引擎可具有存储分派到其线程的唯一线程 ID 的专门的线程 ID 寄存器，而不使用局部虚拟寄存器中的空间用于此目的。

[0079] 作为另一实例，虚拟结构可指定关于虚拟核心 308 的内部结构的或多或少的细节。举例来说，可能指定虚拟核心 308 包含用于执行 P 向 SIMD 群组中的 CTA 线程的 P 个多线程虚拟处理引擎，其中多达 G 个 SIMD 群组共存于核心 308 中使得 P*G 确定 T(CTA 中线程的最大数目）。还可指定不同类型的存储器和共享等级。

[0080] 虚拟结构可实现于使用硬件和 / 或软件元件的任何组合的多种计算机系统中以定义和控制每一组件。虽然已以举例的方式描述使用硬件组件的一个实现方案，但应了解，本发明涉及使编程任务与特定硬件实现脱离。

[0081] 4. 对虚拟结构进行编程

[0082] 图 4 是根据本发明实施例使用虚拟结构 300 来操作目标处理器或平台 440 的概念模型 400。如模型 400 所展示，虚拟结构 300 的存在使经编译的应用程序和 API 与目标处理器或平台的硬件实施脱离。

[0083] 应用程序 402 定义数据处理应用，其利用上述虚拟编程模型（包含单一 CTA 和 / 或 CTA 栅格）。一般来说，应用程序 402 包含多个方面。第一，程序定义单一 CTA 线程的行为。第二，程序定义 CTA 的维度（以 CTA 线程的数目计），且如果将使用栅格，那么定义栅格的维度（以 CTA 的数目计）。第三，程序定义待由 CTA（或栅格）处理的输入数据集和输出数据集将被存储在的位置。第四，程序定义总体处理行为，包含（例如）何时启动每一 CTA 或栅格。程序可包含动态确定 CTA 或栅格的维度、是否不断启动新的 CTA 或栅格等的额外代码。

[0084] 可用例如 C/C++、FORTRAN 等高级编程语言编写应用程序 402。在一个实施例中，C/C++ 应用程序直接指定一个（虚拟）CTA 线程的行为。在另一实施例中，使用数据并行语言（例如，Fortran 90、C* 或数据并行 C）编写应用程序，且应用程序指定对阵列和聚合数据结构的数据并行操作；此程序可编译成指定一个（虚拟）CTA 线程的行为的虚拟 ISA 程序代码。为了允许定义 CTA 线程的行为，可提供语言扩充或函数库，编程人员可经由所述语言扩充或函数库来指定并行 CTA 线程行为。举例来说，可定义特殊符号或变量以对应于线程 ID、CTA ID 和栅格 ID，且可提供函数，编程人员可经由所述函数来指示 CTA 线程应何时与其它 CTA 线程同步。

[0085] 当编译应用程序 402 时，编译器 408 针对应用程序 402 的定义 CTA 线程行为的那些部分产生虚拟 ISA 代码 410。在一个实施例中，以图 3 的虚拟结构 300 的虚拟 ISA 表达虚拟 ISA 代码 410。虚拟 ISA 代码 410 是程序代码，但不一定是可在特定目标平台上执行的形式的代码。如此，虚拟 ISA 代码 410 可作为任何其它程序代码被存储和 / 或分布。在其它实施例中，可将应用程序完全或部分指定为虚拟 ISA 代码 410，且可完全或部分避免使用编译器 408。

[0086] 虚拟指令翻译器 412 将虚拟 ISA 代码 410 转换为目标 ISA 代码 414。在一些实施例中，目标 ISA 代码 414 是可由目标平台 440 直接执行的代码。举例来说，如由图 4 中的虚线框所示，在一个实施例中，目标 ISA 代码 414 可由 PPU 122 中的指令单元 430 接收和正确解码。视目标平台 440 的特殊性而定，虚拟 ISA 代码 410 可能被翻译成待由目标 平台 440 上的 n_0 个线程的每一者执行的每线程代码。或者，虚拟 ISA 代码 410 可能被翻译成将在少于 n_0 个的线程中执行的程序代码，其中每一线程包含与 CTA 线程中的一者以上有关的处理任务。

[0087] 在一些实施例中，CTA 和 / 或栅格的维度的定义以及定义输入数据集和输出数据集是由虚拟 API 处理的。应用程序 402 可包含对于虚拟 API 函数库 404 的调用。在一个实施例中，将虚拟 API 的规范（包含（例如）函数名称、输入、输出和作用，但不包含实施细节）提供给编程人员，且编程人员将虚拟 API 调用直接并入到应用程序 402 中，藉此直接产生虚拟 API 代码 406。在另一实施例中，通过编译使用某一其它语法来定义 CTA 和栅格的应用程序 402 来产生虚拟 API 代码 406。

[0088] 部分地通过提供虚拟执行驱动器 416 来实现虚拟 API 代码 406，虚拟执行驱动器 416 将代码 406 的虚拟 API 命令翻译为可由目标平台 440 处理的目标 API 命令 418。举例来说，如由图 4 中的虚线框所示，在一个实施例中，目标 API 命令 418 可由 PPU 驱动器 432 接收和处理，PPU 驱动器 432 将相应命令传送到 PPU 122 前端 434。（在此实施例中，虚拟执行驱动器 416 可以是 PPU 驱动器 432 的一方面或部分。）在另一实施例中，虚拟执行驱动器可能不对应于用于协处理器的驱动器；其可能只是在运行所述虚拟执行驱动器的同一处理器上启动其它程序或线程的控制程序。

[0089] 应了解，可针对能够支持 CTA 执行的任何平台或结构创建虚拟指令翻译器 412 和虚拟执行驱动器 416。就针对不同平台或结构的虚拟指令翻译器 412 可从同一虚拟 ISA 进行翻译来说，同一虚拟 ISA 代码 410 可与任何平台或结构一起使用。因此，不需要针对每一可能的平台或结构重新编译应用程序 402。

[0090] 此外，目标平台 440 不必包含如图 4 所示的 PPU 和 / 或 PPU 驱动器。举例来说，在

一个替代实施例中，目标平台是使用软件技术仿真大量线程的并发执行的 CPU，且目标 ISA 代码和目标 API 命令对应于待由目标 CPU 执行的程序（或互相通信的程序的群组）中的指令，所述目标 CPU 可以是（例如）单核心或多核心 CPU。

[0091] 5. 虚拟 ISA 实例

[0092] 现在将描述根据本发明实施例的虚拟 ISA 的实例。如上所述，虚拟 ISA 有利地对应于上述虚拟编程模型 (CTA 和栅格）。因此，在本实施例中，由编译器 408 产生的虚拟 ISA 代码 410 定义要由图 3 的虚拟核心 308 中的虚拟处理引擎 310 之一执行的单个 CAT 线程的行为；所述行为可包含与其它 CTA 线程的协作性交互，例如同步和 / 或数据共享。

[0093] 应了解，本文中描述的虚拟 ISA 只用于说明的用途，且本文中描述的特定要素或要 素组合并不限制本发明的范围。在一些实施例中，编程人员可用虚拟 ISA 编写代码；在其它实施例中，编程人员用另一高级语言（例如，FORTRAN，C，C++）编写代码，且编译器 408 产生虚拟 ISA 代码。编程人员也可编写“混合”代码，其中代码中有些部分是用高级语言编写而其它部分是用虚拟 ISA 编写。

[0094] 5.1 特殊变量

[0095] 图 5 是列举由实例虚拟 ISA 定义的“特殊”变量的表格 500（本文中用前缀“%”来表示特殊变量）。这些变量与图 2 的编程模型有关，其中通过每个线程 204 在 CTA 202 内的位置来识别所述线程，而 CTA 202 又位于某一数目的栅格 200 中的特定一者内。在一些实施例中，表格 500 的特殊变量对应于图 3 的虚拟结构 300 中的特殊寄存器 311。

[0096] 在表格 500 中，假设 CTA 和栅格各自用三维空间来定义，且不同的栅格在一维空间中循序编号。虚拟 ISA 预期图 5 的特殊变量将在启动 CTA 时被初始化，且虚拟 ISA 代码可简单地使用这些变量而无需初始化。以下参看虚拟 API 论述特殊变量的初始化。

[0097] 如图 5 所示，特殊变量的第一个 3 向量 % ntid = (% ntid.x，% ntid.y，% ntid.z) 定义 CTA 的维度（以线程数目计）。CTA 中的所有线程将共享同一 % ntid 向量。在虚拟结构 300 中，预期 % ntid 向量的值将经由虚拟 API 函数调用提供给虚拟处理器 302，所述虚拟 API 函数调用如下所述地建立 CTA 的维度。

[0098] 如图 5 所示，特殊变量的第二个 3 向量 % tid = (% tid.x，% tid.y，% tid.z) 是指 CTA 内的给定线程的线程 ID。在图 3 的虚拟结构 300 中，预期虚拟处理器 302 将在启动 CTA 的每个线程时分派满足制约条件 0 ≤ % tid.x < % ntid.x、0 ≤ % tid.y < % ntid.y 和 0 ≤ % tid.z < % ntid.z 的唯一 % tid 向量。在一个实施例中，% tid 向量可定义成使得其可存储在压缩的 32 位的字中（例如，对于 % tid.x 为 16 个位，对于 % tid.y 为 10 个位，且对于 % tid.z 为 6 个位）。

[0099] 如图 5 所示，特殊变量的第三个 3 向量 % nctaid = (% nctaid.x，% nctaid.y，% nctaid.z) 定义栅格的维度（以 CTA 的数目计）。在图 3 的虚拟结构 300 中，预期 % nctaid 向量的值将经由虚拟 API 函数调用提供给虚拟处理器 302，所述虚拟 API 函数调用建立 CTA 的栅格的维度。

[0100] 如图 5 所示，特殊变量的第四个 3 向量 % ctaid = (% ctaid.x，% ctaid.y，% ctaid.z) 是指栅格内的给定 CTA 的 CTA ID。在图 3 的虚拟结构 300 中，预期当启动 CTA 时将把满足 CTA 的限制条件 0 ≤ % ctaid.x < % nctaid.x、0 ≤ % ctaid.y < % nctaid.y 和 0 ≤ % ctaid.z < % nctaid.z 的唯一 % ctaid 向量提供给虚拟处理器 302。

[0101] 特殊变量还包含提供 CTA 所属的栅格的栅格识别符的标量 % gridid 变量。在图 3 的虚拟结构 300 中，预期将把 % gridid 值提供给虚拟处理器 302，以便识别包括当前 CTA 的栅格。例如当正在使用多个栅格来解决较大问题的不同部分时，有利地在虚拟 ISA 代码中使用 % gridid 值。

[0102] 5.2 程序定义的变量和虚拟状态空间

[0103] 虚拟 ISA 使得编程人员（或编译器）可定义任意数目的变量以代表正被处理的数据项目。通过类型和指示如何使用变量以及变量在何种程度上共享的“虚拟状态空间”来定义变量。使用目标平台中可用的寄存器或其它存储器结构来实现变量；在许多目标平台中，状态空间可能会影响对用来实现特定变量的存储器结构的选择。

[0104] 图 6 是列举实例虚拟 ISA 实施例中支持的变量类型的表格 600。支持四个类型：未指定类型的位、带符号的整数、无符号整数和浮点。未指定类型的变量就是单个位或具有指定长度的位的群组。可根据常规格式（例如，IEEE 754 标准）来定义带符号整数和无符号整数格式以及浮点格式。

[0105] 在本实施例中，针对每个类型支持多个宽度，其中使用参数 <n> 来指定宽度；因此，举例来说，.s16 指示 16 个位的带符号的整数，.f32 指示 32 位的浮点数字等。如表格 600 所示，有些变量类型限于特定的宽度；举例来说，浮点变量必须至少为 16 个位；且整数类型必须至少为 8 个位。预期虚拟 ISA 的实现支持所有指定宽度；如果处理器的数据路径和 / 或寄存器比最宽宽度窄，那么如此项技术中已知的可使用多个寄存器和处理器循环来处置较宽类型。

[0106] 应了解，本文中使用的数据类型和宽度是说明而不是限制本发明。

[0107] 图 7 是列举实例虚拟 ISA 中支持的虚拟状态空间的表格。定义了九个状态空间，其对应于不同的共享级别和在图 3 的虚拟结构 300 中的可能的存储位置。

[0108] 在线程级别上共享前三个状态空间，这意味着每个 CTA 线程将具有单独的变量实例，且没有任何 CTA 线程将可存取任何其它 CTA 线程的实例。有利地使用虚拟寄存器 (.reg) 状态空间来定义要由每个 CTA 线程执行的计算的运算数、临时值和 / 或结果。程序可宣称任何数目的虚拟寄存器。只能通过静态编译时间名称而不是计算出来的地址来寻址虚拟寄存器。这个状态空间对应于图 3 的虚拟结构 300 中的局部虚拟寄存器 314。

[0109] 特殊寄存器 (.sreg) 状态空间对应于图 5 的预定义的特殊变量，其存储在虚拟结构 300 中的特殊寄存器 311 中。在一些实施例中，虚拟 ISA 代码可能不会宣称 .sreg 空间中的任何其它变量，而是可使用特殊变量作为对计算的输入。所有 CTA 线程均可读取 .sreg 状态空间中的任何变量。对于 % tid（或其分量），每个 CTA 线程将读取其唯一的线程识别符；对于 .sreg 状态空间中的其它变量，同一 CTA 中的所有 CTA 线程将读取相同值。

[0110] 每线程局部存储器 (.local) 变量对应于全局存储器 304 中在每 CTA 线程基础上分配和寻址的区域。换句话说，当 CTA 线程存取 .local 变量时，其存取其自身的变量实例，且在一个 CTA 线程中对 .local 变量作出的改变不会影响其它 CTA 线程。与 .reg 和 .sreg 状态空间不同，每线程局部存储器可使用计算出来的地址来寻址。

[0111] 接下来的两个状态空间定义每 CTA 变量，这意味着每个 CTA 将具有变量的一个实例，其任何（虚拟）线程均可存取所述实例。任何 CTA 线程均可读取或写入共享 (.shared) 变量。在一些实施例中，这个状态空间映射到虚拟结构 300（图 3）的虚拟共享存储器 316。

在虚拟结构 300 的实现中，.shared 状态空间可映射到芯片上共享存储器实施方案（例如，共享寄存器堆或共享高速缓冲存储器）上，而在其它实现中，.shared 状态空间可映射到芯片外存储器的分配和寻址为任何其它可全局存取存储器的每 CTA 区域上。

[0112] 参数 (.param) 变量是只读的，且可由 CTA 中的任何（虚拟）线程读取。这个状态空间映射到虚拟结构 300 的参数存储器 318，且可（例如）在芯片上共享参数存储器或高速缓冲存储器中或者在可全局存取的芯片外存储器的分配和寻址为其它任何可全局存取存储器的区域中实现。预期这些变量将响应于来自虚拟驱动器 320 的驱动器命令来初始化。

[0113] 常量 (.const) 状态空间用来定义可由栅格中的任何 CTA 中的任何（虚拟）线程读取（但不可修改）的每栅格常量。在虚拟结构 300 中，.const 状态空间可映射到全局存储器中可由 CTA 线程进行只读存取的区域。.const 状态空间可在芯片上共享参数存储器或高速缓冲存储器中或在可全局存取芯片外存储器的分配和寻址为任何其它可全局存取存储器的每栅格区域中实现。与 .param 状态空间一样，预期 .const 状态空间中的变量将响应于来自虚拟驱动器 320 的驱动器命令而被初始化。

[0114] 其余三个状态空间定义“上下文”变量，其可供与应用程序相关联的任何 CTA 中的任何（虚拟）线程存取。这些状态空间映射到虚拟结构 300 中的全局存储器 304。全局 (.global) 变量可用于一般用途。在一些实施例中，也可定义用于共享纹理 (.tex) 和表面 (.surf) 的特定状态空间。这些可用于（例如）图形相关应用程序的状态空间可用来定义并提供对图形纹理和像素表面数据结构的存取，所述数据结构提供对应于 2D（或在一些实施例中为 3D）阵列的每个像素的数据值。

[0115] 在图 4 的虚拟 ISA 代码 410 中，通过指定状态空间、类型和名称来宣称变量。名称是占位符，且可由编程人员或编译者来选择。因此，例如：

[0116] .reg.b32 vr1；

[0117] 宣称虚拟寄存器状态空间中名为 vr1 的 32 个位的未指定类型变量。虚拟 ISA 代码的后续行可引用 vr1（例如）作为操作的来源或目的地。

[0118] 实例虚拟 ISA 还支持虚拟变量的阵列和向量。举例来说：

[0119] .global.f32 resultArray[1000][1000]；

[0120] 宣称 32 位浮点数字的虚拟可全局存取 1000×1000 阵列。虚拟指令翻译器 412 可将阵列映射到对应于分派的状态空间的可寻址存储器区域中。

[0121] 可使用向量前缀 .v<m> 来定义一个实施例中的向量，其中 m 是向量的分量的数目。举例来说：

[0122] .reg.v3.f32 vpos；

[0123] 宣称每线程虚拟寄存器状态空间中的 32 位浮点数字的 3 分量向量。一旦宣称了向量，便可使用例如 vpos.x，vpos.y，vpos.z 等后缀来识别其分量。在一个实施例中，允许 m = 2、3 或 4，且可使用例如 (.x，.y，.z，.w)、(.0，.1，.2，.3) 或 (.r，.g，.b，.a) 等后缀来识别分量。

[0124] 由于变量是虚拟的，所以虚拟 ISA 代码 410 可定义或引用任何状态空间（除了 .sreg，其中变量是预定义的）中的任何数目的变量。可能针对虚拟 ISA 代码 410 中的特定状态空间定义的变量的数目可能会超过特定硬件实施方案中的相应类型的存储量。虚拟指令翻译器 412 有利地经配置以包含合适的存储管理指令（例如，在寄存器与芯片外存储

器之间移动数据），以便使变量在需要时可用。虚拟指令翻译器 412 也可能能够检测到不再需要临时变量的情况并允许为其分配的空间供另一变量重新使用；可使用常规的用于分配寄存器的编译器技术。

[0125] 此外，虽然实例虚拟 ISA 定义向量变量类型，但不要求目标平台支持向量变量。虚拟指令翻译器 412 可将任何向量变量实施为适当数目（例如，2、3 或 4）个标量的集合。

[0126] 5.3. 虚拟指令

[0127] 图 8A-8H 是列举实例虚拟 ISA 中定义的虚拟指令的表格。通过指令的效果来定义指令，所述效果例如是使用一个或一个以上运算数计算出特定结果并将所述结果放置在目的地寄存器中、设置寄存器值等。将大多数虚拟指令分类以识别输入和 / 或输出的格式，且指令执行的各方面可取决于类型。指令的一般格式是：

[0128] name.<type>result, operands ;

[0129] 其中 name 是指令名称；.<type> 是图 6 中列举的任一类型的占位符；result 是存储结果的变量；且 operands 是提供为对指令的输入的一个或一个以上变量。在一个实施例中，虚拟结构 300 是寄存器到寄存器处理器，且用于除存储器存取之外的操作的 result 和 operands（图 8F）被要求是在虚拟寄存器状态空间 .reg（或在一些运算数的情况下是特殊寄存器状态空间 .sreg）中的变量。

[0130] 预期目标平台实现虚拟 ISA 中的每个指令。指令可实现为产生指定效果的相应的机器指令（本文中称为“硬件支持”），或实现为在执行时产生指定效果的机器指令序列（本文中称为“软件支持”）。特定目标平台的虚拟指令翻译器 412 有利地经配置以识别对应于每个虚拟指令的机器指令或机器指令序列。

[0131] 以下子章节描述图 8A-8H 中列举的各种类别的指令。应了解，本文中提供的指令列表是说明性的，且虚拟 ISA 可包含本文中未明确描述的额外指令，且可不包括本文中描述的一些或所有指令。

[0132] 5.3.1. 虚拟指令 - 算术

[0133] 图 8A 是列举实例虚拟 ISA 中定义的算术运算的表格 800。在此实施例中，虚拟结构只支持寄存器到寄存器算术，且所有算术运算均操纵一个或一个以上虚拟寄存器运算数（图 8A 中表示为 a、b、c），以产生写入到虚拟寄存器的结果 (d)。因此，算术运算的运算数和目的地始终在虚拟寄存器状态空间 .reg 中，除了图 5 的特殊寄存器（在特殊寄存器状态空间 .sreg 中）可用作运算数以外。

[0134] 表格 800 中的算术运算的列表包含四个基本的算术运算：加法 (add)、减法 (sub)、乘法 (mul) 和除法 (div)。这些运算可对所有整数和浮点数据类型执行，且产生相同类型的结果作为输入；在一些实施例中，也可将舍入模式的限定符 (qualifier) 添加到指令中，以允许编程人员指定应当如何对结果进行舍入，以及在整数运算数的情况下是否应当强加饱和极限。

[0135] 也支持 a、b 和 c 运算数的三个复合算术运算：乘加 (mad)、熔合乘加 (fma) 以及绝对差和 (sad)。乘加计算 a×b 的乘积（具有括号指示的舍入），且将结果与 c 相加。熔合乘加与 mad 的区别在于，在与 c 相加之前，不对 a×b 的乘积进行舍入。绝对差和计算绝对值 |a-b|，然后与 c 相加。

[0136] 余数 (rem) 运算只对整数运算数执行，且当将运算数 a 除以运算数 b 时计算余数

(amod b)。绝对值 (abs) 和求反 (neg) 是可应用于浮点格式或带符号的整数格式的运算数 a 的一元运算。可应用于整数或浮点运算数的最小值 (min) 和最大值 (max) 运算将目的地寄存器设置成较小运算数或较大运算数；也可指定对一个或两个运算数是非正规数（例如，根据 IEEE 754 标准）的特殊情况的处置。

[0137] 表格 800 中的其余运算只对浮点类型执行。分数 (frc) 运算返回其输入的分数部分。正弦 (sin)、余弦 (cos) 和比率的反正切 (atan2) 提供对应于三角函数的方便的指令。也支持以 2 为底的对数 (lg2) 和取幂 (ex2)。也支持倒数 (rcp)、平方根 (sqrt) 和倒数平方根 (rsqrt)。

[0138] 应注意，这个算术运算列表是说明而不是限制本发明。可支持其它运算或运算组合，其中包含任何预期会以足够的频率被调用的运算。

[0139] 在一些实施例中，虚拟 ISA 还定义向量运算。图 8B 是列举实例虚拟 ISA 支持的向量运算的表格 810。向量运算包含点积 (dot) 运算，其计算运算数向量 a 和 b 的标量点积 d；叉积 (cross) 运算，其计算运算数向量 a 和 b 的向量叉积 d；以及量值 (mag) 运算，其计算运算数向量 a 的标量长度 d。向量归约 (vred) 运算通过以迭代方式对向量运算数 a 的要素执行指定操作 <op> 来计算标量结果 d。在一个实施例中，对于浮点向量只支持归约运算 add、mul、min 和 max；对于整数向量，也可支持额外的归约运算（例如，and、or 和 xor，如下文所述）。

[0140] 除了这些运算之外，也可在虚拟 ISA 中定义例如向量加法、向量定标等其它向量运算（图 8B 中未列举）。

[0141] 如上所述，虚拟结构 300 的一些硬件实现可能不支持向量处理。此种实现的虚拟指令翻译器 412 有利地适合于产生标量机器指令的适当序列以执行这些运算；所属领域的技术人员将能够确定适当的序列。

[0142] 5.3.2 虚拟指令 - 选择和设置寄存器

[0143] 图 8C 是列举实例虚拟 ISA 中定义的选择和设置寄存器运算的表格 820。这些运算可对任何数值数据类型执行，且基于比较运算的结果设置目的地寄存器。如果 c 是非零那么基本选择 (sel) 运算选择运算数 a，且如果 c 是零则选择运算数 b。比较与设置 (set) 对运算数 a 和 b 执行比较运算 <cmp> 以产生比较结果 t，接着基于比较结果 t 是真（～ 0）还是假 (0) 而将目的地寄存器 d 设置成布尔 (Boolean) 真（～ 0）或假 (0)。一个实施例中允许的比较运算 <cmp> 包含等于（如果 a ＝ b 则 t 为真）、大于（如果 a>b 则 t 为真）、小于（如果 a<b 则 t 为真）、大于或等于（如果 a ≥ b 则 t 为真）、小于或等于（如果 a ≤ b 则 t 为真），以及其它比较，其中包含例如 a 和 / 或 b 是数字值还是未定义的值。

[0144] Setb 运算是对比较与设置的变化形式，其在比较运算 <cmp> 的结果 t 与第三运算数 c 之间执行进一步的布尔运算 <bop>；布尔运算 t<bop>c 的结果确定是将目的地寄存器 d 设置成布尔真还是布尔假。一个实施例中允许的布尔运算 <bop> 包含 and、or 和 xor（见下述图 8C）。setp 运算与 setb 相似，区别只是设置了两个 1 位“断言”(predicate) 目的地寄存器：将目的地寄存器 d1 设置成 t<bop>c 的结果，而将目的地寄存器 d2 设置成 (!t)<bop>c 的结果。

[0145] 5.3.3. 虚拟指令 - 逻辑和位操纵

[0146] 图 8D 是列举实例虚拟 ISA 中定义的逻辑和位操纵运算的表格 830。按位的布尔运

算 and、or 和 xor 通过以下方式来执行 :对运算数 a 和 b 的每个位执行指定运算,且将寄存器 d 中的相应位设置成结果。按位求反 (not) 运算反转运算数 a 的每个位,而如果 a 是零(布尔假)则逻辑求反 (cnot) 运算将目的地寄存器设置成 1(布尔真),否则设置成 0(布尔假)。

[0147] 通过左移 (shl) 和右移 (shr) 运算来支持移位,所述运算将运算数 a 中的位字段左移或右移运算数 b 指定的位数。对于带符号的格式,右移有利地基于符号位来填补引导位 ;对于无符号格式,右移用零来填补引导位。

[0148] 5.3.4. 虚拟指令 - 格式转换

[0149] 图 8E 是列举实例虚拟 ISA 中定义的格式转换运算的表格 840。格式转换 (cvt) 指令将第一类型 <atpye> 的运算数 a 转换成目标类型中的均等值 <dtpye>,且将结果存储在目的地寄存器 d 中。图 6 中列举一个实施例中的有效类型 ;无法将未指定类型值 (.b<n>) 转换成整数或浮点类型或者将整数或浮点类型转换成未指定类型值。格式转换指令的变化形式使得编程人员可指定舍入模式 <mode> ;也可指定对当表达为目标类型时饱和的数字的处置。

[0150] 5.3.5. 虚拟指令 - 数据移动和数据共享

[0151] 图 8F 是列举实例虚拟 ISA 中定义的数据移动和数据共享指令的表格 850。移动 (mov) 操作将目的地寄存器 d 设置成立即运算数 a 的值,或者如果运算数 a 是寄存器,则设置成寄存器 a 的内容。移动操作可限于虚拟寄存器类型的状态空间,例如图 7 中的 .reg 和 .sreg。

[0152] 加载 (ld) 指令将来自存储器中的源位置的值加载到目的地寄存器 d 中,所述目的地寄存器 d 在一个实施例中必须在虚拟寄存器 (.reg) 状态空间中。.<space> 限定符指定源位置的状态空间,且可限于图 7 中的可寻址状态空间,例如 .reg 和 .sreg 之外的空间(其中可改为使用移动操作)。由于这个实施例中的虚拟结构 300 是寄存器到寄存器处理器,所以加载指令有利地用来将变量从可寻址的状态空间转移到虚拟寄存器 .reg 状态空间中,使其可用作运算数。

[0153] 使用可用各种方式定义的来源参数 <src> 来识别特定的源位置,以便支持不同的寻址模式。举例来说,在一些实施例中,来源参数 <src> 可能是以下中的任何一者 :其值存储在 d 中的经命名的可寻址变量、对保存来源地址的寄存器的引用、对保存将与偏移值(提供为立即运算数)相加的地址的寄存器的引用,或立即绝对地址。

[0154] 类似地,存储 (st) 操作将来源寄存器 a 中的值存储到由目的地参数 <dst> 识别的存储器位置。一个实施例中的来源寄存器 a 必须在 .reg 状态空间中 ;目的地必须在可写入和可寻址的状态空间(例如,图 7 中的 .local、.global 或 .shared) 中。目的地参数 <dst> 可用各种方式定义以支持不同的寻址模式,这与加载指令中的来源参数 <src> 相似。例如,可使用存储指令将操作结果从寄存器转移到可寻址的状态空间。

[0155] 在提供纹理和表面状态空间的实施例中,可使用额外的虚拟指令从纹理存储器状态空间进行读取 (tex) 和从表面存储器状态空间进行读取 (suld) 和对其进行写入 (sust)。纹理读取的运算数 (t,x,y) 指定纹理识别符 (t) 和坐标 (x,y) ;同样,表面读取或写入的运算数 (s,x,y) 指定表面识别符 (s) 和坐标 (x,y)。

[0156] CTA 线程可通过与其它 CTA 线程共享数据而与其它 CTA 线程协作。举例来说,为

了在CTA内共享数据,CTA线程可使用加载与存储虚拟指令(以及下述原子更新指令atom)以将数据写入到每CTA虚拟状态空间和从中读取数据。因此,一个CTA线程可使用具有合适定义的目的地地址的st.shared指令将数据写入到.shared状态空间;相同CTA内的另一CTA线程可随后通过在ld.shared指令中使用相同地址来读取所述数据。下述同步指令(例如,bar和membar)可用来确保CTA线程间的数据共享操作的正确序列,例如,产生数据的CTA线程在消耗数据的CTA线程读取数据之前先写入数据。类似地,st.global和ld.global指令可用于相同CTA中的CTA线程、相同栅格中的CTA和/或相同应用程序中的不同栅格之间的协作和数据共享。

[0157] 5.3.6. 虚拟指令-程序控制

[0158] 图8G是列举实例虚拟ISA中提供的程序控制操作的表格860。这些控制操作是所属领域的技术人员所熟悉的,且其使得编程人员可将程序执行重定向。分支(bra)将程序流重定向到目标位置<target>。在一些实施例中,通过将字母标签放置在虚拟ISA代码中的目标指令之前并使用所述标签作为分支指令的目标识别符<target>未定义分支目标。举例来说,在一个实施例中:

[0159] label:add.int32 d, vr1,vr2;

[0160] 将add指令识别为具有标签label的分支目标。在代码中的其它位置的指令:

[0161] bra label;

[0162] 将执行重定向到带有标签的指令。

[0163] Call与返回(ret)指令支持函数和子例行程序调用;fname识别函数或子例行程序。(在一个实施例中,"子例行程序"就是其返回值被忽略的函数。)可使用func指示(directive)来宣称函数fname,且也提供定义所述函数的虚拟ISA代码。波形括号{}或其它分组符号可用来将定义函数或子例行程序的代码与其它虚拟ISA代码隔开。

[0164] 对于函数来说,可指定参数列表<rv>来识别应当将返回值存储在何处。对于函数和子例行程序来说,在自变量列表<args>中指定输入自变量。当执行call时,存储下一指令的地址;当执行ret时,进入到所存储地址的分支。

[0165] Exit指令终止遭遇其的CTA线程。陷阱指令调用处理器定义的或用户定义的陷阱例行程序。断点(brkpt)指令使执行暂停,且可用于(例如)调试用途。无操作(nop)是在执行时没有影响的指令。例如,其可用来控制可在多久以后执行下一操作。

[0166] 5.3.7. 虚拟指令-并行线程

[0167] 图8H是列举在根据本发明实施例的实例虚拟ISA中提供的明确并行的虚拟指令的表格870。这些指令支持CTA执行所需要的协作线程行为,例如在CTA线程之间交换数据。

[0168] 屏障(bar)指令指示:到达其的CTA线程应当在执行任何进一步的指令之前等待,直到其它所有CTA线程(相同CTA中的)也已经到达同一屏障指令时为止。可在CTA程序中使用任何数目的屏障指令。在一个实施例中,屏障指令不需要任何参数(不论使用多少个屏障),因为必须在所有CTA线程均到达第n个屏障之后,任何线程才可前进到第(n+1)个屏障,依此类推。

[0169] 在其它实施例中,可例如通过指定应当在特定屏障处等待的CTA线程(或特定CTA线程的识别符)的数目来使屏障指令参数化。

[0170] 另外其它实施例提供“等待”和“不等待”两种屏障指令。在等待屏障指令下，CTA 线程一直等到其它相关 CTA 线程也已经到达屏障为止；在不等待指令处，CTA 线程指示其已到达，但可在其它 CTA 线程到达之前继续。在给定屏障处，一些 CTA 线程可能等待而其它 CTA 线程不等待。

[0171] 在一些实施例中，bar 虚拟指令可用来同步 CTA 线程，所述 CTA 线程正在使用共享存储器状态空间进行协作或共享数据。举例来说，假设一组 CTA 线程（其可包含 CTA 的一些或全部线程）各自在每线程变量（例如，.fp32 虚拟寄存器变量 myData）中产生一些数据，然后读取集合中的另一 CTA 线程产生的数据。指令序列：

[0172] st.shared.fp32 myWriteAddress，myData；

[0173] bar；

[0174] ld. shared.fp32 myData，myReadAddress；

[0175] 提供所需的行为，其中 myWriteAddress 和 myReadAddress 是对应于 .shared 状态空间中的地址的每线程变量。在每个 CTA 线程将其产生的数据写入到共享存储器之后，其一直等到所有 CTA 线程已经存储其数据为止，然后继续从共享存储器中读取数据（其可能已经由不同的 CTA 线程写入）。

[0176] 存储器屏障 (membar) 指令指示每个 CTA 线程应当等待其先前请求的存储器操作（或至少所有写入操作）完成。这个指令保证在 membar 指令之后发生的存储器存取将看到在其之前的任何写入操作的结果。Membar 指令在一个实施例中使用可选的状态空间名称 <space> 以将其范围限制于瞄准指定状态空间的存储器操作，所述指定状态空间应当是存储器状态空间（例如，不是 .reg 或 .sreg 状态空间）。如果未指定任何状态空间名称，那么 CTA 线程等待瞄准所有存储器状态空间的所有待决操作完成。

[0177] 原子更新 (atom) 指令致使对由引用 <ref> 识别的共享变量 a 的原子更新（读取 - 修改 - 写入）。所述共享变量 a 可处于任何共享状态空间中，且与其它存储器引用一样，可使用各种寻址模式。举例来说，<ref> 可为以下中的任一者：命名的可寻址变量 a、对保存变量 a 的地址的寄存器的引用、对保存将添加到偏移值（提供为立即操作数）以定位变量 a 的地址的寄存器的引用，或变量 a 的立即绝对地址。CTA 线程将变量 a 从共享状态空间位置加载到目的地寄存器 d 中，接着使用对运算数 a 和（取决于操作）第二和第三运算数 b、c 执行的指定操作 <op> 来更新变量 a，并将结果存储回由 <ref> 识别的位置。目的地寄存器 d 保持 a 的原先加载的值。原子地执行加载、更新和存储操作，保证没有其它任何 CTA 线程在第一 CTA 线程执行原子更新时存取变量 a。在一个实施例中，变量 a 限于 .global 或 .shared 状态空间，且可用与上述加载和存储操作一样的方式来指定。

[0178] 在一些实施例中，只有特定的操作可作为原子更新来执行。举例来说，在一个实施例中，如果 a 是浮点类型那么只可指定以下操作 <op>：将 a 与 b 相加；用 a 和 b 的最小值或最大值来替换 a；以及三元比较与交换操作，其在 a 等于 b 的情况下用 c 替换 a，且否则使 a 保持不变。对于整数 a，可支持额外操作，例如运算数 a 与 b 之间的按位的 and、or 和 xor，以及使运算数 a 递增或递减。也可支持其它原子操作或操作组合。

[0179] Vote 指令在预定义组的 CTA 线程间对布尔（例如，.b1 类型）运算数 a 执行归约运算 <op>。在一个实施例中，虚拟结构指定 CTA 线程在 SIMD 群组中执行，且预定义的群组对应于 SIMD 群组；在其它实施例中，其它群组的 CTA 线程可由虚拟结构或编程人员来定义。

归约运算<op>要求基于在群组中的CTA线程间对运算数a的归约以及.<op>限定 符指定的归约运算将结果值d设置成布尔真或布尔假。在一个实施例中，所允许的归约运算是：(1).all，其中如果a对于群组中的所有CTA线程为真则d为真，且否则为假；(2).any，其中如果a对于群组中的任何CTA线程为真则d为真；以及(3).uni，其中如果a对于群组中的所有活动的CTA线程具有相同值（真或假）则d为真。

[0180] 5.3.8. 虚拟指令－断言执行

[0181] 在一些实施例中，虚拟ISA支持任何指令的断言执行。在断言执行中，将布尔“保护断言(guard predicate)”值与指令相关联，且指令只有当在执行的时候保护断言评估为真时才执行。

[0182] 在实例虚拟ISA中，保护断言可为任何1位布尔虚拟寄存器变量（本文中表示为P）。通过将断言保护@P或不断言保护@！P放置在指令的操作码前面来指示断言执行。例如通过将P识别为产生了布尔结果的指令（例如，表820(图8C)中的setp指令）的目的地寄存器来在断言寄存器中建立一个值。在遇到@P或@！P保护断言时，虚拟处理器读取P寄存器。对于@P保护，如果P为真，则执行指令，否则将其跳过；对于@！P保护，如果P为假则执行指令，否则跳过。在遇到断言指令的每个CTA线程的执行时间评估断言P；因此，一些CTA线程可能执行断言指令而其它CTA线程不执行。

[0183] 在一些实施例中，可将断言设置为指令执行。举例来说，表800-870(图8A-8H)中的特定虚拟指令可能接受指定断言寄存器作为输出的参数；此种指令基于指令结果的某种性质来更新指定的断言寄存器。举例来说，断言寄存器可用来指示算术运算的结果是不是特殊数字（例如，零、无穷大或IEEE 754浮点运算中的非数字）等等。

[0184] 6. 虚拟指令翻译器。

[0185] 如参看图4所述，虚拟指令翻译器412瞄准特定的平台结构。虚拟指令翻译器412可实施为（例如）在例如图1的CPU 102等的处理器上执行的软件程序，所述虚拟指令翻译器412接收虚拟ISA代码410并将其翻译成目标ISA代码414，所述目标ISA代码414可在虚拟指令翻译器412瞄准的特定平台结构上执行（例如，通过图1的PPU 122)。虚拟指令翻译器412将在虚拟ISA代码410中宣称的虚拟变量映射到可用的存储位置上，其中包含处理器寄存器、芯片上存储器、芯片外存储器等。在一些实施例中，虚拟指令翻译器412将每个虚拟状态空间映射到特定类型的存储设备上。举例来说，可将.reg状态空间映射到线程特定的数据寄存器上，将.shared状态空间映射到处理器的可共享存储器上，将.global状态空间映射到分配给应用程序的虚拟存储器区域，等等。其它映射也是可能的。

[0186] 将虚拟ISA代码410中的虚拟指令翻译成机器指令。在一个实施例中，虚拟指令翻译器412经配置以将每个虚拟ISA指令映射成相应的机器指令或机器指令序列，这取决于在将执行CTA线程的处理器的指令集中是否存在相应的机器指令。

[0187] 虚拟指令翻译器412也将CTA线程映射到目标平台结构中的“物理”线程或处理上。举例来说，如果目标平台结构支持至少n_0个并发线程，那么每个CTA线程可映射到一个物理线程上，且虚拟指令翻译器412可为单个CTA线程产生虚拟指令代码，并预期目标平台440将为具有n_0个唯一识别符的n_0个线程执行代码。如果目标平台结构支持少于n_0个线程，那么虚拟指令翻译器412可产生虚拟ISA代码410，所述虚拟ISA代码410并入有对应于多个CTA线程的指令，并预期这个代码将对每个CTA执行一次，因而将多个CTA线程映

射到单个物理线程或处理。

[0188] 确切地说，将与数据共享（例如，存取 .shared 或 .global 状态空间的加载、存储和原子更新指令）和 / 或协作线程行为（例如，图 8H 中的屏障、原子更新和其它指令）有关的虚拟指令翻译成机器指令或机器指令序列。针对 CTA 执行而优化的目标平台结构有利地包含硬件支持的屏障指令，例如其中用指令单元中的计数器和 / 或寄存器来计数已经到达屏障指令的线程的数目，并设置旗标以防止在线程在屏障处等待时发布线程的进一步的指令。其它目标结构可能不提供对线程同步的直接硬件支持，在此情况下可使用其它线程间通信技术（例如，信号量、存储器中的状态阵列等）来创建所需的行为。

[0189] 也将断言指令翻译成机器指令。在一些实例中，目标硬件直接支持断言执行。在其它实例中，可将断言与有条件的分支指令等一起存储在（例如）处理器寄存器中，所述有条件的分支指令等用来询问寄存器并通过有条件地围绕断言指令产生分支而创建所需的运行时间行为。

[0190] 图 9 是使用根据本发明实施例的虚拟指令翻译器的过程 900 的流程图。在步骤 902 处，编程人员用高级语言编写 CTA 程序代码。在一个实施例中，所述 CTA 程序代码定义单个 CTA 线程的所需行为，且可使用线程 ID（包含 CTA ID 和 / 或栅格 ID）作为参数来定义或控制 CTA 线程的行为的各个方面。举例来说，可将要读取或写入的共享存储器位置确定为线程 ID 的函数，使得相同 CTA 中的不同 CTA 线程将对共享存储器中的不同存储器位置进行读取和 / 或写入。在一个实施例中，包含 CTA 程序代码作为应用程序代码（例如，图 4 的程序代码 402）的一部分。除了定义 CTA 线程行为之外，应用程序代码还可定义 CTA 和 / 或栅格、设置输入和输出数据集等。

[0191] 在步骤 904 处，编译器（例如，图 4 的编译器 408）根据高级语言代码产生定义单个（虚拟）CTA 线程的行为的虚拟 ISA 代码。如果代码包含 CTA 程序代码和其它代码两者，那么编译器 408 可将 CTA 程序代码与其余代码分开，以便只使用 CTA 程序代码来产生虚拟 ISA 代码。可使用常规的用于将用一种语言编写的程序代码编译成另一（虚拟）语言的技术。应注意，由于所产生的代码是使用虚拟语言，所以编译器不需要束缚于特定硬件或针对特定硬件而优化。编译器可优化根据特定的输入代码序列而产生的虚拟 ISA 代码（例如，更偏好虚拟 ISA 指令的较短序列）。可将虚拟 ISA 中的程序代码存储在盘上的存储器中和 / 或分布到各种各样的平台结构，其中包含在物理上不同于图 3 的虚拟结构 300 的结构。虚拟 ISA 中的代码是独立于机器的，并且可在任何有虚拟指令翻译器可用的目标平台上执行。在替代实施例中，编程人员可用虚拟 ISA 直接编写 CTA 程序代码，或者虚拟 ISA 代码可由程序自动产生；如果程序代码起初创建为虚拟 ISA 代码，那么可省略编译步骤 904。

[0192] 在步骤 906 处，虚拟指令翻译器（例如，图 4 的翻译器 412）读取虚拟 ISA 代码，并以目标 ISA 产生可在目标平台上执行的代码。与编译器不同的是，虚拟指令翻译器瞄准特定的（真实）平台结构且有利地经配置以调适并优化目标 ISA 代码以便在所述结构上实现最好的性能。在目标结构支持至少 n_0 个线程的一个实施例中，虚拟指令翻译器产生目标线程程序，所述目标线程程序可由 n_0 个线程中的每一者并发执行以实现 CTA。在另一实施例中，虚拟指令翻译器产生目标程序，所述目标程序使用软件技术（例如，指令序列）来仿真 n_0 个并发线程，所述线程各自执行对应于虚拟 ISA 代码的指令。翻译器可在程序安装时、在程序初始化期间或在程序执行期间在及时的基础上操作。

[0193] 在步骤 908 处，目标平台中的处理器（例如，图 1 的 PPU 122）执行目标 ISA 代码以处理数据。在一些实施例中，步骤 908 可包含将命令和状态参数提供给处理器，以便控制其行为，如下文进一步描述的。

[0194] 将明白，过程 900 是说明性的，且可进行更改和修改。描述为循序的步骤可并行执行，步骤次序可更改，且步骤可被修改或组合。举例来说，在一些实施例中，编程人员可直接使用虚拟 ISA 来编写 CTA 程序代码，从而无需产生虚拟 ISA 代码的编译器。在其它实施例中，将 CTA 程序代码编写为较大的应用程序的一部分，且所述 CTA 程序代码还包含（例如）定义要执行以解决特定问题的 CTA 和 / 或 CTA 栅格的维度的代码。在一个实施例中，只将代码中那些阐述 CTA 程序的部分编译到虚拟 ISA 代码中；可将其它部分编译到其它（真实或虚拟）指令集中。

[0195] 在其它实施例中，一个虚拟指令翻译器可经配置以产生适合于不同目标平台的目标 代码的多个版本。举例来说，翻译器可产生高级语言（例如，C）的程序代码、用于 PPU 的机器代码和 / 或用于使用软件技术来仿真 PPU 行为的单核或多核 CPU 的机器代码。

[0196] 7. 虚拟执行驱动器

[0197] 在一些实施例中，使用虚拟 ISA 代码 410 和虚拟指令翻译器 412 来产生要对 CTA 的每个线程执行的 CTA 程序代码。就图 2A-2B 的编程模型来说，指定 CTA 程序会为每个 CTA 线程 204 定义处理任务。为了完成所述模型，也有必要定义 CTA 202 的维度、栅格中的 CTA 的数目、要处理的输入数据集等。此种信息在本文中称为“CTA 控制信息”。

[0198] 如图 4 所示，在一些实施例中，应用程序 402 通过使用对虚拟库 404 中的函数的调用来指定 CTA 控制信息。在一个实施例中，虚拟库 404 包含各种函数调用，编程人员可经由所述函数调用来定义 CTA 或 CTA 栅格并指示何时应当开始执行。

[0199] 图 10 是列举实例虚拟库 404 中可用的函数的表格 1000。第一群组的函数涉及定义 CTA。具体来说，initCTA 函数是第一个要调用以创建新 CTA 的函数。这个函数使得编程人员可定义 CTA 的维度 (ntid.x, ntid.y, ntid.z)，并向新 CTA 分派识别符 cname。SetCTAProgram 函数指定要由 CTA cname 的每个线程执行的 CTA 程序；参数 pname 是对应于所需的 CTA 程序（例如，用虚拟 ISA 代码编写的程序）的逻辑程序识别符。SetCTAInputArray 函数使得编程人员可指定全局存储器中 CTA cname 将从中读取输入数据的源位置（开始地址和大小），且 setCTAOutputArray 函数使得编程人员可指定全局存储器中 CTA cname 将向其写入输出数据的目标位置（开始地址和大小）。使用 setCTAParams 函数来设置 CTA cname 的运行时间常量参数。编程人员向函数提供参数列表，例如作为（名称，值）对。

[0200] 在一个实施例中，setCTAParams 函数也可由编译器 408 在产生虚拟 ISA 代码 410 时使用。由于 setCTAParams 函数定义 CTA 的运行时间参数，所以编译器 408 可将这个函数解译为将每个参数定义为 .param 状态空间中的虚拟变量。

[0201] 表格 1000 还列举与定义 CTA 的栅格有关的函数。InitGrid 函数是第一个调用来创建新栅格的函数。这个函数使得编程人员可定义栅格的维度 (nctaid.x, nctaid.y, nctaid.z)，识别将在栅格上执行的 CTA cname，并将识别符 gname 分派给新定义的栅格。setGridInputArray 和 setGridOutputArray 函数与 CTA 级函数相似，允许针对栅格中的所有 CTA 的所有线程定义单个输入和 / 或输出阵列。setGridParams 函数用来为栅格 gname

中的所有 CTA 设置运行时间常量参数。编译器 408 可将这个函数解译为将每个参数定义为 .const 状态空间中的虚拟变量。

[0202] launchCTA 和 launchGrid 函数指示应开始指定 CTA cname 或栅格 gname 的执行。

[0203] 虚拟 API 也可包含其它函数。举例来说，一些实施例提供可用来协调多个 CTA 的执行的同步函数。举例来说，如果第一 CTA（或栅格）的输出将用作第二 CTA（或栅格）的输入，那么 API 可包含一个函数（或启动函数的参数），可经由所述函数来指示虚拟执行驱动器：在第一 CTA（或栅格）的执行完成之前不应启动第二 CTA（或栅格）。

[0204] 根据本发明的实施例，表格 1000 中的任何或所有函数调用均可包含在应用程序中，所述应用程序也定义要执行的 CTA 程序（或者如果在应用程序中存在多个 CTA 则要定义多个 CTA 程序）。在编译时，将函数调用视为对应用程序接口（API）库 404 的调用，因而产生虚拟 API 代码 406。

[0205] 使用实施虚拟库中的每个函数的虚拟执行驱动器 418 来实现虚拟 API 代码。在一个实施例中，虚拟执行驱动器 418 是在图 1 的 CPU 102 上执行的驱动器程序，所述驱动器程序控制实现 CTA 线程的 PPU 122。实施图 10 的表格 1000 中的各种函数调用，使得其导致驱动器经由推送缓冲器（pushbuffer）向 PPU 122 提供命令。在另一实施例中，CPU 执行一个或一个以上程序以实现 CTA，且虚拟执行驱动器 418 设置参数并控制 CPU 对此种程序的执行。

[0206] 将了解，本文中描述的虚拟 API 是说明性的，且变更和修改是可能的。可支持其它函数或函数组合。此项技术中已知的虚拟 API 技术可适合于本发明的用途。

[0207] 进一步的实施例

[0208] 虽然已经参照特定实施例描述了本发明，但所属领域的技术人员将认识到许多修改是可能的。举例来说，不需要本文中描述的特定虚拟结构、虚拟指令和虚拟 API 函数；可用其它支持并发、协作线程的虚拟结构、指令和 / 或函数来代替。此外，上述实施例可引用所有区块具有相同数目的元件、所有 CTA 具有相同数目的线程且执行相同 CTA 程序等的情况。在一些应用程序（例如，其中使用多个依赖性栅格的情况）中，可能需要使不同栅格中的 CTA 执行不同 CTA 程序或具有不同数目和 / 或大小的栅格。

[0209] 虽然本文中引用“协作线程阵列”，但应了解，一些实施例可使用其中不支持并发线程之间的数据共享的线程阵列；在支持此种数据共享的其它实施例中，针对给定应用程序定义的线程可能或可能不在实际上共享数据。

[0210] 此外，虽然上述实施例可将线程阵列称为具有多个线程，但应了解，在“退化”的情况下，线程阵列可能只具有一个线程。因此，本发明可应用于在要在具有一个或一个以上单线程或多线程核心的 CPU 上执行的程序中提供可扩展性。通过使用本文中描述的 技术，可用可在任何数目的可用 CPU 核心上分布线程（例如，使用操作系统功能性）而无需对虚拟 ISA 代码的修改或重新编译的方式编写程序。

[0211] 本文中使用术语“虚拟”和“真实”来反映编程人员用来描述问题解决方案的概念性编程模型与可在上面最终执行程序的实际计算机系统的脱离。“虚拟”编程模型及其相关联的结构使得编程人员可对并行的处理任务采用高级视角，且应了解，可能存在或可能不存在其组件一一对应地映射到本文中描述的虚拟结构组件的实际计算系统或装置。有利地将包含虚拟 ISA 代码和虚拟 API 代码的虚拟代码实现为用一种语言编写的可能或可能不与

任何实际处理装置的指令集一一对应的代码。与所有程序代码一样，本文中所指的虚拟代码可存储在有形的媒体（例如，存储器或盘）中、通过网络传输等。

[0212] 并入有本发明的各种特征的计算机程序——包含但不限于虚拟 ISA 和 / 或虚拟 API 代码、虚拟指令翻译器、虚拟驱动器、编译器、虚拟函数库等——可编码在各种计算机可读媒体上以供存储和 / 或传输；合适的媒体包含磁盘或磁带、光学存储媒体，例如光盘 (CD) 或 DVD（多功能数字光盘）、闪速存储器等。此种程序也可使用适合经由符合各种协议的有线、光学和 / 或无线网络（包含因特网）传输的载波信号。用程序代码编码的计算机可读存储媒体可用兼容装置封装，或者程序代码可与其它装置独立地提供（例如，经由因特网下载）。

[0213] 此外，本文中可将特定动作描述为由“编程人员”作出。预期编程人员可以是人类、通过很少或没有人为介入来产生程序代码的自动过程、或人类交互与自动或半自动处理的用以产生程序代码的任何组合。

[0214] 此外，虽然本文中描述的实施例可能引用了特定目标平台的特征，但本发明不限于这些平台。实际上，虚拟结构可用硬件和 / 或软件组件的任何组合来实现。所属领域的技术人员将明白，可预期相同虚拟结构的不同实现在效率和 / 或输出量方面不同；然而，此种差异与本发明无关。

[0215] 因此，虽然已经参考特定实施例描述了本发明，但应明白，本发明并不意图涵盖属于随附权利要求书范围内的所有修改和等效物。

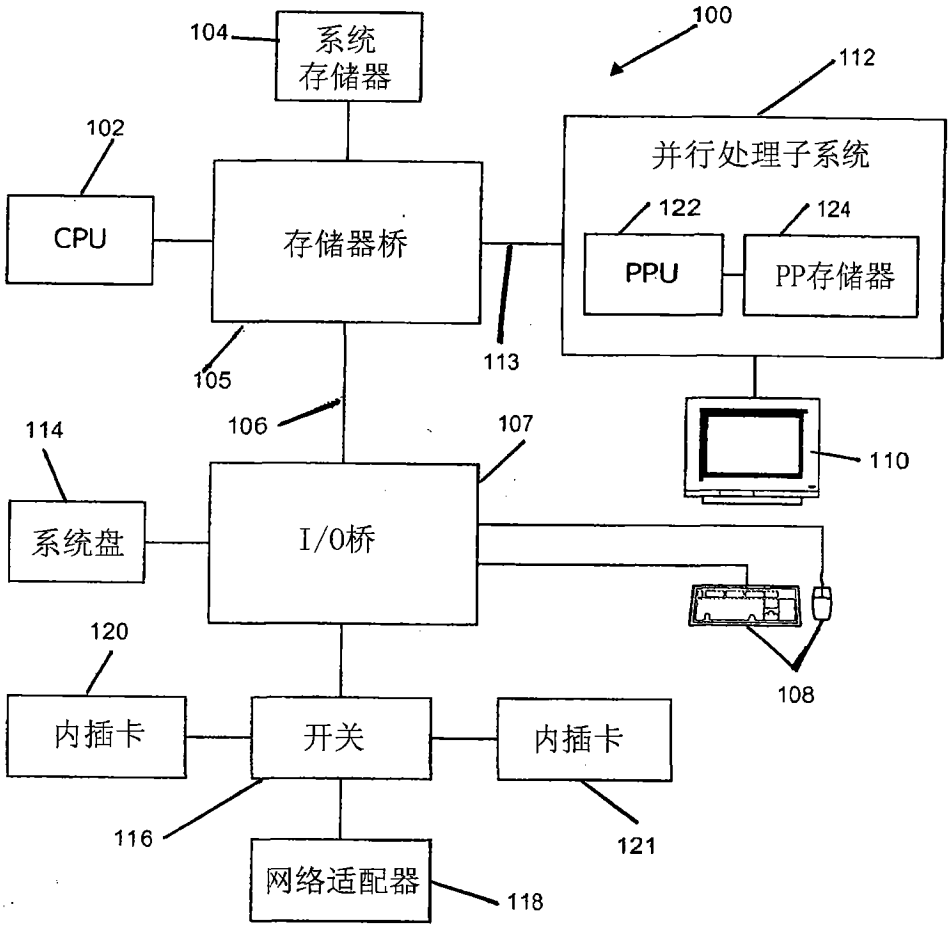

104
系统
存储器
100
112
102
并行处理子系统
CPU
存储器桥
122
124
PPU
PP存储器
105
113
106
107
110
114
系统盘
I/O桥
108
120
内插卡
开关
内插卡
116
121
网络适配器
118

图1

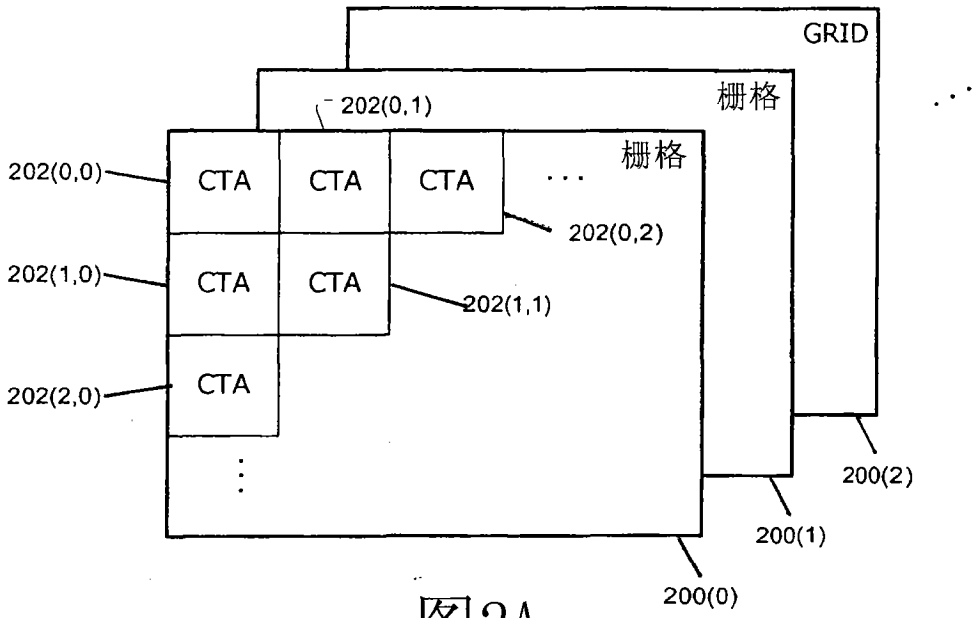

GRID
栅格
栅格
202(0,1)
202(0,0)
CTA
CTA
CTA
202(0,2)
202(1,0)
CTA
CTA
202(1,1)
202(2,0)
CTA
200(2)
200(1)
200(0)

图2A

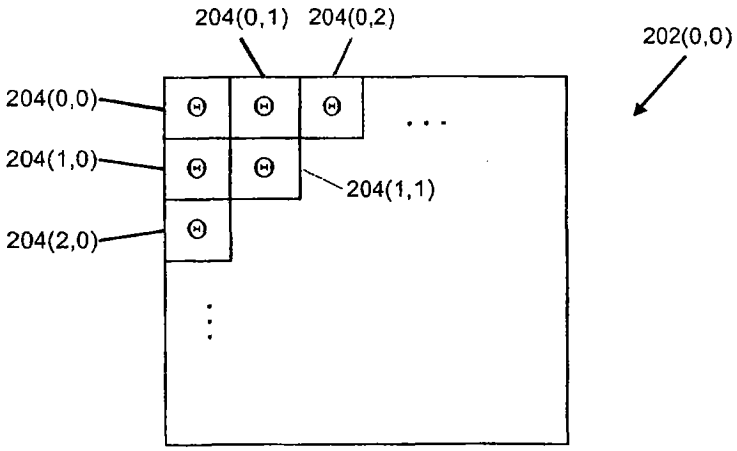

204(0,1)
204(0,2)
202(0,0)
204(0,0)
204(1,0)
204(1,1)
204(2,0)

图2B

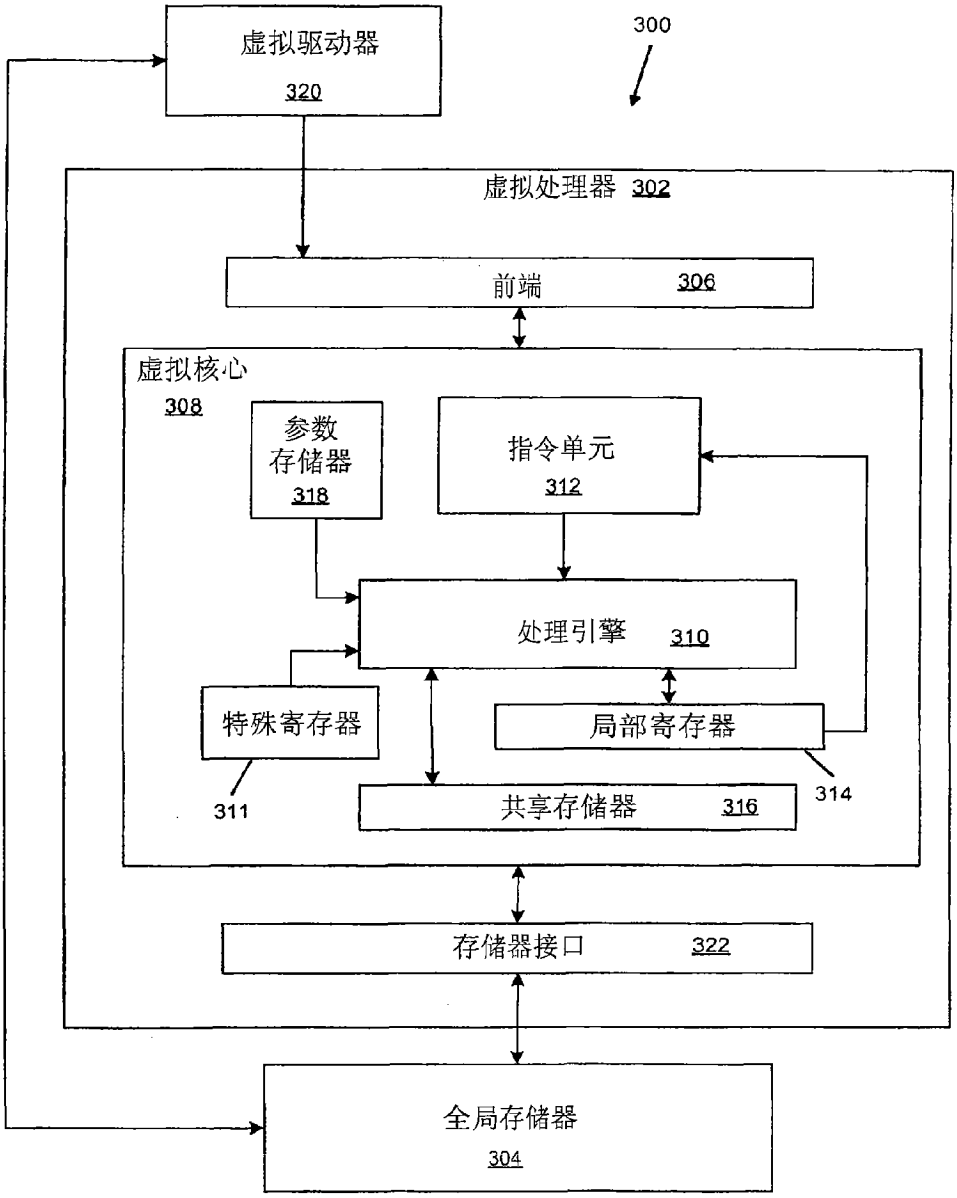

虚拟驱动器
320
300
虚拟处理器 302
前端 306
虚拟核心
308
参数
存储器
318
指令单元
312
处理引擎 310
特殊寄存器
311
局部寄存器
314
共享存储器 316
存储器接口 322
全局存储器
304

图3

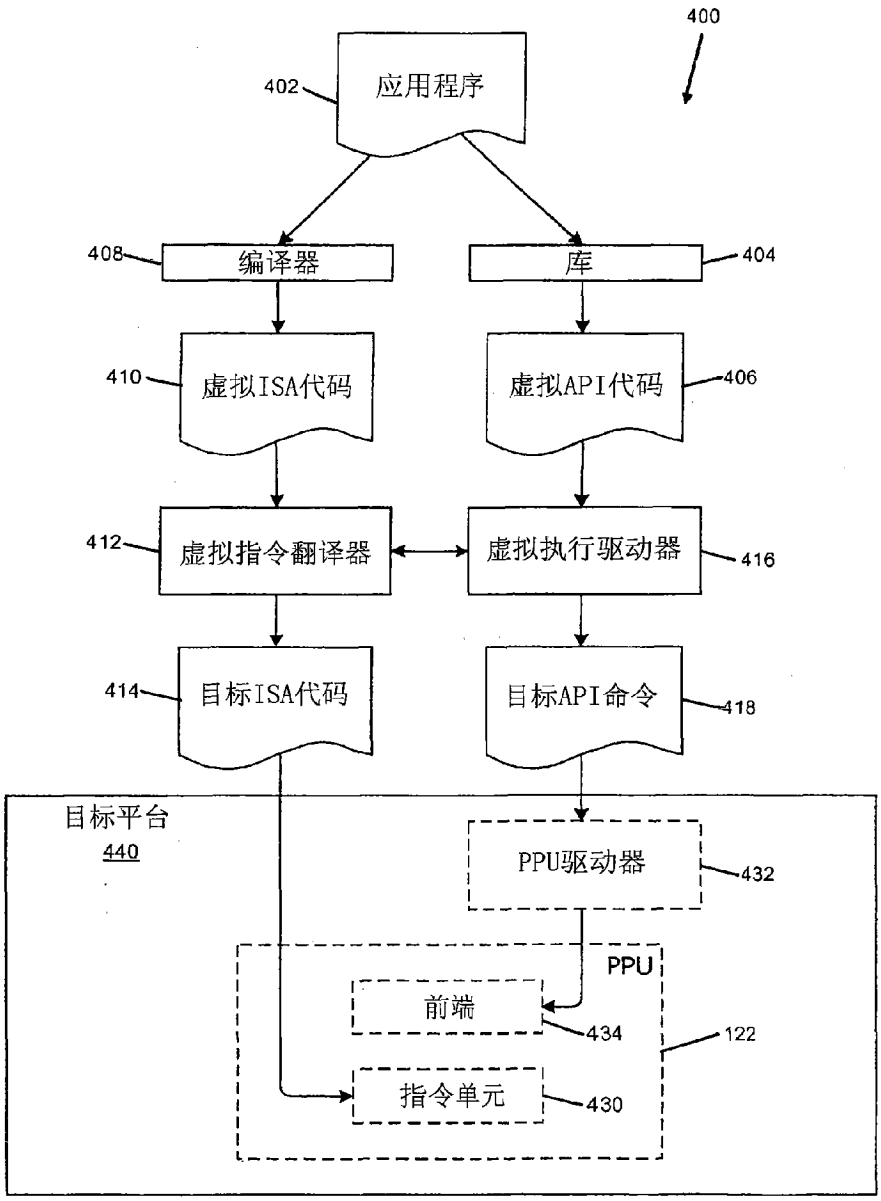

400
402
应用程序
408
编译器
404
库
410
虚拟ISA代码
406
虚拟API代码
412
虚拟指令翻译器
416
虚拟执行驱动器
414
目标ISA代码
418
目标API命令
目标平台
440
PPU驱动器
432
PPU
前端
434
122
指令单元
430

图4

500

名称	说明
%ntid.x, %ntid.y, %ntid.z	CTA维度
%tid.x, %tid.y, %tid.z	CTA内的线程ID
%nctaid.x, %nctaid.y, %nctaid.z	栅格维度
%ctaid.x, %ctaid.y, %ctaid.z	栅格内的CTA ID
%gridid	栅格ID

图5

600

类型	说明	允许的<n>
.b<n>	未指定类型变量	1, 8, 16, 32, 64
.s<n>	带符号的整数	8, 16, 32, 64
.u<n>	无符号整数	8, 16, 32, 64
.f<n>	浮点数	16, 32, 64

图6

700

名称	共享	说明	映射到
.reg	线程	运算数寄存器	局部寄存器
.sreg	线程	特殊寄存器（见图5）	局部寄存器
.local	线程	每线程不共享存储器	全局存储器
.shared	CTA	共享存储器、读取/写入存取	共享存储器
.param	CTA	可进行只读存取的共享值	参数或共享存储器
.const	栅格	可进行只读存取的共享值	参数存储器
.global	上下文	在CTA间共享的值	全局存储器
.tex	上下文	在CTA间共享的纹理	全局存储器
.surf	上下文	在CTA间共享的表面	全局存储器

图7

800

指令	效果
add.<type> d, a, b	d = a+b
sub.<type> d, a, b	d = a-b
mul.<type> d, a, b	d = a*b
div.<type> d, a, b	d = a/b
mad.<type> d, a, b, c	d = [a*b] + c
fma.<type> d, a, b, c	d = a*b + c
sad.<type> d, a, b, c	d = |a-b| + c
rem.<type> d, a, b	d = a mod b
abs.<type> d, a	d = |a|
neg.<type> d, a	d = -a
min.<type> d, a, b	d = min(a,b)
max.<type> d, a, b	d = max(a,b)
frc.<type> d, a	d = a - floor(a)
sin.<type> d, a	d = sin a
cos.<type> d, a	d = cos a
atan2.<type> d, a, b	$d = \tan^{-1}(a/b)$
lg2.<type> d, a	$d = \log_2 a$
ex2.<type> d, a	$d = 2^a$
rcp.<type> d, a	d = 1/a
sqrt.<type> d, a	d = sqrt(a)
rsqrt.<type> d, a	d = 1/sqrt(a)

图8A

810

指令	效果
dot.<type> d, **a**, **b**	d = sum(a[i]*b[i])
cross.<type> **d**, **a**, **b**	**d = a X b**
mag.<type> d, a	d = sqrt(sum(a[i]*a[i]))
vred.<op>.<type> d, a	d=a[0]; 对于i=1到长度{d=<op>(d,a[i])}

图8B

820

指令	效果
sel.<type> d, a, b, c	d = c ? a : b
set.<cmp>.<type> d, a, b	t = a <cmp> b; d = t ? ~0 : 0
setb.<cmp>.<bop>.<type> d, a, b, c	t = a <cmp> b; d = (t <bop> c) ? ~0 : 0
setp.<cmp>.<bop>.<type> d1 | d2, a, b, c	t = a <cmp> b; d1 = (t <bop> c) d2 = (!t <bop> c)

图8C

830

指令	效果
and.<type> d, a, b	d = a & b
or.<type> d, a, b	d = a | b
xor.<type> d, a, b	d = a ^ b
not.<type> d, a	d = ~a
cnot.<type> d, a	d = !a
shl.<type> d, a, b	d = a << b
shr.<type> d, a, b	d = a >> b

图8D

840

指令	效果
cvt.<dtype>.<atype> d, a	d = (dtype) a
cvt.<mode>.<dtype>.<atype> d, a	d=(dtype)a, 每<mode>舍入

图8E

指令	效果
mov.<type> d, a	d = a
ld.<space>.<type> d, <src>	向寄存器d中加载通过<src>识别的数据
st.<space>.<type> <dst>, a	将寄存器a中的数据存储到通过<dst>识别的位置
tex d, [t, x, y]	d=纹理(t, x, y)
suld d, [s, x, y]	d=表面(s, x, y)
sust [s, x, y], a	表面(s, x, y)=a

850

图8F

指令	效果
bra <target>	去往<target>
call <rv> fname <args>	函数/子例行程序调用
ret	从函数/子例行程序返回
exit	终止线程
trap	调用处理器陷阱例行程序
brkpt	暂停执行
nop	无操作

860

图8G

指令	效果
bar	在屏障处等待CTA的其它所有线程
Membar <space>	等待存储器写入完成
atom.<space>.<op>.<type> d, <ref>, b, c	以原子方式读取、更新和写回共享存储器变量
vote.<op> d, a	基于在线程间a是真还是假，为群组中的每个线程设定d

870

图8H

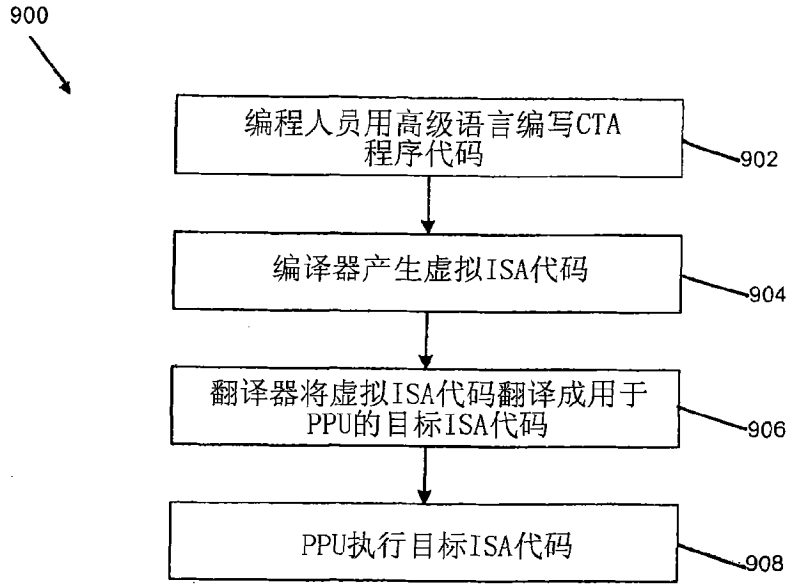


图9

1000

函数	参数
initCTA	cname, ntid.x, ntid.y, ntid.z
setCTAProgram	cname, pname
setCTAInputArray	cname, baseAddr, size
setCTAOutputArray	cname, baseAddr, size
setCTAParams	cname, np,<list>
initGrid	gname, cname, nctaid.x, nctaid.y, nctaid.z
setGridInputArray	gname, baseAddr, size
setGridOutputArray	gname, baseAddr, size
setGridParams	gname, np,<list>
launchCTA	cname
launchGrid	gname

图10