(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 103534686 B
(45)授权公告日 2017.07.11

(21)申请号 201280023687.X
(22)申请日 2012.05.16
(65)同一申请的已公布的文献号
申请公布号 CN 103534686 A
(43)申请公布日 2014.01.22
(30)优先权数据
13/108,438 2011.05.16 US
(85)PCT国际申请进入国家阶段日
2013.11.15
(86)PCT国际申请的申请数据
PCT/US2012/038057 2012.05.16
(87)PCT国际申请的公布数据
WO2012/158753 EN 2012.11.22

(73)专利权人 超威半导体公司
地址 美国加利福尼亚州
(72)发明人 毛里西奥·布莱特尼特斯
帕特里克·卡名斯基
基思·洛韦里 迪斯-清·具
(74)专利代理机构 上海胜康律师事务所 31263
代理人 李献忠

(51)Int.Cl.
G06F 9/48(2006.01)
G06F 9/50(2006.01)

审查员 孙蕾

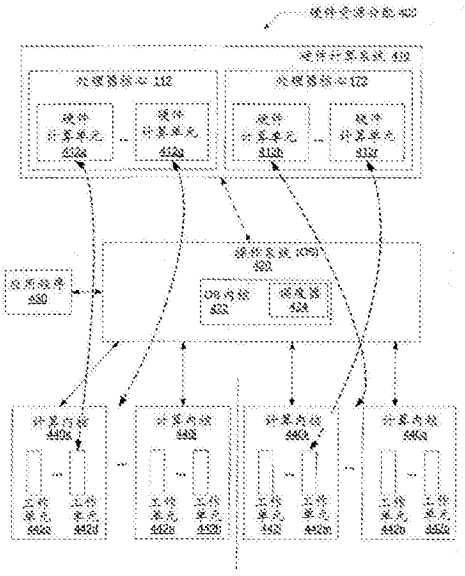
权利要求书3页 说明书12页 附图11页

(54)发明名称

异构核心的自动内核迁移

(57)摘要

一种用于在多个异构核心之间自动地迁移工作单元的执行的系统和方法。计算系统包括具有单指令多数据微架构的第一处理器核心以及具有通用微架构的第二处理器核心。编译器预测在程序中的给定位置处的函数调用的执行迁移到不同的处理器核心。编译器创建数据结构以支持移动与在给定位置处的函数调用的执行相关联的实时值。操作系统(OS)调度器至少将在程序顺序中的给定位置之前的代码调度到第一处理器核心。响应于接收迁移条件得到满足的指示，OS调度器将实时值移动至由数据结构指示的位置以便由第二处理器核心访问并且将在给定位置之后的代码调度至第二处理器核心。



CN 103534686 B

1.一种迁移计算内核的方法,包括:

在计算内核的编译期间识别在包括多条指令的所述计算内核内的位置,在所述位置处,所述计算内核的执行可以在所述计算内核的执行期间迁移;

创建数据结构以维持和迁移所述计算内核的上下文;

调度在所述计算内核内的在所述位置之前的代码以便在具有第一微架构的第一处理器核心上执行;

在所述第一处理器核心可访问的第一存储位置存储所述计算内核的所述上下文;以及

响应于接收迁移条件得到满足的指示:

将所述上下文从所述第一存储位置移动至由具有不同于所述第一微架构的第二微架构的第二处理器核心可访问的第二存储位置;以及

将在计算内核中的在所述位置之后的代码调度至所述第二处理器核心;

其中为了判定迁移条件得到满足,所述方法进一步包括判定已经到达出口点的所述计算内核的多次并行执行迭代在给定的阈值之上,

其中所述计算内核是包括一个或多个可执行的程序指令的函数。

2.如权利要求1所述的方法,进一步包括为对应于所述第一处理器核心的计算内核生成第一版本的代码,以及为对应于所述第二处理器核心的计算内核生成第二版本的代码。

3.如权利要求2所述的方法,其中,所述第一微架构是单指令多数据(SIMD)微架构,并且所述第二微架构是通用微架构。

4.如权利要求2所述的方法,进一步包括至少基于下列项中的一项来执行所述识别:配置文件运行时间信息和静态信息。

5.如权利要求2所述的方法,进一步包括:

利用指令来对所述第一处理器核心的所述第一版本的代码进行检测以判定所述迁移条件是否得到满足;以及

利用指令来对所述第二处理器核心的所述第二版本的代码进行检测以在所述第二处理器核心可访问的所述第二存储位置处找到实时值并且开始执行,其中所述第二存储位置是由所述数据结构指示的。

6.如权利要求1所述的方法,其中,在所述计算内核内的所述位置紧接在条件转移指令的前面。

7.一种包括异构多核架构的计算系统,所述计算系统包括:

具有第一微架构的第一处理器核心;

具有不同于所述第一微架构的第二微架构的第二处理器核心;

包括多条指令的计算内核,包括所述计算内核内的位置,在所述位置处,所述计算内核的执行可以在所述计算内核的执行期间迁移;

可用于维持和迁移所述计算内核的上下文的数据结构;

包括调度器的操作系统,其中,所述调度器被配置成:

调度在所述计算内核中的在所述位置之前的代码以便在具有第一微架构的第一处理器核心上执行;

在所述第一处理器核心可访问的第一存储位置存储所述计算内核的所述上下文;以及

响应于接收迁移条件得到满足的指示:

将所述上下文从所述第一存储位置移动至由具有不同于所述第一微架构的第二微架构的第二处理器核心可访问的第二存储位置;以及

将在所述计算内核中的在所述位置之后的代码调度至所述第二处理器核心;

其中为了判定迁移条件得到满足,所述第一和所述第二处理器核心中的每一个被配置成判定已经到达出口点的所述计算内核的多次并行执行迭代在给定的阈值之上,

其中所述计算内核是包括一个或多个可执行的程序指令的函数。

8.如权利要求7所述的计算系统,进一步包括编译器,所述编译器进一步被配置成为对应于所述第一处理器核心的计算内核生成第一版本的代码,以及为对应于所述第二处理器核心的计算内核生成第二版本的代码。

9.如权利要求8所述的计算系统,其中,所述第一微架构是单指令多数据(SIMD)微架构,并且所述第二微架构是通用微架构。

10.如权利要求8所述的计算系统,其中,所述编译器进一步被配置成至少基于下列项中的一项来执行识别所述计算内核内的位置:配置文件运行时间信息和静态信息。

11.如权利要求8所述的计算系统,其中,所述编译器进一步被配置成:

利用指令来对所述第一处理器核心的所述第一版本的代码进行检测以判定所述迁移条件是否得到满足;以及

利用指令来对所述第二处理器核心的所述第二版本的代码进行检测以在所述第二处理器核心可访问的所述第二存储位置处找到实时值并且开始执行,其中所述第二存储位置是由所述数据结构指示的。

12.如权利要求8所述的计算系统,其中,所述编译器进一步被配置成:

响应于预测所述计算内核的多次稍后平行执行迭代满足所述迁移条件,将所述计算内核划分成在所述计算内核内的所述位置处的两个计算子内核;

将第一计算子内核调度到所述第一处理器核心,其中,所述第一计算子内核包括在所述计算内核内的所述位置之前的代码;以及

将第二计算子内核调度到所述第二处理器核心,其中,所述第二计算子内核包括在所述计算内核内的所述位置之后的代码。

13.如权利要求7所述的计算系统,其中,所述计算内核内的位置紧接在条件转移指令的前面。

14.一种迁移计算内核的系统,包括:

第一装置,用于在计算内核的编译期间识别在包括多条指令的所述计算内核内的位置,在所述位置处,所述计算内核的执行可以在所述计算内核的执行期间迁移;

第二装置,用于创建数据结构以维持和迁移所述计算内核的上下文;

第三装置,用于调度在所述计算内核中的在所述计算内核内的所述位置之前的代码以便在具有第一微架构的第一处理器核心上执行;

第四装置,用于将所述计算内核的所述上下文存储在所述第一处理器核心可访问的第一存储位置;以及

第五装置,用于响应于接收迁移条件得到满足的指示:

将所述上下文从所述第一存储位置移动至由具有不同于所述第一微架构的第二微架构的第二处理器核心可访问的第二存储位置;以及

将在计算内核中的在所述位置之后的代码调度至所述第二处理器核心；

其中为了判定迁移条件得到满足，所述系统进一步包括第六装置，用于判定已经到达出口点的所述计算内核的多次并行执行迭代在给定的阈值之上，

其中所述计算内核是包括一个或多个可执行的程序指令的函数。

15.如权利要求14所述的系统，其中，所述系统进一步包括第七装置，用于为对应于所述第一处理器核心的计算内核生成第一版本的代码，以及为对应于所述第二处理器核心的计算内核生成第二版本的代码。

16.如权利要求15中所述的系统，其中，所述系统进一步包括：

第八装置，用于利用指令来对所述计算内核内的所述位置处的第一处理器核心的所述第一版本的代码进行检测以判定所述迁移条件是否得到满足；以及

第九装置，用于利用指令来对所述计算内核内的所述位置处的第二处理器核心的所述第二版本的代码进行检测以在所述第二处理器核心可访问的所述第二存储位置处找到实时值并且开始执行，其中所述第二存储位置是由所述数据结构指示的。

17.一种迁移计算内核的方法，包括：

在计算内核的编译期间识别在包括多条指令的所述计算内核内的位置，在所述位置处，所述计算内核的执行可以在所述计算内核的执行期间迁移；

创建数据结构以维持和迁移所述计算内核的上下文；

响应于预测所述计算内核的多次稍后平行执行迭代满足迁移条件，将所述计算内核划分成在所述位置处的两个计算子内核；

将第一计算子内核调度到第一处理器核心，其中，所述第一计算子内核包括在所述位置之前的代码；以及

将第二计算子内核调度到第二处理器核心，其中，所述第二计算子内核包括在所述位置之后的代码，

其中所述计算内核是包括一个或多个可执行的程序指令的函数。

18.一种包括异构多核架构的计算系统，所述计算系统包括：

具有第一微架构的第一处理器核心；

具有不同于所述第一微架构的第二微架构的第二处理器核心；

包括多条指令的计算内核，包括所述计算内核内的位置，在所述位置处，所述计算内核的执行可以在所述计算内核的执行期间迁移；

可用于维持和迁移所述计算内核的上下文的数据结构；

编译器，被配置成响应于预测所述计算内核的多次稍后平行执行迭代满足迁移条件，将所述计算内核划分成在所述位置处的两个计算子内核；

包括调度器的操作系统，其中，所述调度器被配置成：

将第一计算子内核调度到所述第一处理器核心，其中，所述第一计算子内核包括在所述位置之前的代码；以及

将第二计算子内核调度到所述第二处理器核心，其中，所述第二计算子内核包括在所述位置之后的代码，

其中所述计算内核是包括一个或多个可执行的程序指令的函数。

异构核心的自动内核迁移

技术领域

[0001] 本发明涉及计算系统，并且更具体地，涉及在多个异构核心之间自动地迁移工作单元的执行。

背景技术

[0002] 任务的并行化用来增加计算机系统的吞吐量。为此，编译器可以从程序代码提取并行化的任务以在系统硬件上并行地执行。在单核心架构的情况下，单核心可以包括被配置成执行多个线程的深管线。为了进一步增加硬件上的并行执行，多核架构可以包括多个通用核心。该类型的架构可以被称为同构多核架构。该类型的架构可以提供比单核心架构更高的指令吞吐量。

[0003] 一些软件应用程序不可以被频繁地划分成并行任务。此外，特定任务可能未在通用核心上有效地执行。计算密集任务的特定指令可能造成共享资源的不成比例的共享，这使得共享资源的重新分配延迟。这样的特定任务的实例可以包括加密、视频图形渲染以及碎片收集。

[0004] 为了克服常规通用核心的性能限制，计算机系统可以将特定任务卸载到专用硬件。该硬件可以包括单指令多数据(SIMD)并行架构、现场可编程门阵列(FPGA)、以及其它专门的核心。具有不同类型的核心的架构的类型可以被称为异构多核架构。取决于任务的调度，该类型的架构可以提供比同构多核架构更高的指令吞吐量。

[0005] 在许多情况下，特定的软件应用程序具有各工作项的执行或并行函数调用在本身内是数据依赖的数据并行度。例如，第一工作项与第二工作项可以是数据独立的，并且第一和第二工作项中的每一个在具有SIMD微架构的核心内在独立的路径上被调度。然而，在第一和第二工作项中的每一个内执行的指令量可能是数据依赖的。依赖于各工作项的数据，实施为分支指令的条件测试对于第一工作项而言可能通过，但是对于第二工作项而言可能未通过。

[0006] 在第二工作项停止执行并且等待且第一工作项继续进行其不间断执行时，并行执行的效率可能下降。当由于已通过的测试仅少许工作项继续执行而由于未通过的测试工作的项目的大部分空闲时，低效率增加。在由异构多核架构中的OS调度器执行的工作项的有效的功能匹配任务之后，系统性能由于特定软件应用程序的数据依赖行为可能仍然下降。

发明内容

[0007] 构想了用于在多个异构核心之间自动地迁移工作单元的执行的系统和方法。

[0008] 在一个实施方案中，计算系统包括具有第一微架构的第一处理器核心以及具有不同于第一微架构的第二微架构的第二处理器核心。在一个实施方案中，第一微架构是单指令多数据(SIMD)微架构，并且第二微架构是通用微架构。计算系统包括耦接到第一和第二处理器核心中的每一个的存储器。存储器存储包括一个或多个计算内核或函数调用的计算机程序。由于编译器横跨给定的函数调用的指令，所以编译器被配置成预测在给定位置处

的函数调用迁移到不同的处理器核心的执行。编译器创建数据结构以支持移动与在给定位置处的函数调用的执行相关联的实时值。这样的实时值可以被称为“上下文”。

[0009] 操作系统(OS)内的调度器至少将在程序顺序中的给定位置之前的代码调度到第一处理器核心。响应于接收迁移条件得到满足的指示，OS调度器将实时值移动至由数据结构指示的位置以便由第二处理器核心访问并且将在程序顺序中的给定位置之后的代码调度至第二处理器核心。为了判定迁移条件是否得到满足，第一和第二处理器核心中的每一个被配置成判定已经到达出口点的函数调用的多次并行执行迭代是否在给定的阈值之上。

[0010] 根据对下列描述和附图的参照进一步解释这些和其它实施方案。

附图说明

[0011] 图1是具有异构多核架构的示例性处理节点的一个实施方案的一般化框图。

[0012] 图2是利用计算内核的源代码的一个实施方案的一般化框图。

[0013] 图3是用条件语句限定计算内核的源代码的一个实施方案的一般化框图。

[0014] 图4是在硬件资源与计算内核之间的调度分配的一个实施方案的一般化框图。

[0015] 图5是对于两个类型的处理器核心的微架构的逻辑布局的一个实施方案的一般化框图。

[0016] 图6是通用管线执行流程的一个实施方案的一般化框图。

[0017] 图7A是SIMD管线执行流程的一个实施方案的一般化框图。

[0018] 图7B是SIMD管线执行流程的一个实施方案的另一个一般化框图。

[0019] 图8是具有迁移标记分支的程序代码的一个实施方案的一般化框图。

[0020] 图9是示出用于检测计算内核迁移的代码的方法的一个实施方案的一般化流程图。

[0021] 图10是示出用于在程序执行期间迁移计算内核的方法的一个实施方案的一般化流程图。

[0022] 虽然本发明易受各种修改和替代形成的影响，但是在附图中以举例方式示出特定实施方案并且在本文中详细地描述了特定实施方案。然而，应理解，附图及其详细描述不旨在将本发明限制于所公开的特定的形式，而是相反，本发明覆盖落入如由所附权利要求限定的本发明的精神和范围内的所有修改、等同物和替代方案。

具体实施方式

[0023] 在下列描述中，阐明许多特殊细节以提供对本发明的彻底理解。然而，本领域的普通技术人员之一应认识到，也可在没有这些特殊细节的情况下实践本发明。在某些情况下，为避免对本发明造成模糊，未详细地示出熟知的电路、结构、和技术。

[0024] 参照图1，示出具有异构多核架构的示例性处理节点110的一个实施方案。处理节点110可以包括一个或多个处理单元115，所述一个或多个处理单元115可以包括一个或多个处理器核心112以及相关联的高速缓存存储器子系统114。在一个实施方案中，处理器核心112利用通用微架构。

[0025] 处理节点110还可以包括一个或多个处理单元170，所述一个或多个处理单元170可以包括一个或多个处理器核心172以及数据存储缓冲器174。处理器核心172可能不是处

理器核心112的镜像硅图像。处理器核心172可以具有不同于由处理器核心112使用的微架构的微架构。在一个实施方案中，处理器核心172可以是与处理器核心112的相同处理器家族的不同代。在另一个实施方案中，处理器核心172可以是电压和/或频率标度版本的处理器核心112。换句话说，处理器核心172不是具有相同功能与指令集架构(ISA)、相同时钟频率、相同高速缓存大小、相同存储器模型诸如此类的处理器核心112的硅拷贝。

[0026] 继续处理器核心172的微架构，在又一个实施方案中，处理器核心172可以包括为计算密集任务提供高指令吞吐量的微架构。处理器核心172可以具有并行架构。例如，处理器核心172可以是单指令多数据(SIMD)核心。SIMD核心的实例包括图形处理单元(GPU)、数字信号处理(DSP)核心或其它。在一个实施方案中，处理节点110包括单指令集架构(ISA)。通常，如本领域中熟知的，已经示出用以为片上多处理器(CMP)提供较高功率和吞吐量的单ISA多核架构。

[0027] 当软件应用程序的线程被有效地调度时，处理节点110上的高指令吞吐量可以以在给定的功率极限内的实测功率消耗来实现。线程可以至少部分地基于处理器核心112和172的运行时间硬件资源在处理器核心112和172中的一个上以这样的方式被调度，使得各线程具有最高的指令吞吐量。

[0028] 继续处理节点110中的部件，处理节点110可以包括存储控制器120和接口逻辑140。在一个实施方案中，处理节点110的图示的功能合并在单个集成电路上。在一个实施方案中，处理器核心112包括用于根据预先定义的通用指令集执行指令的电路。例如，可以选择SPARC®指令集架构(ISA)。替代地，可以选择x86、x86-64®、Alpha®、PowerPC®、MIPS®、PA-RISC®或任意其它指令集架构。一般地，处理器核心112分别访问数据和指令的高速缓存存储器子系统114。如果在高速缓存存储器子系统114或在共享高速缓存存储器子系统118中未找到被请求的块，则读取请求可以生成并且转移到在缺失块被映射到的节点内的存储控制器。

[0029] 在一个实施方案中，处理单元170是图形处理单元(GPU)。现代GPU在操纵以及显示计算机图形学上是非常有效的。GPU的高度并行结构使得它们比用于复杂算法的范围的诸如处理单元115的通用中心处理单元(CPU)更加有效。通常，GPU执行图形和视频所使用的计算，并且CPU为比图形本身更多的系统处理执行计算。常规GPU利用非常宽的单指令多数据(SIMD)架构以在图像渲染应用中实现高吞吐量。这样的应用一般而言必需在大量的对象(顶点或像素)上执行相同的程序，诸如顶点着色或像素着色。由于各对象独立于其它对象被处理，但是使用了相同序列的运算，所以SIMD微架构提供相当大的性能增强。GPU也已经被认为用于非图解计算。

[0030] 在一个实施方案中，GPU 170可以位于显卡上。在另一个实施方案中，GPU 170可以集成在母板上。在又一个实施方案中，处理节点110的图示的功能可以合并在单个集成电路上。在这样的实施方案中，CPU 115和GPU 170可以是来自不同的设计中心的专用核心。另外，GPU 170可能现在能够经内存控制器120从处理节点110直接访问两个本地存储器114和118和主存储器，而非经接口140执行片外内存访问。该实施方案可能使GPU 170的内存访问的延迟减少，这可能转化为较高的性能。

[0031] 继续图1中的处理节点110的部件，高速缓存子系统114和118可以包括被配置成存储数据块的高速缓存存储器。高速缓存存储器子系统114可以集成在相应的处理器核心112

内。替代地，根据需要，高速缓存存储器子系统114可以以后方高速缓存配置或内联配置耦接到处理器核心114。再者，高速缓存存储器子系统114可以实现为高速缓存的层次结构。若需要，则位于较靠近处理器核心112(在层次结构内)的高速缓存可以集成到处理器核心112中。在一个实施方案中，高速缓存存储器子系统114各自表示L2高速缓存结构，并且共享高速缓存存储器子系统118表示L3高速缓存结构。两个高速缓存存储器子系统114和共享高速缓存存储器子系统118可以包括耦接到对应的高速缓存控制器的高速缓存存储器。

[0032] 一般地，包处理逻辑116被配置成响应于在处理节点110所耦接到的链路上所接收的包，响应于处理器核心112和/或高速缓存存储器子系统114以生成控制包，响应于由内存控制器120选定的事务以生成探测器命令和响应包以便维修，以及对其节点110是通过接口逻辑140到其它节点的中间节点的包进行路由。接口逻辑140可以包括用以接收包并且使包与由包处理逻辑116使用的内部时钟同步的逻辑。

[0033] 现在转向图2，示出利用计算内核的源代码的一个实施方案。OpenCL™(开放计算语言)是异构计算的低级应用程序设计接口(API)的一个实例。OpenCL包括限定执行队列的C类语言，其中各队列与OpenCL设备相关联。OpenCL设备可以是CPU、GPU、或在异构多核架构内具有至少一个处理器核心的其它单元。函数调用可以被称为OpenCL内核，或简称为“计算内核”。OpenCL构架可以提高在游戏、娱乐、科学和医疗领域中使用的宽范围的数据并行应用的计算性能。对于异构架构，计算机程序通常包括一些计算内核和内部功能。软件程序员可以限定计算内核，而内部函数可以被限定在给定的库中。

[0034] 对于数据并行软件应用程序，N维计算域可以限定“执行域”的组织。N维计算域也可被称为N维栅格或N维范围(“NDRange”)。NDRange可以是一维、二维、或三维空间。注意，一些实施方案可以允许大于三维数据。该尺寸空间也可被称为索引空间。例如，软件应用程序可能在诸如图像文件的二维(2D)数据阵列上执行数据处理。软件应用程序可以在2D图像的逐像素基础或二维矩阵的逐元素基础上执行由软件程序员研发的算法。给定的计算内核可以在索引空间(NDRange)上被调用。在其它实施方案中，软件应用程序可以包括利用在3D晶格上映射的静电势和在大分子建模中使用的直接库仑求和的数据并行编程的算法。

[0035] 通常在编译之后，设置各计算内核的实参和形参。另外，创建关联的内存对象和缓冲器。计算内核的给定实例可以作为其自身的软件线程执行。然而，计算内核可以包括创建fork函数的控制流转移指令，而计算机程序中的fork函数通常通过普通定义创建软件线程。在索引空间中在给定的点处的计算内核的给定实例可以被称为“工作项”。工作项也可以被称为工作单元。工作单元可以利用在对应于2D图像的给定的像素(给定的索引)的数据的记录上的计算内核中的一个或多个指令来操作。通常，工作单元具有相关联的唯一标识符(ID)。在另一个实例中，处理字符串“Hello World(世界，你好)”的引导计算机程序可以具有用于计算字符串中的各字母的一个工作单元。

[0036] 如果存在充分的硬件支持，则NDRange可以限定并行执行的工作单元的总量。例如，NDRange可以限定多个280工作单元，但是GPU可以支持64个工作单元在任意给定的时间的同时执行。工作单元的总数可以限定全局工作大小。如本领域的技术人员所熟知的，工作单元还可以进一步分组成工作组。各工作组可以具有唯一标识符(ID)。在给定的工作组内的工作单元可以能够彼此通信并且使执行同步并且协调内存访问。许多工作单元可以聚集到波前中以便以SIMD方式在GPU上同时执行。关于280个总工作单元的以上实例，波前可以

包括64个工作单元。

[0037] OpenCL框架是各种计算设备或OpenCL设备的开放的编程标准。软件程序员可以避免写入供应商特定的代码，从而提高代码可移植性。其它框架是可用的并且可以为异构架构提供更多供应商特定的编码。例如，NVIDIA提供计算统一设备架构(CUDA®)，并且AMD提供ATI流®。在CUDA框架的情况下，当计算机程序被编译时，计算内核通常被静态编译。在OpenCL框架的情况下，计算内核通常以Just-In-Time(JIT)方法编译。JIT方法在获得系统硬件配置之后可以生成适当的二进制代码。在JIT编译法的情况下，编译时间以总执行时间被包括。因此，编译器优化可以增加执行时间。此外，在运行时间时，OpenCL编译器可以生成多个版本的计算内核。一个版本的计算内核可以针对诸如通用CPU、SIMD GPU诸如此类的各类型的OpenCL设备类型生成。

[0038] 两个框架OpenCL和CUDA在术语上在它们的相应的执行模型之间具有差异。例如，在OpenCL中的工作单元、工作组、波前和NDRange具有在CUDA中对应的术语，诸如线程、线程块、warp和栅格。贯穿描述的其余部分，使用了对应于OpenCL的术语。然而，所描述的系统和方法可以应用于CUDA、ATI流和其它框架。

[0039] 如图2中所示，代码210限定一般标题为“doWorkA(干工作A)”和“doWorkB(干工作B)”的两个函数调用。各函数调用可以被称为“计算内核”。计算内核可以与数据的一个或多个记录匹配以产生一个或多个工作计算单元。因此，两个或更多个工作单元可以利用单个函数调用的相同的指令，但是在不同的数据记录上工作。例如，可以使用代码220中的函数调用“Power2”来执行10个工作单元，阵列“INPUT(输入)”中的每个数据值一个。在此处，记录包括单个数据值。在其它实例中，记录可以包括两个或更多个字段，其中各字段包括数据值。SIMD微架构可以有效地执行内核“Power2”的这些指令，为INPUT阵列中的值计算2的功率并且将输出写入到RESULT阵列。

[0040] OpenCL构架可以并行调用计算内核的实例多次。对计算内核的每次调用具有可以通过调用被命名为get_global_id(0)的内部函数而获取的一个相关联的唯一的ID(工作单元ID)。关于代码220中的以上实例，为INPUT阵列中的各数据值调用计算内核“Power2”一次。在这种情况下，调用计算内核“Power2”10次。据此，获取10个唯一的工作单元ID。在JIT编译方法的情况下，在运行时间时调用这些实例。OpenCL构架可以通过利用唯一的工作单元ID来区分这些不同的实例。将被在(记录)上运算的数据也可以被指定诸如为在INPUT阵列中的特定的数据值。因此，在运行时间时，在相关联的计算内核被调度时，工作单元可以通过默认相同的OpenCL设备而被调度。

[0041] 现在转向图3，示出以条件语句限定计算内核的源代码的一个实施方案。类似于代码210，图3中所示的代码230限定一般标题为“doWorkA”和“doWorkB”的两个函数调用。另外，各函数调用可以被称为“计算内核”。在此处，两个计算内核中的仅一个在运行时间期间被执行。哪个计算内核被执行的选择是基于由函数调用“EvaluateFunction”提供的条件测试。关于对应于相关联的记录的先前指令和数据的执行，给定的指令的结果或给定的指令是否被执行是数据依赖的。如果条件测试的结果在工作单元的波前之间不是一致的，则SIMD微架构的益处可能减小。例如，给定的SIMD核心可以具有可用于64个工作单元的同时执行的64个平行的计算单元。然而，如果64个工作单元中的一半通过条件测试而另一半未通过条件测试，则并行计算单元的仅一半在给定阶段的处理期间被利用。

[0042] 现在转向图4，示出图示出在硬件资源与计算内核之间的调度分配400的一个实施方案的一般化框图。在此处，示出在一个或多个软件应用程序430的执行期间硬件资源与软件资源的分区以及它们的相互关系和分配。在一个实施方案中，操作系统420为计算内核440a-440j和440k-440q分配存储器区域。当应用程序430或计算机程序执行时，各应用程序可以包括多个计算内核。例如，第一执行应用程序可以包括计算内核440a-440j，并且第二执行应用程序可以包括计算内核440k-440q。内核440a-440q中的每一个通过与数据的一个或多个记录(未示出)合并可以被用来生成一个或多个工作单元。例如，计算内核440a可以生成工作单元442a-442d，计算内核440j可以生成工作单元442e-442h，计算内核440k可以生成工作单元442j-442m，并且计算内核440q可以生成工作单元442n-442q。一个工作单元可以独立于其它工作单元执行和与其它工作单元同时执行。

[0043] 在应用程序执行之前，图4中所示的计算内核中的每一个可以拥有其自身的资源诸如存储器的图像、或指令和数据的实例。计算内核中的每一个也可以包括进程专用信息，诸如对代码、数据、和可能地堆和堆栈寻址的地址空间；诸如堆栈指针的数据和控制寄存器、通用寄存器和浮点寄存器、程序计数器等等中的中的变量；操作系统描述符诸如stdin、stdout、等等；和安全性属性，诸如一组权限。

[0044] 在一个实施方案中，硬件计算系统410合并通用处理器核心112和SIMD处理器核心172，所述通用处理器核心112和SIMD处理器核心172各自被配置成处理一个或多个工作单元。在另一个实施方案中，系统410包括两个其它异构处理器核心。一般而言，对于给定的应用程序，操作系统420为应用程序设置地址空间，将应用程序的代码加载到存储器中，为程序设置堆栈，分支到应用程序内部的给定位置，以及开始执行应用程序。通常，操作系统420操控这样的活动的部分是操作系统(OS)内核422。OS内核422被称为“OS内核”以便不将它与计算内核或函数调用混淆。当对于应用程序的执行可用的内存不足时，OS内核422可以进一步判定行动的过程。如前所述，应用程序可以被划分成多于一个计算内核，并且系统410可以运行多于一个应用程序。因此，可能存在几个并行运行的计算内核。OS内核422可以在任何时候决定同时执行的计算内核中的哪一个被分配给处理器核心112和172。对于被称为时间片的给定的时间量，OS内核422可以允许过程在处理器的核心上运行，所述处理器可以具有一个或多个核心。操作系统420中的OS调度器424可以包括用于将计算内核分配到核心的决策逻辑。

[0045] 在一个实施方案中，仅一个计算内核能够在任何时候在硬件计算单元412a-412g和412h-412r中的任一个上执行。这些硬件计算单元包括能够应付具有相关联的数据的给定的工作单元的给定指令的执行的硬件。该硬件可以包括算术逻辑单元，该算术逻辑单元被配置成执行加法、乘法、零检测、位元移位、分割、视频图形和多媒体指令或处理器设计领域的那些技术人员已知的其它操作。这些硬件计算单元可以包括多线程处理器中的硬件线程、SIMD微架构中的并行硬件列、诸如此类。

[0046] 图4中的虚线指示分配并且不一定指示直接物理连接。因此，例如，硬件计算单元412a可以被分配以执行工作单元442d。然而，稍后(例如，在上下文切换之后)，硬件计算单元412a可以被分配以执行工作单元442h。在一个实施方案中，OS调度器424可以利用循环方案将工作单元442a-442q调度至硬件计算单元412a-412r。替代地，OS调度器424可以利用循环方案将工作单元442a-442q调度至核心112和172。给定的工作单元到给定的硬件计算单

元的分配可以由相关联的处理器核心执行。在另一个实施方案中，OS调度器424可以基于处理器核心112和172的可用性执行调度。在又一个实施方案中，OS调度器424可以根据由程序员利用OpenCLTMAPI或另一个类似的API生成的分配来执行调度。当在工作单元分配与硬件资源之间存在不匹配时，这些调度方案可以限制可移值性和性能。

[0047] 参照图5，示出图示出两个类型的处理器核心的微架构的逻辑布局的一个实施方案的一般化框图。尽管示出通用核心510和单指令多数据(SIMD)核心560中的每一个，但是其它类型的异构核心是可能的并且可设想的。核心510和560中的每一个具有用于存储数据和指令的动态随机存取存储器(DRAM)550a和550b。在一个实施方案中，核心510和560共享相同DRAM。在另一个实施方案中，除了DRAM之外，高速缓存存储器子系统(未示出)的给定的电平也被共享。例如，再次参照图1，高速缓存存储器子系统118由核心112和172共享。

[0048] 核心510和560中的每一个可以包括高速缓存存储器子系统530。如示出的，通用核心510在逻辑上具有与控制逻辑单元520和算术逻辑单元(ALU)540分离的高速缓存存储器子系统530。核心510内的数据流可以被管线化，而诸如管线寄存器的存储元件未示出以便简化图示。在给定的管线阶段中，如果在该阶段中的指令不利用某一类型的ALU或如果另一个工作(或通用核心的另一个线程)在该阶段期间消耗ALU，则可能未使用ALU。

[0049] 如示出的，SIMD核心560具有与各排的计算单元542的控制逻辑单元520分组的高速缓存存储器子系统530。核心560内的数据流可以被管线化，而诸如管线寄存器的存储元件未示出以便简化图示。在给定的管线阶段中，如果在该阶段中相关联的指令基于诸如不采取分支的先前未通过的试验而未被执行，则可能未使用计算单元。

[0050] 现在参照图6，示出图示出通用管线执行流程600的一个实施方案的一般化框图。指令602-608可以被获取并且进入通用管线。指令606可以是计算密集的指令。在管线执行流程的特定阶段期间，指令602-608中的一个或多个消耗通用处理器核心112中的资源，诸如解码器逻辑、指令调度器项、重排序缓冲项、ALU、寄存器文件项、分支预测单元、诸如此类。

[0051] 在平衡的方案中，这些指令602-608中的每一个在各阶段消耗相等量的资源。然而，通常，由于半导体不动产成本、功率消耗和其它设计考虑，通用核心不复制各指令的资源。因此，工作量可能变得不平衡。例如，对于一个或多个管阶段，指令606可能由于其计算密集行为而消耗更多资源。如示出的，由该指令消耗的资源630可能变得远大于由其它指令消耗的资源。事实上，计算密集指令可能阻碍由其它指令对硬件资源的使用。

[0052] 一些计算密集任务可能对图1中所示的通用核心112内的共享资源施加压力。因此，对于计算密集过程和等待共享资源的其它过程两者而言，出现吞吐量损失。此外，一些指令可能占据共享资源和其它资源以支持正在共享资源上执行的计算。这样的长延迟指令可能同时阻碍其它过程在长延迟期间使用一些资源。

[0053] 现在参照图7A，示出图示出SIMD管线执行流程700的一个实施方案的一般化框图。指令702-708可以被获取并且进入具有相关联的数据的SIMD管线。指令704可以是控制流转移指令，诸如条件分支。指令706可以是当条件为真时在路径中执行的第一指令。指令708可以是当条件为假时在路径中执行的第一指令。例如，分支指令704可以与高级语言程序中的IF语句相关联。指令706可以与高级语言程序中的THEN语句相关联。指令708可以与高级语言程序中的ELSE语句相关联。

[0054] 在给定的排内的计算单元中的每一个可以是相同的计算单元。这些计算单元中的每一个可以在相同的指令上运算，但是不同的数据与不同的工作单元相关联。如示出的，工作单元中的一些通过由条件转移指令704提供的试验，并且其它工作单元未通过该试验。SIMD核心172可能执行可用的路径中的每一个并且选择性地禁用对应于未选择当前路径的工作项的执行单元，诸如计算单元。在If-Then-Else(如果-则-否则)构造语句的执行期间，在SIMD架构的各列内的是被配置成执行“Then(则)”(路径A)路径和“Else(否则)”(路径B)路径的执行单元。在第一和第二工作单元暂停执行并且等待且第三工作单元继续进行其不间断执行时，并行执行的效率可能下降。因此，在分支指令704的执行之后，并不所有的计算单元是给定的排中的有源计算单元710。如示出的，一个或多个计算单元是已经被禁用执行的无源计算单元711。如果大量计算单元在给定的管阶段期间是非活动的，则SIMD核心的效率和吞吐量减小。

[0055] 在一个实施方案中，“Else(否则)”路径是计算内核的回程。计算内核的执行结束，并且对应的工作单元变得空闲。然而，SIMD核心中的邻近的工作单元可能继续执行。现在参照图7B，示出图示出SIMD管线执行流程720的另一个实施方案的一般化框图。类似于执行流程700，指令702-706可能导致一个或多个计算单元在SIMD核心的特定排中被禁用。在此处，各“Else(否则)”路径可以是计算内核的回程。因此，对于给定的工作单元，不采取方向上的分支解析可能导致给定的工作单元停止对计算内核的进一步执行。在执行流程720中，为了图示的便利性，仅一个指令被示出为在第一分支指令704与第二分支指令712之间。然而，多个指令可以在分支指令704与712之间。无论分支704与712之间指令的数量，解析在不采取方向上的第一分支704的工作单元可以完成执行。同样地，对于分支712，解析在不采取方向上的第二分支的工作单元可以完成执行。SIMD核心的稍后阶段的计算单元可以针对这些工作单元被禁用。如果大量计算单元在给定的管阶段期间是非活动的，则SIMD核心的效率和吞吐量减小。

[0056] 可能导致多个工作单元未通过试验并且停止执行而邻近的工作单元可能继续的应用程序的一个实例是脸部检测。如本领域的技术人员已知的，如在OpenCv(开放的计算机视觉库)中实施的脸部检测是Viola-Jones对象检测算法的一个应用程序。Viola-Jones算法显示数据依赖的执行模式。检索计算内核应用于可以包括一个或多个像素数据的记录。检索计算内核检索二维(2D)或三维(3D)图像的子窗口中的脸部。在计算内核内，可能存在被实施为诸如分支指令的控制流转移指令的一连串的测试。在一个典型的实例中，一连串的测试包括22个阶段或22个测试。这一连串的测试可以判定输入窗口是否包含脸部。

[0057] Viola-Jones算法中的一连串的测试可以被设计成迅速地删除没有希望的路径。因此，大多数工作单元可能判定脸部的不存在并且结束。工作单元的执行在有可能包容脸部的其余的像素上继续。在几个初始阶段测试之后，一小部分像素(即，工作单元执行)可能继续通过22个阶段，而发现大多数像素未包含脸部。即使有大任务平行度，在波前上的几个继续工作单元的存在可能导致低的SIMD核心利用率。下文描述的一个方法利用独立的异构核心，同时释放SIMD核心用于进一步的处理。当检测到小量的SIMD并行度存在时，该方法可能增强整体计算性能。

[0058] 现在转向图8，示出包括用以限定迁移点的标记分支的代码800的一个实施方案。代码800包括一般标题为“foo”的计算内核。在执行期间，代码800的一部分可以被迁移到独

立的异构核心。在示出的实例中，外侧环路是数据依赖的。在一个实施方案中，通过使用在对应于“当(while)”循环测试的分支指令中的标签位元，编译器告知数据依赖性的SIMD核心。在执行期间，当检测到迁移的条件时，诸如实测SIMD利用率是在给定的阈值下方，中间局部值可以被移动至将由独立的异构核心访问的存储器中的数据结构。例如，从标记的分支迁移点的角度，通用核心可能继续计算内核的执行。例如，在当语句中隐含的条件分支以标签“辅助入口”标记。独立的异构核心可能使用编译器生成的数据结构。在另一个实施方案中，该数据可以被缓存，减小迁移成本。在一个实例中，实时数据可以包括“tmp”阵列的局部切片以及局部温度变量的当前值两者。在迁移期间，该数据可以被传达至运行时间环境，其引导计算内核到由标签“辅助入口”指示的辅助入口点的连续执行。

[0059] 现在转向图9，示出用于通过利用预运行时间数据信息优化在处理器中的多个工作单元的并行执行的方法900的一个实施方案。在上述图4中所示的处理节点110和硬件资源分配中体现的部件一般可以根据方法900工作。出于论述的目的，在该实施方案和稍后描述的方法的随后的实施方案中的步骤以顺序次序示出。然而，在其它实施方案中，一些步骤可能以不同于示出的次序发生，一些步骤可能被同时执行，一些步骤可能与其它步骤合并，并且一些步骤可能不存在。

[0060] 在块902中，可以对软件程序或子程序进行定位和分析。该软件程序可以被写入以便在异构多核架构上编译和执行。程序代码可以指软件应用程序、子程序、动态链接库等等的任意部分。路径名可以由用户在命令提示符处输入，路径名可以从给定的目录位置读取，或者说，以便开始编译源代码。程序代码可以由设计者以诸如C语言、诸如OpenCL™的C类语言、诸如此类的高级语言写入。在一个实施方案中，源代码被静态地编译。在这样的实施方案中，在静态前端编译期间，可以将源代码翻译成中间表示(IR)。后端编译步骤可能将IR翻译成机器代码。静态后端编译可能执行更多变换和优化。在另一个实施方案中，以Just-In-Time(即时)(JIT)方法编译源代码。JIT方法在获得系统硬件配置之后可以生成适当的二进制代码。在任一种方法的情况下，编译器可能识别程序代码中的计算内核。在一个实施方案中，编译器，诸如OpenCL编译器，可以生成多个版本的计算内核。一个版本的计算内核可以针对诸如通用CPU、SIMD GPU、诸如此类的各类型的OpenCL设备类型生成。

[0061] 在块904中，编译器可能读取计算内核的一个或多个指令并且分析它们。条件语句可以是控制流转移指令，诸如分支。不同类型的控制流转移指令可以包括正向/反向分支、引导/间接的分支、跳线、诸如此类。对于编译器或其它工具而言，静态地判定分支的方向和/或分支的目标也是可能的。然而，在一个实施方案中，在运行时间期间通常在相关联的数据上执行的一些处理可以在编译期间执行。例如，可以执行用以判定分支的方向(采取、不采取)的简单的测试。虽然编译可以被称为“静态编译”，但是可以执行一个或多个小动态操作。该编译也可以被称为“预运行时间编译”。在此次执行的动态步骤的另一个实例识别下一个指令以执行If-Then-ElseIf-Else(如果-则-否则如果-否则)构造的THEN(则)、ELSE IF(否则如果)和ELSE(否则)块中的每一个。例如，如果条件分支未通过，则可以执行返回语句。因此，编译器知道，在执行期间，当分支试验未通过时，该计算机内核的对应的工作单元可能变得空闲。

[0062] 在块906中，计算内核中特定的代码行数被选定以便创建迁移点。迁移点可以是计算机内核中在飞行执行转移到不同的异构核心的情况下的位置。在一个实施方案中，该计

算子内核迁移可以通过类似于过程迁移的机制来实现，其中，执行状态从第一异构核心移动至具有与第一核心可能不同的微架构的第二异构核心。在另一个实施方案中，该计算子内核迁移可以通过创建稍后发送的的多个计算子内核来实现。

[0063] 在一个实施方案中，编译器可能自动地识别迁移点。如本文所使用的，迁移点也可被称为切换点。编译器可能使用控制流程分析。识别迁移点可以包括利用静态控制流程分析来找到导致计算内核退出或返回的数据依赖的环路。不是识别具有包括退出或返回的路径的各分支，而是编译器可能使用计数来减少多个迁移点。例如，在计算内核中找到的前五个分支可能不是标记为迁移点的候选者。在前五个分支之后的每三个分支可以是标记为迁移点的候选者。基于计数的其它滤波算法是可能的并且是可设想的。

[0064] 此外，编译器可能使用来自先前执行的配置文件输入来识别迁移点。例如，对于在给定的阈值之上的数据的多个记录，与给定的分支相关联的条件测试可能未通过。因此，该分支可以被识别为迁移点。此外，指示迁移点的程序员附注可能被添加为“pragmas”或作为OpenCL框架的扩展。

[0065] 在块908中，对于各版本的编译代码，编译器可能对代码中选定的点加标签。对于相应的OpenCL设备，各版本可以被称为目的计算内核。另外，编译器可能编译识别的计算内核以生成两个或更多个版本的编译代码，所述两个或更多个版本的编译代码各自能够在OpenCL设备中的相应的一个上运行。再次参照图9中的代码800，由标签”辅助入口”指示的辅助入口点是分支的迁移标签的实例。编译器内的代码生成器可能插入标签和插入其它代码以在迁移期间调用实时值。调用实时值可以包括将实时值转移至目的OpenCL设备并且初始化目的OpenCL设备上的值。代码生成和插入过程可以类似于在调式点处被插入的调试器代码和用于测量动态行为的插装。

[0066] 在一个实施方案中，计算内核可以被标记以识别如上文所描述的迁移点。在另一个实施方案中，计算内核可以被划分成被独立地调度并且被分派的多个计算子内核。对于由分支指令实施的条件测试，可以使用运行时间配置文件信息或编译器静态估计使用来判定通过/未通过统计。“热”执行路径可以包括在数据的多个记录的条件测试的给定的阈值之上的大量通过。“冷”执行路径可以包括在数据的多个记录的条件测试的第二给定的阈值之下的少量通过。基于“热”和“冷”执行路径，计算内核可以被划分成计算子内核。

[0067] 对应的计算子内核的生成可能利用类似的运行时间代码生成机制，除继续在通用核心上执行的诸如“冷”执行路径的那些计算子内核的对应的执行范围(NDRange)的创建之外。这可以通过创建包含可能利用OpenCL指定的计算子内核标识符(ID)的将在通用核心上执行的潜在稀疏阵列。给定的计算内核可能利用对该阵列的间接访问以识别适当的计算子内核和稍后的工作单元。替代地，编译器可以生成这些ID的列表和对于执行工作单元中的每一个而言将被调用和映射的对应的计算子内核。

[0068] 在配置文件运行或静态估计之后，对应于“热”执行路径的计算子内核可以为SIMD核心编译。对应于“冷”执行路径的计算子内核可以是为通用核心编译。一连串测试的早期阶段可以具有通过的高可能性。因此，这些执行路径可以在SIMD核心上执行的“热”计算子内核中实施。在这些特定的“热”计算子内核的执行之后，相关联的产生的数据可以在内存中移动。该数据移动使为全局数据而存活的局部数据升级。对应于“热”计算子内核的工作单元可能基于其工作单元ID写入位阵列以指示相关联的“冷”计算子内核随后是否继续在

通用核心上执行。

[0069] 在块910中，编译器识别在识别的迁移点处的一组实时值。实时值可以包括中间计算值和局部阵列。现在再次参照图8中的代码800，实时数据可以包括代码内的“tmp”阵列的局部切片以及局部温度变量的当前值两者。如果迁移在在相关联的工作单元的执行期间稍后发生，则实时值可以被转移并且在目的OpenCL设备上被初始化。如上所述，编译器内的代码生成器可能插入标签和插入其它代码以在迁移期间调用实时值。在目的地OpenCL设备处，用于迁移输入点的代码生成初始化包含实时值的数据结构并且继续进行内核执行。替代地，编译器可以创建计算子内核以继续进行如上文所描述的执行。在块912中，编译器完全至少两个异构处理器核心的计算内核的编译。可以插入其它调试和插装代码。

[0070] 在一个实施方案中，编译器生成多个数据结构。两个或更多个数据结构包括在诸如通用核心和SIMD核心的给定的目标OpenCL设备上的各计算子内核的可执行目标代码。另一个数据结构包括将在迁移时被转移和访问的实时数据。给定被指定为计算内核中的潜在迁移点的标签，编译器利用数据流分析来判定可以被转移的实时值。在诸如在寄存器中被高速缓存的执行中在那个点处未定义的实时值被放置在可访问运行时间环境的位置中。这些位置的实例包括相关联的初始的内存位置和保存被保留的内容的寄存器。在一个实施方案中，可以利用启发式检查来判定数据传送的大小是否允许在异构核心之间有益的改变执行。

[0071] 另外，编译器可以生成由运行时间环境释放的另一个数据结构以将实时数据转移至相关联的目的OpenCL设备。该数据结构可以提供将被输送的实时数据的位置和尺寸和它们的在源OpenCL设备和目的OpenCL设备两者地址空间中的位置。另外，编译器生成用于目的设备的对应的版本的内核。OpenCL设备中的每一个的相应的编译代码访问在指定的位置处的实时数据并且开始在迁移点处的执行。

[0072] 现在转向图10，示出用于通过利用预运行时间数据信息优化在处理器中的多个工作单元的并行执行的方法1000的一个实施方案。在上述图4中所示的处理节点110和硬件资源分配中体现的部件一般可以根据方法1000工作。出于论述的目的，在该实施方案和稍后描述的方法的随后的实施方案中的步骤以顺序次序示出。然而，一些步骤可能以不同于示出的次序发生，一些步骤可能被同时执行，一些步骤可能与其它步骤合并，并且一些步骤在另一个实施方案中可能不存在。

[0073] 在块1002中，将数据的相关联的记录分配给给定的计算内核的各工作单元。在块1004中，OS调度器424将工作单元调度至异构核心。在块1006中，异构处理器核心执行对应调度的工作单元。

[0074] 在块1008中，到达给定的标记迁移点。在一个实施方案中，可以执行对当前使用OpenCL设备的利用率的测量。如果测量指示利用率或性能在给定的阈值下方，则相关联的计算内核或计算子内核可以被迁移到另一个OpenCL设备，诸如具有不同的微架构的异构核心。在一个实施方案中，该测量是在SIMD核心上的达到相关联的计算内核或计算子内核内的退出或返回的多个当前执行工作单元的计数。替代地，在波前中的多个禁用计算单元的计数可以提供相同的数量。如果该计数在给定的阈值之上，则尚未到达出口点的工作单元可以迁移到另一个异构核心，诸如通用核心。然后，在SIMD核心上的波前可以释放并且可用于其它调度的工作单元。

[0075] 在其它实施方案中，以上技术可以扩展以在确定在SIMD核心上的波前中的大部分并行执行工作单元是空闲的并且其余的工作单元被预期大致大致执行的任意情形下启动迁移。例如，生成的数据结构可能是在共用存储器中以及在一个或多个高速缓存中。在具有虚拟内存支持的系统，工作单元的一个子集可能击中高速缓存，而其余的工作单元经历虚拟内存击不中，这是长延迟事件。在这种情况下，由于进一步的执行可能从由当前执行启用的预取技术中受益。

[0076] 如果确定执行效率不在给定的阈值(条件块1010)下方，则方法1000的控制流返回至块1006并且执行继续。如果确定执行效率在给定的阈值(条件块1010)下方，则在块1012中，一个或多个工作单元被识别以迁移到具有不同于第一处理器核心的微架构的微架构的第二处理器核心。识别的工作单元可能导致将在给定的阈值下方的上述测量。在块1014中，由第一处理器核心产生的相关联的局部数据被提升为全局数据。在块1016中，编译版本的迁移工作单元被调度至在第二处理器核心上在迁移的标记点处开始执行。

[0077] 应注意的是，上述实施方案可以包括软件。在这样的实施方案中，可以将实施方法和/或机制的程序指令传送或存储在计算机可读介质上。被构造成存储程序指令的许多类型的介质是可用的并且包括硬盘、快盘、CD-ROM、DVD、闪存存储器、可编程ROM(PROM)、随机存取存储器(RAM)、和各种其它形式的易失性或非易失存储器。一般来说，计算机访问存储介质可以包括在使用期间可由计算机访问的任意存储介质以对计算机提供指令和/或数据。例如，计算机访问存储介质可以包括存储介质，诸如磁性或光学介质，例如，磁盘(固定的或可移除的)、带、CD-ROM、或DVD-ROM、CD-R、CD-RW、DVD-R、DVD-RW、或Blu-Ray。存储介质可能进一步包括易失性或非易失性存储器介质，诸如RAM(例如，同步动态RAM(SDRAM)、双数据速率(DDR、DDR2、DDR3等)SDRAM、低功率DDR(LPDDR2等)SDRAM、Rambus DRAM(RDRAM)、静态RAM(SRAM)等)、ROM、闪存存储器、经诸如通用串行总线(USB)接口等的外部接口可访问的非易失性存储器(例如，闪存存储器)。存储介质可以包括微机电系统(MEMS)、以及经诸如网络和/或无线链路的通信介质可访问的存储介质。

[0078] 另外，程序指令可以包括在诸如C的高级编程序语言、或诸如Verilog、VHDL的设计语言(HDL)、或诸如GDSII流格式(GDSII)的数据库格式中的硬件功能的行为级描述或寄存器转移级(RTL)描述。在一些情况下，描述可以由可能综合描述以产生包括来自综合库的栅极列表的网表的综合工具读取。网表包括一组栅极，该一组栅极也表示包括系统的硬件的功能性。然后，网表可能被放置并且被路由以产生描述将被应用于掩模的几何形状的数据集。然后，可能在各种半导体制造步骤中使用掩膜以产生对应于系统的半导体电路或电路。替代地，根据需要，在计算机可访问存储介质上的这些指令可以是网表(有或没有综合库)或数据集。另外，通过来自这样的供应商如Cadence®、EVE®、和MentorGraphics®的基于硬件类型的仿真器，可以利用这些指令用于仿真的目的。

[0079] 虽然已经以相当大的细节描述了以上实施方案，对于本领域的技术人员而言，一旦充分地领悟以上公开内容，许多变化和修改将变得明显。意图是，解释下列权利要求以包含所有这样的变化和修改。

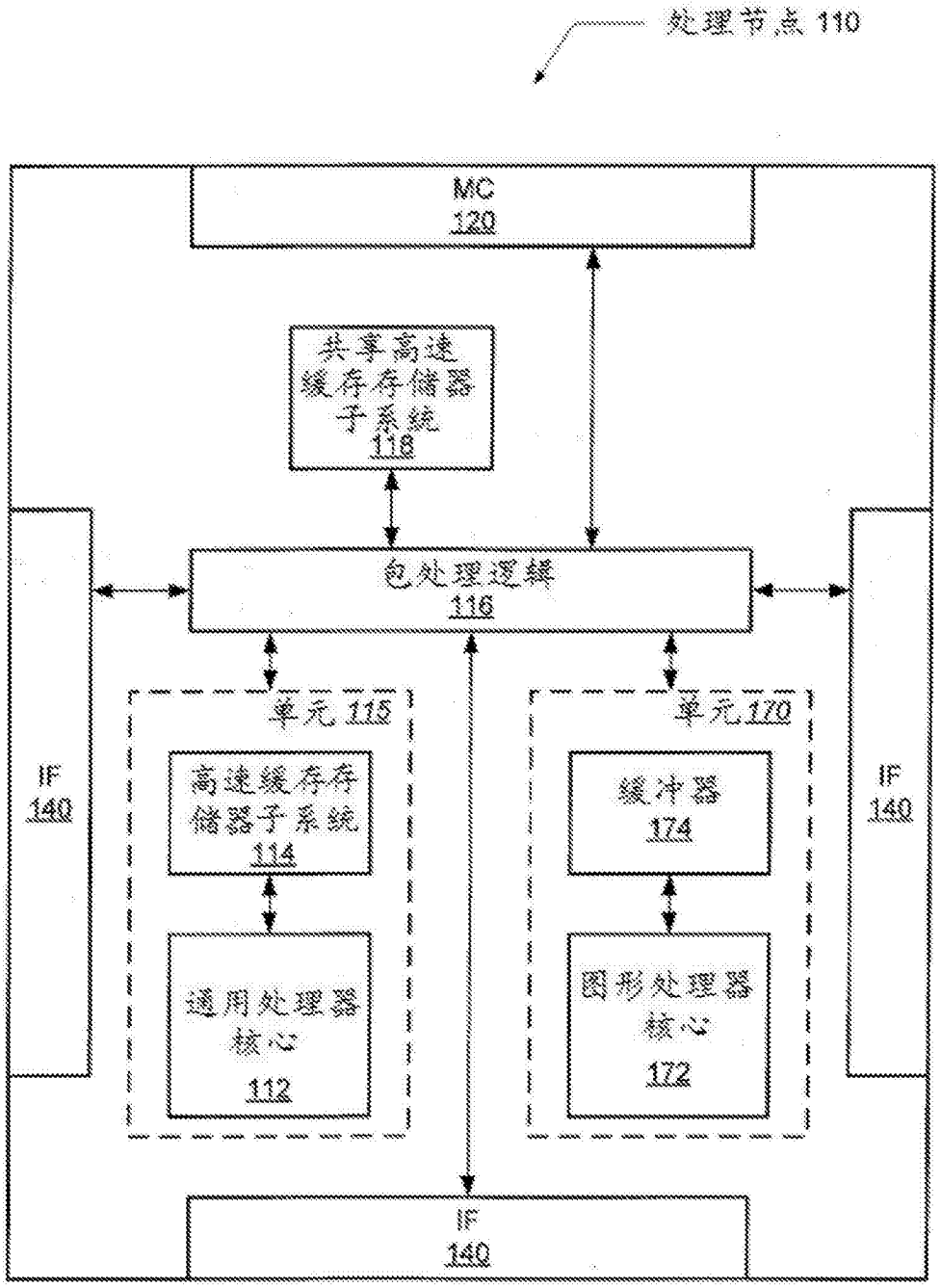

处理节点 110
MC
120
共享高速
缓存存储器
子系统
118
包处理逻辑
116
单元 115
高速缓存存
储器子系统
114
通用处理器
核心
112
单元 170
缓冲器
174
图形处理器
核心
172
IF
140
IF
140
IF
140

图1

代码 210

```
void DoWorkA(Record * record)
{
   // perform steps of a  first algorithm
   // on the record
}

void DoWorkB(Record * record)
{
   // perform steps of a  second algorithm
   // on the record
}

void KernelFunction(Record * recordsArray)
{
   const int unitId = get_global_id(0);

   DoWorkA(&recordsArray[unitId]);

   DoWorkB( &recordsArray[unitId]);
}
```

代码 220

```
INPUT = [1, 3, 2, 7, 8,
         1, 3, 5, 2, 4]

KERNEL Power2(INPUT,
RESULT) {
   N = get_global_id();
   V = INPUT[N];
  RESULT[N] = V * V;
}
```

图2

代码 230

```
void DoWorkA(Record * record)
{
   // perform steps of a  first algorithm
   // on the record
}

void DoWorkB(Record * record)
{
   // perform steps of a  second algorithm
   // on the record
}

void KernelFunction(Record * recordsArray)
{
   const int unitId = get_global_Id(0);
   If (EvaluateFunction(recordsArray[unitId])
   {
      DoWorkA(&recordsArray[unitId]);
   }
   else
   {
      DoWorkB( &recordsArray[ unitId]);
   }
}
```

图3

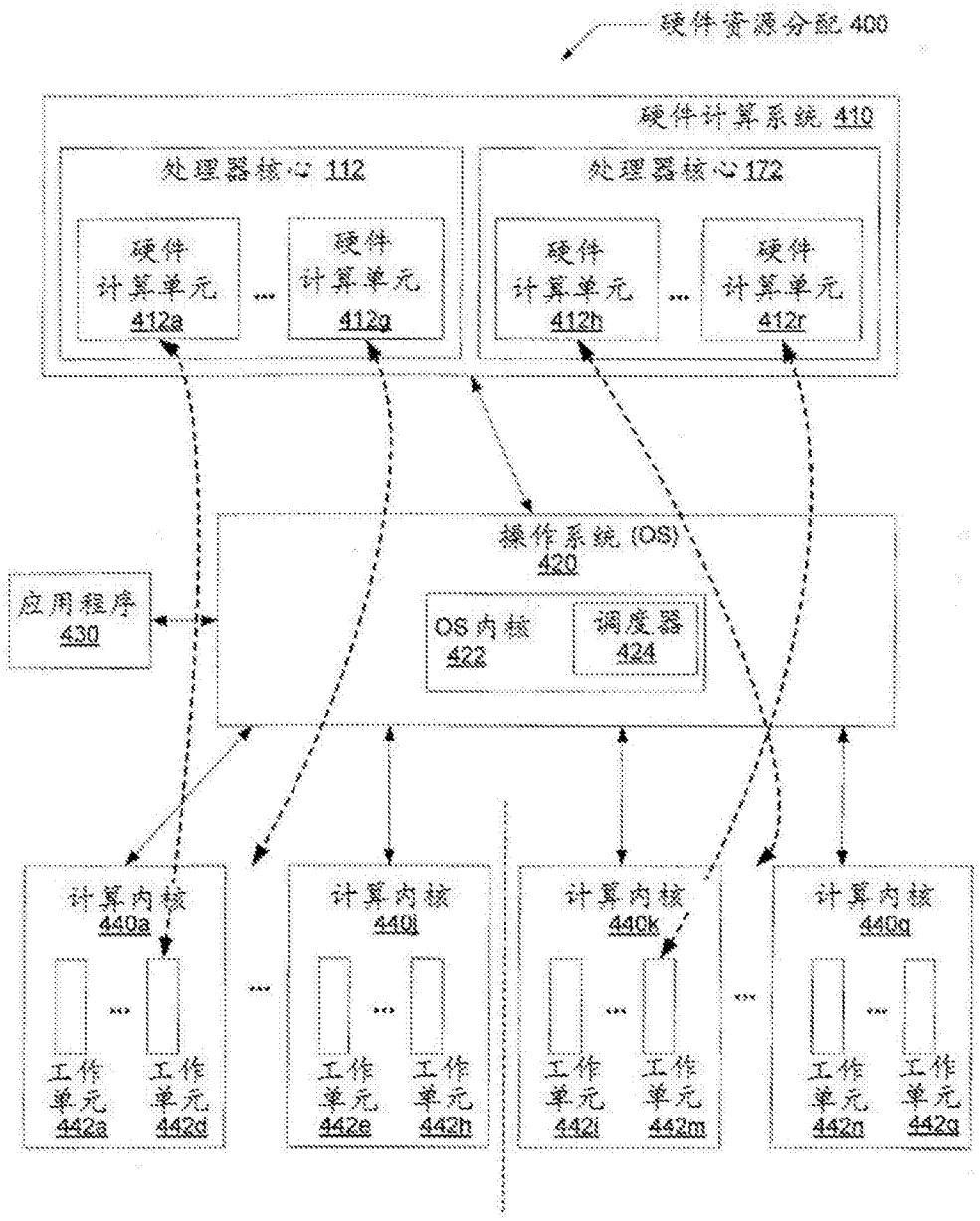

硬件资源分配 400
硬件计算系统 410
处理器核心 112
处理器核心 172
硬件计算单元 412a
硬件计算单元 412g
硬件计算单元 412h
硬件计算单元 412r
操作系统 (OS) 420
应用程序 430
OS 内核 422
调度器 424
计算内核 440a
计算内核 440i
计算内核 440k
计算内核 440q
工作单元 442a
工作单元 442d
工作单元 442e
工作单元 442h
工作单元 442i
工作单元 442m
工作单元 442n
工作单元 442q

图4

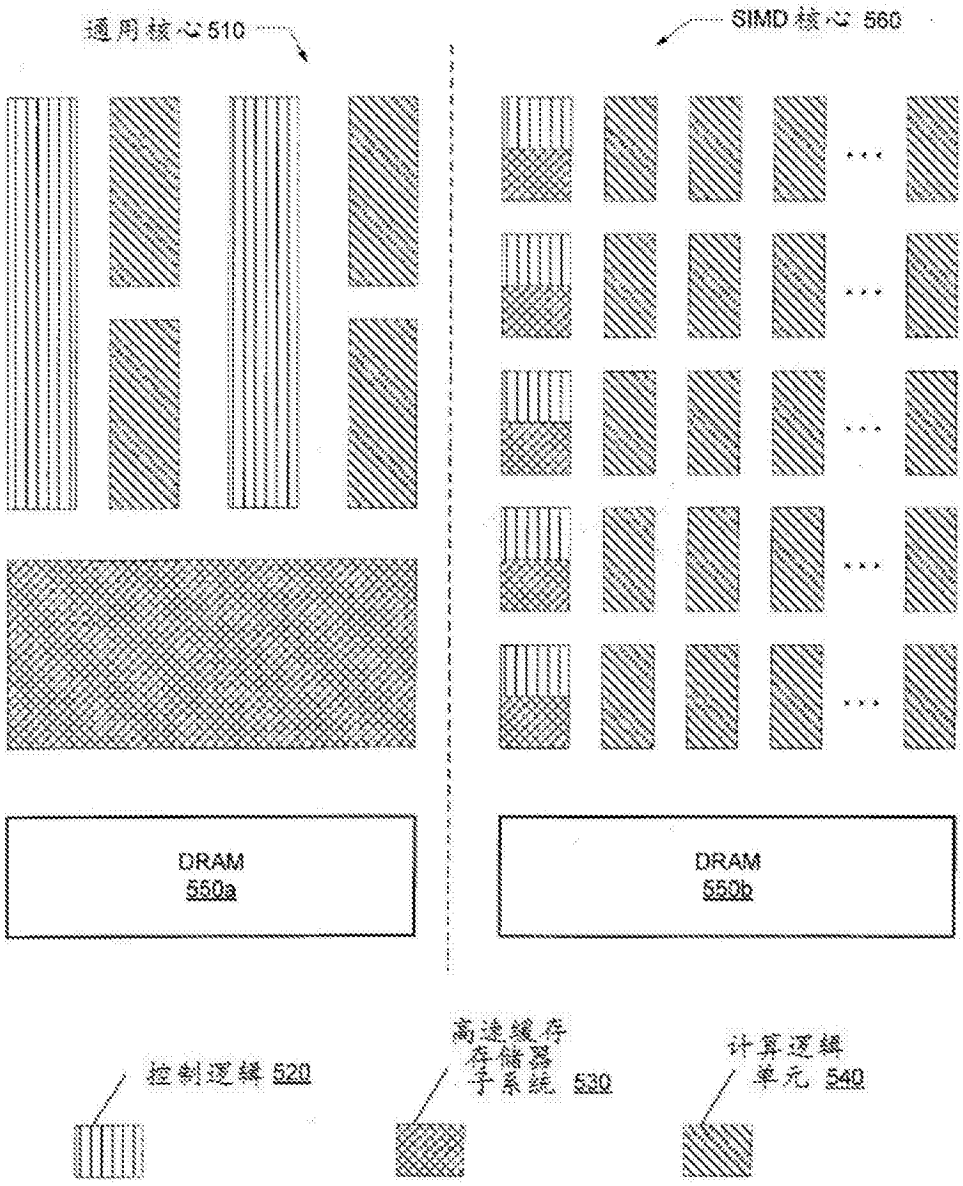

通用核心 510
SIMD 核心 560
DRAM
550a
DRAM
550b
控制逻辑 520
高速缓存
存储器
子系统 530
计算逻辑
单元 540

图5

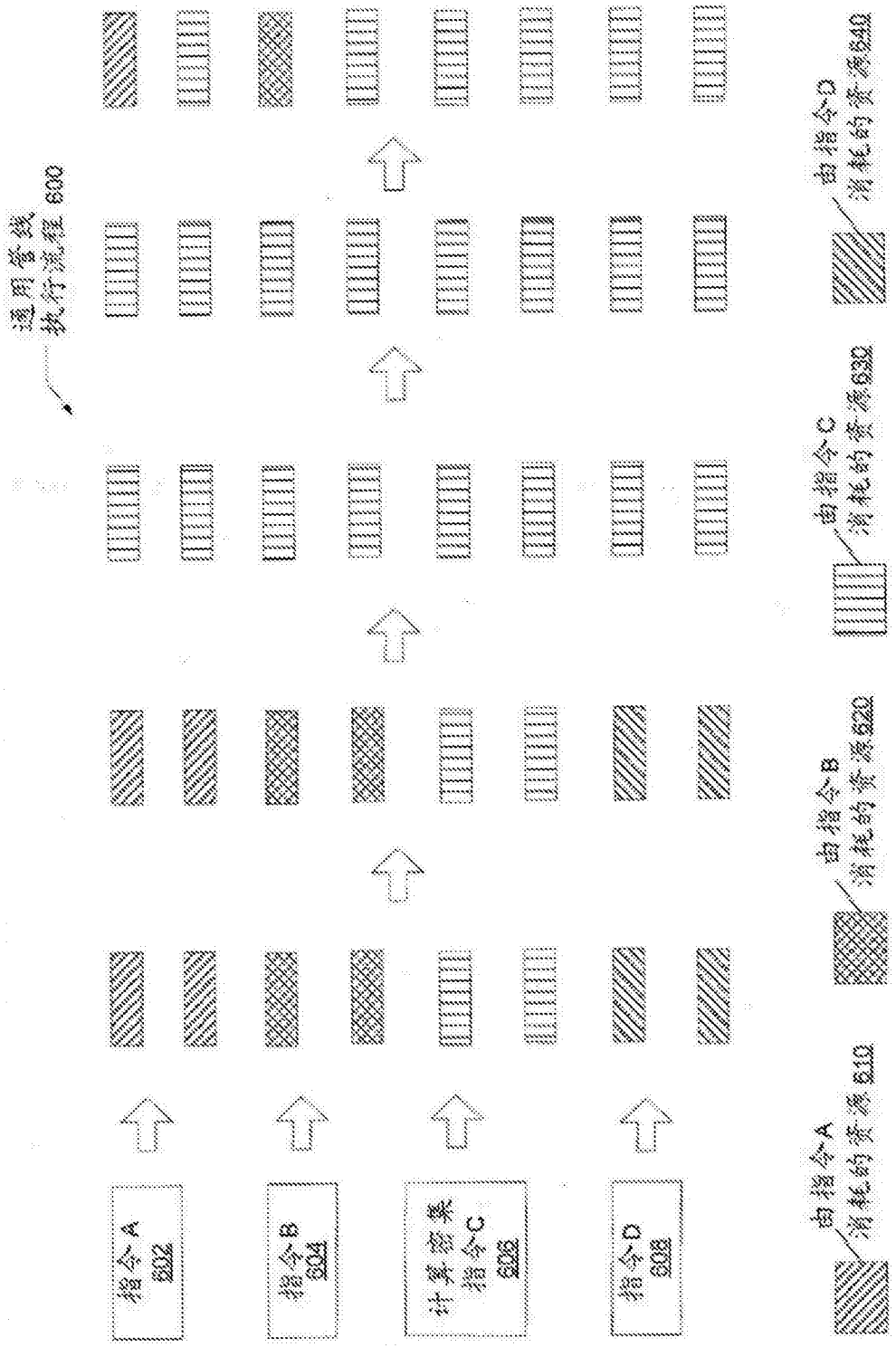

通用管线执行流程 600
指令A 602
指令B 604
计算密集指令C 606
指令D 608
由指令A消耗的资源 610
由指令B消耗的资源620
由指令C消耗的资源630
由指令D消耗的资源640

图6

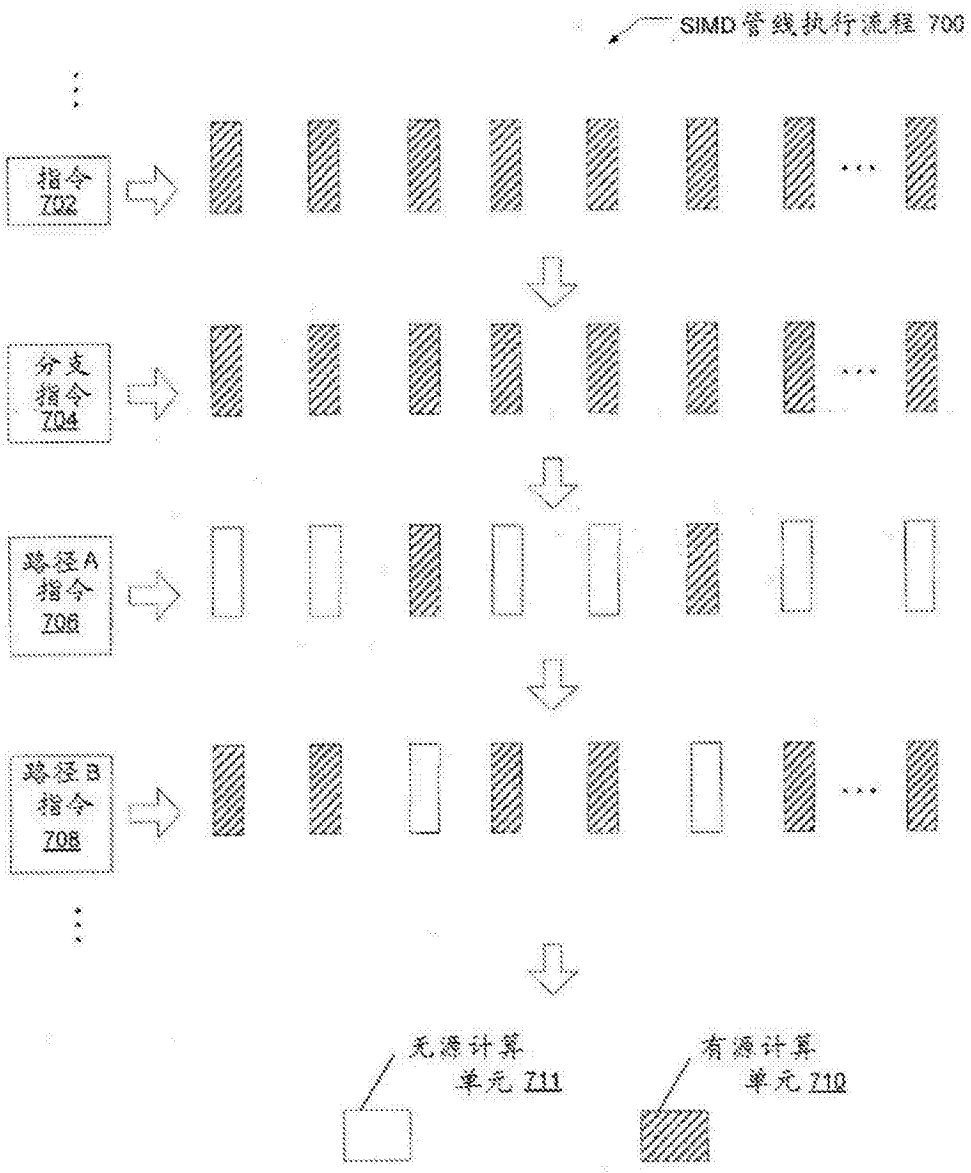

SIMD管线执行流程 700
指令 702
分支指令 704
路径A指令 706
路径B指令 708
无源计算单元 711
有源计算单元 710

图7A

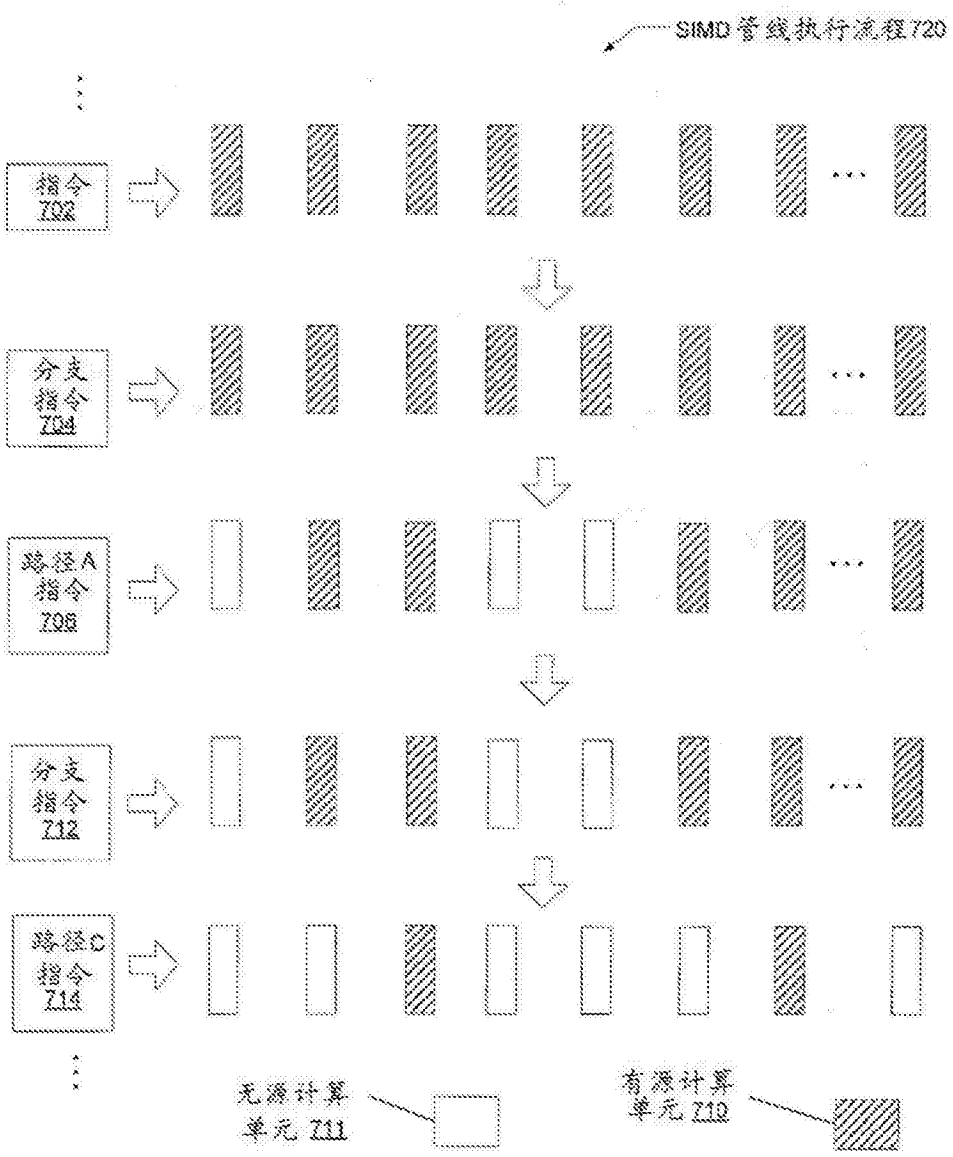

SIMD管线执行流程720
指令 702
分支指令 704
路径A指令 706
分支指令 712
路径C指令 714
无源计算单元 711
有源计算单元 710

图7B

代码 800

```
int f (int arg) { ... }
int g(int arg){_}
kernel foo (
        global int input *, // input array
        local int tmp*, // local
        global int output *)  // result array
{  int id = get_local_id( ... );
   int local_temp = 0;
   tmp[id] input [id] ;
   secondary_entry:
   do {
      tmp[id] = f( input [id] );
      local_temp = f( tmp[id], local_temp);
   }  while ( g(input[id]) ) ;
output [id] = h(local_temp);
}
```

图8

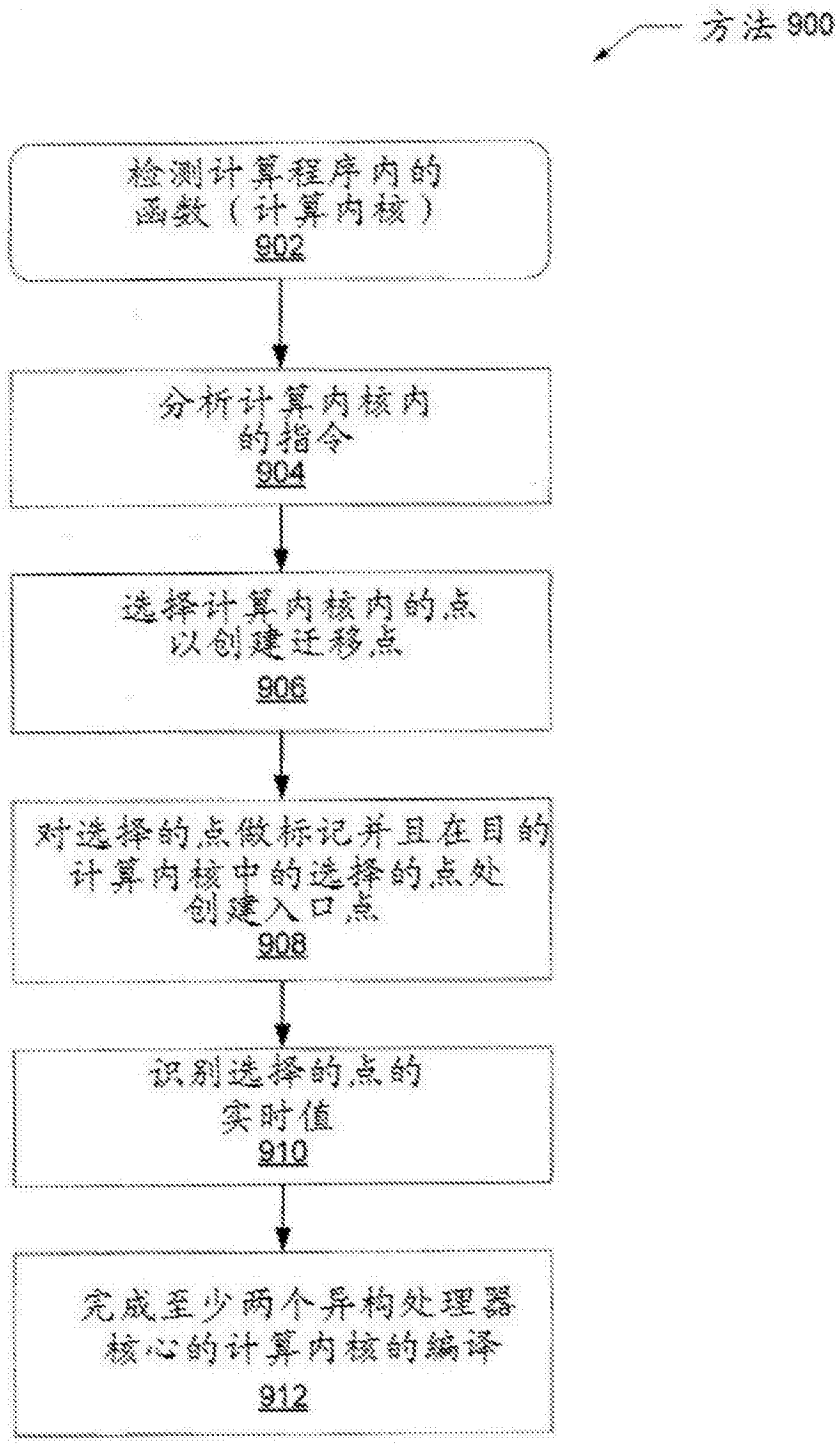

方法900
检测计算程序内的函数（计算内核）
902
分析计算内核内的指令
904
选择计算内核内的点以创建迁移点
906
对选择的点做标记并且在目的计算内核中的选择的点处创建入口点
908
识别选择的点的实时值
910
完成至少两个异构处理器核心的计算内核的编译
912

图9

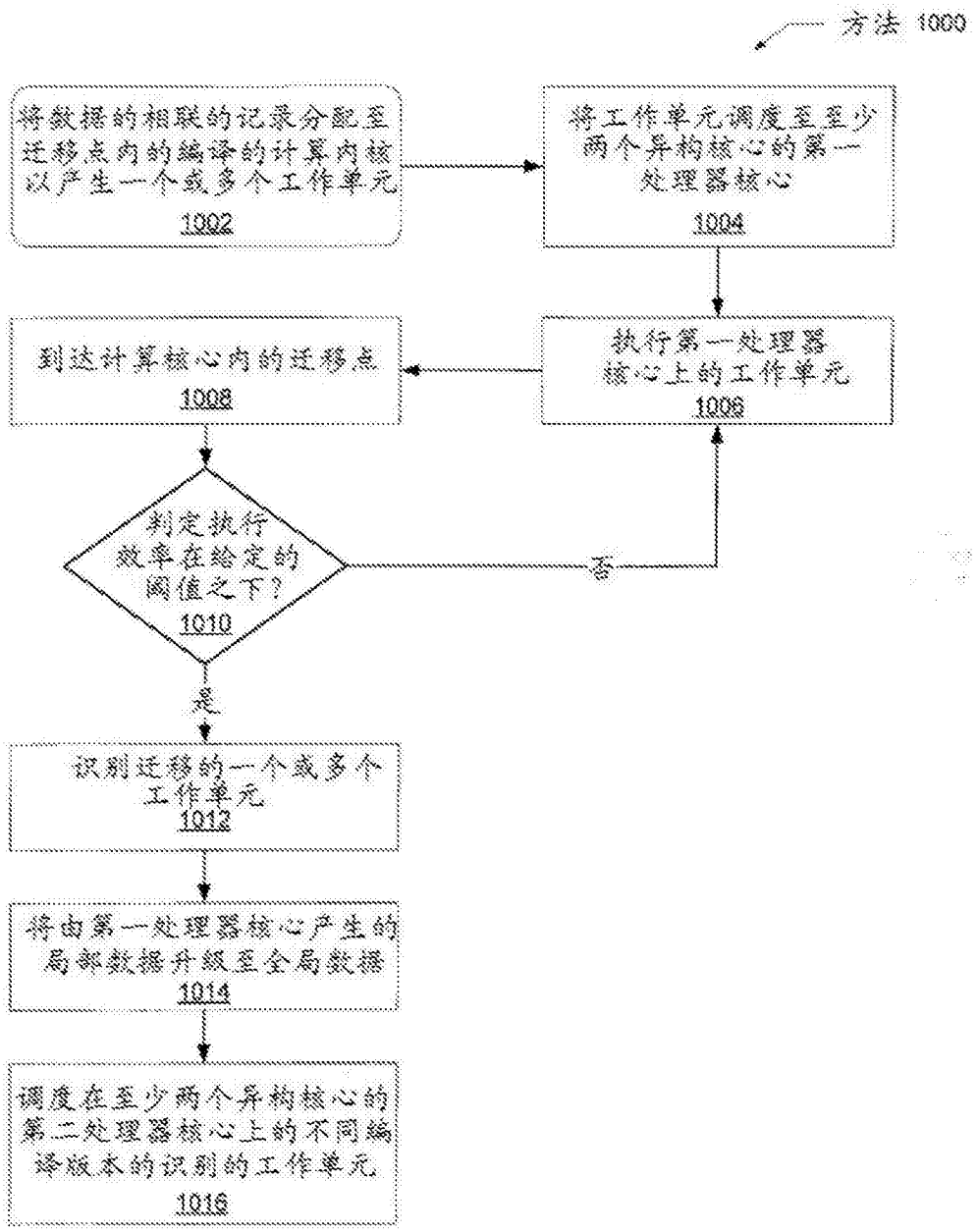

方法 1000
将数据的相联的记录分配至迁移点内的编译的计算内核以产生一个或多个工作单元
1002
将工作单元调度至至少两个异构核心的第一处理器核心
1004
执行第一处理器核心上的工作单元
1006
到达计算核心内的迁移点
1008
判定执行效率在给定的阈值之下?
1010
否
是
识别迁移的一个或多个工作单元
1012
将由第一处理器核心产生的局部数据升级至全局数据
1014
调度在至少两个异构核心的第二处理器核心上的不同编译版本的识别的工作单元
1016

图10