(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 105653243 B
(45)授权公告日 2018.03.30

(21)申请号 201510977450.0

(22)申请日 2015.12.23

(65)同一申请的已公布的文献号
申请公布号 CN 105653243 A

(43)申请公布日 2016.06.08

(73)专利权人 北京大学
地址 100871 北京市海淀区颐和园路5号

(72)发明人 梁云 李秀红

(74)专利代理机构 北京万象新悦知识产权代理事务所(普通合伙) 11360
代理人 张肖琪

(51)Int.Cl.
G06F 9/38(2006.01)
G06T 1/00(2006.01)

(56)对比文件
CN 103064657 A,2013.04.24,
US 2012185671 A1,2012.07.19,
US 8566647 B2,2013.10.22,
CN 105117369 A,2015.12.02,
丑文龙等.ＡＲＭ ＧＰＵ的多任务调度设计与实现.《西安交通大学学报》.2014,第48卷(第12期),
Adriaens等.The Case for GPGPU Spatial Multitasking.《IEEE International Symposium on High Performance Computer Architecture》.2012,
Lee等.Improving GPGPU resource utilization through alternative thread block scheduling.《IEEE International Symposium on High Performance Computer Architecture》.2014,
姚远等.基于通用图形处理器的Jacobi算法研究.《信息工程大学学报》.2010,第11卷(第3期),

审查员 陈敏

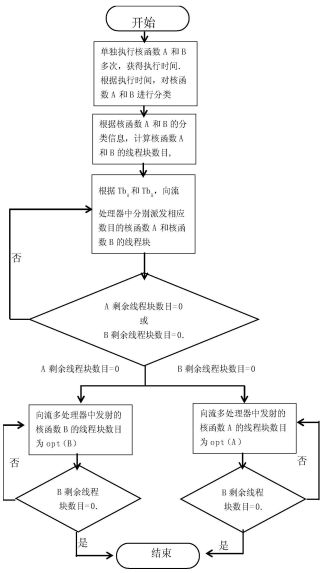
权利要求书3页 说明书9页 附图5页

(54)发明名称

一种通用图形处理器多任务并发执行的任务派发方法

(57)摘要

本发明公布了一种通用图形处理器多任务并发执行的任务派发方法,通过线程块派发引擎方法首先将核函数进行分类,然后根据核函数分类计算得到向一个流处理器分别派发的核函数的线程块数目,将不同核函数相应数目的线程块派发到同一个流多处理器中,以达到提高通用图形处理器中每个流多处理器资源的利用率,提升系统性能和能效比的目的。本发明还可进一步利用一种一级数据缓存旁路方法,该方法首先通过动态方法来确定旁路哪一个核函数的线程块,根据相应核函数的旁路的线程块的数目进行旁路,以达到减轻一级数据缓存的压力、进一步提高性能的目的。



CN 105653243 B

1.一种通用图形处理器多任务并发执行的任务派发方法,通过线程块派发引擎方法首先将核函数进行分类,然后根据核函数分类计算得到向一个流多处理器分别派发的核函数的线程块数目,将不同核函数相应数目的线程块派发到同一个流多处理器中,以达到提高通用图形处理器中每个流多处理器资源的利用率,提升系统性能和能效比的目的;所述线程块派发引擎方法包括如下步骤:

A1)将核函数进行分类,核函数的种类包括Type Down、Type Up和Type Optimal;所述核函数进行分类,分类过程执行操作A11)～A13):

A11)单独执行每个核函数Kernel多次,每次向流多处理器中派发不同数目的线程块,得到相应的执行时间;

A12)将A11)中执行时间最短时每个流多处理器上的线程块数目定义为opt(Kernel);

A13)当A12)中opt(Kernel)等于1时,该核函数Kernel的分类为Type Down;当opt(Kernel)等于max(Kernel)时,该核函数分类为Type Up,所述max(Kernel)是一个流多处理器最多可以容纳的来自一个核函数的线程块数目;当opt(Kernel)不等于1且opt(Kernel)不等于max(Kernel)时,该核函数分类为Type Optimal;

A2)针对多个核函数的线程块,将所述多个核函数看作一个任务池,先从这些核函数中选择两个核函数作为核函数{A,B},设定Tb_A和Tb_B分别表示线程块派发引擎向一个流多处理器中分别派发的核函数A和核函数B的线程块数目,根据A1)分类得到的核函数的种类信息,计算得到Tb_A和Tb_B;

A21)对于两个并发执行的核函数组合{A,B},两个核函数A、B的类型分别设为$Type_A$和$Type_B$,当两个核函数$Type_A$和$Type_B$都属于Type Up类型时,结束操作;当所述$Type_A$和$Type_B$中至少有一个核函数的类型属于Type Down或者Type Optimal时,继续执行如下操作;

A22)根据核函数A、B的类型$Type_A$和$Type_B$,分别计算得到Tb_A和Tb_B;包括如下情形:

当核函数A的类型为Type Down,核函数B的类型为Type Up时,Tb_A=opt(A),Tb_B等于利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r;

当核函数A的类型为Type Down,核函数B的类型为Type Optimal时,Tb_A=opt(A),Tb_B等于利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r与opt(B)中的较小者;

当两个核函数A、B的类型都为Type Down时,Tb_A=opt(A),Tb_B=opt(B);

当两个核函数A、B的类型都为Type Optimal时,Tb_A=opt(A),Tb_B为利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r与opt(B)中的较小者;

当核函数A的类型为Type Optimal,核函数B的类型为Type Up时,Tb_A=opt(A),Tb_B为利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r;

A3)根据Tb_A和Tb_B,向通用图形处理器(GPGPU)的一个流多处理器中分别派发相应数目的核函数A和核函数B的线程块;

A4)循环执行上述A3),当其中有核函数的剩余线程块数目小于向流多处理器派发的相应线程块数目时,向流多处理器派发该核函数所有剩余线程块数目,该核函数的剩余线程块数目为0,该核函数执行完成;此时从两个核函数{A,B}的并发执行变为一个核函数的单独执行;当任务池中的核函数数目为0时,执行步骤A5);当任务池中的核函数数目不为0时,从任务池中选择一个与未完成的核函数重新组合成{A,B},继续按照上述步骤A2) ～A4)

进行线程块派发；

A5)在一个核函数Kernel单独执行期间，向流多处理器中派发该核函数的线程块数目为opt(Kernel)，当核函数的剩余线程块数目小于opt(Kernel)时，向流多处理器派发该核函数的所有剩余线程块数目；直到该核函数执行完成。

2.如权利要求1所述通用图形处理器多任务并发执行的任务派发方法，其特征是，所述利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r通过以下过程得到：

设定流多处理器中线程、共享存储和寄存器的总资源分别为T_M、S_M和R_M；

设定核函数A的每一个线程块在执行期间所占据的线程、共享存储和寄存器资源分别为T_A、S_A和R_A；设定核函数B的每一个线程块在执行期间所占据的线程、共享存储和寄存器资源分别为T_B、S_B和R_B；

所述剩下的计算资源Compute Remain(A) 具体指的是在流多处理器中派发了Tb_A个核函数A的线程块之后还能派发的核函数B的线程块数目，计算方法为取所述最大线程块数目r，同时满足以下不等式：$r\times T_B+Tb_A\times T_A\leqslant T_M$；$r\times S_B+Tb_A\times S_A\leqslant S_M$；$r\times R_B+Tb_A\times R_A\leqslant R_M$。

3.如权利要求1所述通用图形处理器多任务并发执行的任务派发方法，其特征是，在所述步骤A3)之后，执行一级数据缓存旁路方法，再继续执行步骤A4)；所述一级数据缓存旁路方法首先通过动态方法来确定旁路哪一个核函数的线程块，然后根据相应核函数的旁路的线程块的数目进行旁路，以达到减轻一级数据缓存的压力、进一步提高性能的目的；所述一级数据缓存旁路方法执行如下操作：

B1)对于两个核函数的组合{A,B}，设定By_A和By_B分别表示旁路的核函数A的线程块的数目和旁路的核函数B的线程块的数目，只选择其中一个核函数的线程块进行旁路操作；当选择核函数A的线程块进行旁路操作时，By_A不等于0，By_B等于0；当选择核函数B的线程块进行旁路操作时，By_B不等于0，By_A等于0；设定$Stall^{A}_{By_A}$代表当By_A个来自核函数A的线程块旁路一级缓存时在一个抽样周期里流多处理器的空闲时钟总数；设定$Stall^{B}_{By_B}$代表当By_B个来自核函数B的线程块旁路一级缓存时，在一个抽样周期里流多处理器的空闲时钟总数；设定$Stall_{none}$代表没有任何线程块旁路一级缓存时，在一个抽样周期里流多处理器的空闲时钟总数；

设定By_A和By_B的初始值均为0；在一个抽样周期之后，得到$Stall_{none}$；将一个抽样周期作为Tb_A个核函数A的线程块和Tb_B个核函数B的线程块的生命期；

B2)分别针对两个核函数进行旁路操作，分别得到相应核函数的线程块旁路一级缓存时在一个抽样周期里流多处理器的空闲时钟总数；

B3)比较$Stall_{none}$、$Stall^{A}_{By_A}$和$Stall^{B}_{By_B}$，选择进行旁路的核函数，将选择进行旁路的核函数设为核函数C；

B4)将选择进行旁路的核函数设为核函数C，选择By_C+1个核函数C的线程块旁路一级数据缓存，经过一个抽样周期之后，收集得到$Stall^{C}_{By_C+1}$；当$Stall^{C}_{By_C+1}<Stall^{C}_{By_C}$时，$By_C$增加1，继续执行步骤B4)；否则，$By_C$减少1，结束操作；当$By_C$达到上限时，停止更新$By_C$，结束操作。

4.如权利要求3所述通用图形处理器多任务并发执行的任务派发方法，其特征是，B3)所述选择进行旁路的核函数，具体是：

当$Stall_{none}$是最小值时，不旁路任一核函数；

当$Stall^{A}_{By_A}$是最小值时，选择核函数A进行旁路操作，设置$By_A=1$；

当$Stall^{B}_{By_B}$是最小值时，选择核函数B进行旁路操作，设置$By_B=1$。

5.如权利要求4所述通用图形处理器多任务并发执行的任务派发方法，其特征是，所述By_A和By_B的范围是$0\leqslant By_A\leqslant Tb_A$，$0\leqslant By_B\leqslant Tb_B$。

一种通用图形处理器多任务并发执行的任务派发方法

技术领域

[0001] 本发明属于高性能计算技术领域，涉及高性能计算中多任务并发执行方法，尤其涉及一种通用图形处理器(GPGPU)多任务并发执行的任务派发方法。

背景技术

[0002] 通用图形处理器(GPGPU)是一种利用图形处理器众核结构、多线程和高访存带宽的特点来处理生物计算、图像处理和物理仿真模拟等高性能计算任务的处理器。在现代计算中，计算任务对高性能和高吞吐率的迫切需求，使通用图形处理器在计算领域中被广泛利用并且扮演着越来越重要的角色。而且，随着云计算的发展以及计算终端的普及，越来越多的任务被同时发送到通用图形处理器中等待执行。因此，通用图形处理器中的多任务并发执行技术，对于高效利用计算资源以及提升计算速度和能效性具有重要影响。

[0003] 中央处理器(CPU)将计算任务分配给通用图形处理器，然后，计算任务以核函数(Kernel)的形式在通用图形处理器中执行。当核函数被发射到通用图形处理器时，核函数会产生一个称为计算网格(Grid)的计算任务实例。一个计算网格包含成百上千的线程(Thread)。这些线程通过层次化的方式被组织管理。每32个线程组成一个线程束(Warp)，若干个线程束进一步被组织成一个线程块(Block)。一个计算网格中线程块的数目和每个线程块中线程的数目，由程序员编程时指定。

[0004] 硬件上，每个通用图形处理器由多个流多处理器(Streaming Multiprocessor，SM)组成，这些流多处理器通过互连网络与片外存储资源相连。每个流多处理器中包含3种存储资源：寄存器堆(Register File)、一级缓存(L1Cache)和共享存储器(Shared Memory)；以及3种单指令流多数据流(SIMD)的执行单元：流处理器(Streaming Processor，SP)、特殊功能单元(Special Function Unit，SFU)和装载储存单元(Load/Store Unit，LDST)。通用图形处理器拥有一个线程块派发引擎，负责将核函数的线程块派发到流多处理器中。每个流多处理器中有若干个线程束调度器，来调度管理线程束的执行。

[0005] 不同的核函数对于计算资源需求存在着明显的差异。例如，一个核函数属于计算密集型，对流多处理器需求高，但不能充分利用通用处理器高的访存带宽；另一个核函数属于访存密集型，对于存储资源需求高，但不能充分利用流多处理器的计算能力。因此，执行单个核函数时，通用图形处理器的计算资源往往不能被充分利用。可采用多任务并发执行提高资源利用率。

[0006] 2012年，美国威斯康星大学麦迪逊分校(University of Wisconsin-Madison)的Adriaens等人提出一种空间多任务并发管理方案(Published on:High Performance Computer Architecture(HPCA)，2012IEEE 18th International Symposium on，Pages 1-12)。该方案通过将这些流多处理器进行空间上的划分，分配给不同的核函数，来平衡不同核函数对流多处理器和片外存储资源的不均衡需求。该方法是一种粗粒度的并发技术，虽然能平衡流多处理器与片外存储资源的利用，然而，一个流多处理器内部较低的计算资源利用率仍然是一个很严重的问题。

[0007] 2014年,韩国科学技术院(Korea Advanced Institute of Science and Technology,KAIST)的Lee等人针对一个流多处理器内部计算资源利用率低的问题,提出了一种混合并发核函数执行方案(Published on:High Performance Computer Architecture(HPCA),2014IEEE 20th International Symposium on,Pages 260-271)。该方案提出不同的核函数可以同时发射到一个流多处理器上,从而提高流多处理器内部计算资源的利用率。但是,该方案没有具体处理流多处理器内部不同核函数线程块的调度问题,也没有提出针对一级数据缓存污染的优化方案。

发明内容

[0008] 为了克服上述现有技术的不足,本发明提供一种通用图形处理器多任务并发执行的任务派发方法,包括线程块派发引擎方法和一级数据缓存旁路方法,可以高效利用流多处理器内部的计算资源。

[0009] 本发明提供的技术方案是:

[0010] 一种通用图形处理器多任务并发执行的任务派发方法,通过线程块派发引擎方法首先将核函数进行分类,然后根据分类信息计算得到向一个流处理器中分别派发的核函数的线程块数目,将不同核函数相应数目的线程块派发到同一个流多处理器中,以达到提高通用图形处理器中每个流多处理器资源的利用率,提升系统性能和能效比的目的;所述线程块派发引擎方法包括如下步骤:

[0011] A1)针对两个核函数的组合{A,B},将核函数进行分类:设定$Type_A$和$Type_B$分别表示核函数A和核函数B的种类,依据每个核函数单独运行时性能最佳的线程块数目opt(Kernel)与一个流多处理器最多可以容纳的来自核函数的线程块数目max(Kernel)的大小关系,对核函数进行分类;分类过程如下:

[0012] A11)单独执行每个核函数多次,每次线程块派发引擎向流多处理器中派发不同数目的线程块。

[0013] A12)对A11)中所有情况,比较它们的执行时间(性能),取执行时间最短的情况,将这种情况中每个流多处理器上的线程块数目定义为opt(Kernel)。

[0014] A13)如果A12)中opt(Kernel)=1,我们将该核函数分类为Type Down;如果opt(Kernel)=max(Kernel)(其中max(Kernel)中指的是一个流多处理器最多可以容纳的来自核函数的线程块数目),我们将该核函数分类为Type Up;其他情况,我们将该核函数分类为Type Optimal。

[0015] A2)设定Tb_A和Tb_B分别表示线程块派发引擎向一个流处理器中分别派发的核函数A和核函数B的线程块数目,根据分类信息计算得到Tb_A和Tb_B;具体步骤如下所示:

[0016] A21)核函数的类型包括Type Down、Type Up和Type Optimal;对于两个并发执行的核函数组合{A,B},两个核函数A、B的类型分别设为$Type_A$和$Type_B$;其中有一个核函数的类型属于Type Down或者Type Optimal;本发明不适用于两个核函数都属于Type Up类型的情况。

[0017] A22)根据核函数A、B的类型,分别计算得到Tb_A和Tb_B;

[0018] 当核函数A的类型为Down,核函数B的类型为Up时,Tb_A=opt(A),Tb_B等于利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r;

[0019] 由于每一个线程块都会占据流多处理器中的线程、共享存储和寄存器，定义流多处理器中线程、共享存储和寄存器的总资源分别为T_M、S_M和R_M，核函数A的每一个线程块在执行期间所占据的三种资源分别为T_A、S_A和R_A，核函数B的每一个线程块在执行期间所占据的三种资源分别为T_B、S_B和R_B。ComputeRemain(A)是在流多处理器中派发了Tb_A个核函数A的线程块之后还能派发的核函数B的线程块数目，具体计算方法为取线程块数目最大值r，同时满足以下3个不等式：$r\times T_B+Tb_A\times T_A\leqslant T_M$；$r\times S_B+Tb_A\times S_A\leqslant S_M$；$r\times R_N+Tb_A\times R_A\leqslant R_M$。

[0020] 当核函数A的类型为Down，核函数B的类型为Optimal时，Tb_A＝opt(A)，Tb_B等于利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r与opt(B)中的较小者；

[0021] 当2个核函数的类型都为Down时，Tb_A＝opt(A)，Tb_B＝opt(B)；

[0022] 当2个核函数的类型都为Optimal时，Tb_A＝opt(A)，Tb_B为利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r与opt(B)中的较小者；

[0023] 当核函数A的类型为Optimal，核函数B的类型为Up时，Tb_A＝opt(A)，Tb_B为利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r；

[0024] A3)根据Tb_A和Tb_B，向通用图形处理器(GPGPU)的一个流处理器中分别派发相应数目的核函数A和核函数B的线程块。

[0025] A4)循环执行上述A3)，直到其中的一个核函数(不妨假设A)执行完成，也就是核函数A剩余线程块数目为0；如果最后核函数(不妨假设A)剩余线程块数目小于Tb_A，向流多处理器派发核函数A所有剩余线程块数目；

[0026] A5)此时从两个核函数{A,B}的并发执行变为核函数B单独执行，B单独执行期间，向流多处理器中发射的核函数B的线程块数目为opt(B)，当核函数B剩余线程块数目小于opt(B)时，向流多处理器派发核函数B所有剩余线程块数目；直到核函数B执行完成。

[0027] 当处理多个核函数的线程块派发时，将这些核函数看作一个任务池，先从这些核函数中选择两个核函数作为核函数{A,B}，按照上述步骤A1～A4进行线程块派发，即向通用图形处理器(GPGPU)的一个流处理器中分别派发相应数目的核函数A和核函数B的线程块。当其中的一个核函数(不妨假设为A)执行完成之后，从未执行的核函数中选择一个与未完成的核函数(B)重新组合成{A,B}，继续按照上述步骤A1～A4进行线程块派发。当任务池没有未执行的任务，即多任务只剩下未完成的核函数(B)时，此时向流多处理器中发射的核函数B的线程块数目为opt(B)，当核函数B剩余线程块数目小于opt(B)时，向流多处理器派发核函数B所有剩余线程块数目；直到核函数B执行完成。

[0028] 本发明还提供一种一级数据缓存旁路方法，该方法在上述步骤A3)之后、A4)之前执行。首先通过动态方法来确定旁路哪一个核函数的线程块，根据相应核函数的旁路的线程块的数目进行旁路，以达到减轻一级数据缓存的压力、进一步提高性能的目的；该方法执行如下操作：

[0029] B1)对于两个核函数的组合{A,B}，设定By_A和By_B分别表示旁路的核函数A的线程块的数目和旁路的核函数B的线程块的数目；By_A和By_B的范围是$0\leqslant By_A\leqslant Tb_A$，$0\leqslant By_B\leqslant Tb_B$。只选择其中一个核函数的线程块进行旁路操作；当选择核函数A的线程块进行旁路操作时，By_A不等于0，By_B等于0；当选择核函数B的线程块进行旁路操作时，By_B不等于0，By_A等于0；设定$Stall^A_{By_A}$代表当By_A个来自核函数A的线程块旁路一级缓存时在一个抽样周期里流多处理

器的空闲时钟总数;设定$Stall^{B}_{By_B}$代表当By_B个来自核函数B的线程块旁路一级缓存时,在一个抽样周期里流多处理器的空闲时钟总数;设定$Stall_{none}$代表没有任何线程块旁路一级缓存时,在一个抽样周期里流多处理器的空闲时钟总数;

[0030] 设定By_A和By_B的初始值均为0,$By_A=By_B=0$;在一个抽样周期之后,得到$Stall_{none}$;将一个抽样周期作为Tb_A个核函数A的线程块和Tb_B个核函数B的线程块的生命期;Tb_A和Tb_B是在上述步骤A22中根据核函数分类计算得到的向一个流处理器中分别派发的核函数A和核函数B的线程块数目;

[0031] B2)分别针对两个核函数进行旁路操作,分别得到相应核函数的线程块旁路一级缓存时在一个抽样周期里流多处理器的空闲时钟总数:

[0032] 针对核函数A进行旁路操作,即设定$By_A=1$,$By_B=0$;在一个抽样周期之后,得到$Stall^{A}_{By_A}$;

[0033] 在抽样周期中,By_A是动态变化的,By_A的范围是$0\leqslant By_A\leqslant Tb_A$;

[0034] 针对核函数B进行旁路操作,即设定$By_B=1$,$By_A=0$;在一个抽样周期之后,得到$Stall^{B}_{By_B}$;

[0035] 在抽样周期中,By_B是动态变化,By_B的范围是$0\leqslant By_B\leqslant Tb_B$;

[0036] B3)比较$Stall_{none}$、$Stall^{A}_{By_A}$和$Stall^{B}_{By_B}$:当$Stall_{none}$是最小值时,不旁路任一核函数;当$Stall^{A}_{By_A}$是最小值时,选择核函数A进行旁路操作,设置$By_A=1$;当$Stall^{B}_{By_B}$是最小值时,选择核函数B进行旁路操作,设置$By_B=1$;

[0037] B4)对选择的核函数进行旁路,该核函数表示为核函数C(可能为核函数A或核函数B):选择By_C+1个核函数C的线程块旁路一级数据缓存,经过一个抽样周期之后,收集得到$Stall^{C}_{By_C+1}$;当$Stall^{C}_{By_C+1}<Stall^{C}_{By_C}$时,$By_C$增加1,继续执行步骤B4);否则,$By_C$减少1,结束操作;当$By_C$达到了上限$Tb_C$,停止更新$By_C$,结束操作。

[0038] 考虑多个核函数并发执行的情况,由于线程块派发引擎方法通过每次选择两个核函数{A,B}来完成执行,因此上述针对两个核函数{A,B}的一级数据缓存旁路技术可以被直接应用到多个核函数并发执行的场景中。

[0039] 与现有技术相比,本发明的有益效果是:

[0040] 本发明提供一种通用图形处理器多任务并发执行的任务派发方法,包括线程块派发引擎方法,还可进一步执行一级数据缓存旁路方法;其中,线程块派发引擎方法通过将不同核函数的线程块派发到同一个流多处理器中,克服了由于单一核函数对于计算资源和存储资源的不均衡利用而导致的资源利用率低的情况,可以提高通用图形处理器中每个流多处理器资源的利用率,从而提升系统性能和能效比。由于将不同核函数的线程块派发到同一个流多处理器中会导致一级数据缓存污染从而影响性能,为解决这一问题,可通过一级数据缓存旁路方法旁路一部分线程块,以减轻一级数据缓存的压力,从而达到进一步提高计算性能的目的。

附图说明

[0041] 图1是本发明通过线程块派发引擎方法将不同核函数的线程块派发到同一个流多处理器中的示意图；

[0042] 其中，(a)为包含多个线程块的不同核函数；(b)为同一个流多处理器中包含不同核函数的线程块。

[0043] 图2是本发明提供的线程块派发引擎方法的流程框图。

[0044] 图3是本发明实施例中通过一级数据缓存旁路方法旁路核函数的一部分线程块，以减轻一级数据缓存的压力的示意图；

[0045] 其中，(a)为包含多个线程块的流多处理器；(b)为线程块访问缓存的两种模式(箭头直接指向二级缓存表示该线程块旁路了一级缓存;箭头先指向一级缓存然后再指向二级缓存表示该线程块访问了一级缓存)。

[0046] 图4是本发明提供的一级数据缓存旁路方法步骤流程框图。

[0047] 图5是本发明实施例中采用一级数据缓存旁路方法的步骤流程示意图。

具体实施方式

[0048] 下面结合附图，通过实施例进一步描述本发明，但不以任何方式限制本发明的范围。

[0049] 图1是本发明通过线程块派发引擎方法将不同核函数的线程块派发到同一个流多处理器中的示意图。如图1所示，(a)中的矩形从上到下分别为包含多个线程块的不同核函数：核函数A和核函数B;其中白色方块代表核函数A的线程块，黑色方块代表核函数B的线程块；(b)为同一个流多处理器中包含不同核函数的线程块;图中线程块派发引擎左边的矩形从上到下分别代表核函数A和核函数B，其中白色方块代表核函数A的线程块，黑色方块代表核函数B的线程块；(b)为同一个流多处理器中包含不同核函数的线程块；(b)中的圆形代表流多处理器，流多处理器中的方块代表派发到该流多处理器上的线程块。线程块派发引擎负责将不同核函数的线程块派发到同一个流多处理器中。线程块派发引擎首先计算流多处理器中同时可以运行的每个核函数的线程块数目；当流多处理器中的一个线程块完成之后，线程块派发引擎会向流多处理器派发一个来自相同核函数的新的线程块，直到核函数中的所有线程块都完成执行。

[0050] 本实施例针对两个核函数的组合{A,B}，通过线程块派发引擎将核函数组合{A,B}的线程块派发到同一个流多处理器中。其中，设定Tb_A和Tb_B分别表示线程块派发引擎向一个流处理器中分别派发的核函数A和核函数B的线程块数目。本发明提供的线程块派发引擎方法首先将核函数进行分类，然后根据分类信息计算Tb_A和Tb_B;具体包括如下步骤：

[0051] A1)首先，将核函数进行分类。设定$Type_A$和$Type_B$分别表示核函数A和核函数B的种类，依据每个核函数单独运行时性能最佳的线程块数目opt(Kernel)与一个流多处理器最多可以容纳的来自核函数的线程块数目max(Kernel)的大小关系，对核函数进行分类;分类过程如下：

[0052] A11)单独执行每个核函数多次，每次线程块派发引擎向流多处理器中派发不同数目的线程块。

[0053] A12)对A11)中所有情况，比较它们的执行时间(性能)，取执行时间最短的情况，将这种情况中每个流多处理器上的线程块数目定义为opt(Kernel)。

[0054] A13)如果A12)中opt(Kernel)＝1,我们将该核函数分类为Type Down;如果opt(Kernel)＝max(Kernel)(其中max(Kernel)中指的是一个流多处理器最多可以容纳的来自核函数的线程块数目),我们将该核函数分类为Type Up;其他情况,我们将该核函数分类为Type Optimal。

[0055] A2)根据分类信息计算得到Tb_A和Tb_B;具体步骤如下所示:

[0056] A21)核函数的类型包括Type Down、Type Up和Type Optimal;对于两个并发执行的核函数组合{A,B},两个核函数A、B的类型分别设为$Type_A$和$Type_B$;其中有一个核函数的类型属于Type Down或者Type Optimal;本发明不适用于两个核函数都属于Type Up类型的情况。

[0057] A22)根据核函数A、B的类型,分别计算得到Tb_A和Tb_B;

[0058] 当核函数A的类型为Down,核函数B的类型为Up时,Tb_A＝opt(A),Tb_B等于利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r(以下伪代码中的1-4行);

[0059] 当核函数A的类型为Down,核函数B的类型为Optimal时,Tb_A＝opt(A),Tb_B等于利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r与opt(B)中的较小者(以下伪代码中的5-8行);

[0060] 当2个核函数的类型都为Down时,Tb_A＝opt(A),Tb_B＝opt(B)(以下伪代码中的9-11行);

[0061] 当2个核函数的类型都为Optimal时,Tb_A＝opt(A),Tb_B为利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r与opt(B)中的较小者(以下伪代码中的12-15行);

[0062] 当核函数A的类型为Optimal,核函数B的类型为Up时,Tb_A＝opt(A),Tb_B为利用剩下的计算资源Compute Remain(A)所能派发的最大线程块数目r(以下伪代码中的16-19行)。

[0063] 以下是不同情况下计算得到Tb_A和Tb_B的方法的伪代码:

[0064]

算法: 线程块派发引擎技术

输入: Kernel A and Kernel B

输出: Tb_A and Tb_B

```
1  if Type_A = Down ^ Type_B = Up then
2  |    Tb_A = opt(A);
3  |    r = Compute Remain(B);
4  |    Tb_B = r;
5  else if Type_A = Down ^ Type_B = Optimal then
6  |    Tb_A = opt(A);
7  |    r = Compute Remain(B);
8  |    Tb_B = min(r,  opt(B));
9  else if Type_A = Down ^ Type_B = Down then
10 |    Tb_A = opt(A);
11 |    Tb_B = opt(B);
12 else if Type_A = Optimal ^ Type_B = Optimal then
13 |    Tb_A = opt(A);
14 |    r = Compute Remain(B);
15 |    Tb_B = min(r,  opt(B));
16 else if Type_A = Optimal ^ Type_B = Up then
17 |    Tb_A = opt(A);
18 |    r = Compute Remain(B);
19 |    Tb_B = r;
```

[0065] 其中，算法的输入为2个核函数的组合{A,B}，以及它们的类型$Type_A$和$Type_B$；在不同情况下分别计算得到线程块派发引擎向一个流处理器中分别派发的核函数A和核函数B的线程块数目Tb_A和Tb_B。

[0066] A3)根据Tb_A和Tb_B，向通用图形处理器(GPGPU)的一个流处理器中分别派发相应数目的核函数A和核函数B的线程块。

[0067] 上述线程块派发引擎方法将不同核函数的线程块派发到通用图形处理器(GPGPU)的同一个流多处理器中进行处理。不同核函数的线程块，由于其访问数据集不同，会对一级数据缓存造成严重的污染和竞争。本发明还提供一种动态一级数据缓存旁路方法，通过选择一些线程块旁路一级数据缓存，从而减轻一级数据缓存的压力。

[0068] 图3是本发明实施例中通过一级数据缓存旁路方法旁路核函数的一部分线程块，以减轻一级数据缓存的压力的示意图；其中，(a)为包含多个线程块的流多处理器；(b)为线程块访问缓存的两种模式(箭头直接指向二级缓存表示该线程块旁路了一级缓存；箭头先指向一级缓存然后再指向二级缓存表示该线程块访问了一级缓存)；图3中有箭头直接指向

二级数据缓存的线程块表示该线程块旁路了一级数据缓存。

[0069] 图4是本发明提供的一级数据缓存旁路方法的流程框图。对于两个核函数的组合{A,B},By_A和By_B分别表示旁路的核函数A和核函数B的线程块的数目。本发明提供的动态一级数据缓存旁路方法选择一个核函数的线程块进行旁路操作,因此,By_A或者By_B会等于0。设定$Stall^A_{By_A}$代表当By_A个来自核函数A的线程块旁路一级缓存时在一个抽样周期里流多处理器的空闲时钟总数;设定$Stall^B_{By_B}$代表当By_B个来自核函数B的线程块旁路一级缓存时,在一个抽样周期里流多处理器的空闲时钟总数。流多处理器空闲适中的数目和性能成反相关,空闲越多,性能越差。本实施例中,选用流多处理器的空闲时钟数作为监测变量。设定$Stall_{none}$代表没有任何线程块旁路一级缓存时,在一个抽样周期里流多处理器的空闲时钟总数。本发明提供的一级数据缓存旁路方法,首先通过动态方法来确定旁路哪一个核函数的线程块,再得到旁路的线程块的数目,根据得到的相应核函数的旁路的线程块的数目进行旁路,以达到进一步提高性能的目的;该方法包括如下步骤:

[0070] B1)设定By_A和By_B的初始值均为0,$By_A=By_B=0$;在一个抽样周期之后,得到$Stall_{none}$;将一个抽样周期作为Tb_A个核函数A的线程块和Tb_B个核函数B的线程块的生命期;

[0071] B2)针对核函数A进行旁路操作,即设定$By_A=1$,$By_B=0$;在一个抽样周期之后,得到$Stall^A_{By_A}$;

[0072] B3)针对核函数B进行旁路操作,即设定$By_B=1$,$By_A=0$;在一个抽样周期之后,得到$Stall^B_{By_B}$;

[0073] 然后,比较$Stall_{none}$、$Stall^A_{By_A}$和$Stall^B_{By_B}$:如果$Stall_{none}$是最小值,那么不旁路任一核函数;如果$Stall^A_{By_A}$是最小值,那么我们选择核函数A进行旁路操作,设置$By_A=1$;对于核函数B,亦然。

[0074] B4)假设我们选择核函数A进行旁路;经过一个抽样周期之后,我们收集$Stall^A_{By_A+1}$;如果$Stall^A_{By_A+1} < Stall^A_{By_A}$,那么$By_A$增加1,继续步骤B4);否则,$By_A$减少1,此时结束操作;如果,$By_A$达到了上限$Tb_A$,将停止更新$By_A$,结束操作。

[0075] 下面通过实施例对本发明作进一步说明。

[0076] 本实施例针对两个核函数spmv和backprop,这两个核函数分别来自于parboil标准测试集和rodinia标准测试集。由于每一个流多处理器的资源有限,而核函数的每个线程块都会占据一定资源,不同核函数由于其计算任务和类型的不同所需资源情况不同。对于spmv来讲,一个流多处理器的资源最多足够容纳8个spmv的线程块。对于backprop来讲,一个流多处理器的资源最多足够容纳6个backprop的线程块。

[0077] 首先,通过线程块引擎派发方法向流多处理器派发不同数目的线程块:第一步,对spmv和backprop进行分类。第二步,根据分类信息,计算线程块数目;然后向流多处理器派发相应数目的spmv和backprop线程块。具体操作如下:

[0078] 对于spmv,我们单独执行spmv多次,每次向流多处理器派发不同数目的线程块。我们得到当向流多处理器派发的线程块数目为3时,spmv的执行时间最短,性能最高,因此opt(spmv)=3。流多处理器上同时可并发执行的最大数目的线程块max(spmv)=8。因此spmv属

于Type Optimal，并且opt(spmv)＝3。对于backprop，执行类似的操作，我们得到当向流多处理器派发线程块数目为6时，backprop的执行时间最短，性能最高，因此opt(backprop)＝6，流多处理器同时可并发执行的最大数目的线程块max(backprop)＝6。因此，backprop属于Type Up，并且opt(backprop)＝6。

[0079] 根据上述不同情况下计算得到Tb_A和Tb_B的方法的伪代码，spmv和backprop分别对应为核函数A和核函数B。spmv的线程块数目为3。此时，用流多处理器剩下的资源最多可以派发2个backprop的线程块。因此，Tb_{spmv}＝3，$Tb_{backprop}$＝2。

[0080] 进一步地，可通过一级缓存旁路方法(策略)对多任务并发执行进行管理。对于spmv和backprop，由线程块派发引擎方法得到Tb_{spmv}＝3，$Tb_{backprop}$＝2。如图5所示，图中线程块标1表示该线程块会旁路一级数据缓存，标0表示该线程块不旁路一级数据缓存。灰色(线程)块为核函数A的线程块，白色(线程)块为核函数B的线程块。从t1开始第一个抽样周期，在t2时刻，已经有至少3个spmv的线程块和2个backprop的线程块执行完成，该时刻就是第一个抽样周期的结束时刻，也是上述步骤B1的完成时刻。从t2时刻开始，只选择spmv的一个线程块旁路一级数据缓存，在t3时刻，第二个抽样周期结束，即上述步骤B2的完成时刻。从t3时刻开始，只选择backprop的一个线程块旁路一级数据缓存，在t4时刻，第三个抽样周期结束，也就是上述步骤B3的完成时刻。此时，通过对比$Stall_{none}$，$Stall_{By_{spmv}}^{spmv}$和$Stall_{By_{backprop}}^{backprop}$，得到$Stall_{By_{backprop}}^{backprop}$最小，因此选择对backprop的线程块进行旁路操作。从t4时刻，我们经过两个周期t4到t5，以及t5到t6，得到By_{spmv}＝By_A＝0，$By_{backprop}$＝By_B＝1。

[0081] 上述一级数据缓存旁路方法，首先通过动态方法来确定旁路哪一个核函数的线程块，再得到旁路的线程块的数目，根据得到的相应核函数的旁路的线程块的数目进行旁路，以达到进一步提高性能的目的。

[0082] 需要注意的是，公布实施例的目的在于帮助进一步理解本发明，但是本领域的技术人员可以理解：在不脱离本发明及所附权利要求的精神和范围内，各种替换和修改都是可能的。因此，本发明不应局限于实施例所公开的内容，本发明要求保护的范围以权利要求书界定的范围为准。

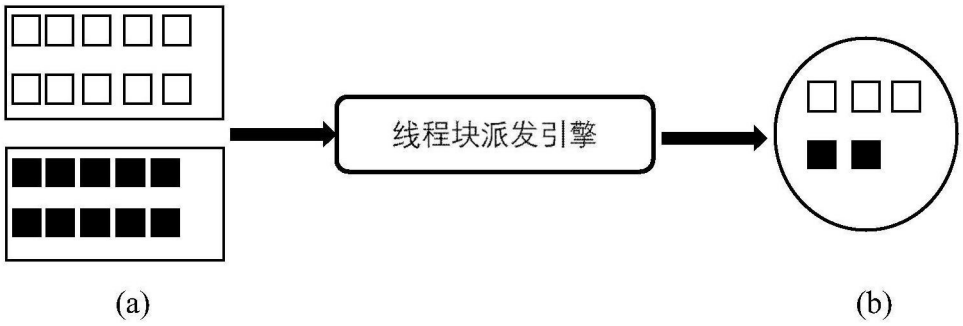


图1

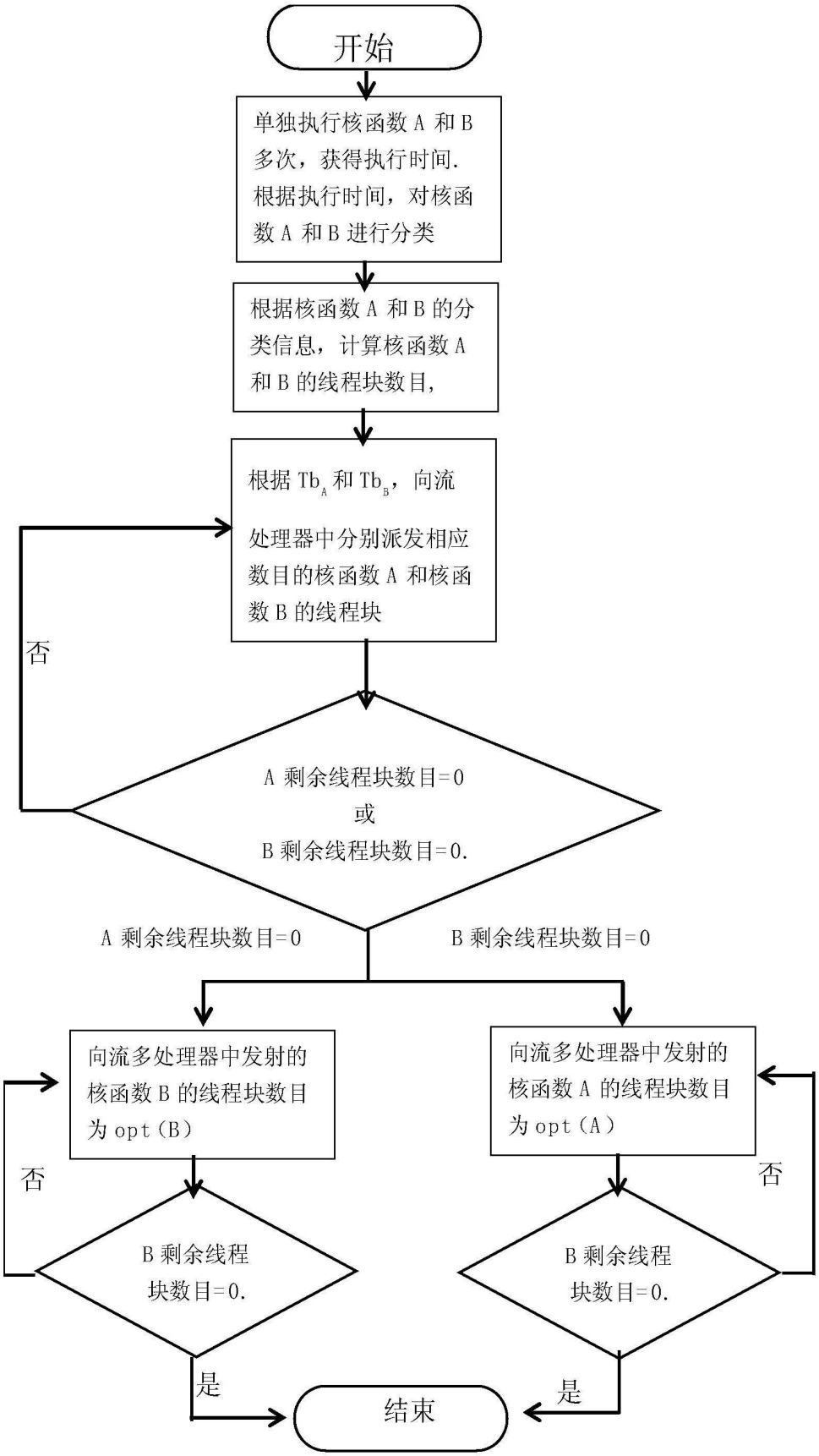
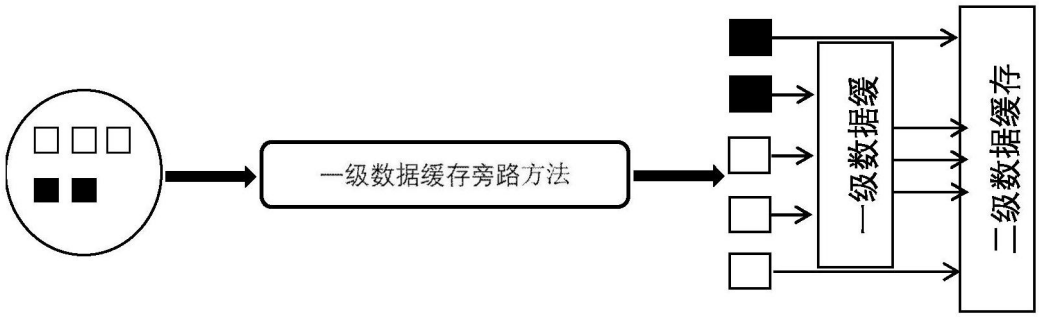

开始
单独执行核函数 A 和 B 多次，获得执行时间. 根据执行时间，对核函数 A 和 B 进行分类
根据核函数 A 和 B 的分类信息，计算核函数 A 和 B 的线程块数目,
根据 Tb_A 和 Tb_B，向流处理器中分别派发相应数目的核函数 A 和核函数 B 的线程块
否
A 剩余线程块数目=0 或 B 剩余线程块数目=0.
A 剩余线程块数目=0
B 剩余线程块数目=0
向流多处理器中发射的核函数 B 的线程块数目为 opt（B）
向流多处理器中发射的核函数 A 的线程块数目为 opt（A）
否
否
B 剩余线程块数目=0.
B 剩余线程块数目=0.
是
是
结束

图2



(a) (b)

图3

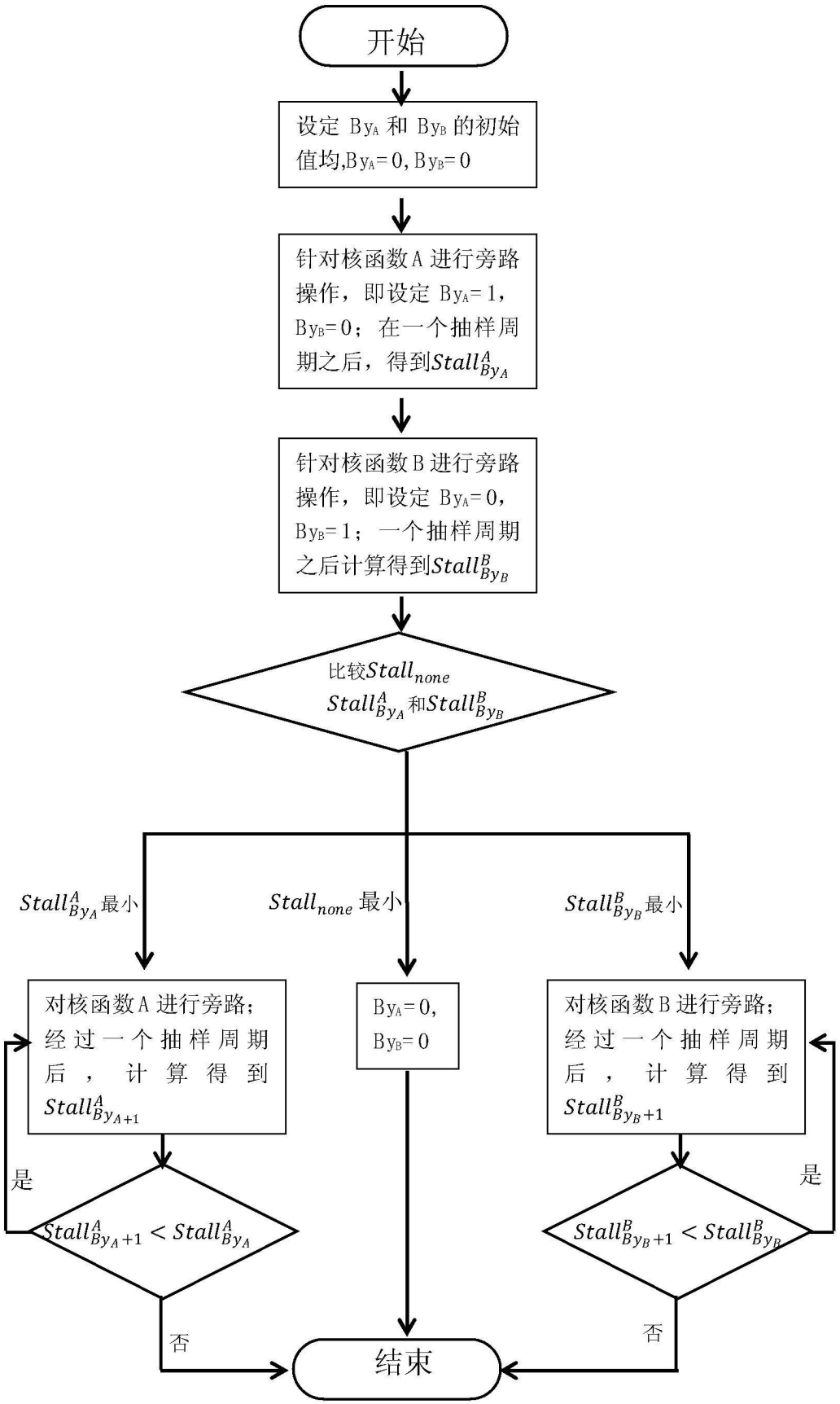

开始
设定 By_A 和 By_B 的初始值均,$By_A=0$, $By_B=0$
针对核函数A进行旁路操作，即设定 $By_A=1$，$By_B=0$；在一个抽样周期之后，得到$Stall^A_{By_A}$
针对核函数B进行旁路操作，即设定 $By_A=0$，$By_B=1$；一个抽样周期之后计算得到$Stall^B_{By_B}$
比较$Stall_{none}$ $Stall^A_{By_A}$和$Stall^B_{By_B}$
$Stall^A_{By_A}$最小
$Stall_{none}$最小
$Stall^B_{By_B}$最小
对核函数A进行旁路；经过一个抽样周期后，计算得到$Stall^A_{By_A+1}$
$By_A=0$, $By_B=0$
对核函数B进行旁路；经过一个抽样周期后，计算得到$Stall^B_{By_B+1}$
是
$Stall^A_{By_A+1} < Stall^A_{By_A}$
否
是
$Stall^B_{By_B+1} < Stall^B_{By_B}$
否
结束

图4

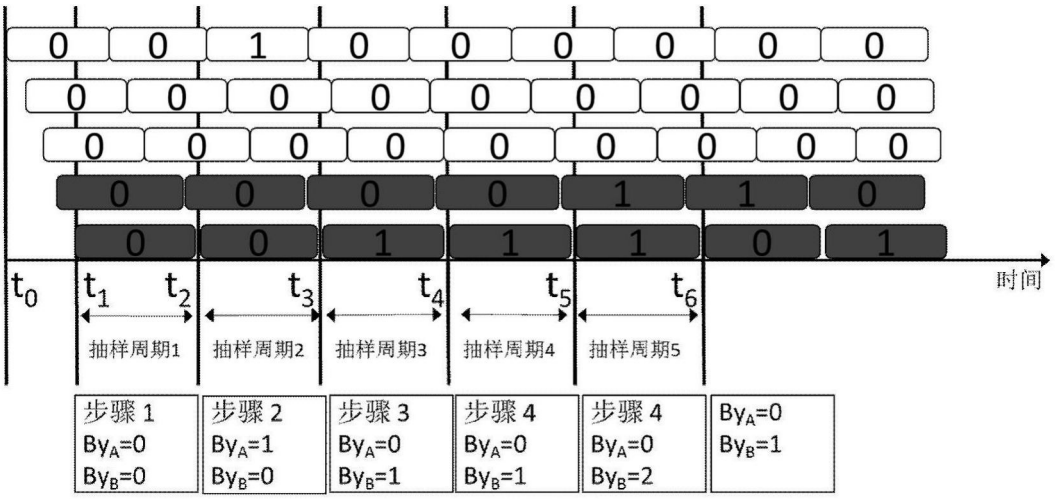

0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0
0 0 1 1 1 0 1
t0 t1 t2 t3 t4 t5 t6
时间
抽样周期1
抽样周期2
抽样周期3
抽样周期4
抽样周期5
步骤 1 ByA=0 ByB=0
步骤 2 ByA=1 ByB=0
步骤 3 ByA=0 ByB=1
步骤 4 ByA=0 ByB=1
步骤 4 ByA=0 ByB=2
ByA=0 ByB=1

图5