第 39 卷 第 9 期 计 算 机 学 报 Vol. 39 No. 9
2016 年 9 月 CHINESE JOURNAL OF COMPUTERS Sept. 2016

通用图形处理器线程调度优化方法研究综述

何炎祥[1),2)] 张 军[1),3)] 沈凡凡[1)] 江 南[1),4)] 李清安[1),2)] 刘子骏[1)]

[1)](武汉大学计算机学院 武汉 430072)

[2)](武汉大学软件工程国家重点实验室 武汉 430072)

[3)](东华理工大学软件学院 南昌 330013)

[4)](湖北工业大学计算机学院 武汉 430068)

摘 要 随着通用图形处理器(GPGPU)并行计算能力的日益增强,其应用范围越来越广.然而由于不规则计算任务使得通用图形处理器资源难以得到充分利用,其性能并未达到最大化.在论述 GPGPU 微体系结构的基础上,该文重点讨论了多种针对 GPGPU 性能提升的线程调度优化方法,主要从针对优化分支转移提升线程级并行度、针对访存效率的提升和针对标量指令执行以提升资源利用率三方面的线程调度优化方法进行了分析和比较.另外,由于功耗问题已成为制约 GPGPU 发展的主要因素之一,该文还分析了影响 GPGPU 功耗效率的主要原因及目前提高 GPGPU 功耗效率的主要低功耗技术,并对目前针对 GPGPU 功耗优化的典型线程调度优化方法进行了分析比较.最后,该文指出了未来线程调度优化方法需要进一步探讨的一些问题.

关键词 通用图形处理器;线程调度优化;性能;功耗

中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2016.01733

Thread Scheduling Optimization of General Purpose Graphics Processing Unit: A Survey

HE Yan-Xiang[1),2)] ZHANG Jun[1),3)] SHEN Fan-Fan[1)]
JIANG Nan[1),4)] LI Qing-An[1),2)] LIU Zi-Jun[1)]

[1)](*Computer School, Wuhan University, Wuhan* 430072)

[2)](*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072)

[3)](*School of Software, East China University of Technology, Nanchang* 330013)

[4)](*Computer School, Hubei University of Technology, Wuhan* 430068)

Abstract Due to their massive parallel computational power, General Purpose Graphics Processing Units (GPGPUs) have become an increasingly popular platform for executing general purpose applications. However the existed irregular workloads make the resource of GPU underutilized and its performance not maximized. A variety of thread scheduling optimization strategies on the performance enhancement of GPGPU are mainly discussed based on the microarchitecture of GPGPU. Specifically, the thread scheduling optimization strategies on branch divergence optimization to improve thread level parallelism, memory access enhancement and scalar instruction execution to increase the resource utilization are analyzed and compared. In addition, the primary causes affecting the power are analyzed and the major methods improving the efficiency of power as well as typical thread scheduling strategies on power optimization are compared as the issue of

收稿日期:2014-12-11;在线出版日期:2015-07-23.本课题得到国家自然科学基金(61170022,61373039,61462004)资助.**何炎祥**,男,1952 年生,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为可信软件、分布并行处理和高性能计算. E-mail: yxhe@whu.edu.cn. **张 军**,男,1978 年生,博士研究生,副教授,中国计算机学会(CCF)会员,主要研究方向为高性能计算、计算机体系结构. **沈凡凡**,男,1987 年生,博士研究生,主要研究方向为可信软件、计算机体系结构、存储系统. **江 南**,女,1976 年生,博士研究生,讲师,中国计算机学会(CCF)会员,主要研究方向为可信软件、可信编译. **李清安**,男,1986 年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为编译优化、嵌入式系统. **刘子骏**,男,1990 年生,硕士研究生,主要研究方向为高性能计算、计算机体系结构.

power has become one of the bottlenecks limiting the development of GPGPU. Finally, it summarizes some research interests need to be discussed and studied in thread scheduling optimization in future.

Keywords general purpose graphics processing unit; thread scheduling optimization; performance; power

1 引 言

图形处理器(Graphics Processing Unit,GPU)拥有强大的并行计算能力,因此,它已经成为目前面向高吞吐量、满足高性能计算需求的主流计算加速部件.GPU 最初用于 3D 图形加速处理和计算,随着处理图形分辨率和图形处理速度要求的日益提高,它的并行计算能力得到了快速发展.2006 年,NVIDIA 公司推出了 GeForce 8800,它首次使用统一的渲染部件代替了各种可编程部件,解决了可编程 GPU 片上负载均衡的问题,成为当今通用图形处理器(General Purpose Graphics Processing Unit,GPGPU)的雏形[1].而后,流式图形处理器的出现为图形处理器在通用计算领域的应用奠定了基础.随着 GPGPU 并行计算体系结构的快速发展,它在面向高吞吐量、高性能计算通用计算领域的应用越来越广泛.

GPGPU 展现出强大的并行计算能力、高吞吐量和优秀的能效比,使它成为了目前构建高性能平台的首选计算加速部件.在 2010 和 2013 年的全球超级计算机 TOP500 排行榜中,由国防科大主导研制的天河一号和天河二号均排在榜首[2-3].它们强大的浮点运算能力来自于其内部采用了大量的图形加速处理部件.另外,根据 2014 年上半年全球超级计算机 TOP500 排行榜,前 500 的超级计算机中有 62 台超级计算机采用了图形加速处理部件,比 2013 年增长了 9 台①.

近年来,VLSI 技术及制造工艺的快速发展使得 GPGPU 的并行计算能力有了很大提升.然而在体系结构和编程模型相同的情况下,GPGPU 性能的高低主要受访存效率和计算任务并行度的影响.对于访存和计算模式不规则的应用程序,它的性能往往无法达到最大化.这种不规则性体现在:(1)应用程序中存在分支转移,使得 GPGPU 在同一时刻的处理任务分散,降低了任务处理的并行度;(2)访存的数据分布离散,使得并行处理任务的数据访问延时增加.

GPGPU 采用 SIMD(Single Instruction Multiple Data)执行模式,即一条指令同时处理多个不同的数据[4-6].为了获得计算的高度并行,主流的图形处理器均派生出多个不同的线程对同一指令的不同数据进行并行处理,这种模式又称为 SIMT(Single Instruction Multiple Thread)执行模式[7-8].SIMT 执行模式获得的并行处理属于线程级并行(Thread Level Parallelism,TLP),它与指令级并行(Instruction Level Parallelism,ILP)和数据级并行(Data Level Parallelism,DLP)并为影响处理器并行计算能力的 3 个不同的并行级别[9-10].GPGPU 通过支持大量线程的同时执行获得很高的 TLP,并通过一定的线程调度策略保证执行任务的高效并行处理.然而由于不规则计算和不规则访存模式的存在,GPGPU 中不同的线程调度策略对各种并行计算任务的执行效率会产生不同的影响.早在 2007 年,国外就有学者展开了针对 GPGPU 性能提升的线程调度优化研究[11].另外,随着功耗问题日渐成为制约 GPGPU 发展的重要瓶颈,也有部分线程调度策略对 GPGPU 的功耗优化进行了考虑.本文重点关注了近几年来国内外关于提升 GPGPU 性能和功耗效率的线程调度优化方法的关键技术和相关问题.

本文第 2 节阐述主流 GPGPU 的体系结构和相关概念,并阐述 GPGPU 中线程层次结构和线程调度层次结构;第 3 节重点分析比较多种针对 GPGPU 性能提升和功耗优化的线程调度优化方法;最后提出 GPGPU 线程调度优化面临的挑战和进一步研究的方向.

2 GPGPU 体系结构及相关概念

NVIDIA、AMD 和 Intel 是目前主流通用图形处理器的三大生产厂商,它们生产的 GPGPU 在宏观结构上相似,但在微观结构上存在较大差异.为了论述的统一,本文以 NVIDIA 公司生产的通用图形

① 世界超级电脑 2014 年上半年排名:中国天河二号三度夺冠,http://technews.cn/2014/06/24/top500-201406/

处理器体系结构为基准进行分析论述，后续的相关概念均以 NVIDIA 公司提出的 GPGPU 相关术语为参考.

2.1　GPGPU 线程层次结构及线程调度层次结构

为了便于线程管理，GPGPU 对线程的组织通常分为三级：即线程组级（Thread Group，TG）、线程块级（Thread Block，TB）、线程网格级（Grid）[12]. 线程组（被称为 warp）是线程并发执行的基本单位，通常由 32 个线程构成. 它一般按锁步的方式执行指令[6,13-14]，即线程组中所有线程执行完当前指令后方可执行下一条指令. 线程块由多个线程组构成，是发射到 GPGPU 核上执行的基本单位. 多个线程块又构成了线程网格. GPGPU 采用三级线程层次结构，主要是为了方便在线程和线程处理的数据元素之间建立映射关系. 图 1 展示了 GPGPU 的线程层次结构[12].

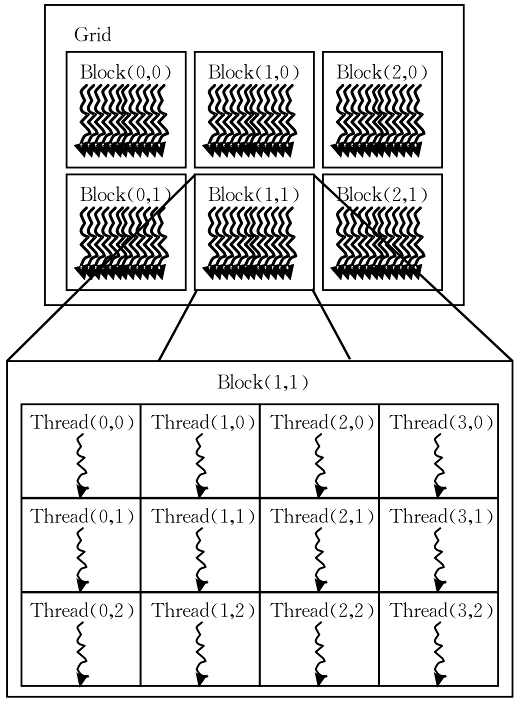

图 1　GPGPU 线程层次结构

线程调度是将计算任务分配到不同的处理单元并按照一定顺序执行的过程. GPGPU 中的线程调度可以分为 3 个层次：kernel 级、线程块级和线程组级. 本文对 GPGPU 线程调度层次结构的描述主要基于文献[12,15-16]. kernel 是在 GPGPU 核上能被多个线程并发执行的函数，它需要在 CPU 端启动并发射到 GPGPU 的流式多处理器（Stream Multi-processor，SM）上执行. NVIDIA 从 Fermi 架构的 GPGPU 开始支持同时发射多个 kernel，之前的 GPGPU 只允许 kernel 的顺序发射. 当 kernel 被发射之后，线程调度器将派生出来的线程块按一定的顺序发送到可用的 SM 上执行，通常情况下按轮转的方式发送. 可以被发送到每个 SM 的线程块数依赖于每个 SM 上的资源使用情况，且不能超过物理规定的上限. 被发送到某个 SM 上的线程组将按照一定的调度顺序在执行通道上执行. 当线程组在执行过程中因访存失效等原因被阻塞时，该线程组将被插入到线程组阻塞队列，然后调度其他准备就绪的线程组继续执行.

目前对线程调度优化的研究主要集中在线程块级和线程组级，对 kernel 级的调度优化研究相对较少.

2.2　GPGPU 宏观体系结构

GPGPU 的宏观体系结构如图 2 所示①，它由多个 SM 组成，每个 SM 又被称为核. 为了有效地控制逻辑控制单元的数量和芯片面积，通常将多个 SM 组织为一个图形处理核组（Graphics Processing Cluster，GPC）. 每个 SM 包含了众多的处理单元（Processing-Unit，PU），每个处理单元负责最终的指令执行.

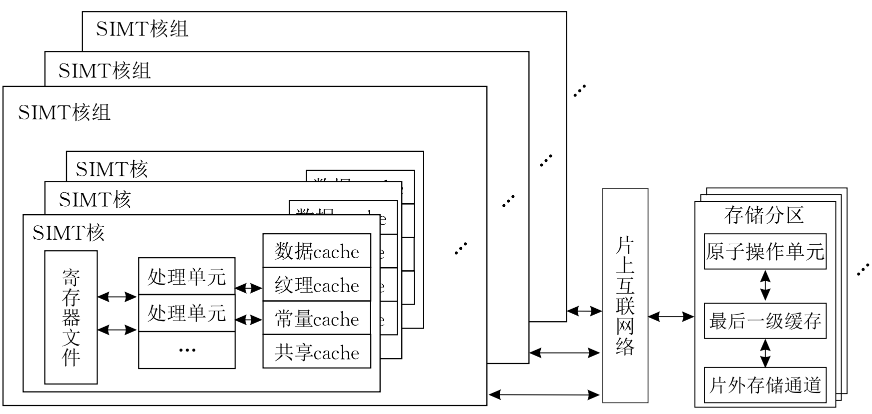

图 2　GPGPU 宏观体系结构

① GPGPU-Sim 3. x Manual. http://gpgpu-sim. org/manual/index. php5/GPGPU-Sim_3. x_Manual

为了提高数据访问效率,GPGPU 采用多级存储层次结构.常见的 GPGPU 存储层次结构分为寄存器级、高速缓存(cache)级和片外存储器级三级.其中高速缓存级又可以分为多级,目前常见的 GPGPU 采用两级高速缓存结构.每个 SM 中包含数量众多的寄存器,称之为寄存器文件,负责直接提供 PU 处理任务所需的数据.通常情况下,寄存器和一级 cache 放置在 SM 内部,二级 cache 和片外主存储器则通过片上互联网络和所有的 SM 相连.为了提高二级 cache 的访问效率,二级 cache 通常被划分为多个分区,每个分区通过片外存储通道与片外主存储器相连.另外,GPGPU 通过片上互联网络将多个 GPC 和片外存储控制器连接起来,主要负责 GPC 与片外存储之间的数据通信.

2.3 SM 微体系结构

由于本文重点讨论 GPGPU 线程调度方法,在此仅重点介绍了 SM 的微体系结构.本文对 SM 微体系结构的描述基于文献[1,17-18].

SM 微体系结构如图 3 所示.整个 SIMD 流水线[19]可以分为前端和后端,前端主要包括取指部件、译码部件、记分板及指令发射器等部件,其余部件均属于后端[20].

取指部件根据每个线程组对应的 PC 值负责从片外存储将指令取到指令 cache 中.指令 cache 中的指令则依次送到指令译码部件,经过译码后的指令被送到指令 buffer 中.指令 buffer 为每个线程组缓冲译码后的指令,便于指令的快速执行,并为每个线程组分配一定数量的专用指令槽.图 3 中的记分板部件用来记录每个线程组当前执行指令的目的寄存器的使用情况,主要用来判断执行指令之间的数据依赖关系.指令发射器按一定的策略选取准备就绪的指令发射.

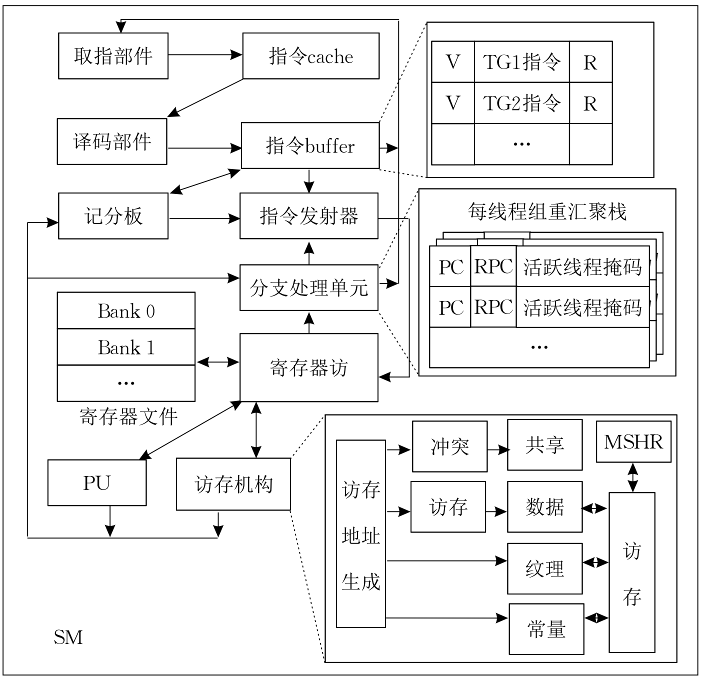

图 3 SM 微体系结构

由指令发射器发射的指令到达 SIMD 流水线后端执行通道后需要通过寄存器访存部件获取执行所需要的操作数.为了满足大量并发线程的同时执行,GPGPU 设置了数量众多的寄存器,并将这些寄存器组织为多个单端口的寄存器块(Bank),以便于多个并发线程同时访问.分支处理单元用于正确处理程序中的分支转移指令,并为每个线程组分配一个栈结构,称之为重汇聚栈.该栈结构由执行指令 PC、分支重汇聚 PC(RPC)和执行指令对应的活跃线程掩码位向量三部分组成.访存部件主要实现数据存取,其中冲突检测部件确保对同一共享 cache 块的多个访问能正确执行;访存合并部件实现对访问同一数据 cache 行的多个访存请求进行合并,以减少访存次数;MSHR(Missing Status Hold Register,失效状态保留寄存器)用于记录访问 cache 失效的请求,并对同一 cache 行的多个访问失效请求进行合并.

3 线程调度优化方法

GPGPU 在越来越多的通用计算领域得到了广泛

应用，然而对于访存和计算模式不规则的应用程序，GPGPU 的性能往往未能达到最大化. 影响 GPGPU 性能达到最大化的原因是多方面的，程序执行过程中出现的线程级并行度降低和访存效率下降是其中的两个主要原因. 其中线程级并行度的降低会使 GPGPU 计算资源不能得到充分利用，它主要由程序中存在的分支转移指令导致. 访存效率下降则会使计算过程延时增加，访存离散和资源竞争导致的数据局部性破坏是其产生的主要原因. 另外，致使 GPGPU 性能不能达到最大化的原因还包括由于线程执行速度不一致等因素导致核上资源利用率下降、标量指令冗余执行而导致的资源利用浪费等.

目前的线程调度优化方法通常都以提高程序执行过程中的线程级并行度、访存效率和资源利用率等为主要目标. 另外，由于功耗问题的日渐突出，越来越多的线程调度方法在提升 GPGPU 性能的同时，也考虑了对 GPGPU 产生的功耗进行优化.

3.1　针对分支转移的线程调度优化

分支转移指令在通用应用程序中非常普遍，它主要分为结构化分支转移指令和非结构化分支转移指令两类. 其中结构化分支转移指令一般包括 if 语句和循环语句中的分支转移指令等，而非结构化分支转移指令一般包括 break 指令、continue 指令、goto 指令以及异常跳转指令等[21]. 如果线程组在执行过程中遇到分支转移指令，那么该线程组中的每个线程将根据分支指令的计算结果而选择后续执行的分支路径，这容易出现部分线程执行不同分支路径的情形. 而且由于 SIMT 执行模式只允许某线程组在某一时刻执行一条指令，这会致使不同的分支路径只能串行执行，从而必然降低任务执行的 TLP. 另外，如果同一线程组中执行不同分支路径的线程在执行完相应的分支路径后没有进行重汇聚处理，那么这也会影响到后续规则指令执行的 TLP. 基于栈的重汇聚机制(Post Dominator，PDOM)可以有效地解决由于分支转移带来的性能降低[22-24]，它在分支重汇聚处将属于同一线程组的线程重组为分支前的线程组，以提高后续指令执行的 TLP. 图 4 展示了一个传统的、基于栈的重汇聚机制的分支转移控制流的处理流程示例. 图中左边部分是一个基本的分支转移控制流结构，其中每个字母代表一个基本语句块；每位数字表示一位掩码位，代表对应 SIMD 通道的状态，“1”表示该通道上有活跃线程执行，“0”表示该通道空闲. 图中右边部分表示左边控制流图中的线程在通道中的执行情况，其中黑色箭头表示通道上有活跃线程执行，灰色箭头表示该通道空闲. 该图表明出现分支转移时，SIMD 通道未能满负载运行.

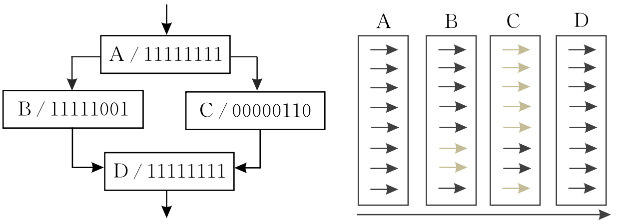

图 4　传统分支转移执行情况

由图 4 分析可以看出分支转移引起性能下降的根本原因在于分支转移降低了 TLP，因此在出现分支转移时，应尽量提高并发执行的线程数. 目前针对分支转移的线程调度优化方法主要有两类：(1) 线程重组机制；(2) 多路并行执行机制.

3.1.1　线程重组调度优化

线程重组机制的基本思想是当遇到分支转移时，将执行相同指令、相同分支路径上的、对应不同 SIMD 执行通道的多个线程重新组合在一起，动态地形成并行执行宽度更大的线程组. 当该分支路径执行结束后，该分支路径上的线程将在重汇聚点等待其他分支路径的线程执行结束，并与之重新组合成原先的线程组继续向前执行. 该机制以提高某一分支路径上并行执行的线程数为目标，进而提高 SIMD 执行通道的利用率. 线程重组又可以分为线程组间的线程重组、线程组内的线程重组及二者结合的线程重组这三类策略.

Cervini[25] 提出了针对包含 SIMD 功能单元的通用微处理器的动态线程重组机制. Fung 等人[11] 则提出了针对 GPGPU 的线程重组策略 DWF (Dynamic Warp Formation). 在 2011 年，Fung 等人[17] 又提出了另一种线程重组调度策略 TBC (Thread Block Compaction). 相对于 DWF，TBC 将线程重组的范围限制在一个线程块中，而且不需要在每个时钟周期进行线程重组检查，仅在出现分支转移指令时进行线程重组，大大地减少了线程重组的次数以及由此产生的开销. 另外，TBC 对基于栈的重汇聚机制进行了扩展，这在一定程度上减少了 SIMD 资源利用率的下降. 实验表明，相对于 DWF，TBC 的平均性能提升了 17%. Narasiman 等人[26] 针对分支路径活跃线程数量不够的问题提出了 LWM (Large Warp Microarchitecture) 策略. 该策略中定义的 large warp 实质上是由多个线程组构成的线程

块，因此与 DWF 和 TBC 一样，LWM 也属于线程组间的线程重组. 但是由于 LWF 是以 large warp 为调度单位，因此相对于 TBC，它能在一定程度上减少因为对不同线程组进行同步而产生的开销. Malits 等人[27]则提出了全局范围内的线程重组策略 ODGS(Oracle Dynamic Global Scheduling)，将线程组间的线程重组从 SM 内扩展至 SM 之间.

虽然线程组间的线程重组可以提高 SIMD 通道的利用率，但是此类方法需要将属于不同线程组的线程重组在一起，这会在一定程度上破坏了线程组内的数据局部性. 另外 TBC 等策略还需要对多个线程组同步，因此会产生一定的开销. 在此基础上，Vaidya 等人[28]提出了线程组内的线程重组机制 BCC(Basic Cycle Compaction)和 SCC(Swizzled Cycle Compaction)，其利用实际 SIMD 物理通道数比标准线程组宽度小的特点，仅将线程组内的线程重组为与实际 SIMD 物理通道宽度相等的线程组. Jin 等人[29]提出的线程重组机制 HWS(Hybrid Warp Size)也是基于这样的思想.

线程组内的线程重组避免了对线程组内的数据局部性产生破坏，也无需对多个线程组进行同步. 然而此类方法也存在着重组率不高、受到实际 SIMD 物理通道宽度的影响等不足，尤其是当 SIMD 硬件宽度与线程组大小相等时，该类策略对系统性能提升的意义不大. 部分研究者将上述两类线程重组调度机制进行了有效地结合. Brunie 等人[30]同时提出了线程组内的线程重组策略 SBI(Simultaneous Branch Interweaving)和线程组间的线程重组策略 SWI(Simultaneous Warp Interweaving)，并将二者进行了有效地结合. 但是与前面所提到的线程组间和线程组内的线程重组不同的是，SBI 和 SWI 可以将执行不同指令的线程重组在一起. Jin 等人在文献[29]中对 DWF 算法进行了改进，允许线程改变执行通道，提升了线程重组率，并将 HWS 和改进的 DWF 进行了有效结合，平均性能比 DWF 提升了 27%.

在进行线程重组时，对应相同 SIMD 通道的线程组合在一起可能会产生冲突，这种情况下进行线程重组并不会提升性能，相反会产生额外开销. 对此，Rhu 等人[31]提出了 CAPRI(Compaction-Adequacy Pridictor)策略，在不同的线程组间进行选择性的重组压缩调度，使得线程重组调度时机的选取更加合理有效，减少了无效的线程重组调度. CAPRI 策略在较大程度上解决了线程重组的冲突问题，但是该问题并没有完全得到避免. 之后 Rhu 等人[32]又提出了 SLP(SIMD Lane Permutation)策略，通过置换线程对应的 SIMD 通道进一步解决了线程重组出现的冲突问题，提高了线程重组策略的有效性. 然而该策略进行的 SIMD 通道置换操作会产生较大开销，而且频繁置换通道产生的开销对系统性能提升的影响是不可忽略的.

线程重组调度优化策略能在出现分支转移的情况下提高不同分支路径上的 SIMD 资源利用率，减少闲置的 SIMD 通道数，它在一定程度上提高了系统性能，但是在提高线程重组的有效性、平衡线程重组的收益和减少开销等方面依然没有得到很好的解决.

表 1 归纳了几种主要的线程重组调度优化策略的特点. 从表中可以看出组间的线程重组能获得较高的 SIMD 资源利用率，而组内的线程重组保持了线程组的数据局部性，其访存失效率更低.

表 1 主要的线程重组调度优化策略特点

线程重组策略	线程重组分类	数据局部性保持特性	线程组间同步特性	受限 SIMD 硬件宽度	重汇聚机制	平均访存失效率	SIMD 资源利用率
DWF[11]	组间	破坏数据局部性	不同步	不受限	延迟重汇聚	较高	较低
TBC[17]	组间	破坏数据局部性	同步	不受限	尽早重汇聚	较高	较高
LWM[26]	组间	破坏数据局部性	同步	不受限	及时重汇聚	较高	较高
SCC+BCC[28]	组内	保持数据局部性	不同步	受限	及时重汇聚	较低	较低
HWS[29]	组内	保持数据局部性	不同步	受限	及时重汇聚	较低	较低
SBI+SWI[30]	组内+组间	破坏数据局部性	不同步	不受限	选择性重汇聚	较高	较高
CAPRI[31]	组间	破坏数据局部性	同步	不受限	及时重汇聚	一般	较高

3.1.2 多路并行执行的线程调度优化

在提高 SIMD 资源利用率的同时，线程重组调度在一定程度上会破坏线程组内的数据局部性，而多路并行执行线程调度则是在不破坏数据局部性的情况下尽量提升 SIMD 资源的利用率，并通过提供更多的线程调度实体来提升隐藏长延时访存的能力. 它是另一类常见的、提升分支转移处理效率的线程调度优化机制，主要通过改变传统分支路径顺序执行的方式，尽量让多个分支路径上的线程交替调度执行或同时调度执行.

当某一个分支路径上的线程组由于长延时访存等待而造成流水线空闲时，交替调度执行其他分支

路径的线程组能够填补该空闲，这样既能提高并发执行的线程数量，也能在一定程度上提升访存级并行度(Memory Level Parallelism，MLP). Meng 等人[33]、Yu 等人[34]和 ElTantawy 等人[35]分别提出了多路交替执行的线程调度优化策略 DWS(Dynamic Warp Subdivision)、PDOM-ASI(PDOM with All Sub-warps Issuable)、MP IPDOM(Multi-Path IPDOM). 其中 DWS 和 MP IPDOM 采用 warp 分片表实现了多个子 warp(warp 中执行相同分支路径的部分线程集合)的交替执行，PDOM-ASI 则通过对分支控制结构进行二叉树分析实现了多个子 warp 的交替执行. 而 Rhu 等人[36]提出的双路径交替执行策略 DPE(Dual-path Execution)通过对重汇聚栈结构的修改，仅实现了双分支路径上的线程交替执行. 另外，多路线程交替执行需要考虑当子 warp 执行结束后，及时对它们进行重汇聚，以保证后续指令保持高的 SIMD 通道利用率. PDOM-ASI、DPE 实现了及时重汇聚策略，MP IPDOM 实现了尽早重汇聚策略，DWS 策略则采用延迟重汇聚机制，但一定程度上降低了后续规则指令执行时的 SIMD 通道利用率. 另外，这 4 种多路线程交替调度策略在一定程度上均能有效地解决 SIMD 执行深度不够(可供调度的线程实体不足)的问题. Lashgar 等人[37]提出的 HARP(Harnessing in Active thReads in many-core Processors)策略则有效地结合了线程重组机制和多路交替调度机制，它能同时解决分支转移引起的 SIMD 执行宽度和执行深度下降的问题.

交替执行的线程调度方式在宏观上实现了多路分支路径的并行执行，增加了可供调度的线程实体，相对于传统的顺序执行，它能进一步提升 TLP. 但在一般情况下，这种优势只有在活跃线程组不够的情况下方能体现出来，因为当存在其他活跃线程分组时，快速切换到其他活跃线程组可以隐藏线程调度实体的长延时访存操作. 另外，交替执行的线程调度也可以看做是将后面调度线程组的调度次序提前，提前执行的线程组产生的访存结果可以被先前调度的线程组使用，从而提高并发线程的访存并行度，有利于提升系统的访存效率.

多路交替调度执行能使不同分支路径上的线程组相互隐藏各自的长延时访存操作，然而微观上某一时刻仍然只有一条分支路径上的线程执行，SIMD 资源的利用率仍然没有达到最大化. 多路线程同时调度执行是另一类提高 SIMD 通道利用率的方法，它可以使不同分支路径上的线程同时执行各自的指令. 相对于多路交替执行的线程调度方法，多路线程同时调度执行能进一步提高 SIMD 通道的利用率. 文献[38]提出了双指令多线程(Dual Instructions Multiple Threads，DIMT)的思想，通过对指令发射等部件的改动来实现双分支路径的同时执行. 3.1.1小节提到的 SBI 和 SWI 策略通过对指令发射部件的修改也实现了双分支路径上同时发射指令. Wang等人[39]提出并实现了MSMD(Multiple SIMD，Multiple Data)执行模型，设置了多个可灵活划分的、独立的 SIMD 数据通道，使得不同分支路径上的线程组可以同时执行，大大地提高了程序执行的 TLP. MSMD 模型实质上是对 MIMD 执行模型的发展，它包含了 SIMD 模型的思想，但在硬件实现上更加复杂. Dasika 等人[40]针对科学计算提出的 PEPSC(Power-Efficient Processor for Scientific Computing)体系结构和文献[41-43]提出的 TSIMT(Temporal SIMT)执行模式均支持多个分支同时执行，与 MSMD 模型的思想相似. 但是，相对于 SIMD 模型，它们和 MSMD 模型一样，在物理实现上要复杂得多，需从面积和功耗等方面进行更多的考虑.

多路并行执行的线程调度优化能在出现分支转移时进一步提升程序执行的 TLP，还能在一定程度上提升系统的 MLP. 但是多路并行执行的线程调度优化策略仍然存在着提升 TLP 有限、硬件物理开销较大、重汇聚时机选取欠合理等不足，可以在这些方面做进一步的研究. 另外，可以综合考虑线程重组和多路交替执行的两类线程调度机制的优势，对它们进行有效地结合，使得各分支路径的 SIMD 执行宽度和执行深度得到更有效的提升.

表 2 归纳了几种常见的多路并行执行线程调度策略的特点. 从表 2 中可以看出，由于 DWS 采用了延迟重汇聚策略，它对 TLP 提升的幅度最小.

表 2　主要的多路并发执行线程调度策略特点

多路并发执行策略	多路并发执行分类	是否结合线程重组	重汇聚时机	TLP 提升程度	实现复杂程度
SBI+SWI[30]	多路同时执行	是	选择性重汇聚	较高	较高
DWS[33]	多路交替执行	否	延迟重汇聚	较低	一般
MP IPDOM[35]	多路交替执行	否	尽早重汇聚	一般	较高
DPE[36]	多路交替执行	否	及时重汇聚	一般	一般
HARP[37]	多路交替执行	是	及时重汇聚	一般	一般
MSMD[39]	多路同时执行	否	及时重汇聚	较高	高

3.1.3 分支转移中的重汇聚机制

将执行不同分支路径的线程在重汇聚点上进行重汇聚同步，将它们重新组合成分支转移前的线程组，以保证后续规则指令执行过程能保持较高的 TLP.

传统的 PDOM 重汇聚机制选择立即后支配块作为重汇聚点[11,44]. 然而重汇聚点的选取对存在分支转移程序的执行效率有较大的影响. 如果重汇聚点选择较晚，会在一定程度上降低后续规则指令执行的 TLP，但是对于某些应用程序尤其是访存密集型程序，选择较晚的重汇聚点对提升整个程序执行的 MLP 是有益的. 这主要得益于此类程序的线程组内部存在数据局部性的可能性较大，提前执行的部分线程获取的数据则可能被落后的线程所使用，从而提高落后线程后续的 cache 访问命中率. 因此，针对不同的计算任务，应选择不同的重汇聚策略. 但是，采用何种重汇聚策略需要综合考虑重汇聚策略带来的性能收益和由此产生的开销等因素，只有当收益大于开销，重汇聚策略才是有益的. Meng 等人[33]提出的 DWS 策略对所有的情形均采用延迟重汇聚策略. 文献[17,30,35]则认为尽早重汇聚以同步不同执行分支路径的线程对提升 TLP 有利，并对尽早重汇聚机制及实现进行了分析讨论. Diamos 等人[45]专门针对非结构化的分支转移，提出了 TF (Thread Frontier)重汇聚机制，结合编译技术实现了尽早重汇聚机制，减少了某些基本指令块的重复执行.

上述重汇聚策略的最大不同在于重汇聚时机的选择，但是它们各自对所有的应用程序均采用统一的重汇聚机制，并未综合考虑性能收益和开销等多种因素. 适宜的方法是根据不同应用程序的特点选择合适的重汇聚时机和相应的重汇聚机制.

3.2 针对访存效率提升的线程调度优化

访存效率是影响 GPGPU 整体性能高低的主要因素之一. 除了本身物理条件的限制，访存离散和片上存储资源访问竞争引起的数据局部性破坏等因素也会影响 GPGPU 的访存效率. 针对访存效率提升的线程调度优化方法通常以减少片上存储资源访问竞争和减少访存离散为主要目的，并通过数据预取等手段来提升数据访问的命中率.

3.2.1 针对片上存储资源访问竞争的线程调度优化

存储系统的带宽是影响 GPGPU 性能的重要瓶颈，在片上集成层次化的存储资源有利于提升 GPGPU 的访存吞吐量. GPGPU 片上存储资源通常包括寄存器文件、一级数据 cache、一级指令 cache 和纹理 cache 等. 由于芯片面积的限制，相对于片上同时执行的成千上万个线程，这些资源显得非常有限. 当调度到核上的线程数超过一定限度，存储资源将会出现不足. 如果继续调度线程到这些核上执行，则会产生对核上存储资源的访问竞争，此时会置换部分已分配的资源，必然会降低被置换资源对应线程的访存效率. 因此当出现资源访问竞争时，系统的性能会出现一定程度的下降，严重时会出现访存“抖动”现象[46-48]. 因此，尽量减少线程对存储资源的占用，或尽快释放对存储资源的占用，是减少片上存储资源访问竞争的两种直观思路. Bakhoda 等人[49]研究发现，有相当部分的应用程序在并发执行的线程块数量受限的情况下能获得更高的性能. 线程调度优化成为了目前解决片上存储资源尤其是 cache 资源访问竞争问题的重要方法，它可以从 3 个方面进行考虑：(1) 通过优化调度到 GPGPU 核上的并发线程数量减少对片上存储资源访问竞争；(2) 通过混合执行不同类型的计算任务减少对片上存储资源访问竞争，并尽可能地提升片上资源的利用率；(3) 通过优化线程的调度顺序减少片上存储资源的访问竞争.

改变线程调度的顺序和时机、有效控制和优化调度到核上的线程数量，是目前线程调度优化解决片上存储资源访问竞争的主要思想. Rogers 等人[50]提出的 CCWS(Cache-Conscious Wavefront Scheduling) 策略通过对线程组 cache 访问失效情况的分析，动态调整调度到 GPGPU 核上的线程组数量，但是它需要等到数据局部性破坏之后才进行相应的调整. 随后 Rogers 等人[48]又提出了另一种降低 cache 资源访问竞争的线程调度策略 DAWS(Divergence-Aware Warp Scheduling). 与 CCWS 策略不同的是，该策略通过分析 cache 的使用情况主动地动态调整调度到核上的线程组数量，破坏数据局部性的情况大幅度地减少. Kayiran 等人[16]则结合不同计算任务的特点动态调整调度到核上的线程块数量. 他们通过分析发现，不同类型的计算任务运行不同数量的线程块时会表现出不同的性能效果，并提出了 DYNCTA(Dynamic CTA scheduling) 线程调度策略. 与 CCWS 和 DAWS 策略不同，该策略定期对 GPGPU 核的状态进行定量分析，然后动态调整调度到 GPGPU 核上的线程数量. Cheng 等人[51]针对多核处理器的访存竞争问题也提出过类似的模型. 不同的是，他们通过对多核处理器性能的定量分析来动态地确定最优并发访存任务数. 与前面几种动态调整调度到核上的线程块数量的方法不同，Lee 等人[52]提出的 LCS(Lazy CTA Scheduling)+BCS

(Block CTA Scheduling)的线程调度策略对调度到核上的最优线程组数量进行静态量化.

限制调度到核上的线程组数量是前面几种算法的主要特点,可以有效地减少 cache 的访问竞争和对数据局部性的破坏,但在一定程度上限制了片上资源的利用率. Xiang 等人[53]提出了 WarpMan(Warp-level resource Management)线程调度策略,充分考虑了片上存储资源竞争的问题,在保证不破坏数据局部性的基础上,尽量调度更多的线程组到核上运行. 文献[54-55]通过实验分析发现,一次运行一个计算任务会造成 GPGPU 资源利用率较低,通过对 GPGPU 资源空间划分或细粒度控制,让多个 kernel 并发执行,可以有效地提高片上资源的利用率. 但是在组合计算任务时,应尽可能选择不同特点的 kernel,以尽可能减少 kernel 之间产生的存储资源竞争. Zheng 等人[47]提出的 CCA(adaptive Cache and Concurrency Allocation)线程调度策略充分利用了这一性质,对于表现出数据局部性的线程组采用动态数量调节调度策略,而对于不具有数据局部性的线程组则采用 cache 绕行策略调度执行,这既限制了访问 cache 的线程数量,又提高了并发执行的线程数量. 相对于 WarpMan,由于有更多的执行任务,CCA 策略具有更大的 SIMD 执行深度,对长延时访存操作的隐藏能力更强. Awatramani 等人[56]也提出了同时调度不同类型 kernel 的线程调度策略 KITBS(Kernel Interleaved Thread Block Scheduling),并明确指出将访存密集型 kernel 和计算密集型 kernel 进行组合调度. Lee 等人[52]在文献中提出的 mCKE(mixed Concurrent Kernel Execution)线程调度策略也支持发射多个 kernel 到同一个核执行,但是文中对于如何组合 kernel 没有进行深入的讨论.

调度多个 kernel 到同一个核上执行确实能提高片上资源的利用率,但是需要同时调度不同类型的 kernel,否则会加剧对片上资源的访问竞争. 如何合理组合 kernel 成为了该类线程调度方法的关键,对于包含 kernel 类型相同的应用程序并不适用,因此它的应用具有一定的局限性.

和上述两类方法一样,改变线程的调度顺序也能有效地减少片上存储资源访问竞争. Narasiman 等人[26]为了提升长延时访存操作的隐藏能力,提出了两级调度策略 TL(Two-Level scheduling),将连续的线程块进行分组,限制某一时间段内同时执行的线程数量,从而有效地避免了片上资源访问竞争,但这会在一定程度上降低程序执行过程的 TLP,也会降低片上资源的利用率. Jog 等人[57]在 TL 的基础上提出了 OWL(Cooperative Thread Array Aware Warp Scheduling)线程调度策略,它也是按线程块组进行两级调度. 另外 OWL 策略还通过调度不连续的线程块组来提高访存的并行度,有利于系统整体访存性能的提升.

优化线程的调度顺序可以在一定程度上减少片上资源的访问竞争,但是如果调度顺序不当,有可能破坏线程组之间的数据局部性,甚至会降低程序执行过程中的 TLP 和片上资源的利用率.

上述三类线程调度优化方法在一定程度解决了片上资源访问竞争问题,有的方法甚至进一步提升了片上资源的利用率. 然而这些线程调度优化方法仍然存在以下不足:限制核上并行执行的线程数量与提高片上资源利用率存在冲突;同时调度多个 kernel 的适用性较差,对 kernel 进行定性分析机制有待完善;改变线程的调度顺序与保持数据局部性存在冲突. 因此,在解决片上存储资源访问竞争的问题上可以在这些方面继续展开研究.

表 3 对几种常见的针对片上存储资源竞争的线程调度优化方法进行了归纳和比较.

表 3　常见的针对片上资源竞争的线程调度优化策略特点

线程调度策略	解决资源竞争的方式	线程调度粒度	是否提升资源利用率	是否提升 TLP	保持数据局部性	提升片上/片外存储资源访问效率
DYNCTA[16]	动态限制调度数量	线程块	否	否	较好	片上存储
TL[26]	改变线程调度顺序	线程块	否	否	较差	片上存储
CCA[47]	混合调度不同计算任务	线程组	是	是	较好	片上存储
DAWS[48]	动态限制调度数量	线程块	否	否	较好	片上存储
CCWS[50]	动态限制调度数量	线程块	否	否	较差	片上存储
mCKE[52]	混合调度不同计算任务	Kernel	是	是	一般	片上存储
WarpMan[53]	动态限制调度数量	线程组	是	否	较好	片上存储
OWL[57]	改变线程调度顺序	线程块	否	是	较好	片上和片外存储

3.2.2　针对访存离散的线程调度优化

GPGPU 通过支持大量并发线程的同时执行来隐藏长延时访存操作,当某个线程组处于长延时访存等待状态,GPGPU 会立即切换到其他准备就绪

的线程组继续执行,因此只要有足够活跃的线程组,SIMD 流水部件就会一直处于工作状态.然而,同一个线程组中不同线程访问的地址可能不在同一个 cache 行,且它们的访存情况也可能存在差异,部分线程由于 L1 cache 访问失效而处于等待状态,另一部分线程的 L1 cache 访问命中.另外,由于 SIMD 执行模式采用锁步执行方式,线程组中 L1 cache 访问命中的线程只能等待 L1 cache 访问失效的线程,导致整个线程组的执行被延时,我们称这种情况为访存离散(Memory Divergence)[33].

通常情况下,访存离散比分支转移发生的概率大很多[58].如果在发生访存离散时没有足够活跃的线程组进行切换,SIMD 流水线将处于空闲状态,从而对系统性能造成影响.另外,访存离散还会导致相应线程组的执行被挂起,延缓了其执行进度,其持续占有的资源对其他线程的执行也会产生影响.文献[59-61]针对 CPU 的 cache 访问失效提出了向前执行(Runahead Execution)的策略,他们将这种向前执行看作是无效的"预执行",但是可以将后续指令及其数据提前预取到 cache 中,减少后续指令的 cache 访问失效率.借助于该思想,在 GPGPU 中,当某个线程组出现访存离散时,也可以允许部分 L1 cache 命中的线程继续向前执行,从而加速整个线程组的执行进程.Tarjan 等人[58]针对访存离散提出了 Adaptive Slip 线程调度策略,重点讨论了针对循环中出现访存离散时的线程调度优化机制.但是由于该策略总是在下一次循环迭代中才对线程组进行重汇聚,容易造成较大的等待延时.在此基础上,Meng 等人[33]提出了 DWS 策略,能对任意发生的访存离散进行处理,并提出了相对更加有效地重汇聚机制.随后 Meng 等人[62]分析认为宽度越大的 SIMD 越容易出现访存离散,他们提出了 Robust-SIMD 自适应调度策略,对不同应用程序的执行性能进行抽样测试分析,以此来调节 SIMD 的执行宽度和执行深度,减少了访存离散的发生.Lashgar 等人[63]同样认为宽度大的线程组容易发生访存离散,他们提出了 DWR(Dynamic Warp Resizing)策略,通过动态调节线程组大小来减少访存离散的发生.

访存离散发生时允许部分访存命中的线程继续向前执行,实质上是以减小 warp 宽度而降低 TLP 为代价的,因此需要考虑尽早重汇聚线程组中的所有线程.而且,当存在足够多活跃的线程组时,切换到其他活跃线程组可以隐藏由于访存离散带来的长延时等待,这种情况下此类方法对系统性能提升的意义不大,因此还需要考虑策略选择的时机.另外,如果能尽量保持数据局部性不被破坏,且尽可能开发线程之间的数据局部性,则能更有效的解决访存离散问题.

表 4 归纳了几种常见的针对访存离散的线程调度优化方法的特点.

表 4 常见的针对访存离散的线程调度优化策略特点

线程调度策略	解决访存离散的方式	重汇聚时机	是否提升 TLP	提升片上/片外存储资源的访问效率
DWS[33]	向前执行	延迟重汇聚	是	片上和片外存储
Adaptive Slip[58]	向前执行	延迟重汇聚	否	片上和片外存储
RobustSIMD[62]	动态调整 SIMD 宽度	及时重汇聚	是	片上存储
DWR[63]	动态调整 SIMD 宽度	及时重汇聚	否	片上存储

3.2.3 针对数据预取的线程调度优化

数据预取是提前将数据从片外存储取到缓存中以提升后续数据访问命中率的技术,它是提高处理器执行效率的重要手段之一.然而 GPGPU 采用的 SIMT 执行模式可以通过众多并行执行的线程来隐藏长延时访存操作,并出于考虑能效优化的目的,在 GPGPU 中去掉了硬件预取部件[64-65].通过线程调度优化提高数据预取能力,有利于进一步地开发线程之间的数据局部性、提高 cache 访问命中率、提升隐藏长延时访存操作能力.

GPGPU 中的数据预取可以通过线程调度优化来实现,其主要思想是在线程执行过程中根据一定的预取距离和预取度,在为自己获取数据的同时,将同一个线程组或其他线程组中其他线程的所需数据提前预取到缓存中.其中预取距离是指相对于特定数据请求而进行预取的提前量,它制约着数据预取的准度;预取度是指一次预取数据的大小,它的大小的选取对片上资源的竞争会产生一定的影响,因为预取过多的数据会挤占其他线程所需的资源.3.2.2 小节中针对访存离散的线程调度优化策略允许 cache 访问命中的线程继续执行,实质上也可以看作是一种数据预取机制,因为继续向前执行的线程后续访存获得的数据有可能成为落后线程后续访问的数据,可以提高落后线程的 cache 访问命中率.这类数据预取机制主要利用了连续线程之间具有数据局部性的特点,然而却以牺牲一定的 TLP 为代价.NVIDIA 公

司 Fermi 系列的 GPGPU 采用的轮转线程调度策略 RR(Round-robin Scheduling)和 Narasiman 等人[26]提出的 TL 线程调度策略同样利用了数据局部性原理来实现数据预取机制. 但是由于连续的线程组之间距离太近，可能会使数据预取“太迟”. Jog 等人[66]提出了预取感知的线程调度策略 PAS(Prefetch-aware Scheduling)，通过改变连续线程组的调度顺序使得原本连续的线程组之间保持一定的预取距离. 该策略在调整预取距离的同时，还考虑了对预取度的控制，仅对访问频度较高的内存块实行数据预取. 他们提出的另一种线程调度策略 OWL 与 PAS 相似，也是通过调整连续线程块的发射距离来进行数据预取. 但是与 PAS 不同的是，OWL 在数据预取时还考虑了如何提升访存并行度.

通常情况下对于规则的访存模型容易确定数据预取的距离，但是对于不规则的访存模型来说，则容易出现数据预取“过早”或“过晚”，导致预取的数据未及时到来或被覆盖. 这种无效的数据预取会造成有限存储带宽的浪费，甚至有可能造成整体访存性能的下降，因此应根据程序执行过程中系统的状态变化动态调整数据预取的距离和预取度，从而进一步提高数据预取的准度和有效性. 前面提到的 PAS 和 OWL 策略均未考虑后一种情况，它们采用静态方法确定数据预取的距离，对于访存不规则的应用程序来说，其预取的准确度势必会下降. Lee 等人[67]则提出了自适应预取机制 MT-prefetching(Many-Thread aware prefetching)，在线程执行过程中对预取数据替换率和预取访存合并率两个度量进行周期性采样分析，并以此自动调节数据预取行为. Dahlgren 等人[68]通过对预取准确度的度量来自适应地调整预取距离. Srinath 等人[69]提出了更复杂的、基于反馈的预取机制，综合考虑了预取的准确度、预取的及时性以及 cache 污染等多个因素.

无论是规则的数据预取，还是基于反馈的动态预取机制，一般只有在活跃线程数较少的情况下才能充分发挥其优势，因为在活跃线程数足够的情况下长延时访存可以被隐藏. Sethia 等人[70]提出的APOGEE(Adaptive Prefetching On GPUs for Energy Efficiency)预取策略着重研究了活跃线程组少的情况下的数据预取，取得了较好的能效比.

另外，Yang 等人[71]通过编译的方式提前将数据预取到寄存器中. 但是在一定程度上增加了对寄存器资源的使用，对系统功耗产生影响.

有效地结合线程调度优化方法，根据系统状态变化进行动态的数据预取能进一步地提升数据预取的有效性. 但是对于不同类型的计算任务，数据预取表现出来的重要性不同. 例如，对于计算密集型的计算任务，数据预取的优势并不能得到很好的体现. 因此，针对不同的计算任务应采取不同的预取策略.

表 5 归纳了几种常见的针对数据预取的线程调度优化方法的特点.

表 5　常见的针对数据预取的线程调度优化策略特点

线程调度策略	数据预取机制	结合预取因素	预取的准确度	实现难易程度
DWS[33]	静态的数据预取	无	较低	容易
OWL[57]	静态的数据预取	预期距离	一般	一般
PAS[66]	静态的数据预取	预取距离＋预取度	一般	一般
MT-prefetching[67]	基于反馈的动态数据预取	预取距离＋预取度＋系统状态	较高	较复杂

3.3　针对标量指令执行的线程调度优化

SIMT 执行模式让同一条指令在不同的数据上执行相同的操作，以此获得高的计算吞吐量和高性能. 然而在应用程序执行过程中存在相当数量的 SIMD 指令，它们对应的线程不仅处理的操作数完全相同，而且其输出结果也完全相同，这些指令被称为标量指令[20,72-73]. 标量指令的执行会产生大量重复冗余操作. 在同一个线程组或线程块中，如果一条标量指令只需要一个线程执行，则可以节省大量的计算资源和存储资源. 对此首先需要识别应用程序中存在的标量指令，其次是对标量指令的执行进行优化.

目前常见的识别标量指令的方法有两类. 一类是通过编译的方式对代码进行静态分析识别[72,74-75]，另一类则是通过硬件机制进行动态识别[76]. 识别后的标量指令一般只需执行一次，如果利用传统的 SIMD 通道执行，则会造成很大的片上资源浪费. 为此有不少做法是对硬件进行修改，增加特定的标量指令执行部件和标量数据存储部件. AMD 发布的 GNU 体系结构中引入了专门的标量处理单元[77]. Xiang 等人[73]则通过增加单独的标量寄存器文件解决了标量指令操作数的存储问题. 然而这些方法都需要对传统的 SIMT 系统结构进行较大的改动，线程调度优化则可在不用对 SIMT 系统结构进行较大改动的情况下实现标量指令的高效执行. Yilmazer 等人[20]提出了SW(Scalar Waving)＋SSSW (Simultaneous Scalar and SIMD group Waving)线

程调度策略.一方面将执行相同 PC 的标量指令组织为标量波执行,另一方面将标量波和线程组组合在一起执行,实质上也是一种线程压缩重组调度策略.由于进行了标量识别,SW+SSSW 策略比普通线程重组调度策略具有更好的"压缩"性能.但是 SW 策略只对线程组内的指令进行标量化执行,对线程组间指令的标量化执行未进行分析讨论.

对标量指令的识别和执行能够大大减少某些指令的执行次数,不仅可以节省大量的计算资源,还为降低系统功耗提供了很大空间.若能从线程调度优化的角度充分利用这两方面的优势,则能更好地提升系统的性能和功效,目前这方面开展的研究工作相对较少.

3.4 针对功耗优化的线程调度优化

相对于 CPU,GPGPU 能获得更高的能效[78-80].目前应用广泛的动态电压频率调节技术(DVFS)能有效地降低 GPGPU 的动态功耗[81-83],但是随着片上集成的晶体管数量越来越多及制作工艺的提升,静态漏电流功耗在整个 GPGPU 功耗中所占的比重越来越大[81-82,84-85].动态功耗和静态漏电流功耗已经成为 GPGPU 功耗的两个主要来源.过高的功耗会产生大量的热量,直接影响电子元器件的稳定性、可靠性和使用寿命,功耗问题成为制约 GPGPU 发展的重要瓶颈之一.

动态功耗通常是由晶体管的充放电产生的功耗,其大小通常与电子元器件的工作电压和频率相关.动态降低不同部件的工作电压和频率是目前有效降低 GPGPU 动态功耗的主要手段之一.Gu 等人[86]针对交互式游戏软件,利用 DVFS 技术对图形处理器的动态功耗进行优化.Mochocki 等人[87]在分析了手机 3D 图形处理器不同流水阶段负载不均衡特性的基础上,结合 DVFS 技术有效地降低了手机 3D 图形处理器的动态功耗.

静态功耗通常是由漏电流产生的功耗.漏电流的产生是由于 CMOS 工艺的提升使得晶体管绝缘层变薄,导致电流容易透过绝缘层.通过门控技术适时关闭或休眠空闲部件是目前降低静态功耗的常用技术.Wang 等人[81]针对着色器部件、固定功能几何单元和非着色执行单元分别提出了 PSS(Predictive Shader Shutdown)、DGP(Deferred Geometry Pipeline)和超时门控的 3 种体系结构级门控策略.Wang 等人[88]着重针对 L1 cache 和 L2 cache 的访问进行能耗门控,将它们的工作状态在激活、休眠和关闭 3 种状态间动态切换.Abdel-Majeed 等人[89]则专门提出了针对 GPGPU 中寄存器文件功耗优化的门控方法,通过设置三模态寄存器访问控制单元和对活跃掩码行为感知的门控单元,降低了静态漏电流功耗和动态功耗.

DVFS 和功耗门控技术能有效地降低 GPGPU 的动态功耗和静态漏电流功耗,然而单一的根据各个部件的运行状态进行功耗控制优化的空间有限.目前有部分对 GPGPU 功耗进行优化的研究工作结合了线程调度优化方法,通过对线程调度的优化使得各部件的运行状态呈现更佳的规律性,从而使之能更好的匹配 DVFS 和功耗门控技术的应用.Abdel-Majeed 等人[82]提出了门控感知的二级 warp 调度策略 GATES(Gating-aware Two-level Warp Scheduler),优先调度发射执行同类型指令的线程组,以使其他执行单元获得较长的空闲时间,从而有利于通过功耗门控技术降低功耗.Xu 等人[85]提出了模式感知的二级 warp 调度策略 PATS(Pattern Aware Two-level Scheduler),优先执行具有相同分支模型的线程组.林一松等人[90]提出的功耗优化模型结合了计算并行度和访存并行度,根据并发执行的线程数确定调频因子的大小,并结合 DVFS 技术对 GPGPU 的动态功耗进行了有效地优化.

上述几种线程调度优化方法均以功耗优化为主要目的,而前面章节分析的线程调度优化方法则以性能优化为主,其中也有部分研究工作分析了这些线程调度方法对功耗产生的影响.Gebhart 等人[91]提出的二级 warp 调度策略 TL 通过设置专门的寄存器缓冲来保留活跃线程组的上下文,减少了对寄存器文件的访问,从而大大降低了访问寄存器产生的功耗.Xiang 等人[73]提出了对标量指令执行标量化,大大减少了某条标量指令执行所需要的各种资源,结合功耗门控技术能有效地降低静态功耗.随后他们提出的 WarpMan 线程调度策略及时回收执行完成的线程组占用的资源,也为功耗门控优化提供了机会.Rogers 等人[48]对其提出的 DAWS 线程调度分析指出,由于提高了 Cache 访问命中率,减少了片外访存次数,能有效地降低由于访存产生的动态功耗.

尽管目前有部分线程调度优化算法考虑了功耗优化,但是以功耗优化为主要目标的线程调度优化研究工作仍然不多,尤其是结合低功耗优化技术的线程调度优化方法的研究还相对较少.

表 6 归纳了几种常见的针对功耗优化的线程调度优化方法的特点.

表 6 常见的针对功耗优化的线程调度优化策略特点

线程调度策略	主要目标	减少功耗目标	结合何种低功耗技术
DAWS[48]	提高性能	降低动态功耗	无
WarpMan[53]	提高性能	降低动态功耗	无
GATES[82]	降低功耗	降低静态功耗	功耗门控技术
PATS[85]	降低功耗	降低静态功耗	功耗门控技术
基于并行度分析模型[90]	降低功耗	降低动态功耗	DVFS
TL[91]	提高性能	降低动态功耗	无

4 结束语

目前研究者提出的线程调度优化方法均在一定程度上解决了影响 GPGPU 性能和功效提升的相关问题,但是这些线程调度优化方法都只侧重考虑解决其中某一个或几个方面的问题,导致 GPGPU 的性能和功效仍然未能达到最大化.

GPGPU 的不断发展使得线程调度优化在以下方面还存在挑战:(1) 随着 GPGPU 微体系结构的发展,尤其是片上融合 CPU-GPU 微体系结构的发展,可以大大减少 CPU 和 GPU 之间数据传输产生的开销,然而还需要针对新的微体系结构的特点对线程调度进行分析并优化,解决在 CPU 和 GPU 间合理分配计算任务、利用 CPU 数据预取优势以减少 GPU 的访存开销及片上资源的访问竞争等方面的问题;(2) 当前新型存储技术在存储密度、访问速度和功耗方面都有一定的优势,为 GPGPU 的性能提升和功耗优化提供了新的空间,需要将线程调度优化方法和新的存储技术进行有效地结合;(3) 在不少线程调度优化方法中,某些参数的选取及某些自适应线程调度机制还有待完善,结合编译技术对代码进行静态或动态的分析,为线程调度优化提供更加有效合理的决策支持,例如通过编译技术实现对标量指令的识别、线程重汇聚点的合理判断等;(4) 随着新的 GPGPU 微体系结构对多 kernel 并行执行的支持,通过线程调度优化更好的解决多个 kernel 混合执行产生的资源竞争等问题;(5) 功耗问题仍然是影响 GPGPU 发展的主要瓶颈之一,需进一步结合线程调度技术更好的为功耗优化服务;(6) 结合多种线程调度优化方法的优点综合考虑影响 GPGPU 性能的多种因素,使线程调度优化方法更加合理有效. 未来以进一步提升 GPGPU 的性能和能效为主要目标的线程调度优化可以在以上方面做进一步的研究.

致 谢 感谢何炎祥教授主持的博士讨论班上各位师生所发表的意见和建议.

参 考 文 献

[1] Nickolls J, Dally W J. The GPU computing era. IEEE Micro, 2010, 30(2): 56-69

[2] Qian Wei. Tianhe-1: China's speed. China Newsweek, 2010, 48: 55-57(in Chinese)
(钱炜. 天河一号:中国速度. 中国新闻周刊, 2010, 48: 55-57)

[3] Wang Tao. "Tianhe-2" Supercomputer. Science, 2013, 65(4): 52(in Chinese)
(王涛. "天河二号"超级计算机. 科学, 2013, 65(4): 52)

[4] Luebke D, Humphreys G. How GPUs work. IEEE Computer, 2007, 40(2): 96-100

[5] Montrym J, Moreton H. The Geforce 6800. IEEE Micro, 2005, 25(2): 41-51

[6] Steffen M, Zambreno J. Improving SIMT efficiency of global rendering algorithms with architectural support for dynamic micro-kernels//Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture. Atlanta, USA, 2010: 237-248

[7] Nickolls J, Kirk D. Graphics and computing GPUs. Computer Organization and Design: The Hardware/Software Interface. 4th Edition. USA: Morgan Kaufmann, 2009

[8] Lindholm E, Nickolls J, Oberman S, et al. NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro, 2008, 28(2): 39-55

[9] Sankaralingam K, Nagarajan R, Liu H, et al. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture//Proceedings of the 30th Annual International Symposium on Computer Architecture. San Diego, USA, 2003: 422-433

[10] Wills L, Taha T, Baumstark Jr L, et al. Estimating potential parallelism for platform retargeting//Proceedings of the 9th Working Conference on Reverse Engineering. Richmond, USA, 2002: 55-64

[11] Fung W W L, Sham I, Yuan G, et al. Dynamic warp formation and scheduling for efficient GPU control flow//Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. Chicago, USA, 2007: 407-420

[12] NVIDIA. CUDA C Programming Guide Version 5.0. USA: NVIDIA, 2012

[13] Nugteren C, van den Braak G J, Corporaal H. Future of GPGPU micro-architectural parameters//Proceedings of the Conference on Design, Automation and Test in Europe. Grenoble, France, 2013: 392-395

[14] Lorie R A, Strong Jr H R. Method for conditional branch execution in SIMD vector processors: U.S. Patent 4435758. 1984-3-6

[15] Nickolls J, Buck I, Garland M, et al. Scalable parallel programming with CUDA. Queue, 2008, 6(2): 40-53

[16] Kayıran O, Jog A, Kandemir M T, et al. Neither more nor less: Optimizing thread-level parallelism for GPGPUs//Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques. Edinburgh, UK, 2013: 157-166

[17] Fung W W L, Aamodt T M. Thread block compaction for efficient SIMT control flow//Proceedings of the 17th International Symposium on High-Performance Computer Architecture. San Antonio, USA, 2011: 25-36

[18] NVIDIA. NVIDIA's next generation CUDA compute architecture: FERMI. Computer System, 2009, 26: 63-72

[19] Moy S, Lindholm J E. Method and system for programmable pipelined graphics processing with branching instructions: U.S. Patent 6947047. 2005-9-20

[20] Yilmazer A, Chen Z, Kaeli D. Scalar Waving: Improving the efficiency of SIMD execution on GPUs//Proceedings of the 28th International Parallel and Distributed Processing Symposium. Phoenix, USA, 2014: 103-112

[21] Wu H, Diamos G, Li S, et al. Characterization and transformation of unstructured control flow in bulk synchronous GPU applications. International Journal of High Performance Computing Applications, 2012, 26(2): 170-185

[22] Woop S, Schmittler J, Slusallek P. RPU: A programmable ray processing unit for realtime ray tracing. ACM Transactions on Graphics, 2005, 24(3): 434-444

[23] Coon B W, Lindholm J E. System and method for managing divergent threads in a SIMD architecture: U.S. Patent 7353369. 2008-4-1

[24] Levinthal A, Porter T. Chap-a SIMD graphics processor//Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH. Minneapolis, USA, 1984: 77-82

[25] Cervini S. System and method for efficiently executing single program multiple data (SPMD) programs: European Patent EP 1531391 A2. 2005-5-18

[26] Narasiman V, Shebanow M, Lee C J, et al. Improving GPU performance via large warps and two-level warp scheduling//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. Porto Alegre, Brazil, 2011: 308-317

[27] Malits R, Bolotin E, Kolodny A, et al. Exploring the limits of GPGPU scheduling in control flow bound applications. ACM Transactions on Architecture and Code Optimization, 2012, 8(4): 29

[28] Vaidya A S, Shayesteh A, Woo D H, et al. SIMD divergence optimization through intra-warp compaction//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel, 2013: 368-379

[29] Jin X, Daku B, Ko S B. Improved GPU SIMD control flow efficiency via hybrid warp size mechanism. Microprocessors and Microsystems, 2014, 38(7): 717-729

[30] Brunie N, Collange S, Diamos G. Simultaneous branch and warp interweaving for sustained GPU performance//Proceedings of the 39th International Symposium on Computer Architecture. Portland, USA, 2012: 49-60

[31] Rhu M, Erez M. CAPRI: Prediction of compaction-adequacy for handling control-divergence in GPGPU architectures//Proceedings of the 39th International Symposium on Computer Architecture. Portland, USA, 2012: 61-71

[32] Rhu M, Erez M. Maximizing SIMD resource utilization in GPGPUs with SIMD lane permutation//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel, 2013: 356-367

[33] Meng J, Tarjan D, Skadron K. Dynamic warp subdivision for integrated branch and memory divergence tolerance//Proceedings of the 37th International Symposium on Computer Architecture. Saint Malo, France, 2010: 235-246

[34] Yu L, Tang X, Wu M, et al. Improving branch divergence performance on GPGPU with a new PDOM stack and multi-level warp scheduling. Journal of Systems Architecture, 2014, 60(5): 420-430

[35] ElTantawy A, Ma J W, O'Connor M, et al. A scalable multi-path microarchitecture for efficient GPU control flow//Proceedings of the 20th International Symposium on High Performance Computer Architecture. Orlando, USA, 2014: 248-259

[36] Rhu M, Erez M. The dual-path execution model for efficient GPU control flow//Proceedings of the 19th International Symposium on High Performance Computer Architecture. Shenzhen, China, 2013: 591-602

[37] Lashgar A, Khonsari A, Baniasadi A. HARP: Harnessing inactive threads in many-core processors. ACM Transactions on Embedded Computing Systems, 2014, 13(3s): 114

[38] Glew A. Coherent vector lane threading. Berkeley Par Lab Seminar. UC Berkeley, Carliforlia, USA, 2009

[39] Wang Y, Chen S, Wan J, et al. A multiple SIMD, multiple data (MSMD) architecture: Parallel execution of dynamic and static SIMD fragments//Proceedings of the 19th International Symposium on High Performance Computer Architecture. Shenzhen, China, 2013: 603-614

[40] Dasika G, Sethia A, Mudge T, et al. PEPSC: A power-efficient processor for scientific computing//Proceedings of the International Conference on Parallel Architectures and Compilation Techniques. Galveston, USA, 2011: 101-110

[41] Keckler S W, Dally W J, Khailany B, et al. GPUs and the future of parallel computing. IEEE Micro, 2011, 31(5): 7-17

[42] Krashinsky R M. Temporal simt execution optimization: U.S. Patent Application 13/209189. 2011-8-12

[43] Lucas J, Lal S, Mesa M A, et al. DART: A GPU architecture exploiting temporal SIMD for divergent workloads//Proceedings of the 9th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems. Fiuggi, Italy, 2013

[44] Muchnick S S. Advanced Compiler Design Implementation. USA: Morgan Kaufmann, 1997

[45] Diamos G, Ashbaugh B, Maiyuran S, et al. SIMD reconvergence at thread frontiers//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. Porto Alegre, Brazil, 2011: 477-488

[46] Li D. Orchestrating Thread Scheduling and Cache Management to Improve Memory System Throughput in Throughput Processors [Ph.D. dissertation]. The University of Texas at Austin, Austin, USA, 2014

[47] Zheng Z, Wang Z, Lipasti M. Adaptive cache and concurrency allocation on GPGPUs. IEEE Computer Architecture Letters, 2014, 13(2): 1-4

[48] Rogers T G, O'Connor M, Aamodt T M. Divergence-aware warp scheduling//Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. Davis, USA, 2013: 99-110

[49] Bakhoda A, Yuan G L, Fung W W L, et al. Analyzing CUDA workloads using a detailed GPU simulator//Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software. Boston, USA, 2009: 163-174

[50] Rogers T G, O'Connor M, Aamodt T M. Cache-conscious wavefront scheduling//Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture. Vancouver, Canada, 2012: 72-83

[51] Cheng H Y, Lin C H, Li J, et al. Memory latency reduction via thread throttling//Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture. Atlanta, USA, 2010: 53-64

[52] Lee M, Song S, Moon J, et al. Improving GPGPU resource utilization through alternative thread block scheduling//Proceedings of the 20th International Symposium on High Performance Computer Architecture. Orlando, USA, 2014: 260-271

[53] Xiang P, Yang Y, Zhou H. Warp-level divergence in GPUs: Characterization, impact, and mitigation//Proceedings of the 20th International Symposium on High Performance Computer Architecture. Orlando, USA, 2014: 284-295

[54] Pai S, Thazhuthaveetil M J, Govindarajan R. Improving GPGPU concurrency with elastic kernels//Proceedings of the Architectural Support for Programming Languages and Operating Systems. Houston, USA, 2013: 407-418

[55] Adriaens J T, Compton K, Kim N S, et al. The case for GPGPU spatial multitasking//Proceedings of the 18th International Symposium on High Performance Computer Architecture. New Orleans, USA, 2012: 1-12

[56] Awatramani M, Zambreno J, Rover D. Increasing GPU throughput using kernel interleaved thread block scheduling//Proceedings of the 31st International Conference on Computer Design. Asheville, USA, 2013: 503-506

[57] Jog A, Kayiran O, Chidambaram Nachiappan N, et al. OWL: Cooperative thread array aware scheduling techniques for improving GPGPU performance. ACM SIGARCH Computer Architecture News, 2013, 41(1): 395-406

[58] Tarjan D, Meng J, Skadron K. Increasing memory miss tolerance for SIMD cores//Proceedings of the ACM/IEEE Conference on High Performance Computing. Portland, USA, 2009: 22

[59] Ramirez T, Pajuelo A, Santana O J, et al. Runahead threads to improve SMT performance//Proceedings of the 14th International Symposium on High-Performance Computer Architecture. Salt Lake City, USA, 2008: 149-158

[60] Dundas J, Mudge T. Improving data cache performance by pre-executing instructions under a cache miss//Proceedings of the 11th International Conference on Supercomputing. Vienna, Austria, 1997: 68-75

[61] Mutlu O, Stark J, Wilkerson C, et al. Runahead execution: An alternative to very large instruction windows for out-of-order processors//Proceedings of the 9th International Symposium on High-Performance Computer Architecture. Anaheim, USA, 2003: 129-140

[62] Meng J, Sheaffer J W, Skadron K. Robust SIMD: Dynamically adapted SIMD width and multi-threading depth//Proceedings of the 26th International Parallel & Distributed Processing Symposium. Shanghai, China, 2012: 107-118

[63] Lashgar A, Baniasadi A, Khonsari A. Dynamic warp resizing: Analysis and benefits in high-performance SIMT//Proceedings of the 30th International IEEE Conference on Computer Design. Montreal, Canada, 2012: 502-503

[64] Rhu M. Performance-Efficient Mechanisms for Managing Irregularity in Throughput Processors [Ph.D. dissertation]. The University of Texas at Austin, Austin, USA, 2014

[65] Wang Hai-Feng, Chen Qing-Kui. General purpose computing of graphics processing unit: A survey. Chinese Journal of Computers, 2013, 36(4): 757-772(in Chinese)
(王海峰，陈庆奎. 图形处理器通用计算关键技术研究综述. 计算机学报，2013，36(4)：757-772)

[66] Jog A, Kayiran O, Mishra A K, et al. Orchestrated scheduling and prefetching for GPGPUs//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel, 2013: 332-343

[67] Lee J, Lakshminarayana N B, Kim H, et al. Many-thread aware prefetching mechanisms for GPGPU applications//Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture. Atlanta, USA, 2010: 213-224

[68] Dahlgren F, Dubois M, Stenstrom P. Sequential hardware prefetching in shared-memory multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1995, 6(7): 733-746

[69] Srinath S, Mutlu O, Kim H, et al. Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers//Proceedings of the 13th International Symposium on High Performance Computer Architecture. Phoenix, USA, 2007: 63-74

[70] Sethia A, Dasika G, Samadi M, et al. APOGEE: Adaptive prefetching on GPUs for energy efficiency//Proceedings of the 22nd International Conference on Parallel architectures and Compilation Techniques. Edinburgh, UK, 2013: 73-82

[71] Yang Y, Xiang P, Kong J, et al. A GPGPU compiler for memory optimization and parallelism management//Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation. Toronto, Canada, 2010: 86-97

[72] Chen Z, Kaeli D, Rubin N. Characterizing scalar opportunities in GPGPU applications//Proceedings of the International Symposium on Performance Analysis of Systems and Software. Austin, USA, 2013: 225-234

[73] Xiang P, Yang Y, Mantor M, et al. Exploiting uniform vector instructions for GPGPU performance, energy efficiency, and opportunistic reliability enhancement//Proceedings of the International Conference on International Conference on Supercomputing. Eugene, USA, 2013: 433-442

[74] Lee Y, Krashinsky R, Grover V, et al. Convergence and scalarization for data-parallel architectures//Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization. Shenzhen, China, 2013: 1-11

[75] Coutinho B, Sampaio D, Pereira F M Q, et al. Divergence analysis and optimizations//Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques. Galveston, USA, 2011: 320-329

[76] Collange S, Defour D, Zhang Y. Dynamic detection of uniform and affine vectors in GPGPU computations//Proceedings of the Euro-Par 2009—Parallel Processing Workshops. Delft, The Netherlands, 2010: 46-55

[77] AMD Accelerated Parallel Processing OpenCL Programming Guide 2.1. USA: AMD, 2012

[78] So H K H, Chen J, Yiu B Y S, et al. Medical ultrasound imaging: To GPU or not to GPU?. IEEE Micro, 2011, 31(5): 54-65

[79] Song S, Su C, Rountree B, et al. A simplified and accurate model of power-performance efficiency on emergent GPU architectures//Proceedings of the 27th International Symposium on Parallel & Distributed Processing. Cambridge, USA, 2013: 673-686

[80] Fung W W L, Aamodt T M. Energy efficient GPU transactional memory via space-time optimizations//Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. Davis, USA, 2013: 408-420

[81] Wang P H, Yang C L, Chen Y M, et al. Power gating strategies on GPUs. ACM Transactions on Architecture and Code Optimization, 2011, 8(3): 13

[82] Abdel-Majeed M, Wong D, Annavaram M. Warped gates: Gating aware scheduling and power gating for GPGPUs//Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. Davis, USA, 2013: 111-122

[83] Cebrián J M, Guerrero G D, Garcia J M. Energy efficiency analysis of GPUs//Proceedings of the 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. Shanghai, China, 2012: 1014-1022

[84] Jing N, Shen Y, Lu Y, et al. An energy-efficient and scalable eDRAM-based register file architecture for GPGPU//Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel, 2013: 344-355

[85] Xu Q, Annavaram M. PATS: Pattern aware scheduling and power gating for GPGPUs//Proceedings of the 23rd International Conference on Parallel Architectures and Compilation. Edmonton, Canada, 2014: 225-236

[86] Gu Y, Chakraborty S, Ooi W T. Games are up for DVFS//Proceedings of the 43rd Annual Design Automation Conference. San Francisco, USA, 2006: 598-603

[87] Mochocki B, Lahiri K, Cadambi S. Power analysis of mobile 3D graphics//Proceedings of the Conference on Design, Automation and Test in Europe. Munich, Germany, 2006: 502-507

[88] Wang Y, Roy S, Ranganathan N. Run-time power-gating in caches of GPUs for leakage energy savings//Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition. Dresden, Germany, 2012: 300-303

[89] Abdel-Majeed M, Annavaram M. Warped register file: A power efficient register file for GPGPUs//Proceedings of the 19th International Symposium on High Performance Computer Architecture. Shenzhen, China, 2013: 412-423

[90] Lin Yi-Song, Yang Xue-Jun, Tang Tao, et al. A GPU low-power optimization based on parallelism analysis model. Chinese Journal of Computers, 2011, 34(4): 705-716 (in Chinese)
(林一松,杨学军,唐滔等. 一种基于并行度分析模型的GPU功耗优化技术. 计算机学报, 2011, 34(4): 705-716)

[91] Gebhart M, Johnson D R, Tarjan D, et al. Energy-efficient mechanisms for managing thread context in throughput processors. ACM SIGARCH Computer Architecture News, 2011, 39(3): 235-246

HE Yan-Xiang, born in 1952, Ph. D., professor, Ph. D. supervisor. His research interests include trusted software, distributed parallel processing and high performance computing.

ZHANG Jun, born in 1978, Ph. D. candidate, associate professor. His research interests include high performance computing and computer architecture.

SHEN Fan-Fan, born in 1987, Ph. D. candidate. His research interests include trusted software, computer architecture and storage system.

JIANG Nan, born in 1976, Ph. D. candidate, lecturer. Her research interests include trusted software and trusted compilation.

LI Qing-An, born in 1986, Ph. D., lecturer. His research interests include compilation optimization and embedded system.

LIU Zi-Jun, born in 1990, M. S. candidate. His research interests include high performance computing and computer architecture.

Background

Thread scheduling optimization is an important way to improve the performance of General Purpose Graphics Processing Unit (GPGPU) and optimize the power efficiency of GPGPU. Many domestic and foreign research institutes and researchers do many researches on thread scheduling optimization aiming at the performance enhancement of GPGPU. Some of them are also relative with power optimization of GPGPU. But few of them analyze and compare all types of thread scheduling methods comprehensively. The main thread scheduling strategies on performance enhancement and power optimization of GPGPU are analyzed and compared in this paper. The interests of thread scheduling optimization that can be continued to study in future are also proposed in the end.

This research is funded by the National Natural Science Foundation of China (NSFC) project "Research on Theory and Methods of Green Compiler for Green Embedded Systems" under Grant No. 61373039 which aims at power optimization of embedded systems via a special green compiler framework. This paper is also supported by the NSFC project "Research on Optimization of Embedded System Based on Compilation" under Grant No. 61170022 which mainly solves the issues on storage, power efficiency and performance optimization via a special optimization framework of embedded systems. In addition, this paper is supported by the NSFC project "Key Techniques of High Throughput Many-Core Microarchitecture for Big Data Computing" under Grant No. 61462004 as well which focuses on several issues of the storage system and on-chip network on many-core processor.

Our research group mainly study how to improve the performance of processor especially of the storage system and optimize power efficiency and builds up a great deal of correspondent research results. This paper focus on analyzing and comparing all types of thread scheduling strategies aiming at improving performance and power efficiency of many-core processor which makes it correspond to the main tasks of these projects. The result can be referenced to be used to optimize the performance and power efficiency of the embedded system in terms of thread scheduling.