

一种易实现的 SIMT 调度模型分析

徐元旭, 张 超, 杨 兵, 喻明艳

(哈尔滨工业大学 电子信息与技术系, 黑龙江 哈尔滨 150001)

摘 要: 以图像处理器(GPU)为基础, 从中提取建立出单指令多线程调度模型, 该模型独立于 GPU, 功能与其类似, 但结构更简单, 更易于理解和实现, 方便向其他处理器结构中移植. 并对该调度模型进行性能分析, 分析结果揭示出该模型的特性, 对该模型的使用和向其他处理器中移植以及优化提供重要参考.

关键词: 多核处理器; 多线程并行处理; 单指令多线程; 调度模型; 性能分析

中图分类号: TP302.7

文献标识码: A

文章编号: 1000-7180(2014)01-0025-04

Analysis of a Kind of SIMT Scheduling Model Easy to Realize

XU Yuan-xu, ZHANG Chao, YANG Bing, YU Ming-Yan

(Department of Electronic Information and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract: This article, based on Graphic Processing Unit (GPU) picks up and builds a kind of single instruction multi-threads scheduling model. This model is independent of GPU, but is similar in function, has more simple architecture, is easier to be understood and realized, and is convenient to be transplanted into other processor architectures. This article also analyzes the performance of this scheduling model, which shows the attribute of this model and is an important reference for the use and optimizing of this model and transplanting into other processors.

Key words: multicore processor; multi-thread parallel processing; single instruction multi-threads; scheduling model; performance analysis

1 引言

高效提升处理器的性能成为了计算机产业发展的重点, 处理器体系结构的研究已经从传统的指令集并行探索转移到线程集并行开发, 多线程并行给处理器界带来了新的活力.

在多线程并行开发中, 并行线程如何高效调度执行, 成为工业界和学术界探讨的重点. 单指令多线程(SIMT)技术是实现多线程调度执行的热门选择, 尤其以 NVIDIA 公司为代表推出的图像处理器(GPU)所使用的 SIMT 多线程调度技术^[1], 因其对用户良好的可编程性和高效的多线程并行数据处理能力吸引了业界的高度关注, 其独特的线程调度体系结构在很多领域的性能已经超越了传统的多核处理器^[2], 将 GPU 中高效的 SIMT 调度技术广泛融合到其他处理器体系结构中, 将很大的提升处理器

的性能.

本文基于 GPU 的 SIMT 调度原理, 从中提取建立一个独立于 GPU 特殊结构便于向其他类型处理器体系结构移植的结构简单易实现的 SIMT 调度模型.

2 GPU 中 SIMT 调度原理

NVIDIA 公司的 GPU 中 SIMT 调度技术是在 CUDA 平台上实现的, 从 CUDA 平台上编写软件程序到最终执行在硬件结构上的多线程软硬件调度全过程比较复杂但是很高效.

在 CUDA 平台上编写的程序主要分为串行部分和并行部分^[3], 分别执行在 CPU 上和 GPU 上, 执行在 GPU 上的并行部分称为内核程序, 内核程序在发射入 GPU 硬件执行前会形成一组并行的线程组, 线程组在 GPU 中并行执行. 执行在 GPU 上的内核程序需要由程序员显示编写, 还需显示编写

一些准备程序,如存储空间的分配,数据从 CPU 存储空间向 GPU 存储空间的传递,线程数量,线程块的划分等.内核程序形成的线程组中所有线程所执行的代码完全相同,所有线程被统一调度管理.

在 GPU 中 SIMT 调度分两种过程,分别是软件调度过程和硬件调度过程,见图 1. 在软件调度过程中,内核代码形成一定数量的线程,线程的数目由程序员在编写时指定,软件调度将这组线程分成若干个线程块,线程块的数目也是由程序员在编程时显示指明,每个线程块中包含一定数量的线程,形成的这些线程块就是软件调度的结果,接着以这些线程块为单位进入硬件调度过程.在硬件调度过程中,线程块中的线程重新组合形成一组 warp, warp 就是 GPU 在硬件中调度执行线程的基本单位^[4],每个 warp 都含有一定数目的线程,一般该数目在 GPU 硬件结构中是固定值.在内核线程被执行时,整个线程块会被发射到 GPU 多线程并行执行硬件上,以 warp 为单位进行 SIMT 调度执行. warp 中线程指令进入指令缓存后被译码单元译码,每个线程取自己的操作数进入 ALU 执行,等待存储器数据访问的 warp 进入等待区等待,当 warp 中线程执行全部完成进入写回操作完成 SIMT 调度执行全部任务.

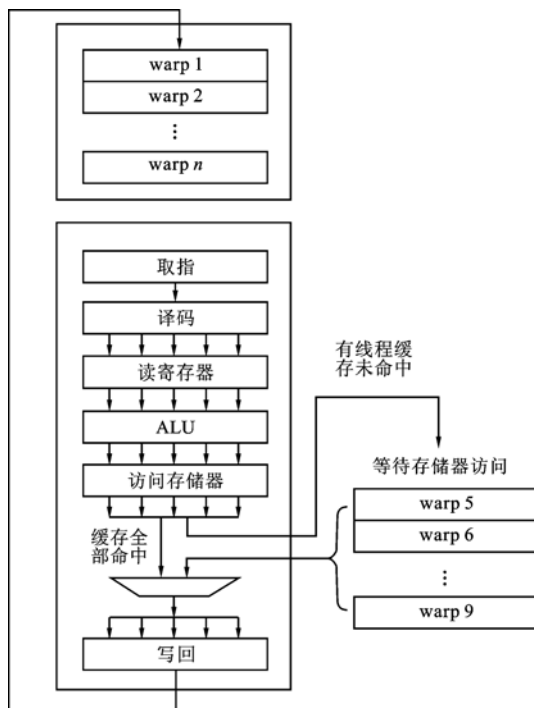


图 1 GPU 单指令多线程调度图

3 SIMT 调度模型

在分析了 GPU 中高效的 SIMT 调度原理后,

可以发现如果将这种技术在其他处理器中广泛应用不失为一种提升现有处理器性能的好方法,故本文将这种技术从 GPU 中提取出来,建立一种便于分析移植的调度模型,具体调度策略结构见图 2,虚线上半部分为 SIMT 软件调度过程,虚线下半部分为 SIMT 硬件调度过程,硬件调度采用 6 级流水线结构.

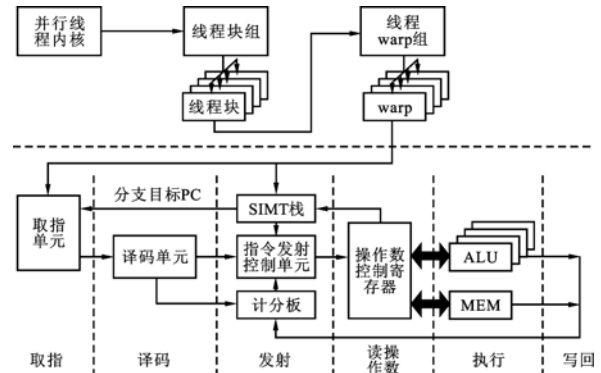


图 2 单指令多线程调度模型

软件调度阶段将大量的并行线程组织成众多的线程块组,每个线程块中的线程在软件调度下形成 warp 组,每个 warp 包含固定数量的线程,在软件调度过程中形成三个关键的数据结构,这三个数据结构中的一些数据会写入 SIMT 模型中硬件单元里的特殊存储单元中,成为组织多线程并行执行的关键参数.这三个关键数据结构分别是内核信息表(图 3(a)),线程信息表(图 3(b))和 warp 信息表(图 3(c)),各个信息表中对整个 SIMT 调度过程起到关键作用的参数已在图中标明.

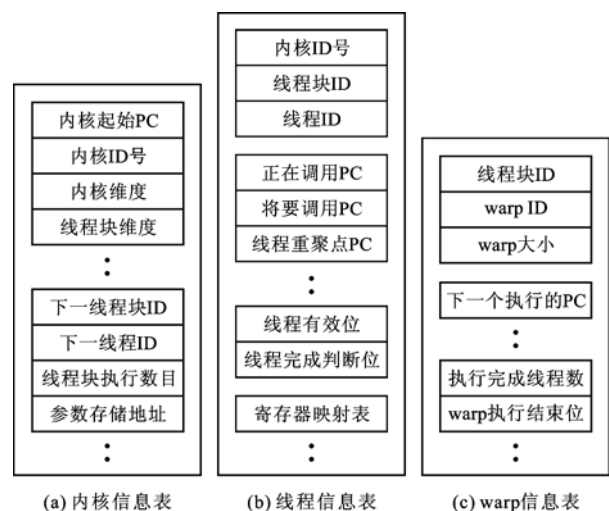


图 3 SIMT 软件调度关键数据结构图

图 3(a)中,内核起始 PC 是整个内核中所有线程指令在指令存储空间中的起始位置;内核维度是整个内核线程被分成的线程块数;线程块维度是一

个线程块中包含的线程数目;下一个线程块 ID 是即将执行的下一个线程块;下一个线程 ID 是即将执行的下一个线程;线程块执行数目是当前内核中正在被执行的线程数目,方便确定内核是否全部执行完成;参数存储地址是当前内核调用的参数数据所在的各自存储空间中的地址,在使用对应数据时方便数据的访问。

图 3(b)中,内核 ID 是当前线程所属内核号,线程块 ID 是当前线程所属块号;线程重聚点 PC 用于 SIMT 栈处理多线程分叉^[5];寄存器映射表是当前线程使用到的每一个寄存器从体系结构寄存器到物理寄存器的映射信息。

图 3(c)中,线程块 ID 标明当前 warp 所属的线程块 ID 号;warp 大小是当前 warp 中包含的线程数;执行完成线程数标明当前 warp 中已经完成的线程数目;warp 执行结束位用于指示 warp 中所有线程是否全部执行完成。

软件调度形成 warp 组之后,接下来以 warp 为单位,将 warp 中的所有线程送入硬件调度阶段。硬件调度阶段的 6 级流水如下:

(1)取指阶段,见图 4,取指单元从 warp 中取出将要执行的线程指令,将其放入指令缓存 I-Cache 中,译码单元从缓存中选取指令。

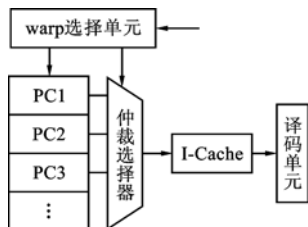


图 4 取指单元

(2)译码阶段,见图 5,对所取的线程指令进行译码,并放入指令缓冲单元中等待指令的发射,每个 warp 对应唯一一个指令缓冲单元,每个指令缓冲单元中存放准备发射的已经译码完成的指令,每个指令前都有一个有效位,表示当前被译码完成的指令是否可以准备发射,该模型中每个 warp 的指令缓冲单元缓冲两条被译码完成的指令。译码器同时将相关信息送入到计分板中,在指令发射时期使用。

(3)发射阶段,指令发射控制单元控制指令的发射,读取 SIMT 栈中的 warp 线程的信息和计分板中的相关线程信息决定当前的指令是否可以发射,其中 SIMT 栈和计分板是整个 SIMT 模型中多线程调度的核心。计分板的作用是检查写后写(write after write, WAW)和写后读(read after write, RAW)

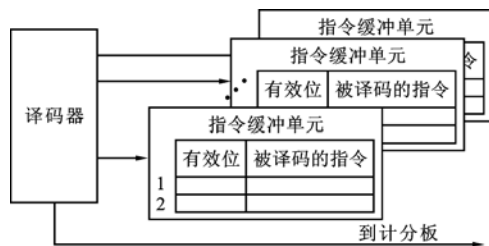


图 5 译码单元

冒险的,其中保存的信息是对应 warp 中所使用的寄存器号,指令发射控制单元在发射指令时会检查计分板是否存在寄存器相关冒险。SIMT 栈用于提升线程调度效率,避免线程分支叉造成的影响,每个 warp 都有自己的 SIMT 栈,组织自己内部线程,其结构如图 6 所示。

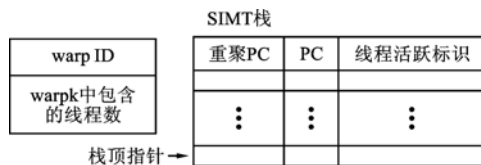


图 6 SIMT 栈

(4)读操作数阶段,负责 warp 中并行线程的独立寄存器访问和操作数选择,为每个线程分配对应的操作数,从这个步骤开始 SIMT 模型的单指令多线程模型的多线程并行特性开始显现。

(5)执行阶段,各线程开始并行在各自 ALU 中执行。SIMT 调度模型的存储器设备中有一特殊单元,称为共享存储单元,由一个线程块中的并行线程共同拥有,即同一线程块中的线程共享该存储单元中的数据,但是不同线程块中的线程不共享该共享存储单元,这是 GPU 高效调度技术之一,本文亦将这种技术应用在建立的 SIMT 模型中,如图 7 所示。

(6)写回阶段,这是整个硬件调度执行部分的最后一步,将相关结果写回到对应寄存器中。标志着流水级执行结束,线程指令执行完成。

4 SIMT 调度模型性能分析

SIMT 调度模型一个重要特点是能充分利用线程的数目来隐藏线程中存储器访问和其他问题导致的线程执行流水线停滞带来的延时,也即当某 warp 中的线程因为相关原因停滞时,其他 warp 中的线程能填充到流水线中,使 SIMT 流水线继续执行,故模型执行过程中线程的组织管理造成模型中线程并行数目的变化会对整个模型的性能产生影响。线程组织管理中对线程数目影响最大的是线程块大小(block)、线程 warp 大小和硬件中运行的线程块上

限数目(cta). 这三者彼此之间具有相关性,共同作用影响模型性能.

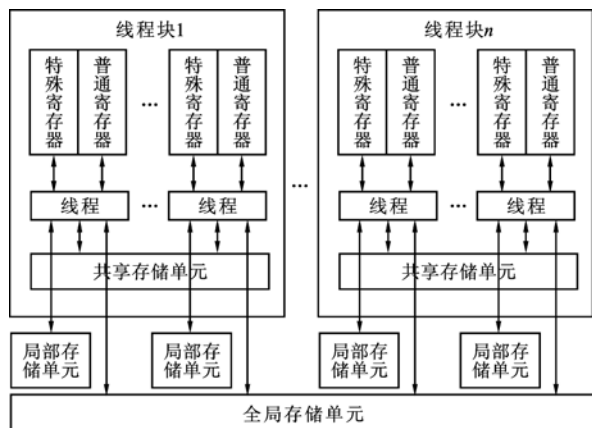


图7 SIMT 调度模型存储设备访问图

线程块大小是SIMT模型在使用时编程人员为每个线程块中设定的包含的线程数目,在SIMT调度执行中,会以线程块为单位向SIMT硬件执行部件上发射线程,故线程块大小会影响SIMT硬件上发射的线程数目.warp大小是SIMT硬件从发射来的线程块中选取一定数量的线程组织在一起,以warp为单位从线程块中取线程调度执行在SIMT的执行单元中,且模型中warp的大小同执行单元数目相关,故warp的大小影响SIMT执行单元中的线程数目,同时线程块的大小和warp的大小都会影响线程块组织成warp的个数,影响SIMT的调度执行状态.cta大小是SIMT硬件允许发射的线程块数目的最大值,该值变化会导致整个SIMT硬件中总并行线程的数目,影响其性能,同时硬件中总线程数目的变化同硬件中寄存器资源紧耦合,后者会影响前者行为,对SIMT性能产生影响.

本文已经介绍过,SIMT调度模型中有个特殊存储部件,即共享存储器,该部件的使用取决于编程人员,由于该部件属性特殊,使用与否会对线程块中的并行线程的存储访问特性产生影响,继而影响SIMT调度执行性能^[6].

5 结束语

本文基于GPU的SIMT调度原理,从GPU中提取并建立出一个独立于GPU特殊结构的SIMT调度模型,该模型结构更简单,易于理解,方便向其他处理器体系结构中移植.

参考文献:

- [1] Luebke D, Humphreys G. How GPUs work[J]. Computer, 2007, 40(2): 96-100.
- [2] Victor W Lee, Changkyu Kim. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU [C]// 37th International Symposium on Computer Architecture (ISCA'10). France, 2010.
- [3] Nvidia. Nvidia CUDA C programming guide version 3.2 [M/OL]. [2013-04-10]. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf.
- [4] Wilson W L Fung, Ivan Sham, George Yuan, et al. Dynamic warp formation and scheduling for efficient GPU control flow [C]// 40th Annual IEEE/ACM International Symposium on Microarchitecture. Chicago, IEEE Press, 2007: 407-420.
- [5] Jiayuan Meng, David Tarjan, Kevin Skadron. Dynamic warp subdivision for integrated branch and memory divergence tolerance [C]// 37th Annual International Symposium on Computer Architecture (ISCA'10). France, Saint-malo, 2010.
- [6] Al-Kiswani S, Gharaibeh A, Santos-Neto E, et al. Store GPU: exploiting graphics processing units to accelerate distributed storage systems [C]// Proc. 17th Int'l Symp. on High Performance Distributed Computing. San Francisco, 2008: 165-174.

作者简介:

徐元旭 男, (1987—), 硕士. 研究方向为高性能计算机体系结构.