ARTICLE IN PRESS

Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Locality based warp scheduling in GPGPUs

Yang Zhang *, Zuocheng Xing, Cang Liu, Chuan Tang, Qinglin Wang

Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, 410073, China

HIGHLIGHTS

- An analysis on the drawbacks of the existing warp schedulers is taken.
- A novel reordering method to maintain the time locality is taken.
- A warp scheduling scheme called LPI is taken. It allows overlapping of long- and short-latency warps to preserve locality.
- LPI can get an improvement of 2.2% over the baseline.

ARTICLE INFO

Article history:
Received 29 February 2016
Received in revised form
12 February 2017
Accepted 21 February 2017
Available online xxxx

Keywords:
GPGPU
Warp scheduling
Locality
Reordering

ABSTRACT

As the need for high performance computing continues to grow, it becomes more and more urgent to design a massive multi-core processor with high throughput and efficiency. However, when the number of cores keeps increasing, the capacity of on-chip memory is always insufficient. In a multi-core processor such as GPGPU (General Purpose Graphic Processor Unit), dozens or hundreds of SMs (Stream Multi-processor) coordinate to gain high throughput with several MB on-chip memory. Furthermore, in one SM, thousands of threads are organized as thread blocks to process instructions in a SIMT (Single Instruction Multiple Threads) manner. As all the threads share the same on-chip memory, the mismatch between large core number and small on-chip memory capacity can easily impair the performance due to excessive thread contention for cache resource.

An efficient thread scheduling method is a promising way to alleviate the problems and to boost performance. From the hardware perspective, the instructions are executed by warps which are made up by a fixed number of threads. So we propose a novel warp scheduling scheme to maintain data locality and to relieve cache pollution and thrashing issues. First, to make full use of time locality, we put the disordered warps into a supervised warp queue and issue the warps from oldest to youngest. To utilize space locality and to hide computation unit stalls, we put forward a new insertion method called LPI (Locality Protected Insertion) to reorder warps in the supervised warp queue to better hide long-latency warps with short-latency warps such as ALU operations and on-chip accesses. Over a wide variety of applications, the new scheduling method gains at most 10.1% and an average of 2.2% improvements over the baseline loose round-robin scheduling.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, GPUs have been a more and more powerful and energy efficient platform for High Performance Computing (HPC). As a kind of stream multi-processor, it is featured with thousands of processors and few control units. For example, state-of-the-art NVIDIA TITAN X with Maxwell architecture has up to 3072 cuda cores and several MB on-chip memory [1]. It relies on massive cores to tolerate memory latency and to deliver high throughput. On one hand, so many cores meet the demand for massive parallel computing, on the other hand, it inevitably introduces a large number of accesses to the memory and causes cache contention and thrashing. As a result, the need for a more intelligent scheduling strategy is much more urgent for the architecture designers.

The early generations of NVIDIA GPU (for example GT200) do not include L1 data cache as on-chip memory for simplicity. The capacity of the only existed scratchpad memory is far from enough, especially for some memory intensive applications. Moreover, the capacity of the scratchpad memory is designated by the programmer at design time and it is not appropriate for the applications with diverse access patterns, which prefer cache than

* Corresponding author.
E-mail address: zhangyang@nudt.edu.cn (Y. Zhang).

http://dx.doi.org/10.1016/j.future.2017.02.036
0167-739X/© 2017 Elsevier B.V. All rights reserved.

Please cite this article in press as: Y. Zhang, et al., Locality based warp scheduling in GPGPUs, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.02.036

scratchpad memory. Therefore, the later generations introduce L1 data cache which shares a memory array with shared memory. In the newest Maxwell architecture, the shared memory has an exclusive memory array and the L1 data cache merges into the texture cache [1]. As the computing ability of new architectures upgrade so fast, we are not able to follow up the features of the newest GPGPU architecture. So we take the Fermi architecture for our target and the scheduling strategies applied on it can be popularized to other new architectures.

Recent studies mainly focus on the decrease of the access to the on chip memory through bypassing and throttling [2–8]. Bypassing is used when the system detects the memory is full of data and any new data insertion will destroy the locality of the memory. So the data skip the memory level and goes into the upper memory level. Throttling is a hotspot in recent years. It can release the cache contention and thrashing by fundamentally decreasing the number of blocks on the cores. There are other studies to research on data locality [9] and reuse distance in GPU [10]. However, L1 data cache is hard to make good use of. For the cache sensitive applications, the enormous thread count makes cache difficult to preserve locality. So we need to apply an intelligent scheduling strategy to reserve the locality.

In addition to locality preserving, latency hiding is also very important in GPGPUs. Preserving data locality can reduce long memory access time and latency hiding can reduce pipeline stalls. Previous works mainly focus on latency hiding. But few work combines latency hiding with locality preserving when scheduling. We design a scheduling method to use warp classification for latency hiding as two-level scheduling does and add warp insertion to make successive warps not too far away from each other for locality preserving.

This paper mainly makes the following contributions:

- We take an analysis on the current warp scheduling methods and show the drawbacks of them. We focus on the way to preserve locality and hide latency at the same time to solve the problems in the existing scheduling methods.
- We propose a novel reordering method to maintain the time locality. This scheme reorders warps from oldest to youngest and puts some especially long-latency warps to the back of the queue.
- Based on the above reordered warp queue, we further propose a scheduling scheme called LPI to allow better overlapping of long-latency warps with short-latency warps to regain localities lost by two-level scheduling and to well hide computation unit stalls.
- We evaluate LPI on a simulated Fermi architecture and an average of 2.2% improvement over the baseline can be achieved over a variety of applications with different characteristics.

2. Background and motivation

2.1. Baseline architecture

A modern unified GPU consists of multiple stream multiprocessors (SMs) or computation units (CUs). An SM runs instructions in a manner of Single Instruction Multiple Threads. At each cycle, an SM fetches and decodes an instruction in the front end, which is further dispatched to multiple warps in the warp pool. After the dispatch, the scoreboard will check the hazard. If a warp (typically 32 threads) has passed the scoreboard, it is qualified to be issued to execute. The instruction execution is performed in Single Instruction Multiple Data (SIMD) lanes, which constitute the execution units. The execution unit is much like a vector processor, that works with a lot of computing resources. As multiple instructions share the same fetch and decode stage, lots of hardware expenses are saved.

Since the performance of modern GPU is always restricted by the bandwidth and the memory latency, it presents a memory system with multi-level to make use of data locality to make the memory access more efficient. As we can see in Fig. 1, in each SM, register file, L1 cache and the shared memory are fast on-chip memories and they belong to the only SM. If one instruction fails to hit on chip, the requests are pushed to L2 cache in the memory partition through memory port and interconnect. The requests are further transferred to the global memory unless the requests hit in the L2 cache. It needs to be emphasized that the on-chip memories are private to each SM but L2 cache and global memory are not private. Because all the blocks in different SMs can read and write data in the global memory, data can be exchanged in the global memory [1].

A kernel executes the GPU code through massive threads, the number of which can be assigned by the programmer. Numerous threads are organized hierarchically. A kernel is consist of several thread blocks which are formed by many threads. A large workload can be first divided into a few blocks and then into many threads. Blocks can communicate through the global memory. But only the blocks in one SM can share data in the shared memory. Threads in blocks execute 32 threads in lockstep which is called a warp. The threads in a block run in an SIMT (Single Instruction Multiple Threads) model and they can synchronize among themselves through barriers.

Nowadays, GPUs are increasingly used in broader application domains where memory access patterns are both hard to analyze and to manage in software-controlled caches. To optimize these applications, we need to find effective ways to make use of the considerable parallelism and the limited cache.

2.2. GPU scheduling method

The GPU scheduling method has a great effect on GPU performance. With the LRR (Loose Round-Robin) scheduling policy, all the warps assigned to a core are given equal priority and are executed in a round-robin fashion. This scheduling policy is the simplest with the least scheduling expenses and can meet the demand of most regular programs. But for the many irregular programs, especially those with intensive memory access, the LRR scheduling policy is always inefficient. That is because for many memory-intensive applications, most of the warps arrive at long-latency memory operations roughly at the same time. The burst of long-latency memory operations leads to core stalling because there are no short latency warps to hide the long latencies. That is the primary cause of inefficiency for modern GPUs.

The two-level warp scheduling is an optimization to the baseline LRR scheduling and we take an implementation of two-level scheduler similar to that in [11]. The scheduling strategy attempts to hide two distinct sources of latency in the system: (1) long operations, often unpredictable latencies, such as waiting to be initialized with a kernel, synchronization, atomic operations, texture operations, access to the DRAM and so on; and (2) short operations, often with predictable and fixed latencies, such as ALU operations and access to the on-chip memory. In the scheduler, an active pool is created to hold warps with short operations and a relatively large warp to hold warps with long operations. In each pool, warps are scheduled in round-robin and the pending pool will not be scheduled until all the active warps have finished. However, it may cause performance decrease on the condition that two continuous warps are separated when they have data locality. In our implementation, we take a similar two-level division to the warps to hide long-latency warps with short-latency ones.

Please cite this article in press as: Y. Zhang, et al., Locality based warp scheduling in GPGPUs, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.02.036

Download English Version:

https://daneshyari.com/en/article/6873226

Download Persian Version:

https://daneshyari.com/article/6873226

Daneshyari.com