Multi-Application Concurrency in GPU

Summer Project Report

Submitted in partial fulfillment of the requirements
for the summer project

by

Navneet

(Roll No. 200070048)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering
Indian Institute of Technology Bombay
October 2020

Acknowledgement

I express my gratitude to my guide Prof. Virendra Singh for providing me the opportunity to work on this topic.

Navneet
Electrical Engineering
IIT Bombay

Contents

List of Figures

Chapter 1

Introduction

1.1 Concurrent execution of Kernels in GPU

Most of the GPUs run single kernel at a time but this lead to resource under utilization. Executing concurrent kernels allow resource sharing but it also increase the complexity in scheduling and execution of kernels

1.1.1 Resource Sharing

To allow resource sharing among different kernels the execution has to be multiplex in space or time
Time Multiplexing - Generally time multiplexing runs kernels one by one without considering concurrency. If two or more kernels are run concurrently and GPU switches among them then the overhead of time multiplexing grows significantly.

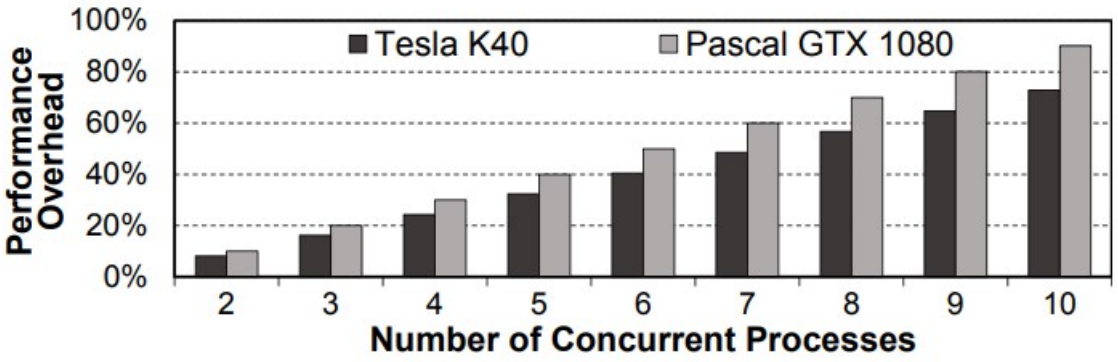

Figure 1.1: Time Multiplexing Overhead

Spacial Multiplexing- Another method of resource sharing in via space multiplexing. This allow to co-schedule kernels but also sacrifice memory protection by merging kernels in a single address space. To simplify this generally resources are partitioned statically prior to program execution. As a result, these systems cannot adapt to changes in demand at runtime, and, thus, can still leave GPU resources underutilized.

1.1.2 Virtualization

To overcome these shortcomings efficient mechanisms for virtualization is needed.
Current state-of-the art GPU implement virtualization but that is not very efficient. Per-

formance of some of the baseline designs are shown below:

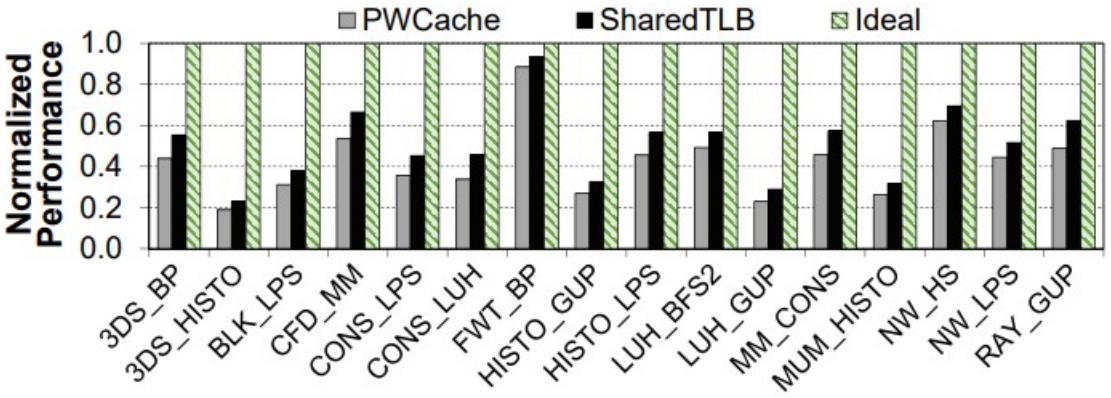

Figure 1.2: Performance of state of the art GPU

1.2 Effect of poor Virtualization

1.2.1 Effect of TLB Misses

TLB carry the virtual to physical address translation information. If one warp address translation request is stalled then their is a high probability that multiple warps executing similar code will get stalled. This affect multi threading significantly and reduce the performance.

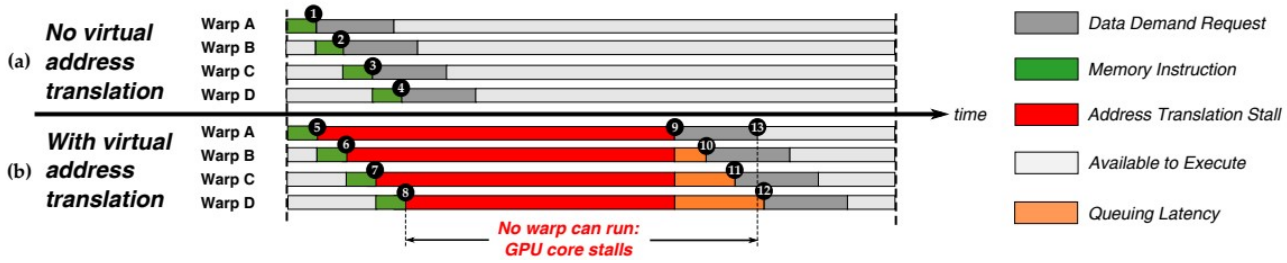

Figure 1.3: Bottleneck created by TLB miss

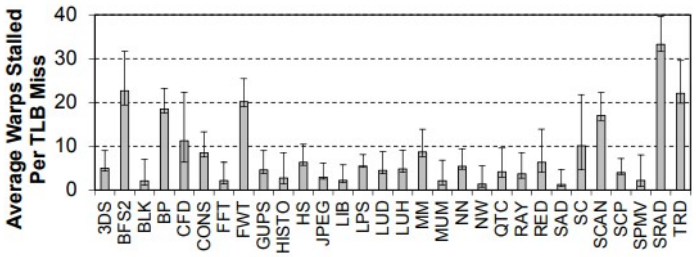

Figure 1.4: Average warp stall per tlb miss

Figure 1.3 and 1.4 shows that Address translation stall can stall multiple warps at once

1.2.2 TLB Interference Effect

When multiple application execute concurrently the address translation overhead increase due to inter-address-space interference. Inter-address-space address translation request

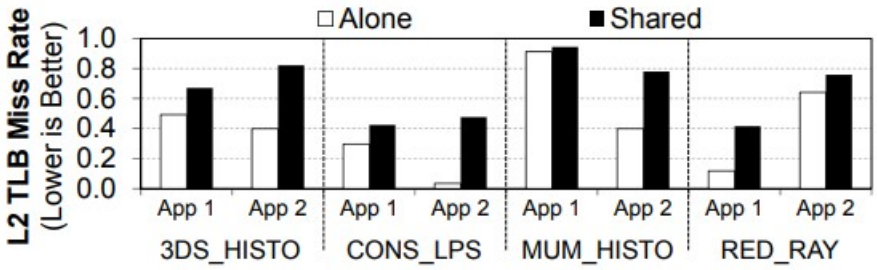

Figure 1.5: Single vs Multiple Application Miss rate

induce thrashing at TLB.Figure 1.5 shows the L2 TLB miss rate difference when only one application is run as compared to when two applications are ruun concurrently

1.2.3 Interference at the Shared Data Cache

When a page table walk hits in the shared L2 cache, the cache hit has the potential to help reduce the latency of other warps that have threads which access the same page in memory. However, TLB-related data can interfere with and displace cache entries housing regular application data, which can hurt the overall GPU performance. It is observed that translation data from page table levels closer to the page table root are more likely to be shared across warps, and typically hit in the cache. For a 4-level page table, the data cache hit rates of address translation requests across all workloads are 99.8%, 98.8%, 68.7%, and 1.0% for the first (root), second, third, and fourth levels of the page table, respectively. This means that address translation requests for the deepest page table levels often do not utilize the cache well.

1.2.4 Interference at Main Memory

s. We see that even though address translation requests consume only 13.8% of the total utilized DRAM bandwidth(2.4% of the maximum available bandwidth), their average DRAM latency is higher than that of data demand requests. This is undesirable because address translation requests usually stall multiple warps, while data demand requests usually stall only one warp.The higher latency for address translation requests is caused by the FR-FCFS memory scheduling policy. so a scheduler that cannot distinguish address translation requests from data demand requests effectively de-prioritizes the address translation requests, increasing their latency, and thus exacerbating the effect on stalled warps.

Chapter 2

Literature Survey

2.1 Multi-Address Space Concurrent Kernels (MASK)

MASK focus on the aspect that poor virtualization affect multi-application concurrency significantly, so the aim is to improve hardware virtualization. There are 3 key modification that mask suggest to make the entire memory hierarchy aware of the address translation request and prioritizing it over data demand request and tries to prevent multiple warps stall.The 3 modifications are:-
TLB fill token
Address translation aware L2 cache bypass
Address space aware memory scheduler

2.2 Proposed Idea

2.2.1 TLB Fill Token

Warps are assigned tokens so that the can put their request in the l2 TLB, warps without token can't fill the l2 TLB but the can probe L2 TLB. This reduce thrashing at L2 TLB and eventually lead to increase in TLB hit rate.

2.2.2 Address Translation Aware Bypass

This block aims at bypassing the address translation request from a lower level of page table. It repeatedly check for the hit rate of the levels of page table and if the hit rate is lower that total hit rate of L2 cache that request is bypassed. This reduce misses of

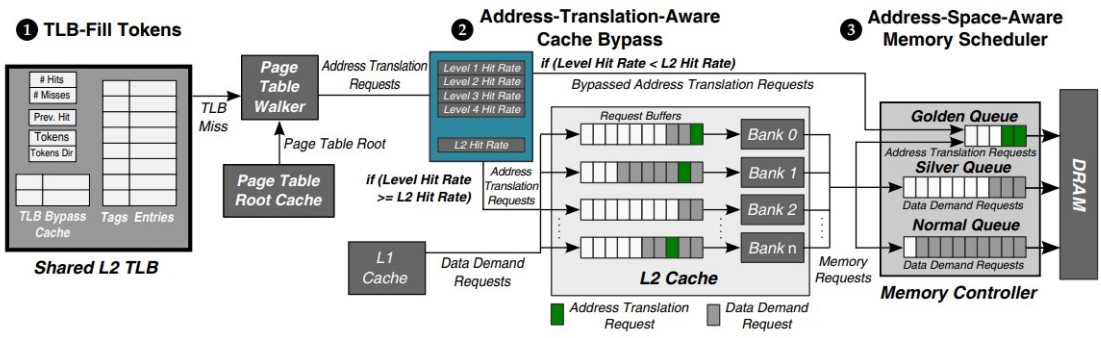

Figure 2.1: Mask Design

lower level PTEs as the hit rate is quiet low for such levels. This also allow to keep more application related data in L2 cache.

2.2.3 Address Space Aware Memory Scheduler

This step separate the address translation request from data demand request. Separate queues are maintained like golden queue for address translation request, silver and normal queue for data demand request. To provide higher priority to applications that are likely to be stalled due to concurrent TLB misses, and to minimize the time that bandwidth-heavy applications have access to the silver queue, each application takes turns being assigned to the Silver Queue based on two per-application metrics: (1) the number of concurrent page walks, and (2) the number of warps stalled per active TLB miss.

2.3 Evaluation

To evaluate the model 35 application-pairs are divided into three workload categories based on the number of applications that have both high L1 and L2 TLB miss rates, as high TLB miss rates at both levels indicate a high amount of pressure on the limited TLB resources. n-HMR contains application-pairs where n applications in the workload have both high L1 and L2 TLB miss rates.

L1 TLB Miss Rate	L2 TLB Miss Rate	Benchmark Name
Low	Low	LUD, NN
Low	High	BFS2, FFT, HISTO, NW, QTC, RAY, SAD, SCP
High	Low	BP, GUP, HS, LPS
High	High	3DS, BLK, CFD, CONS, FWT, LUH, MM, MUM, RED, SC, SCAN, SRAD, TRD

Figure 2.2: Categorization of workload

The performance of MASK is compared against (1) Static, (2) PWCache Model, (3) Shared TLB Model, (4) Ideal, which represents a hypothetical GPU where every TLB access is a hit. The performance of the individual components of MASK: TLB-Fill Tokens (MASK-TLB), Address-Translation-Aware L2 Bypass (MASK-Cache, and AddressSpace-Aware DRAM Scheduler (MASK-DRAM) is also reported.

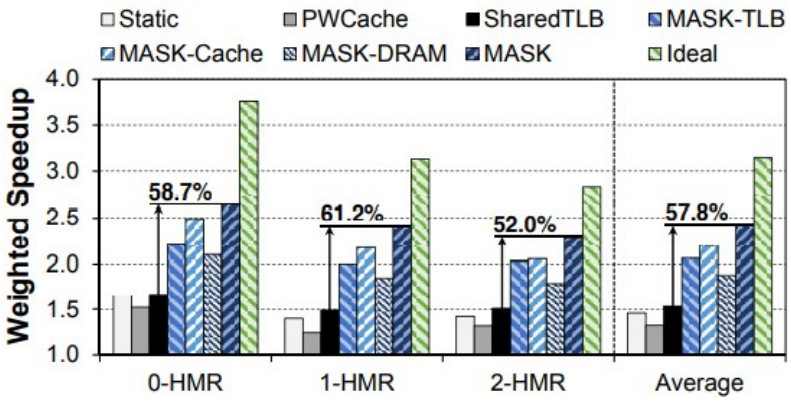

Figure 2.3: Multiprogrammed workload performance

MASK provides significant performance improvements over both PWCache and SharedTLB regardless of the workload type (i.e.,0-HMR to 2-HMR). This indicates that MASK is effective at reducing the address translation overhead both when TLB contention is high and when TLB contention is relatively low

2.3.1 Component analysis

Effectiveness of TLB-Fill Tokens- MASK uses TLB-Fill Tokens to reduce thrashing. MASK-TLB increases shared L2 TLB hit rates by 49.9% on average over SharedTLB , because the TLB-Fill Tokens mechanism reduces the number of warps utilizing the shared L2 TLB entries, in turn reducing the miss rate.

Effectiveness of Address-Translation-Aware L2 Bypass - MASK uses Address-Translation-Aware L2 Bypass with the goal of prioritizing address translation requests Address-Translation-Aware L2 Bypass minimizes the impact of long L2 cache queuing latency [16], leading to a 43.6% performance improvement compared to SharedTLB

Effectiveness of Address-Space-Aware DRAM Scheduler - To characterize the performance impact of MASK's DRAM scheduler,the DRAM bandwidth utilization and average DRAM latency of (1) address translation requests and (2) data demand requests is compared for the baseline designs and MASK. MASK is effective at reducing the DRAM latency of address translation requests, which contributes to the 22.7% performance improvement of MASK-DRAM over SharedTLB

Chapter 3

Future Plan

I have studied basics of GPGPU, SIMT core, and programming related to GPGPU. I have also completed reading the paper on MASK, and get myself familiar with GPGPU Sim by running various benchmarks. Also, I tried to implement MASK on GPGPU using the Mosiac simulator.
In, coming weeks I will try to model the Mosiac simulator on top of GPGPU since the available one is very old and has outdated requirements.

3.1 End Goal

The end goal of my project in to implement MASK on GPGPU and compare the results.