3J-02

Reducing Thread Divergence in GPU Applications through Memory Partitioned Streams

Oki Yoshitake† Osaki Keiji†

International Christian University Graduate School†

1. Introduction

The colossal computing capabilities of graphics processing units (GPUs) have increasingly emerged as a powerful tool for high performance computing platforms. However, the parallel architecture of GPUs has exposed performance issues under conditional branch scenarios commonly seen in GPGPU applications, such as the Monte Carlo simulation of photon migration in Multi-Layered media (MCML). The lack of complex branch interpreters on GPUs forces the multi-core hardware to execute thread-level divergence codes serially, inflicting serious performance degradations.

This paper introduces a mechanism for eliminating thread divergence through CUDA Streams on the NVIDIA CUDA programming model [1]. This software-level optimization remaps threads that take different paths to alternate Streams, allowing divergent codes to potentially overlap and result in performance improvement.

2. Control Flow Divergences

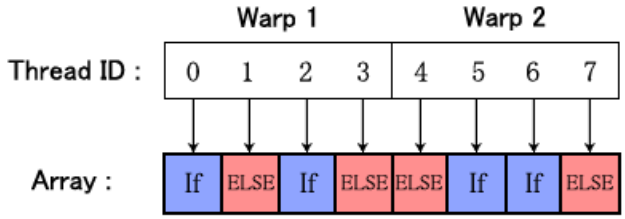


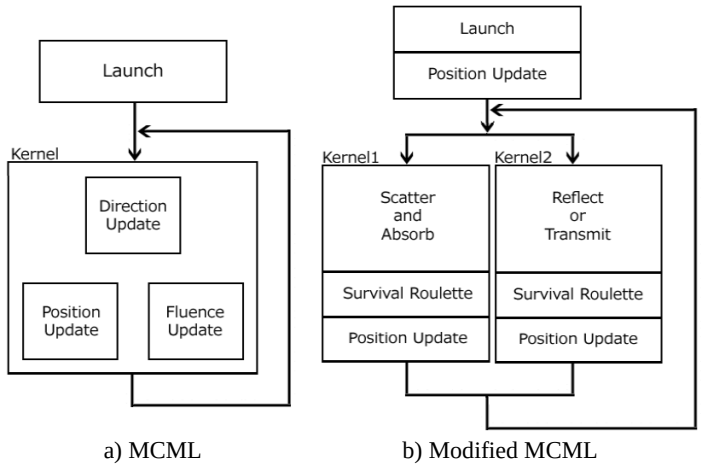
Figure 1. Divergence in Warps

2.1 SIMD Architecture

The architectural philosophy of GPU that inherit single instruction multiple data (SIMD) architecture requires that a bundle of threads, or warps, to run a single issued instruction in a lock-step fashion. As shown in Figure 1, control flow divergence often occur in scenarios where some threads execute different If-else paths of a divergent branch from other threads that reside in the same warp. Thread divergence within warps forces the divergent paths to be executed serially, creating stalled and idle threads during execution Control flow divergences are known to cause significant performance degradation due to its irregular load balancing between threads that take different paths. This irregular workload limits and underutilizes the available GPU resource, and is a main source of bottlenecks for various GPU applications. Given the advantages of parallelism on SIMD architectures, an efficient control flow optimization mechanism is necessary.

3. MCML Simulation

A typical GPU application that suffers such irregular control divergences is MCML. MCML is a Monte Carlo method for modeling steady state light transport in multi-layered media. MCML on GPUs begin by launching and injecting millions of photon packets into a multi-layered media, where each photon packet motion corresponds to a work done by a single thread. A possible action for each photon packet at every time step is a direction update, position update, or a fluency update. Figure 2a depicts the overall flow of the photon packets. Each action is dependent on the outcome of a Pseudo Random Number Generator. Since each photon, or the corresponding thread, has its own random number sequence, the propagation of each thread within a warp is a prime source of control flow divergence.



a) MCML b) Modified MCML

Figure 2. Graph a) shows the flow of the original MCML, graph b) shows the modified flow of MCML

4. Related Works

4.1 Hardware Optimizations

Dynamic Warp Subdivision rearranges threads within warps into subdivided warps [2]. This redirection of threads aims to hide latency by occupying multiprocessors with newly divided non-divergent warps. Dynamic Warp Formation combines threads from multiple warps that suffer from thread divergence [3]. This optimization regroups any threads within the whole SIMD scope by matching warps that share the same Program Counter (PC).

4.2 Software Optimizations

G-Streamline is a framework that integrates a job swapping mechanism of threads along with data layout transformations [4]. This method aims to remove warp-level irregularities and improve coalesced memory access.

5. CUDA Streams for Removing Irregularity

In this work, a practical optimization solution is presented to eliminate control flow divergences through CUDA Streams. CUDA Streams are a sequence of operations deployed by the host CPU that can execute simultaneously on the device GPU. By exploiting the concurrency mechanism of CUDA Streams, kernels with branch divergences can be split and launched simultaneously on the GPU. The modified flow of this optimization for MCML is shown in Figure 2b.

5.1 Splitting Divergent Kernels

The main idea for utilizing CUDA Streams to eliminate branch divergence is to divide the divergent path into two independent kernels, as shown in Figure 3. After all photon packets are initialized at the initialization kernel, work will be split into two new kernels. Each divided kernel is only responsible for a single path in the original divergent code, and will reference the GPU memory that is divided according to the two corresponding CUDA Streams. Furthermore, the two independent kernels created under different CUDA Stream instances are executed simultaneously.

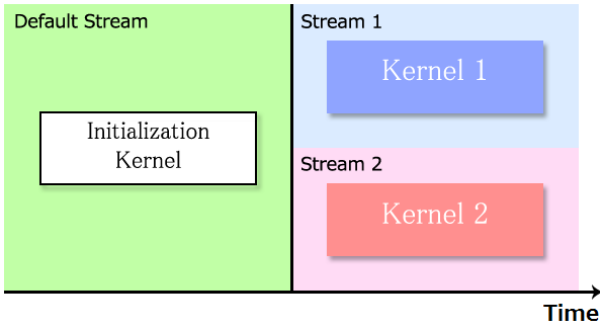


Figure 3. Kernel1 and Kernel2 executed under different CUDA Streams

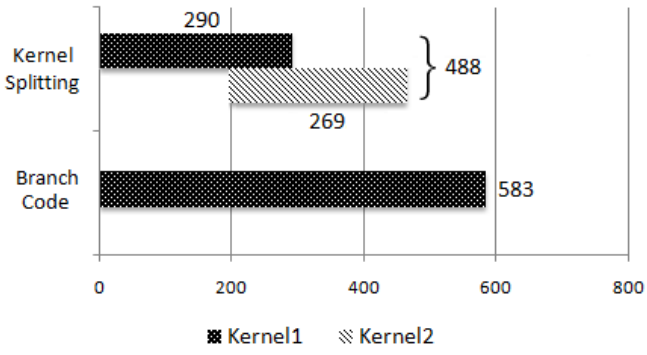


Figure 4. Execution time of kernel splitting with CUDA Stream and the original branch code in microseconds

5.2 Analysis

Rearranging the main kernel into two non-divergent kernels reduces the serialization of branch condition statements as well as the amount of registers required per thread, preventing thread-level resources from limiting the occupancy of the SIMD multiprocessors.

The results shown in Figure 4 does not reflect a full implementation of the presented MCML optimization, but a test case for simulating a single iteration step with 100,000 photon packets. The results show the two split kernels working in an overlapping fashion, leading to a 1.2x speedup from the original execution. This benefit will be obtained for every subsequent time step of the simulation, resulting to a more significant speedup to the overall execution. Unlike inter-warp or intra-warp thread rearrangements done in previous studies, this kernel splitting optimization allows multiprocessors to execute both kernels without any transformations prior to each time step, minimizing overhead for every kernel launch.

6. Conclusion and Future Work

In this paper, an optimization technique is proposed to eliminate branch divergences and improve control flow irregularities. The method employs CUDA Streams to allow kernels to run each path of the divergent branch concurrently. After splitting the kernel, warps can exploit hardware resources without being bounded by registers or transformation overheads. For future work, the optimization should be fully implemented along with an efficient mechanism for improving photon packet data movements between resources for every iteration step.

7. References

[1] NVIDIA Corporation. NVIDIA CUDA Programming Guide, 6.5 edition, 2014.

[2] W. Fung, I. Sham, G. Yuan, and T. Aamodt. Dynamic warp formation and scheduling for efficient gpu control flow. In MICRO, 2007.

[3] J. Meng, D. Tarjan, and K. Skadron. Dynamic warp subdivision for integrated branch and memory divergence tolerance. In ISCA, 2010

[4] E. Z. Zhang, Y. Jiang, Z. Guo, K. Tian, and X. Shen. On-the-fly elimination for dynamic irregularities for gpu computing. In ASPLOS, 2011.