Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021, 8(11):95-98

Research Article ISSN: 2394 - 658X

Latency Throughput Linear Analysis for GPU Architecture Pipeline

Apoorva Reddy Proddutoori

San Diego
Email: apoorvaproddutoori@gmail.com

ABSTRACT
Recent focus has been on eliminating redundant network worload through content-based cracking and utilizing low latency components, using a computationally heavy algorithm. A GPU-based implementation of this algorithm is proposed in this paper, with optimization strategies to enhance performance and throughput of the pipeline architecture. Despite ongoing research on GPU-to-GPU communication, achieving performance on multi-GPU systems remains a challenge due to data workload transfer latency and memory access or allocation issues. Additionally, GPU parallel computation with memory allocation optimization shows significant improvements in fast Fourier transform calculation speed, with ratios better when compared to CPU speeds.

Key words: GPU, latency, throughput, pipeline architecture, memory allocation

---

INTRODUCTION

The elimination of redundant network traffic has recently attracted much attention from both academia and industry. A key challenge and enabling technique in implementing redundancy elimination is content-based cracking, which typically involves a computationally algorithm. In this paper, we propose a GPU-based implementation of fingerprinting to solve this problem. Various optimization strategies are proposed to maximize performance, such as efficient buffer management, GPU memory hierarchy optimization, and balanced load distribution by exploiting hardware characteristics or solving domain-specific challenges. Extensive evaluations of both overall and microscopic performance reveal the effectiveness of the GPU-accelerated Rabin fingerprinting algorithm, and we can achieve up to % Gpbs of performance on a graphics card. Performance shows a better speedup compared to the latest equivalent hardware. Although some optimization schemes are problem specific, the techniques proposed in this paper, including the indexed compact buffer model and approximate sorting, would be useful and applicable to other web applications using GPU acceleration. Despite ongoing research on GPU-to-GPU communication mechanisms, achieving performance on multi-GPU systems remains a major challenge. Inter-GPU data transfer via DMA-based bulk transfers exposes data transfer latency on the GPU's critical execution path because these large transfers logically overlap between computer cores. Conversely, fine-grained peer memory access during kernel execution causes memory stalls that can exceed the ability of GPUs to cover these operations with multiple threads due insufficient memory allocation leading to high workload traffic generation, thereby increasing the latency, and worsening the throughput. Even worse, these underline moves are very inefficient with current GPU interfaces. This study compares the performance of fast Fourier transforms on a host processor, GPU parallel computing, and GPU parallel computing with memory allocation optimization.

Furthermore, GPU parallel computing has been shown to be effective in increasing FFT calculation speed, when using 1-bit complex input data, the ratio of GPU parallel computation to CPU speed can reach. In addition, optimizing GPU memory allocation, FFT calculation speed can be further improved.

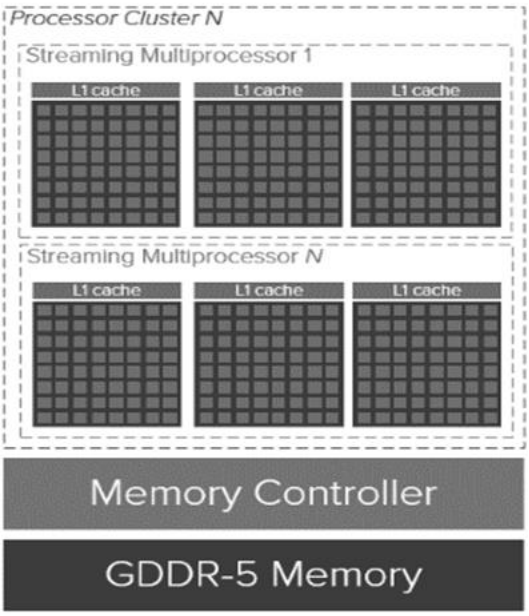

Figure 1: GPU Memory Controller Architecture

RELATED WORK

Common Reason Realistic Handling Units are presently a principal component in any high-performance computing framework due to the tall capacity of these architectures to perform complex computations productively. Over the final decade, NVIDIA has presented seven contrasting GPU generations/architectures. Each engineering architecture pipeline possess its individual microarchitecture and equipment characteristics. Be that as it may, the rate of undisclosed characteristics beyond what GPU merchants have archived is little. Subsequently, inquiries about have proposed distinctive micro-benchmarks compose in programming dialects, such as CUDA or OpenCL to get it the covered-up characteristics of the equipment for nearly each GPU generations/architectures. In this paper, we demystify the idleness of distinctive instructions executing within the pipeline and the distinctive memory chains of command found in different NVIDIA GPUs. Since CUDA (nvcc) compiler optimization influence the informational, we moreover, appear the impact of the CUDA compiler optimizations on the execution of all informational. In this study, we discuss the use of GPU pipeline architecture in parallel computing to accelerate Fast Fourier Transform (FFT) calculations. In addition, memory allocation methods are used to optimize calculations and achieve low latency.

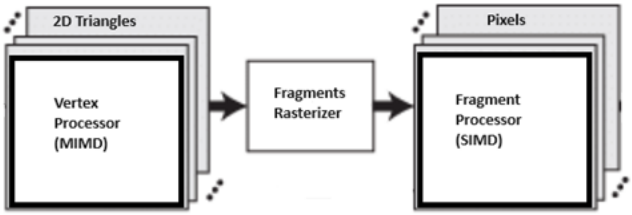

Figure 2: Basics of GPU Pipeline Architecture

Regardless of the achievements in the GPU pipeline architecture and its programming, attaining high throughput and high performance in multi-core GPU systems still holds a challenge for most of the developers. To methodologically parallelize the pipeline architecture efficiently across all the cores of the GPU system, developers potentially utilize the concept of data structures across the L1 cache and L2 cache of GPU memory. Furthermore, the developed algorithm needs to synchronize between the duty cycle of the instructions provided to the multi-core GPU and the data traffic allocation of the memory. In other words, the connection is idle during the computing time and the computing device is idle during the communication time.

Moreover, the implementation of the linear equations with creative iterative and directive methods has been the go-to resource for most of the developers in recent times. To improve and optimize the memory allocation for benefiting the latency-throughput, the main objective focuses on reduction of the sparse matrix of the linear

equations, along with speeding up the computational times. With the development of parallel computers, general purpose technical computers, computers can achieve higher computing speeds through parallel processing. Therefore, the parallelization of spatial matrices for linear equations is of great interest. Some research efforts have been devoted to converting existing algorithms to parallel computing.

INTRODUCTION TO ALGORITHM

A. Memory Allocation Optimizations

- Global Memory – Globally accessible memory, Global memory data can be potentially utilized for multi-core, multi-threaded GPU Pipeline architecture system, but with maximum latency.
- RF Memory – The quickest accessible memory buffer, providing scope for usage of threads running in parallel within the registers. This has lowest latency but not higher throughput due to delay caused when all the registers are inaccessible for storage, then secondary memory storage needs to be added into the pipeline architecture.
- Persistent Memory – Can be used for accessing data that doesn't iterate during kernel execution.
- Common Shared Memory – Only threads with L2 cache can access memory from L2, meaning threads within each of the blocks can individually access memory within the block or buffer. Further, synchronization is required to avoid race around conditions.

In terms of latency, RF is much lower compared to that of any of the local and global memory.

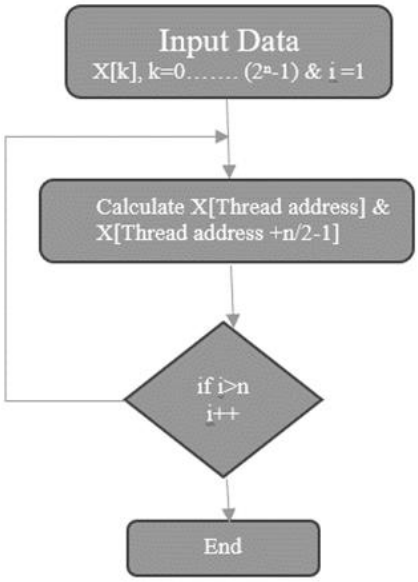

Figure 3: Parallel FFT Algorithm

B. Parallelization of GPU Pipeline Architecture

The rise in the workload size raises the number of threads accessing the memory, either local or global leading to high latency processing. Accessing the shared memory potentially minimizes the execution time of these threads thereby lowering the latency and improving the synchronization. But the drawback of this methodology is that shared memory has limited access, hence restructuring GPU pipeline architecture would highly help manage the workload.

Prepare the data that the processor needs to perform the FFT operation. Put data into global memory. Split data into several parts. Then select the appropriate number of threads in one block. Here we use only its first dimension. Put data into shared memory. And do a butterfly operation on each block. At this point, there is no bank conflict in shared memory and the previous level is executed. Put data in shared memory back into global memory. This is the first exchange. Keep dividing the previous data into several parts. Then join the small pieces. Select the number of threads in one block. This step ensures that the first dimension is 16. Because addition is required. Do the butterfly operation column by column. Currently there is no bank conflict either. Put the data into shared memory and perform a butterfly operation on each block. At this point, the later level is filled. And the whole butterfly operation is done. At this point, the next level is started. And the whole butterfly

operation is over. Put data in shared memory back into global memory. This is the last shift. Return the calculation results in global memory to the CPU.

CONCLUSION

The GPU pipeline architecture performs better computations when the workload size is minimal, meeting the requirements of its ability to be performed in single thread computation. As the workload traffic rises, more number of threads are needed to be perform the computations, rising the level of synchronization in the GPU pipeline architecture, improving the concurrency in return. Therefore, the optimization technique using the Fast Fourier Transform, alleviates the workload access for low latency memory allocations. Furthermore, FFT algorithms have been widely applied into the field of science vastly providing great benefits achieving low latency pipeline architecture systems.

FUTURE SCOPE

The analysis of this paper can be extensively used to develop memory optimizations using 2D FFT and 3D FFT algorithm, relatively improving the latency and throughput of the system design. Algorithms can also be developed to minimize the workload data traffic congestion amongst the threads to maximize performance gains. In addition, indexed buffer methodology and sorting approximation can be applied to multi-core multi-threaded networks to leverage the GPU pipeline architecture computations.

REFERENCES

[1]. Yehia Arafa1, Abdel-Hameed A. Badawy, Gopinath Chennupati, Nandakishore Santhi, Stephan Eidenbenz, "POSTER: GPUs Pipeline Latency Analysis", IEEE 30th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), 2019

[2]. Fan Zhanga, Chen Hua, Qiang Yina, Wei Hua, "A GPU Based Memory Optimized Parallel Method for FFT Implementation", Semantic Scholar, July 2017

[3]. Chu-Hsing Lin, Jung-Chun Liu, Po-Kai Yang, "Performance Enhancement of GPU Parallel Computing Using Memory Allocation Optimization", IEEE, April 2020.

[4]. Harini Muthukrishnan, David Nellans, Daniel Lustig, Jeffrey A. Fessler, Thomas F. Wenisch, "Efficient Multi-GPU Shared Memory via Automatic Optimization of Fine-Grained Transfers", ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), 2021

[5]. Jianhua Sun, Hao Chen, Ligang He, Huailiang Tan, "Redundant Network Traffic Elimination with GPU Accelerated Rabin Fingerprinting", IEEE Transactions on Parallel and Distributed Systems, Vol. 27, No. 7, July 2016

[6]. Dongxu Yan, Haijun Cao, Xiaoshe Dong, Bao Zhang, Xingjun Zhang, "Optimizing Algorithm of Sparse Linear Systems On Gpu", Sixth Annual ChinaGrid Conference, 2011