Paper Analysis Project for CS 575

on

General Transformations for GPU Execution of Tree Traversals

By
Hyun-Taek Oh

ohhyun@oregonstate.edu

11 June 2024

1. **What is the general theme of the paper you read? What does the title mean?**

According to the article "General Transformations for GPU Execution of Tree Traversals", Michael Goldfarb et al. introduce novel general-purpose techniques for implementing irregular algorithms like tree traversals on GPUs that exploit similarities in algorithmic structure rather than application-specific knowledge, achieving speedups of up to 38x over 32-thread CPU versions.

In this paper, the title implies that GPU execution is not suitable for tree traversals since exploiting parallelism and optimization for the GPU memory hierarchy is widely comprehended for regular algorithms or structures that runs on dense data structure such as arrays, and thus transformations for general purpose are needed to compute in different ways. Tree Traversals are not regular algorithms since its data structure called irregular structure is data-dependent, meaning that it performs unexpected memory accesses, and can be changed dynamically, requiring to come up with novel approaches.

2. **Who are the authors? Where are they from? What positions do they hold? Can you find out something about their backgrounds?**

The authors of the paper are Milind Kulkarni, Michael Goldfarb, and Youngjoon Jo from the Purdue University. Milind Kulkarni, who is a senior member of IEEE, received the bachelor's degree from North Carolina State University, and the Ph.D. degree from Cornell University. He is currently an Associate Professor with the School of Electrical and Computer Engineering, Purdue University. His research interests focus on developing languages, compilers and systems that can efficiently and effectively exploit locality and parallelism in complex applications on complex computation platforms. He got the Department of Energy Early Career Research Award in 2013 for his work on optimizing irregular applications.

Michael Goldfarb was a Ph.D. student in a school of Electrical and Computer Engineering at Purdue University. He is currently working on both hardware and software for machine learning as a performance architect for deep learning training at NVIDIA. He was previously the compiler lead and one of the first employees at EnCharge AI where I helped bootstrap the SW engineering organization and build ML compiler for a novel in-memory computing technology. At Qualcomm Research, he also worked on optimizing AI/ML applications for Snapdragon powered devices and developed new accelerator architectures for low power on device inference.

Youngjoon Jo was a Ph.D. student in Electrical and Computer Engineering at Purdue University. He is currently working at Google as a staff software engineer for 11 years. Previously, he did research intern at Microsoft and software engineering intern at Google. Interestingly, he was a game developer and embedded engineer in South Korea.

3. What experiments did the paper present?

The researchers proposed the primary transformation: “autoroping.” They explain that the main overheads in GPU implementations performing general tree traversal is the cost of repeatedly moving up and down the tree during traversal. To avoid unnecessary cost by visiting interior nodes redundantly, previous research used auxiliary pointers called “ropes”, obviating the stack-management that is otherwise necessary to traverse the tree. However, encoding the ropes into the tree requires developing algorithm, sacrificing generality for efficiency. Due to this reason, autoroping the researchers proposed is a generalization of ropes that can be applied to any recursive traversal algorithm. Autoroping can help a point never need to return to interior nodes to perform its traversals, be performed without semantics-based knowledge, apply to any traversal algorithm, even those with complex traversal patterns, and does not require any preprocessing of the tree.

Furthermore, the researchers suggest an approach called “lockstep traversal” to improving throughput of traversal algorithms on GPUs. They describe that effective execution on GPUs typically requires carefully managing two aspects of an application’s behavior, memory coalescing and thread divergence. Memory coalescing issues arise because of a consequence of the GPU’s memory architecture. GPUs rely on high throughput memory to service an entire warp of threads. To achieve this throughput, the GPU memory controller requires threads within the same warp to access contiguous regions of memory so that a single wide block of memory can be transferred in a single transaction. Thread divergence arises when different threads in the same warp execute different instructions. When different threads must execute different instructions, some of the threads in the warp are disabled and sit idle while the other threads perform useful work. Lockstep traversal the researchers introduced is a transformation for unguided traversals which deliberately forces threads to diverge in order to keep them in sync in the tree, promoting memory coalescing.

Point sorting is an important concept of computing tree traversal to improve the performance on GPUs, which can be of great benefit to lockstep traversal. Sorting guarantees that nearby points have similar traversals. This ensures that the union of the warp's traversals is not much larger than any individual traversal, minimizing the load balance penalty incurred by lockstep traversal. On the other hand, unsorted points make a warp be likely to have highly divergent traversals, and the penalty for lockstep traversal will outweigh the load-balancing benefits. The researchers use lockstep implementation if the points are sorted, otherwise use non-lockstep version.

For the experiment, the first step is to identify the key components of the algorithmic structure by translating traversal algorithms to GPUs. After identifying a repeated recursive traversal that can be parallelized onto the GPU, the original CPU implementation was replaced with a GPU kernel call. Then, there are transformation into two steps: one is transformation the recursive function call into an iterative GPU traversal kernel, and another is replacing the point loop and recursive function call with GPU kernel invocation.

For transformation the recursive function call into an iterative GPU traversal kernel, preparing the recursive function for the autoropes transformation, only requiring that a function be expressed in pseudo-tail-recursive form to be correctly applied. In the paper, the researchers provide an example of Barnes-Hut recursive traversal to illustrate the transformation and describe how to lay out the rope stack and the nodes of the tree. By optimizing the stack storage, traversals are performed in lockstep. All threads in a warp will perform the same traversal, allowing any data which is not dependent on a particular point to be saved per warp rather than per thread.

Replacing the point loop with GPU kernel is another way to transform the function. In this step, since the GPU memory resides in a separate address space , it must also be copied any data to and from the GPU that is live-in and -out of the point loop, and update CPU-resident data after the GPU kernel exits. Before the traversal kernel is invoked, an identical linearized copy of the tree is constructed using a left-biased linearization, with the nodes structured according to layout strategy, and copied to the GPU's global memory.

To evaluate proposed techniques on four important scientific and engineering traversal algorithms by comparing the overall performance of multi-threaded CPU codes against GPU implementations. For each algorithm, non-lockstep and lockstep versions of the algorithms are evaluated. Then, these implementations are compared against two alternatives implementations: a naïve GPU implementation and parallel CPU implementations of the same traversal algorithms.

4. **What conclusions did the paper draw from them?**

The authors conclude that their transformations produce GPU implementations, which are superior to naïve GPU implementations and competitive with large-scale multithreaded CPU implementations. In general, the researchers found out that suggested GPU implementations are far faster than naïve recursive implementations on GPUs. In almost all cases, their autoropes transformation is able to deliver significant improvements. Overall, they also found that the best variant between lockstep and non-lockstep for each benchmark/input pair for outperforms the single-threaded CPU version. Furthermore, the best GPU variants of their benchmarks outperform the CPU implementation up to at least 8 threads, and in most cases outperform the CPU implementation even at 32 threads. On the whole, by choosing the right set of optimizations, the researchers can automatically map traversal algorithms to GPUs and achieve results that are substantially better than naïve GPU implementations; much faster than single-threaded CPU implementations; and competitive with even highly threaded CPU implementations without taking advantage of application-specific knowledge.

5. **What insights did you get from the paper that you did not already know?**

At first, I believe that the method to compute irregular algorithms such as trees and graphs is similar to computing regular algorithms such as arrays and matrices. However, after reading and analyzing this paper, I learned that especially tree traversals are needed to address in a different way. Unlike arrays and matrices that are linearly organized and structured, tree structures should travel from the root to child nodes, which are irregularly organized and dynamically changed. Because of GPU architecture, the plan to handle irregularity from non-uniform, unstructured data plays a significant role in processing them and reducing possible overheads.

GPUs are not always perfect for specific computation. Generally, GPUs have a powerful ability to compute massive data accurately and fast. Nevertheless, it needs to handle the irregularity, which depends on the type of data structures. After finding irregular data structures, graphs can also fall into the irregular algorithms. In the related paper about graphs, GPUs should also handle these algorithms in a different way. From these findings, GPUs do not always guarantee the general-purpose processing unit because of its architecture.

6. **Did you see any flaws or short-sightedness in the paper's methods or conclusions?**

To reduce the overhead of superfluous loads, in the paper, unlike Barnes-Hut algorithm, the authors tried to preprocess the tree itself rather than traversals. In this preprocessing step, I expect that the entire cost of preprocessing can be higher than the cost of Barnes-Hut algorithm, causing unnecessary processing. In general, proposed algorithms show the better performance than naïve and one or multi-threaded CPU in given algorithms. However, there are no explanations about why some cases perform less than naïve or CPU cases. If the authors articulate it, this paper will be more persuasive and accurate.

7. **If you were these researchers, what would you do next in this line of research?**

If I were these researchers, I would like to handle graph algorithms, which are also irregular structures and have dynamical data. Graphs are important concepts to represent real-world information and are used in many different applications in daily life such as searching routes and social networking, meaning that its irregularity can cause some flaws like tree traversals with overheads. To solve the problem of graph algorithms, further research should be conducted.

Reference

Goldfarb, M., Jo, Y., & Kulkarni, M. (2013, November). *General transformations for GPU execution of tree traversals*. IEEE Xplore. https://ieeexplore.ieee.org/document/6877443