Optimization Strategy for SRME on Highly Parallel Hardware

Marcel Nauta[1], Lorenzo Casasanta[1]

1. *Shearwater GeoServices*

SHORT ABSTRACT – In this work, we review the optimization compromises made for running surface-related multiple elimination (SRME) on GPUs and high-core count CPUs. SRME is a computationally intensive, data-driven, stage of typical seismic data processing workflows. SRME is parallelizable at multiple levels, similar to modern hardware. This allows a task-based dynamic scheduling approach with subsets of the hardware working on independent outputs, but shared inputs. We found that the most important optimization is reducing disk reads by caching input data in the fastest levels of memory. Since the parallelization is broken up at multiple levels, the data movement also occurs in stages to exploit redundancies in the input requirements. Other optimizations include running asynchronous tasks, buffering the pipeline of data movement, and recycling redundant calculations when outputting results in common-shot order. However, care must be taken not to introduce an excessive memory footprint for large datasets. Therefore, we use a total memory constraint and reduce the number of disk reads by serializing some steps in the data movement pipeline and splitting large shot records with an offset based binning scheme. When memory is not an issue (e.g large CPU memory compared to GPU memory), the data movement pipeline passes through a short FIFO circular buffer to make the buffering more resilient to variations in system performance and reduce the cost of synchronization points for dynamically scheduled work. The presentation will highlight analogies between multi-GPU machines and NUMA architectures. Details of the data movement pipelines will be shown for a variety of field datasets.

INDEX TERMS – geophysics, seismic processing, multithreading, GPU, NUMA

I. MOTIVATION

Surface-related multiple elimination (SRME) is a powerful algorithm that predicts sea surface-related reverberations for a trace in a dataset relying on nearby traces from the same dataset, and eventually attenuates those multiples from the original dataset. Although the compute cost of a single prediction is tiny, the size of modern datasets makes the problem computationally expensive, especially in the context of iterative methods and multi-dimensional convolution operators.

In recent years, hardware platforms have become increasingly reliant on scalability and parallelism. SRME has multiple levels of parallelism that can be exploited. Although not published to our knowledge, common industry practice on CPUs in the 2010s was to multithread the innermost loop over bounce points for a single prediction. In this work we generalize this algorithm to exploit multiple levels of parallelism based on available hardware. This includes high-core count CPUs as well as GPUs. Focus is on the compromises made to exploit data movement redundancy, compute redundancy, and concurrent operation of all hardware components.

II. HYPOTHESIS

In the simplest case, SRME can be implemented on a single-core machine, with a small amount of RAM, assuming access to the entire dataset. For a given source-receiver pair, the vanilla algorithm computes contributing traces on a pre-determined grid of bounce locations. At each bounce location, two traces are selected from a look-up dataset based on a header distance measure, these traces are then adjusted for any kinematic error and convolved with each other. A final summation over all the bounce point locations creates the multiple prediction for the chosen single source-receiver pair.

In practice, redundant reads and redundant compute are removed at the cost of an increased memory footprint. Exploiting redundancy is extremely important because disk reads occur at 100MB/s – 10GB/s whereas memory movement inside a GPU is now at speeds of 1TB/s – 10TB/s. Similarly, searching for required nearby traces, reading data from disk, and processor compute can all be done simultaneously at the cost of increased memory to remove data dependencies. For large datasets, the fastest levels of memory are so scarce that the various optimizations need to be prioritized.

III. METHODS AND RESULTS

The starting point for this project was a production-ready code that had been heavily optimized for clusters of x86 CPUs with good MPI connectivity. The end goal was a feature complete, backwards compatible tool additionally optimized for both GPUs and larger core count CPUs. We understood that this required a near re-write of the code, but we strove to minimize the deviation of any new code from the main development branch. To ease the testing burden, we stacked over bounce points in a multi-threaded, but deterministic way. The chosen order is irrelevant, and has some additional cost, but allows automated testing of many optimizations and often relieves the need for manual QC by a domain expert.

The algorithm was pipelined at several levels. This is commonly implemented with double buffering, but we extended the concept to a short FIFO circular buffer. This allows highly variable tasks such as disk reads to work further ahead and reduce the risk of intermittent stalls. The longer the circular buffer, the higher the memory footprint of that stage in the pipeline. When memory at some level is scarce, the length of the buffer can be reduced to 1. This serializes part of the pipeline but can be essential to exploiting as much redundancy as possible, which also has a significant memory footprint.

There are two main levels where we exploit the redundancy of different predicted traces. The first, which is common industry practice, is to organize the outputs in common-shot and avoid recomputing source side bounce points resulting in a maximum theoretical speedup of half the compute. Unfortunately, for large shots, this once again has a significant memory footprint. Therefore, we break up large shots in the offset domain and apply apertures to the bounce points to save memory at the cost of some recompute.

The second level of redundancy is in the data read from disk into fast levels of memory. To exploit this, CPU threads work ahead to determine a set of outputs that rely on a similar subset of input traces that fit in the memory available. This optimization is so important on GPU that we typically double buffer disk reads into CPU memory and then retain a single copy on GPU. The maximum theoretical speedup of this optimization is the ratio between the fast levels of memory and disk, which are typically orders of magnitude different. Therefore, when memory is scarce, this optimization takes priority.

IV. CONCLUSION

Leveraging an incremental development approach, we upgraded an existing production SRME tool to run on GPU and re-optimized for NUMA architectures. We found the most important optimization is to minimize the number of times input data is moved from slow levels of memory such as disks, to fast levels of memory, such as GPU memory. Additionally, we leveraged task-based execution at several levels of parallelism to implement dynamic scheduling with reproducible results.

BIO

Marcel Nauta has been working in high performance computing for seismic exploration for over 10 years but will often draw from his upbringing in a family bakery to illustrate the importance of HPC concepts. When memory access patterns are compared to the physical distance someone would walk to grab a cookie the importance becomes laughably obvious.

REFERENCES

[1] B. Dragoset, E. Verschuur, I. Moore, R. Bisley, A perspective on 3D surface-related multiple elimination. Geophysics, 2010. https://doi.org/10.1190/1.3475413