Introducing Prefetching and Data Compression to Accelerate Checkpointing for Inverse Seismic Problems

Thiago Maltempi[1], Sandro Rigo[1], Marcio Pereira[1], Hervé Yviquel[1], Jessé Costa[2] and Guido Araujo[1]
[1] Computer Systems Laboratory, University of Campinas (UNICAMP) [2]Faculty of Geophysics, Pará Federal University

SC23 Denver, CO | i am hpc.

CONTEXT

Reverse Time Migration (RTM) is a seismic imaging technique that presents significant computational challenges, requiring hundreds of gigabytes of memory and processing times that can take several days. RTM can be a critical component of a Full Waveform Inversion (FWI) solution.

Our RTM implementation processes three-dimensional fields decomposed across multiple NVIDIA GPUs and uses the Revolve algorithm as an optimizing checkpointing strategy. However, storing checkpoints in host memory and transferring data between the host and GPU becomes a bottleneck. We propose a checkpoint prefetching mechanism and GPU data compression to alleviate this bottleneck. The experimental results show that we can significantly improve performance, with a speedup of 1.98x – 2.53x in our current benchmark dataset.

REVERSE TIME MIGRATION (RTM)

Reverse Time Migration (RTM) is a powerful technique used in seismic imaging to generate images from seismic surveys as illustrated in the figure below. At a high level, the RTM algorithm involves solving the wave equation by discretizing the wave pulse into a series of timesteps.

During the forward phase, the wave propagation is computed in a forward-in-time manner. Subsequently, the backward phase is initiated, where the same timesteps are processed in reverse, enabling the correlation of forward and backward data to generate the final image.

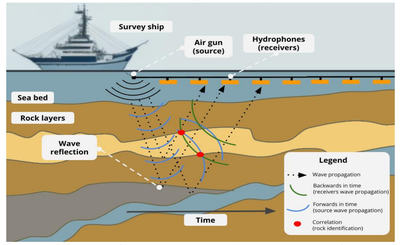

Figure 1. Diagram of a seismic survey with an explanation of how the recorded data is computed using RTM.

TRADITIONAL CHECKPOINTING

Checkpointing is a technique that involves saving certain points of computation during the forward phase to prevent having to redo them during the backward phase. As checkpoint data requires a lot of memory storage, we use the memory of the host, once the GPU memory capacity in our setup is limited to 32 GB.

However, the data transfer between the GPU memory and the host memory creates a bottleneck, directly impacting the overall performance. Checkpointing is managed by the Revolve algorithm, which defines the number of checkpoints and acts as a controller for our application.

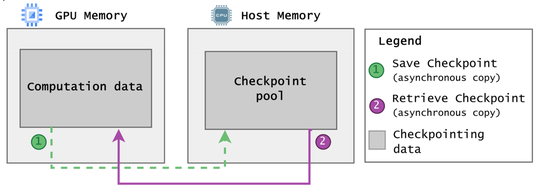

Figure 2. Checkpointing architecture diagram

CHECKPOINT PREFETCHING

Checkpoint Prefetching takes advantage of Revolve's checkpoint reuse to reduce the frequency of data transfers. To minimize the intensive memory copy between the host and GPU, we introduced a GPU dual buffer data structure capable of storing the two most recently used checkpoints on the GPU side.

However, Revolve sometimes requires checkpoints older than those in the dual buffer, thus resulting in "buffer misses". These misses necessitate transfers from the host to the GPU, which can be slow.

To address this, we took advantage of Revolve's cheap and deterministic nature. By dry-running Revolve, we traced the iterations where buffer misses would occur.

We then schedule asynchronous host-to-GPU data transfers ahead of time, a process we refer to as prefetching. This proactive approach further optimizes the checkpointing process, avoiding around 75% of buffer misses, minimizing GPU idle time, and enhancing overall application performance.

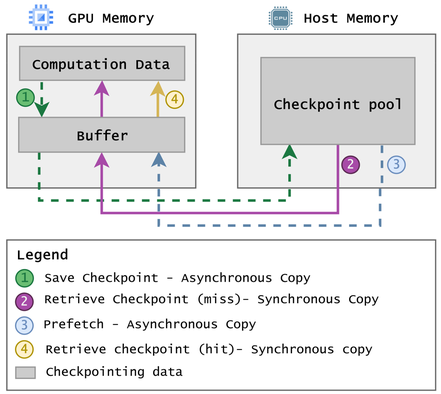

Figure 3. Checkpoint Prefetching Architecture

CHECKPOINT DATA COMPRESSION

To further improve overall performance, we proposed a data compression technique that reduces data transfer through PCI-e and mitigates buffer misses' effects. Our optimization goal is to ensure that compression, decompression, and transferring of compressed data take less time than the current buffer misses.

We employ cuZFP, an open-source library that performs lossy compression on GPUs. cuZFP supports fixed-rate compression, allowing users to specify a maximum number of bits per value during the float point quantization phase. A compression ratio of 4 (or 8 bits per value) strikes a favorable balance between enhanced performance and quality loss.

The compression is seamlessly integrated into the prefetching implementation. Checkpoint data is compressed and then stored in the buffer and then in the host memory. When it is needed for computation, compressed data is brought from the host memory to the GPU and decompressed.

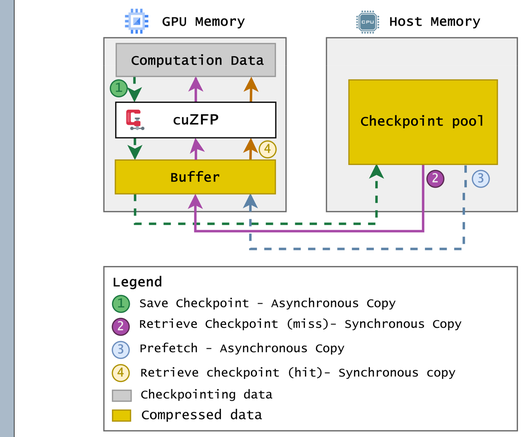

Figure 4. Checkpoint Prefetching with Compression Architecture

EXPERIMENTAL RESULTS

In our experimental setup, we examined three profiles (Checkpointing, Prefetching, and Prefetching with Compression) on three datasets (Salt, Marmousi, and Large), using "Checkpointing" as our baseline.

The results demonstrate notable speedup for Prefetching (around 1.42x) and even greater improvement for Prefetching + Compression (ranging from 1.98x to 2.53x). For example, the Salt dataset's execution time was reduced from approximately 33 hours to 15 hours with Prefetching + Compression.

Prefetching reduces data transferred from the host to the GPU by 4x. When combined with compression, host-to-device transfers can be reduced by 16x compared to the traditional checkpointing implementation, when using a compression rate of 4.

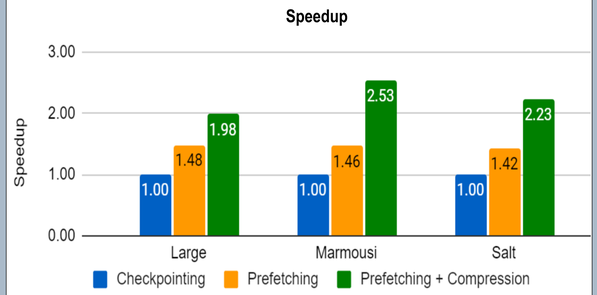

Figure 5. Chart comparing speedup of the proposed enhancements with respect to traditional checkpointing.

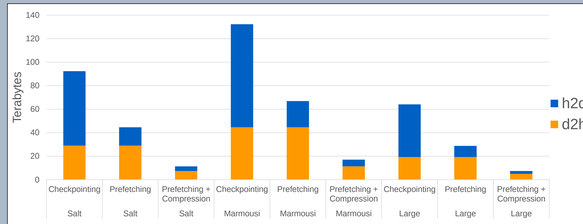

Figure 6. Host-to-device (h2d) and device-to-host (d2h) data transfer through PCI-e

The degradation in the final rendered image caused by lossy compression is hardly noticeable to the human eye. The images below show the migration results of both the baseline execution without compression and the execution using compression.

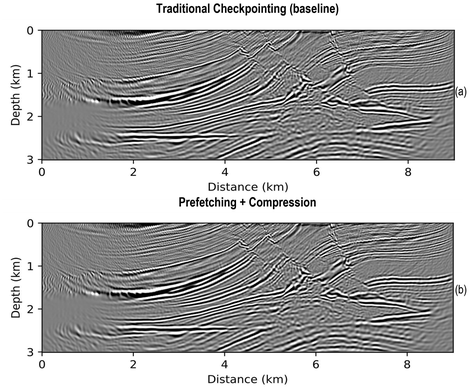

Figure 7. Comparison of Marmousi migrated images. (a) baseline, and (b) prefetching+compression.

Dataset	ABS	PSNR
Salt	7.73	74.67
Marmousi	18.68	87.54
Large	3.99	55.97

Table 1. Average absolute error and Peak Signal-to-Noise Ratio from baseline and prefetching+compression.

FOLLOW UP

Scan the QR code or visit geospeed.gitlab.io/gpuzip_sc23 for detailed multimedia content about our technique and results.

UNICAMP LSC COMPUTER SYSTEMS LABORATORY