A Scalable Framework for Flow Based Processing of Irregular N-Dimensional Seismic Volumes

Marcel Nauta, Andreas Rueger, Lorenzo Casasanta, Shearwater GeoServices

Summary

We present a novel framework for handling large, irregularly shaped and sampled, N-Dimensional seismic volumes that remains compatible with graph-flow-based seismic processing applications commonly used in our industry. Leveraging an abstract I/O framework, we demonstrate how to interconnect standard trace-wise and gather-wise processing steps with large volume-wise processing steps in a hardware-agnostic but scalable way. This is crucial to allow algorithmic flexibility and quality control without compromising the HPC performance of the processing sequence. In fact, the abstracted nature of the framework decouples the HPC infrastructure from geophysical algorithms, allowing the same code to run performantly on a single small machine or supercomputers of various architectures. We demonstrate use cases ranging from HPC crucial algorithms spanning data intense LS-RTM and LS-KDM down to simple filtering steps such as tapering irregular volumes.

A Scalable Framework for Flow Based Processing of Irregular N-Dimensional Seismic Volumes

Introduction

Flexible flow graphs are a fundamental concept in seismic processing and imaging software because they enable workflows tailored to specific acquisition conditions, data characteristics and project goals (Mosher et al., 1996). The key feature lies in allowing processors to connect a set of individual processing steps to form a larger workflow, allowing tweaks such as filters, quality checks, and access to intermediate results. While implementation of a general flow graph is straightforward for simple processing sequences that have little data inter-dependency, such as filtering individual traces, other common processing sequences require access to many inputs and generate large amounts of output. These datasets are typically irregular, and the input and output sizes can differ in both size and dimensionality. Determining the size and dimensionality of intermediate steps in the graph can put a significant burden on users and developers. Therefore, we propose a framework that naturally handles arbitrary datasets by separating the data movement infrastructure from geophysical algorithms without compromising scalability or burdening users.

Processing steps that require large amounts of input data are often implemented as standalone monolithic tools that work semi-independent of the general graph-flow seismic processing framework. This allows HPC optimizations to be tailored to the specifics of the application but compromises workflow flexibility. Standalone tools also lose some future compatibility as algorithms, once considered computationally expensive, eventually become a routine component in a larger processing sequence, as is the case with Least-Squares Kirchhoff Depth Migration (LS-KDM) or Least-Squares Reverse Time Migration (LS-RTM). In contrast, we propose a framework that allows these HPC intensive tools to either be used in standalone mode or used within the general graph-flow framework with minimal compromise to HPC performance. In fact, the generalized framework can offer novel optimizations to reduce disk traffic when working with multiple tools that require large amounts of data.

Adopting a common data movement and management framework further facilitates splitting processing algorithms into individual, testable components. This modularity not only simplifies maintenance and increases software longevity but also ensures that algorithms can be updated to keep up with the latest advances in geophysical technology.

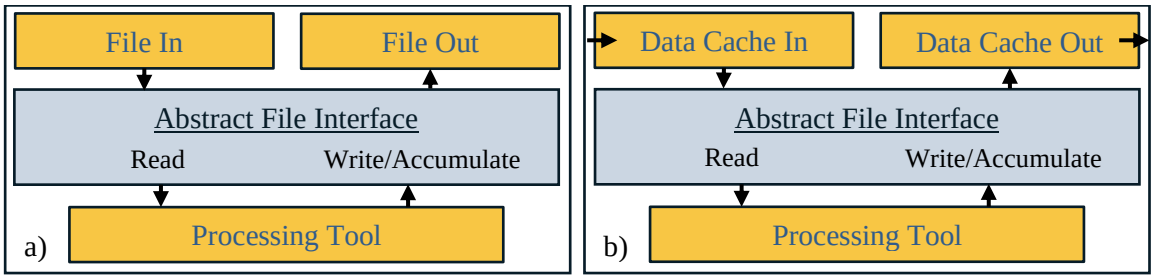

Figure 1 *a) Standalone tools interface directly with the filesystem but an abstract file interface separates data formats from processing tools. b) The same approach can be used in flow graphs by presenting cached data as a file-like object.*

Theory and Methods

Unknowingly, this project began with an effort to develop a unified framework for reading and writing data files. The seismic processing industry relies on many file types, especially when considering compression. This complexity convolutes and gets entangled with geophysical software, making it unnecessarily complicated. Therefore, we defined an abstract interface with three main functions; read, write, and accumulate (Figure 1a). Underneath the hood, the data is pipelined to be read or written in the correct format, including chainable optimization strategies such as blocking and caching. Using this abstraction, developers can focus on geophysics. Secondary functions of the interface provide important metadata, but unlike other data flow solutions, no sorting, dimensionality or regularity constraints are imposed on the datasets at this level. This deliberate design choice significantly reduces the user burden as it can be difficult to determine the size and shape of intermediate datasets in advance.

With an abstracted I/O framework, we can interface the tools that require large amounts of input data with a flow graph by caching incoming data and forming intermediate file-like objects (Figure 1b). Once the input file-like object is formed, the tool dynamically specifies the size and headers of the output file-like object, again ensuring the user does not need to provide a priori information about intermediate results. These intermediate file-like objects can take various forms depending on the available hardware. For example, small problems can cache the incoming data in RAM, or compute nodes with a large amount of local disk space can form temporary files. For large problems, we leverage the popular Message Passing Interface (MPI) to distribute the memory footprint over a large number of computers (MPI 3.1, 2015). Since no assumption is made on the sort or regularity of the data at the infrastructure level, reads and writes are performed with one-sided MPI Remote Memory Access (RMA). Using one-sided MPI-RMA is crucial because it eliminates the need for MPI ranks to synchronize each operation but still gives every rank access to the entire dataset. While one rank reads or writes to the memory of another rank, the target rank is not explicitly involved. This emulates the behaviour of ranks independently reading from a file. All forms of temporary memory are exposed to the tool through the abstract I/O framework described above, allowing each tool to be implemented as if it were a standalone tool.

Once the temporary file is formed, optimization techniques tailored for each specific algorithm can be applied. Many optimizations are readily available within the base framework. The first class of optimizations assumes that the algorithm is embarrassingly parallel with respect to the inputs, the outputs, or both. In that case, automated splitting and dynamic scheduling can be leveraged assuming the derived application specifies a list of partitionable headers. For embarrassingly parallel algorithms, or those with a simple race condition when accumulating into the output file-like object, the input and/or output datasets can be arbitrarily subdivided to fit on available hardware (Figure 2a). Every pair of input and output subsets (Figure 2b) constitutes a work item that is dynamically scheduled over available hardware. Intermediate results are written or accumulated into the output file-like object with hardware-specific locking implemented within the I/O infrastructure to remove stacking race conditions without synchronizing ranks.

A second important optimization is to apply apertures around the input and output areas of each work item to remove unnecessary I/O with negligible compute cost as shown in Figure 2c-d. To remain general in an N-Dimensional framework, apertures are defined as a set of key-value pairs where the key is a header for each trace, and the value is a scalar distance. For every work item and dimension of the aperture specified, a bounding box extended by the aperture distance can be determined for the input dataset. Any output trace with a header outside the aperture-extended input bounding box can be safely discarded from that work item (Figure 2c,d top row). Similarly, in reverse, input traces can be cropped according to an aperture-extended bounding box around the outputs (Figure 2c,d bottom row). Applying apertures in this way crops most of the unnecessary data movement, and entire work items can be discarded if the inputs and outputs are sufficiently separated. For output-oriented algorithms, this process carves out the cropped input dataset required to compute a subset of the output area.

Figure 2b-d also shows that work items vary significantly in the total number of inputs and outputs after aperture cropping. Therefore, the dynamic scheduler should be optimized to prioritize the most computationally expensive tasks first. For all examples we have encountered to date, algorithms have compute costs that decrease when there is less overlap between input and output bounding boxes. Although volumes are assumed irregular, a practical observation is that the N-dimensional distance between the center point of the inputs and the center point of the outputs is a good measure of relative compute cost in seismic imaging. Therefore, a weighted L2-norm of the distance between center points in each relevant dimension is used to prioritize work. Since the dimensionality of the inputs and outputs can differ, only partitionable dimensions common to both are considered. Additionally, a second-order prioritization is applied per MPI rank to further sort work items with an equivalent estimated cost. Although no assumption is made on regularity, the prioritization allows the dynamic scheduler to degenerate into a consistent static scheduler if the dataset is regular and actual runtimes per rank are consistent. For irregular datasets this remains approximately true and keeps runtimes consistent between jobs.

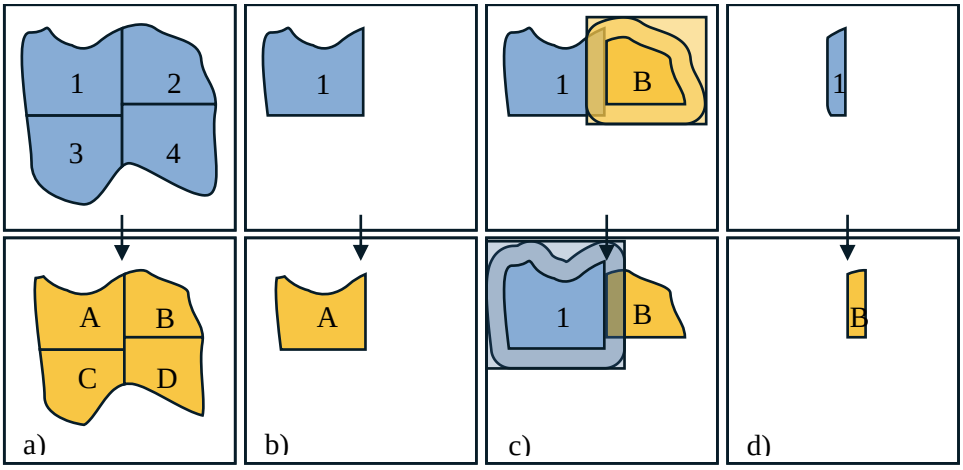

Figure 2 *a) Bird's eye view of a hypothetical processing step, such as KDM, that stacks inputs (top) within a nearby aperture to the output volume (bottom) but can be split into 4 input and output partitions to make 16 work items. b) Work item 1A stacks inputs from area 1 into output area A. c) Work item 1B with superimposed aperture-extended input and output bounding boxes. d) Cropped work item 1B showing the subset of input area 1 that stacks into a subset of output area B after apertures are considered. This also demonstrates the highly variable cost of work items.*

For algorithms that are embarrassingly parallel with respect to inputs, a crucial memory optimization is available. The file-like object does not need to hold the entire input dataset; it only needs a subset of it. This allows a memory constraint to be imposed on the size of the input file-like object and the higher-level flow logic is iterated until all of the partial results have been accumulated in the output file-like object. E.g. in KDM, the flow logic could theoretically be paused for every individual trace and the output image would be a stack of the individual impulse responses. That approach would mitigate most KDM-specific optimizations within a node (Rastogi et al. 2017), so the number of traces processed simultaneously is balanced with the memory footprint. Likewise, if all inputs can be cached and the algorithm is embarrassingly parallel with respect to outputs, the same methodology can be applied to the output flow logic. The only caveat is that algorithms that re-read from the filesystem can only use a reduced memory footprint in standalone mode because it is generally an error to re-request the same data from the higher level flow.

One important and common exception to the limitation that data cannot be re-requested from the flow is algorithms with no data inter-dependencies in inputs or outputs in one or more dimensions. E.g., in KDM or RTM, common-offsets or shots, respectively, are processed independently. In such cases, we assert that the higher-level flow logic is sorted on these dimensions to allow them to be processed sequentially. There is no need to simultaneously load the entire input dataset for RTM; it can still be iterated one shot at a time. Enforcing a sort on the incoming data does put some burden on the user, but we believe it is intuitive to setup and there are checks in the software to provide a meaningful error if data is received in the wrong order. The memory savings of processing independent sub-volumes sequentially is essential to reduce memory footprints. There is no need for the user to sort data within each independent sub-volume.

Not all algorithms are embarrassingly parallel and may require additional locking logic to yield an appropriate level of parallelization. If parallelism can still be expressed as a list of work items that can be equivalently computed by any MPI rank, then we can disable splitting features in the base infrastructure but still leverage the dynamic scheduler without prioritization. For algorithms with complicated depenencies, such as RTM, the derived tool still benefits from the flexibility of robust higher-level flow logic and abstracted file-like input and outputs. All standard approaches to memory transfer between ranks are still available while processing the volume, such as Finite-Difference partition transfers in RTM (Bisbas et al, 2023) or the synchronous data exchange techniques described in Mosher et al. (1996).

Finally, the base infrastructure offers additional flexibility for MPI jobs when there is a separable dimension to the problem such as shots in RTM. In those cases, the number of ranks used in the job might be excessive for processing a small number of shot records. Therefore, MPI communicators are leveraged to range between using a single rank per shot and all ranks per shot.

Examples

We have applied this framework to a wide variety of tools in the seismic processing sequence, ranging from high compute cost tools such as LS-RTM, LS-KDM, and structure tensors down to seemingly trivial tools such as tapering the edges of an irregular volume. Tapering the edges of a volume is a pointwise operation, but only after the full geometry of the volume is known. When the flow graph includes common operations, such as stacking shifted rectangular images, it is very difficult to know the exact geometry in advance. Therefore, we cache all incoming traces with headers and then form geometry information at runtime. If the tapering is applied on a subvolume of an N-Dimensional dataset, such as individual 3D single shot images in a multi-shot RTM flow, only the innermost 3 dimensions are cached. Once the full geometry is known, the output volume can be divided for dynamic scheduling across ranks. An aperture equal to the taper length limits the amount of input data each work item needs to consider, and the actual tapering mask can be cheaply computed and applied to each rank without synchronized communication between ranks. A similar approach is used for spatial domain filters.

A higher compute cost example from LS-KDM is a demigration-remigration step. In this case, an irregular 3D image is demigrated into irregular 3D gathers and then remigrated to the original image geometry. The operation is implemented as two distinct steps in the flow graph and the intermediate gathers are available for QC or modification between steps, but the user does not need to worry about sizing the intermediate dataset nor does the intermediate dataset need to be saved to the filesystem. In contrast, using migration and demigration standalone tools requires the intermediate dataset to be saved on disk, and a single combined tool does not provide convenient access to the intermediate dataset.

Conclusions

Flow-based seismic processing and imaging software is an essential tool because of its flexibility. Our scalable framework for processing irregular N-Dimensional volumes enhances existing approaches by encapsulating higher-level flow logic. It presents each tool requiring access to large amounts of data with an isolated input and output file-like object. This removes the burden on users to specify intermediate dataset sizes but retains full flexibility in the data flow. For MPI jobs, one-sided MPI-RMA is used to obtain file-like behaviour and removes rank synchronization from common data-movement patterns. The separation between flow logic and geophysal processing allows many optimizations to be provided in a common, testable, and robust base infrastructure. Best of all, tools that adopt a framework with intermediate file-like objects remain backwards compatible with existing flow-based tools and can also work as standalone tools that read directly from the filesystem. There is no need for massive refactoring of a large code base because the framework can be introduced incrementally in a sustainable way.

Acknowledgements

The authors thank Shearwater GeoServices for supporting this research.

References

Bisbas, G., Nelson, R., Louboutin, M., Kelly, P.H.J., Luporini, F., Gorman, G. [2024] Automated MPI-X Code Generation for Scalable Finite-Difference Solvers. RICE Energy HPC Conference 2024

Charles, C. M., Calvin, L. J., and Hassanzadeh, S. [1996]. Scalable parallel seismic processing. The Leading Edge, 15(12), 1363-1366.

Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 3.1 (2015).

Rastogi, R., Londhe, A., Srivastava, A., Sirasala, K. M., Khonde, K. [2017]. 3D Kirchhoff depth migration algorithm: A new scalable approach for parallelization on multicore CPU based cluster. Computers & Geosciences, 100, 67-75.