Oil & Gas Science and Technology - Rev. IFP Energies nouvelles **73**, 74 (2018)

https://doi.org/10.2516/ogst/2018048

Available online at:
ogst.ifpenergiesnouvelles.fr

**Numerical methods and HPC**
A. Anciaux-Sedrakian and Q. H. Tran (Guest editors)

REGULAR ARTICLE OPEN ACCESS

# ExSeisDat: A set of parallel I/O and workflow libraries for petroleum seismology

Meghan A. Fisher[1,2,*], Pádraig Ó. Conbhuí[1,2], Cathal Ó. Brion[1,2], Jean-Thomas Acquaviva[3], Seán Delaney[4], Gareth S. O'Brien[4], Steven Dagg[4], James Coomer[5], and Ruairi Short[1]

[1] The Irish Centre for High-End Computing (ICHEC), Ireland
[2] Lero: The Irish Software Research Centre, Ireland
[3] DDN Storage, France
[4] Tullow Oil Limited, Ireland
[5] DDN Storage, UK

Received: 23 February 2018 / Accepted: 31 July 2018

**Abstract.** Seismic data-sets are extremely large and are broken into data files, ranging in size from 100s of GiBs to 10s of TiBs and larger. The parallel I/O for these files is complex due to the amount of data along with varied and multiple access patterns within individual files. Properties of legacy file formats, such as the de-facto standard SEG-Y, also contribute to the decrease in developer productivity while working with these files. SEG-Y files embed their own internal layout which could lead to conflict with traditional, file-system-level layout optimization schemes. Additionally, as seismic files continue to increase in size, memory bottlenecks will be exacerbated, resulting in the need for smart I/O optimization not only to increase the efficiency of read/writes, but to manage memory usage as well. The ExSeisDat (Extreme-Scale Seismic Data) set of libraries addresses these problems through the development and implementation of easy to use, object oriented libraries that are portable and open source with bindings available in multiple languages. The lower level parallel I/O library, ExSeisPIOL (Extreme-Scale Seismic Parallel I/O Library), targets SEG-Y and other proprietary formats, simplifying I/O by internally interfacing MPI-I/O and other I/O interfaces. The I/O is explicitly handled; end users only need to define the memory limits, decomposition of I/O across processes, and data access patterns when reading and writing data. ExSeisPIOL bridges the layout gap between the SEG-Y file structure and file system organization. The higher level parallel seismic workflow library, ExSeisFlow (Extreme-Scale Seismic workFlow), leverages ExSeisPIOL, further simplifying I/O by implicitly handling all I/O parameters, thus allowing geophysicists to focus on domain-specific development. Operations in ExSeisFlow focus on prestack processing and can be performed on single traces, individual gathers, and across entire surveys, including out of core sorting, binning, filtering, and transforming. To optimize memory management, the workflow only reads in data pertinent to the operations being performed instead of an entire file. A smart caching system manages the read data, discarding it when no longer needed in the workflow. As the libraries are optimized to handle spatial and temporal locality, they are a natural fit to burst buffer technologies, particularly DDN's Infinite Memory Engine (IME) system. With appropriate access semantics or through the direct exploitation of the low-level interfaces, the ExSeisDat stack on IME delivers a significant improvement to I/O performance over standalone parallel file systems like Lustre.

## 1 Motivation and introduction

Overall resolution and size (Stanghellini and Carrara, 2017) of seismic studies have increased of the past several decades; individual files range in size from 100s of GiBs to greater than 10 TB. The high resolution of these studies have allowed for better identification and delineation of shallow anomalies and structures related to hydrocarbons and migration paths, thus leading to more accurate estimations and extractions of petroleum products (Hustoft *et al.*, 2007; Lin *et al.*, 2013). With the increase in file size, geophysicists have increased both the computational and personnel time spent handling and optimizing I/O for these files.

While these large and dense datasets have improved seismic interpretation, they have also lead to I/O and memory bottlenecks. Parallel I/O, essentially a requirement for

* Corresponding author: meghan.fisher@ichec.ie

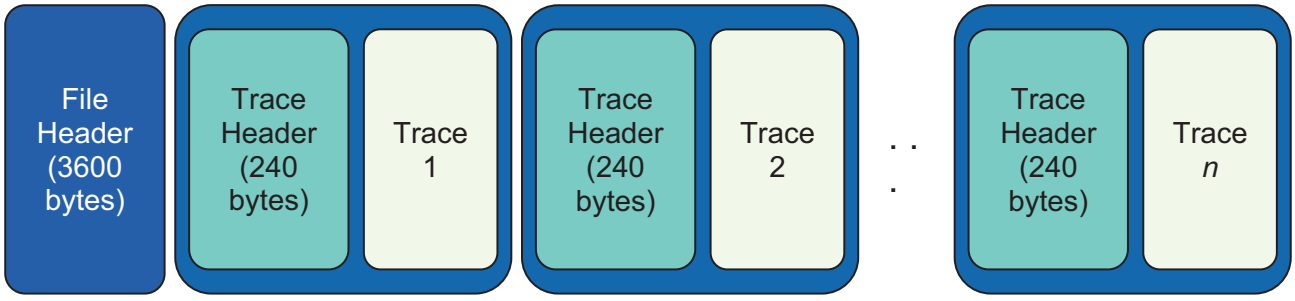


**Fig. 1.** Layout of SEG-Y file with the file header followed by sets of trace headers and trace data. This data arrangement means that no traces are continuous in memory, nor are any individual trace parameters, which drastically decreases I/O efficiency.

files of this size, is complex due to the total volume of data, and the varied access patterns within individual files. Various access patterns for different processing stages can lead to effectively random access patterns, causing contention for disk access (in both serial and parallel file systems), along with limiting the effectiveness of caching and pre-fetching by the operating system. The size of these files also mean that reading entire files into memory for preprocessing is not a scalable solution, particularly for files in the terabyte and, looking forward, petabyte range. While files are growing larger, more processes are involved in the computation. Shared access, specifically for concurrent write, is the most difficult access pattern to deal with for parallel file systems due to lock issue. Write congestion on shared files is a challenging pattern for lock-based parallel file systems (Virtual Institute for I/O, 2018). Addressing write congestion is one of the motivations of check-point oriented file systems such as PLFS (Bent *et al.*, 2009), an inspirational predecessor of IME.

Thus, while the computing part is scaling with the data size the I/O pattern itself is becoming more and more pathological for traditional file systems. Solutions for these problems will be file format dependent since every file format has different storage patterns. One of the most common file formats for exploration seismology is SEG-Y (https://doi.org/10.1190%2F1.1440530). Structurally, SEG-Y stores a file header at the beginning that contains attributes shared by all traces within the file followed by a (typically large) number of individual trace data with metadata for each trace placed at the start of each trace block (Fig. 1). As a result, neither the trace attributes nor sets of trace data are stored contiguously on disk, even when written in a continuous block of storage. This layout strategy leads to poor cache usage and pre-fetching by the operating system, thus increasing I/O time.

In dealing with SEG-Y files, programmers are forced to deal with legacy data formats, *e.g.* IBM floating point numbers and EBCDIC character encoding, along with keeping track of metadata distributed throughout the file. For serial code, alone, these incur a lot of developer overhead in conversion and tracking. For this reason, most commercial seismic data processing tools, *e.g.* OpenCPS and Geosoft, use their own, more regular data formats for processing like SEIS and Geosoft Database. A number of tools exist which can convert SEG-Y files to other formats, *e.g.* Segpy and ObsPy. However, almost all freely available programs that either convert or process SEG-Y files are serial programs. Since the SEG-Y file system cannot be modified due to legacy code and industrial constraint, the way to address the access pattern issue has to be dealt at the library level. Thus shielding geologists from the complexity of the underlying file system but embedding the necessary logic to provide efficient access.

The *Ex*treme-Scale *Seis*mic *Dat*a (ExSeisDat) project addresses the I/O and memory bottlenecks associated with seismic processing within the oil and gas sector though the use of state-of-the-art parallel techniques paired with I/O hardware and software technologies. Designed in partnership with *Tullow Oil plc* and *DataDirect Networks (DDN)* to be both scalable and user friendly, ExSeisDat includes two open source libraries: ExSeisPIOL, a low-level seismic parallel I/O library, and ExSeisFlow, a high level parallel seismic workflow library.

## 2 Background

Parallel I/O allows for multiple processors on a system to perform input/output operations at the same time. While I/O parallelism can offer a speedup, parallel reading and writing from disks has particular challenges; for parallel I/O to be done efficiently, it is vital to understand both the hardware from which it is read or written and the data file format.

### 2.1 Parallel File Systems

Parallel File Systems (PFSs) store data across multiple storage nodes, connecting to a compute node over a network. Unlike serial file systems that stores a file on a single disk, a PFS decomposes the file in blocks that are written to multiple nodes. Writing single files to multiple disks facilitates parallel I/O while still protecting data; each disk locks its portion of the file during a parallel read or write process while allowing other computational nodes to read or write to other disks. While a file is written to multiple disks, its parallel structure is hidden from end users.

Lustre is POSIX compliant PFS that is used on a majority of the Top500 supercomputers. A system consists of a number of "Object Storage Servers" (OSS) – a node containing a number of hard disks, or other storage media, referred to as "Object Storage Targets" (OST) – connected

to a network, with one node, the MetaData Server (MDS), keeping track of the distribution of files, or parts of files, over the OSTs. Unlike other PFSs, like global parallel file systems, allocating a specific server to handle metadata allows for the dedication of OSTs for data I/O resulting in better performance especially for large files.

Lustre uses a "striping" strategy, where a file is split into fixed size chunks, and placed on a predetermined set of OSTs in a round-robin fashion. Lustre-aware applications, particularly parallel applications, can leverage its parallel nature and the striping of a file to achieve a very high throughput. Lustre performance is strongly coupled to file striping. Aligning file access with stripe boundaries minimizes the number of OSTs with which a process must communicate. When stripes are unaligned, an OST receives data from multiple processes causing contention with the read or write; I/O on these OSTs under these conditions are essentially serial. Optimal striping depends not only on the number of OSTs but also the data access pattern. If the size of data accessed does not fit within a stripe, the stripe can become unaligned.

The addition of a fast and intermediate storage layer between compute nodes and the PFS can further improve I/O performance. A burst buffer (Liu *et al.*, 2012) accelerates I/O through the use of Solid State Drives (SSD) that deliver faster read and writes than a traditional Hard Disk Drives (HDD). The underlying operating system later flushes the data to the HDDs when it is convenient. However, these SSDs do not deal any better with random write patterns than HDDs other than their inherent greater speed.

DDN's Infinite Memory Engine (IME) addresses the issues of varying read and write patterns not handled by traditional burst buffers with deployment of a software solution in tandem with its caching system. IME intercepts I/O commands to the underlying PFS from the compute nodes like other burst buffers. However, since IME is composed of SSDs, which are flash devices, incoming data must be written to clean memory blocks rather than rewriting over existing data. Therefore, IME utilizes a logging-based file system where data is written in the order which it is received rather than how that data is stored. IME's software then aligns and orders data for optimal I/O for the underlying PFS. In the case of input data, the IME's SSD cache allows for efficient data access from the PFS and handles all read requests from the compute nodes. The overall I/O time of an application only reflects the amount of time it takes for the compute nodes to read and write to IME not the underlying PFS, like Lustre.

## 2.2 MPI-I/O

The MPI library is a widely-used approach to building parallel, high performance applications. MPI implements the Single Program Multiple Data (SPMD) approach to parallelism, where a programmer writes a single program, a copy of which is run on a number of processes in parallel, and which communicate together via a message-passing protocol.

The MPI library contains a number of routines for performing parallel I/O, collectively referred to as MPI-I/O. When accessing data in parallel with MPI-I/O, processes can perform the access using "collective I/O" and "non-collective I/O." Non collective I/O accesses the data directly from the file system without communicating or coordinating with the other processes, and is most efficient when reading or writing independent, contiguous blocks of data. Collective I/O leverages the typically very fast network interconnect in HPC systems to move data around between processes before writing to disk. This is done to aggregate a number fragmented accesses to the file system into larger, more contiguous ones that result in much better disk access times. Collective I/O can be implemented directly using a POSIX I/O interface, however MPI-I/O can automatically distribute the data for optimal disk access. Both methods require the precise location (in bytes) where data must be written.

MPI-I/O can greatly increase the overall performance on parallel systems, at the cost of an in-depth understanding of the data layout. However, it can instead slow performance when the implementation is not completely tuned. Indeed, one of the recommended best practices for parallel I/O is to use higher level libraries that are based on MPI-I/O (Ching *et al.*, 2007). HDF5 and other high level I/O libraries (Folk *et al.*, 2011; Vishwanath *et al.*, 2011) are examples of efforts made to alleviate the difficulties for end-users in dealing with the low-level details required for efficient MPI-IO utilization.

## 2.3 SEG-Y

The SEG-Y format is a seismic data storage format, originally devised in 1973 (Barry *et al.*, 1975) used for storing seismic reflection data. The format consists of a global, 3600 byte file header containing data common to all the traces and a number of blocks of reflection data, consisting of an individual, 240 byte trace header containing the metadata as well as the trace data itself (Fig. 1). The file header contains information such as trace size, and number of traces; the trace header contains information common to that block of trace data, *e.g.* $x$ and $y$ coordinates, elevation, scaling, etc.

Many existing SEG-Y enabled programs run in serial, because implementing parallel processing for such an unordered file is a complex problem, and implementing efficient parallel processing even more so. Indeed, for poorly ordered disk reads, parallelisation can actually be a pessimization, rather than an optimization. For older, more established programs in particular, efficient parallel I/O of SEG-Y files would require a significant change in software architecture would incur significant development time. Thus far, many applications have relied on vertical scaling – simply using a larger, faster machine with more memory – to handle larger SEG-Y files.

Open source formats, like HDF5 (Folk *et al.*, 2011) and Adaptable Seismic Data Format (ASDF; Krischer *et al.*, 2016), and proprietary formats, like SEIS and Geosoft Database, store metadata and trace data in a more contiguous manner. Typically, a seismic data processing program will instead work with its own proprietary format that's more easily parallelized, and expect a geophysicist to

convert their trace data to a format supported by the program. For very large surveys, this conversion can take considerable time. However, SEG-Y is the defacto standard for exploration seismology and most broadly used file format, with data typically distributed among geophysicists in this format (Stanghellini and Carrara, 2017). It is therefore vital to develop an easy-to-use SEG-Y parallel I/O library to minimize conversion times and run times of SEG-Y enabled applications.

A typical parallel I/O pattern for SEG-Y files involves collective I/O for trace headers and trace data, and non-collective I/O can be used to access the file header. This is because every process will need access to the file header, while trace headers and data will be split among the different processes. However, for good load balancing, a parallel application might want to distribute the data among the processes in a manner that does not reflect the order the data is stored to disk.

## 3 ExSeisDat

ExSeisDat is a open source, portable set of object oriented I/O and workflow libraries designed to simplify parallel I/O and parallel file system optimization for seismic files in order to increase application performance and developer productivity. Source code is available at https://github.com/ICHEC/ExSeisDat under the LGPLv3 license. The project aims to alleviate the bottlenecks in I/O both in runtime and in development time. The libraries completely encapsulate the details of the SEG-Y file format while still providing the flexibility for file and trace header customization that is prevalent when working with SEG-Y, abstracting the specific details away from the developer. To increase usability, the libraries include a language binding interface in C, allowing for future development of *e.g.* python and matlab bindings.

While traditional I/O optimizations libraries such as MPI I/O tend to be blind to the semantic of the application, the ExSeisDat library seeks to bridge the gap between the file format and the purpose of the application and simultaneously address the I/O performance constraints. This domain specific approach has been applied successfully to other application area in HPC, such as FTI (Bautista-Gomez *et al.*, 2011) a high level checkpoint restart library taking care of the storage architecture, or PDI (Roussel *et al.*, 2017). Tailoring the libraries specifically for seismic file formats and applications allows for greater in-depth I/O optimization than would be found in general parallel I/O or parallel file system libraries. It also allows users to focus on domain specific development and optimizations using geophysically intuitive operations. I/O optimizations are paired with smart caching, targeted at typical geophysical workflows, reducing redundant I/O calls while minimizing memory usage.

ExSeisDat also leverages software and hardware solutions to maximize one process reading and writing to one disk. It will communicate and coordinate across processes to coordinate data access as much as possible. As this is not always possible, it also takes advantage of DDN's IME hardware to transform random read and write access to an IME middleware layer into contiguous access to the Lustre layer. This maximizes throughput from the ExSeisDat to IME, and then IME can reorder these accesses to maximize throughput to the filesystem.

ExSeisDat contains two core libraries, ExSeisPIOL and ExSeisFlow, that handle explicit file I/O and seismic workflows respectively. It also contains auxiliary APIs to handle communications between computational processes and to handle error logging. While more complex MPI calls are hidden within the two core libraries, the communication auxiliary API provide calls to retrieve the number of processes called by MPI and the rank of the current process. These calls are simplifications of the MPI calls. The barrier command wraps the MPI_Barrier call to force all processes to wait until all processes call the command before continuing, thus preventing premature data access by allowing data synchronization. Additionally, the auxiliary layer contains the error log that stores verbose MPI errors. Both libraries are designed to work without any further tuning to specific hardware by the end user, although this type of tuning could further improve performance.

### 3.1 ExSeisPIOL

ExSeisPIOL is a low-level parallel I/O library that efficiently handles file access, targeting SEG-Y and other proprietary formats. The API is simple in abstraction, where the end-user employs the API to directly extract traces and parameters. Using ExSeisPIOL, the three aspects of I/O left to the end-user are memory limits, data selection, and decomposition of the data across the processes. Internally, the API uses this information to leverage MPI-I/O middleware to allow for efficient, multi-processor access to the parallel file system.

The internal architecture of ExSeisPIOL, which underlies the API, consists of an I/O stack that allows for effective communication between storage and compute processes. Each component layer of the library has a downward dependency on lower layers that can be removed individually from the topmost layer down to the bottom layer while maintaining a library which can both be compiled and used. These are split into the data layer, the object layer, and the API layer.

The data layer is the lowest level API within the ExSeisPIOL and utilizes MPI-IO. Using the data layer, data on the file system can be accessed by each process reading a contiguous data block, even if a block overlaps. Alternatively, it can be read and written as a collection of data blocks of a fixed width which are separated by a fixed offset. Each process may write a different number of data blocks.

The object layer directs which data is to be read, and prepares data to be written by the data layer. It works on the level of the file header, trace header, and trace data which are then converted into a block description. The object layer is separated from higher level layers so that access pattern optimisations can be performed independently from the packing operation used to fill the contents of the objects. The object layer maps file and header fields to their byte locations, allowing end users to call parameters

by their names instead of their file location. The specific details of SEG-Y file access are addressed in this layer.

The outward facing Parallel I/O Library (PIOL) API is designed to simplify reading and writing of file headers, trace headers, and traces. Each of these values is read or written individually, allowing for only the necessary values to aid in memory management. Parameters are read and written through calling the name of the parameter (*e.g.* xSrc, ySrc) from the object layer. Since deviations from the standard SEG-Y file and trace header fields are common, ExSeisPIOL provides a framework for introducing custom header data. End-users can be define new parameters that overload the SEG-Y standard; they do not need to refer to standard SEG-Y fields while referring to their custom fields.

### 3.2 ExSeisFlow

ExSeisFlow leverages ExSeisPIOL to create parallel seismic workflows. It implicitly handles all parallel I/O, internally optimizing the memory limits, data, and data decomposition across MPI processes. Individual header values are implicitly read and written, whereas in ExSeisPIOL, each field must be explicitly written. End users are then able to focus on operation driven development rather than any I/O requirements.

Operations in ExSeisFlow target pre-stack preprocessing and are categorized as trace, gather, or survey wide operations. These designations indicate the level of communication required; trace wide operations require communication within individual traces or trace headers, gather wide between all traces or trace headers in a gather, and survey wide between all traces or trace headers in a survey. Examples of trace wide operations include trace filtering and trace muting, gather wide include performing gathers and radon to angle transformations, and survey wide include 4D binning and sorting. Multiple operations can be called during a single workflow.

Within an ExSeisFlow workflow, all MPI processes are initialized implicitly and I/O performed at the end of the workflow. When an operation is called within an application, it is entered into a queue. Only at the end of an application, when an implicit destructor is called, any data required for operations is read in and the operations are performed.

Since ExSeisFlow is designed to work on large scale data sets, the library takes special care to implicitly optimize internal memory usage. A benefit of the operation queuing system is that all data needed for the operations is known before any I/O is performed. Therefore, only data that is needed is read in during the operation.

Each operation specifies its the data type dependencies, modification level, the modification dependencies, and the communication is tagged when it is submitted to the operation queue. These tags describe the type of data accessed (header data or trace values), how that data is modified (traces added, traces deleted, trace values modified, trace lengths modified, header values modified, or traces reordered), modification dependencies (number of traces, order of traces, value of traces, or value of trace header), and the operation communication level (trace, single gather, or survey). Using these tags, ExSeisFlow determines when a metadata or trace value will change.

Operations are performed hierarchically based on the modification dependencies and then communication tags. Any operations with modification dependencies are performed after any operations that modify that dependency. In the communication hierarchy, trace wide operations are performed first, followed by gather wide, then survey wide.

Throughout a workflow, trace and header data that will be used by multiple operations is cached to prevent redundant reads. When a trace or header value will not be modified by any subsequent operations, it is written to disk, which minimizes the amount of data in memory. After all modified data from all operations is written to disk, any remaining header or trace data that was unmodified is read in and written to the output file.

## 4 Benchmarking

The benchmarking of a parallel I/O library is strongly dependant on the specific parallel storage system on which the application is tested. The type of parallel file system, number of OSTs, number and size of file stripes, and interconnect speed, which vary between individual systems, all strongly affect the overall I/O time for an application (Carns *et al.*, 2011). Even though benchmarking numbers vary from system to system, it does demonstrate the effects of tuning the library to specific hardwares and the effects of increasing the number of processes reading and writing on overall I/O time.

ExSeisDat was benchmarked on DDN's benchmarking cluster with their Lustre PFS with IME both mounted and unmounted. The file system consists of 40 OSTs with 14 TB of storage each. All files had a stripe size of 1 MB and the MPI-I/O was optimized for Lustre access. Profiled runs on the Lustre and Lustre+IME stacks were done using 4 nodes with 16 cores per node (and one process per core) for 100 GiB, 500 GiB, 1 TB, 2 TB, 10 TB, and 20 TB SEG-Y files. While the file size was varied, the number of trace blocks was kept the same to ensure the benchmarks focused on the I/O, rather than the compute time of the sort algorithm itself.

The I/O was characterized using the profiler Darshan (Carns *et al.*, 2009), which tracks the various MPI and POSIX based calls to provide an accurate picture of the I/O-access patterns and I/O operations. The library wraps MPI function calls with a set of user space libraries using LD_PRELOAD that substitute I/O calls with its own implementation; no modifications are made to the source code. The overall overhead introduced is very low for small files (less than a GB) and continues to decrease as file sizes increase.

The library was benchmarked using the ExSeisFlow's sort utility, which sorts traces from an existing SEG-Y file based on trace header data. The minimum amount of data from each trace header is read in a distributed manner by each process. This means the memory needed per process for the entire sort is O (number of traces / number of

processes). The actual sort algorithm is a variation of a parallel quicksort (Miller and Boxer, 2012). Each process sorts its own header data, keeping track of each value's original position in the SEG-Y file. The processes then send the lower half of it's sorted values to the next lowest ranked process, also receiving data from the next highest ranked process. The data is sorted, and then the upper half of these sorted values are sent to the next highest process. This is repeated until the values are unchanged on all processes for an iteration of the algorithm. In the worst case scenario for this sort algorithm, for N trace data blocks and P processes, each process performs $O(N + (N/P) \log(N/P))$ operations with O(P) communications.

The sort utility is ideal for benchmarking the ExSeisDat libraries due to its multiple I/O patterns. With the given stripe size, individual traces are essentially contiguous reads while all file and trace headers are read and written non-contiguously. These varied read and write patterns best showcase the power of ExSeisDat.

A sort program using ExSeisFlow API can be written as:

```
int main() {
    auto piol = ExSeis::New();
    Set set(piol, ''input.segy'', ''output.segy'');
    set.sort(type);
    return 0;
}
```

It is worth noting the sort function is built into ExSeisFlow, so it is, perhaps, unsurprising the snippet is so short. However, this is indicative of the simplicity of the ExSeisFlow interface, and the expressivity of programs using the library over bespoke implementations.

## 5 Results

In order to show how ExSeisPIOL is ideal for use in existing seismic codes with complex I/O the Kirchhoff migration with a time-shift extended imaging condition used in (O'Brien *et al.*, 2017) was ported to use ExSeisPIOL for its I/O back end. The statistics presented here are not directly representative of the development effort, but they represent a rough indication. The code consisted of 6.3 KLOC and 44% of all lines were related to I/O. The port reduced the lines related to I/O by 25% and the overall code base by 16%. I/O blocks replaced by ExSeisPIOL calls included all MPI-I/O calls and the type conversions and scaling of trace header data. It is worth noting that around 36% of all code commits made to the project during its initial development were related to I/O handling and I/O optimization, with the developers reporting they were particularly tough to write. In absolute terms, the I/O development and optimization took around 450 commits while the porting to ExSeisPIOL took around 30. The run times for the Kirchhoff/ExSeisPIOL application were within 10% of the original, an expertly hand-optimized program explicitly for the Kirchhoff migration algorithm.

Leveraging IME with Lustre demonstrates significant improvement over the pure Lustre system (Fig. 2). There is a 2.8× increase in read performance and an 86× increase

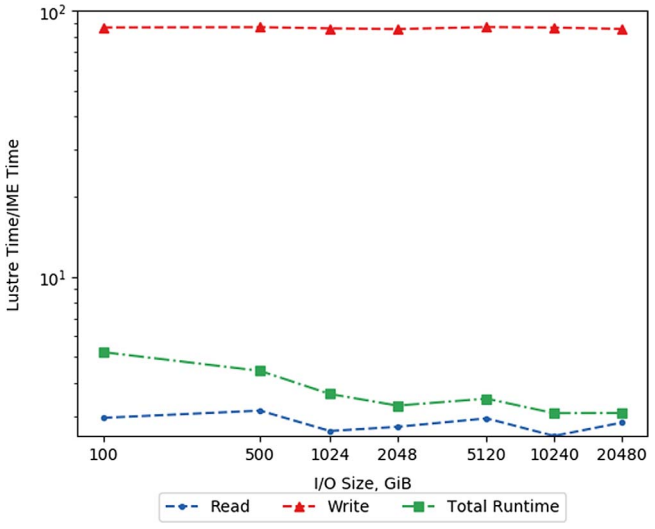


**Fig. 2.** Hardware speedup (Lustre time/Lustre + IME time) of read, write, and total runtime. The write speedup is steady across all file sizes, whereas the read and overall runtime speedup does decrease with larger files.

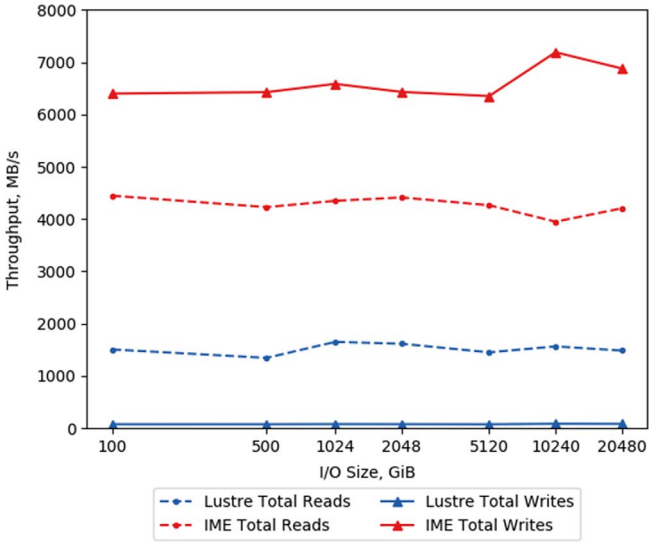


**Fig. 3.** Comparison of reading and writing throughput on the Lustre filesystem and on a Lustre-backed IME filesystem for a range of file sizes for the ExSeisFlow sort benchmark run on 32 cores. The throughput represents how quickly data can be passed between the filesystem and the parallel program. The Lustre + IME, as expected is faster than the pure Lustre system.

in write performance. As a result of the I/O speedup, leveraging the IME hardware reduced the overall runtime of the sort utility by around 27%. This reduction in runtime is solely the result of using Lustre + IME *versus* a pure Lustre filesystem; there is no speedup from the compute portion of the sort utility. Notice that the decoupling of I/O part from the compute fraction could be considered as artificial in the case where computation and I/O can be partially overlapped. The throughput rate, or the rate

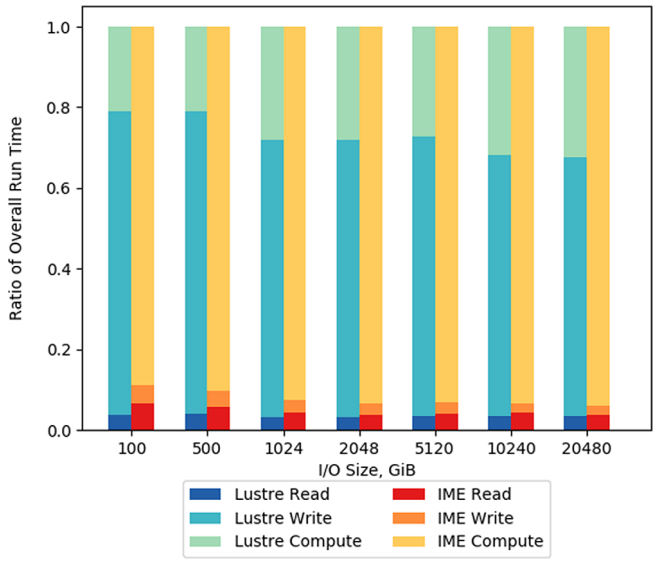


**Fig. 4.** Comparison of the fraction of runtime spent on I/O and on compute for SEG-Y files on a Lustre filesystem and on a Lustre-backed IME filesystem for a range of file sizes for the ExSeisFlow sort benchmark run on 32 cores. The execution time for the Lustre and Lustre + IME runs are individually scaled to unity, and the read, write, and compute time displayed as a fraction of the overall time. While the sort is I/O bound on the Lustre system, it is compute bound on the Lustre + IME system.

in which a processes reads from or writes to disk, remains stable for all file sizes (Fig. 3). On a Lustre system, reads are faster than writes while the reverse is true on a Lustre + IME system.

On a Lustre PFS, the application is I/O bound, averaging 73% of runtime being spent on reads and writes (Fig. 4). In particular, 70% of the total runtime is spent on writes. However, for the Lustre + IME system, the I/O consumes less an average of 8% of the runtime, making it compute bound.

## 6 Conclusion

The ExSeisDat project has produced a set highly performant libraries, immediately useful for seismic data processing in industry, and immediately usable by industry geophysicists. The libraries can be used to port existing seismic analysis code, like the Kirchhoff migration code, with significant improvements to maintainability as well as portability on new architectures and with execution times close to expertly hand-optimized codes. This porting work is expected to uncover a number of functionalities that will be useful to include in ExSeisDat. Most importantly, ExSeisDat removes obstacles for geoscience programmers, while delivering high performance. ExSeisDat approach is comforted by other effort toward shielding numericians from low level storage architectural details (Roussel *et al.*, 2017).

The benchmarking results presented here show that ExSeisDat is well suited to parallel processing of large files, up to 20 TB in this benchmark. In particular, the performance achieved by a relatively small snippet of code is remarkable. One goal of the ExSeisDat project is that most common operations on seismic data files should require as short a snippet as this, and achieve highly performant code. As other seismic data files tend to be more coherent than SEG-Y files, we expect the simplicity of code and performance to remain at least at this level with the introduction of new file formats to ExSeisDat. Unfortunately, since there are no other existing parallel implementations of SEG-Y I/O library, a direct and meaningful comparison of benchmarks between ExSeisDat and other libraries could not be included in this study.

While the ExSeisDat libraries do not necessarily require any additional hardware tuning, it would likely further increase I/O performance, and is worth further study. For example, ExSeisDat could be used for managing I/O on a cloud-based cluster, but would require further study to identify optimal strategies for parallel I/O on systems with their networking characteristics. However, jobs running on a single node with many processors, reading from a local disk should see the performance described in this paper.

Leveraging novel storage systems, like DDN's IME, further increases the I/O performance of ExSeisDat. In particular, IME greatly increases the throughput for non-contiguous writes, which describes the write pattern for many SEG-Y based processing algorithms. IME's log-based file system plays a significant role in that improvement, essentially turning non-contiguous writes to contiguous writes on its own disks, and further combines them into even more contiguous writes to the Lustre filesystem, which happens independent of the running application. The speedup of less than 3× is expected; the increase is due solely to the SSD hardware, which can read 2.7x to 3x faster than an HDD. The transition of the sort application from I/O bound to compute bound on Lustre *versus* Lustre+IME presents the opportunity for any compute optimizations (*e.g.* modifying the sort algorithm) to have a significant impact; improving compute performance would have little impact on the overall performance of I/O bound applications, common with very large data files. The 86× speedup for Lustre + IME writes is very promising for future applications burst buffer technology to seismic data files.

In summary, ExSeisDat reduces development overhead by shielding end-users from the details of the data layout, without compromising on performance. This is illustrated by our results on a modern storage architecture. We believe that the combination of high-level libraries and smarter I/O middlewares, as illustrated by ExSeisDat and IME, is a more performance portable approach to HPC development than heavily hand-tuned applications.

*Acknowledgments.* This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

## References

Barry K.M., Cavers D.A., Kneale C.W. (1975) Recommended standards for digital tape formats, *Geophysics* **40**, 2, 344–352.

Bautista-Gomez L., Tsuboi S., Tsuboi S., Komatitsch D., Cappello F., Maruyama N., Matsuoka S. (2011, Nov) FTI: high performance fault tolerance interface for hybrid systems. *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, IEEE 1-12*.

Bent J., Gibson G., Grider G., McClelland B., Nowoczynski P., Nunez J., Polte M., Wingate M. (2009) PLFS: a checkpoint file system for parallel applications. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC* '09), ACM.

Carns P., Latham R., Ross R., Iskra K., Lang S., Riley K. (2009) 24/7 characterization of petascale I/O workloads, In *Cluster Computing and Workshops, 2009. CLUSTER*'09, IEEE International Conference, 1–10.

Carns P., Harms K., Allcock W., Bacon C., Lang S., Latham R., Ross R. (2011) Understanding and improving computational science storage access through continuous characterization, *ACM TOS* **7**, 3, 8.

Ching A., Coloma K., Liao W.K., Choudhary A.N., Li J. (2007) High-performance techniques for parallel I/O, *Handbook of Parallel Computing*.

Folk M., Heber G., Koziol Q., Pourmal E., Robinson D. (2011) An overview of the HDF5 technology suite and its applications, *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, ACM, pp. 36–47.

Hustoft S., Mienert J., Bünz S., Nouzé H. (2007) High-resolution 3D-seismic data indicate focussed fluid migration pathways above polygonal fault systems of the mid-Norwegian margin, *Mar. Geol.* **245**, 1–4, 89–106.

Krischer L., Smith J., Lei W., Lefebvre M., Ruan Y., de Andrade E.S., Podhorszki N., Bozdağ E., Tromp J. (2016) An adaptable seismic data format, *Geophy. Sup. to the MNRAS* **207**, 2, 1003–1011.

Lin F.C., Li D., Clayton R.W., Hollis D. (2013) High-resolution 3D shallow crustal structure in Long Beach, California: Application of ambient noise tomography on a dense seismic array Noise tomography with a dense array, *Geophysics* **78**, 4, Q45–Q56.

Liu N., Cope J., Carns P., Carothers C., Ross R., Grider G., Crume A., Maltzahn C. (2012, April) On the role of burst buffers in leadership-class storage systems, *MSST, 2012 IEEE 28th Symposium*, IEEE, pp. 1–11.

Miller R., Boxer L. (2012) *Algorithms sequential & parallel: A unified approach*, Cengage Learning.

O'Brien G., Delaney S., Igoe M., Doherty J. (2017) Kirchhoff migration with a time-shift extended imaging condition: A synthetic example. *2017 SEG International Exposition and Annual Meeting*, Society of Exploration Geophysicists.

Roussel C., Keller K., Gaalich M., Gomez L.B., Bigot J. (2017) PDI, an approach to decouple I/O concerns from high-performance simulation codes.

Stanghellini G., Carrara G. (2017) Segy-change: The swiss army knife for the SEG-Y files, *SoftwareX* **6**, 42–47.

Vishwanath V., Hereld M., Papka M.E. (2011) Toward simulation-time data analysis and I/O acceleration on leadership-class systems, *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on 9–14*.

Virtual Institute for I/O (2018, 24 May). IO500. Retrieved from https://www.vi4io.org/io500/start