Scalable parallel seismic processing

CHARLES C. MOSHER and CALVIN L. JOYNER
ARCO Exploration and Production Technology
Plano, Texas

SIAMAK HASSANZADEH
Sun Microsystems
Mountain View, California

Seismic methods, the primary tool in the search for oil and gas, are used to produce images of the earth's subsurface that explorationists can analyze for geologic structures with hydrocarbon potential.

Two distinguishing attributes of seismic processing are size of the data and the computational complexity. A typical 3-D seismic survey can yield terabytes (more than a million floppy disks) of data with computation requirements that exceed 10^{18} floating point operations. A high end personal computer would require hundreds of years to deliver that many computations. As might be imagined, the processing of such seismic data sets places a great demand on the petroleum industry's high performance computing capacity.

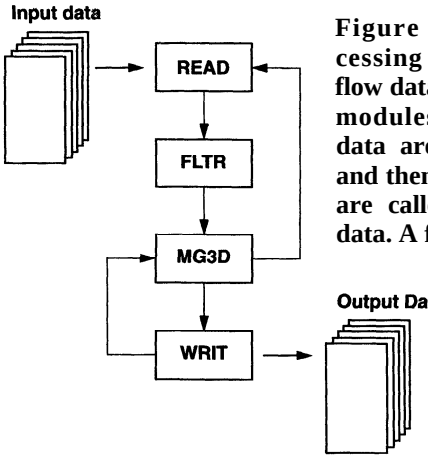


Figure 1. Seismic processing is structured to flow data through software modules. 2-D arrays of data are read from disk, and then software modules are called to process the data. A final module writes the processed data back to disk. Loops are used to create more complex flows.

Seismic data processing, however, is ideally suited for parallel computations. Several of the processing tasks are trivial to parallelize. These generally need to access a block of the data, process it in-place, and write it back to the secondary storage devices. There is no interprocessor communication during the execution of these tasks. On the other hand, a significant percentage needs to be restructured for parallel computation. This can be simplified by adopting a uniform parallel computing strategy that addresses parallelism for all parts of the processing data flow. With this approach, significant increases in processing speed can be accomplished on nearly any available commercial parallel computing platform.

The design and implementation of scalable parallel seismic data processing are illustrated by the ARCO Seismic Benchmark Suite (SBS), a public domain software system that provides an environment for development and performance analysis of parallel processing algorithms. Recently, the ARCO suite was included in the Systems Performance Evaluation Corporation's high performance computing benchmark suite (SPECseis96).

The following sections describe some common parallel programming models used in seismic processing and illustrate the use of the models with examples from SBS.

Types of parallelism. A conventional seismic processing sequence operates as a pipeline (Figure 1). A block (record or records) of seismic data is read, one at a time, from secondary storage devices and is then passed through a chain of data processing routines. A final step writes the processed data back to secondary storage devices. Loops are used to create more complex flows. There are several ways to parallelize this processing sequence.

One common approach is the *master/slave* model, where one processor collects data for all other processors and assigns tasks to the slave processors as well. In this approach, the host or master controls the sequence of operations on the slaves, with all data transfers and interprocessor communications handled by the master. Another commonly used approach is the *fan-in/fan-out* method whereby parallel tasks are spawned by a single processor to other processors. Synchronizations and data transfers are handled by the control processor, as illustrated in Figure 2.

These two approaches, often referred to as examples of control parallelism, both introduce I/O bottlenecks that can limit scalability to a handful of processors, although the underlying model is very simple and puts few demands on the programmer. This approach is also limited to those cases where little or no communications is required between processors.

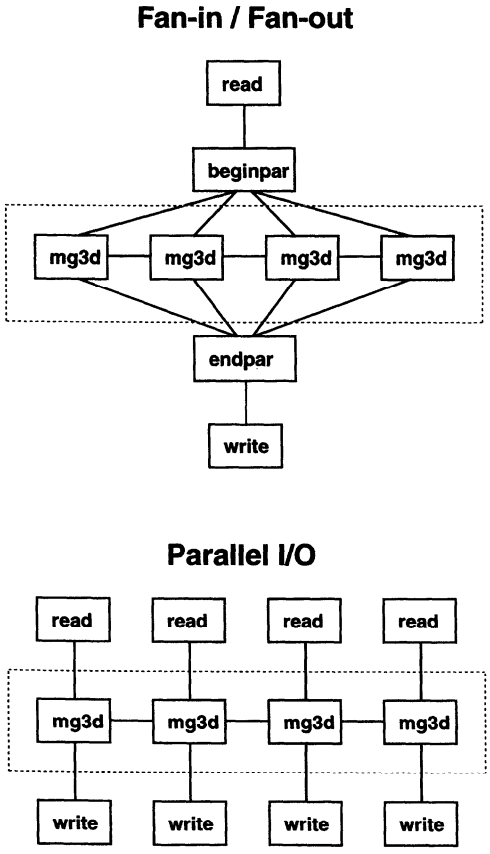


Figure 2. Parallel decomposition models for seismic processing.

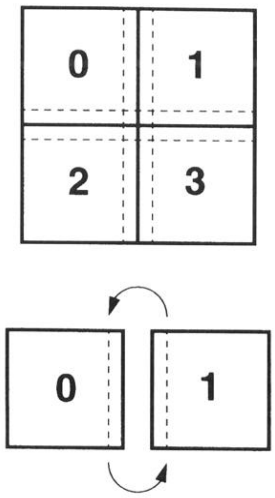


Figure 3. Domain decomposition for finite difference calculations. The finite difference grid is distributed across the processors. At the edges, data must be exchanged to complete the difference calculations.

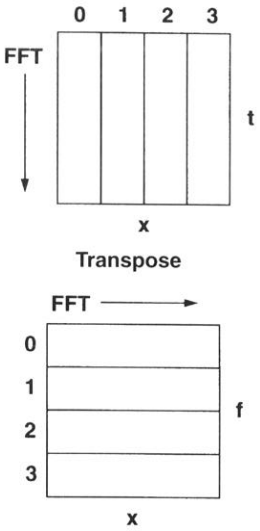


Figure 4. Transpose parallelism for a 2-D Fourier transform. Initially, the x dimension is spread across the processors, so that a block of traces is owned by each processor. The first FFT in the time direction is computed in parallel "sing simple parallelism (owner computes). A distributed transpose is then used to change the parallel dimension, so that frequency is spread across the processors. The second FFT is then computed in parallel.

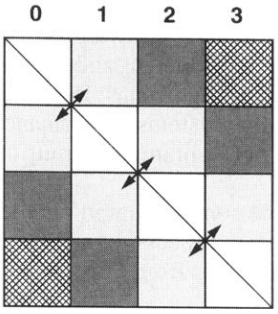


Figure 5. Distributed memory transpose routines are used to change the state of the parallel decomposition. A tiled algorithm is used (where processors exchange tiles of data) followed by a local transpose of each tile to complete the operation. The fill patterns indicate which tiles are exchanged.

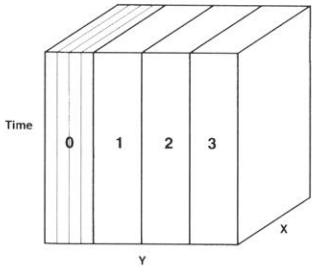


Figure 6. Initial parallel decomposition of input 3-D seismic data. The y dimension is spread across the processors, and each node manages a "slab" of t-x slices.

Seismic data, which can have as many as five dimensions in prestack form (two source coordinates, two receiver coordinates, and one time coordinate) offers many opportunities *for data* parallelism. In this model, parallelism is described in terms of the number of independent (or nearly independent) pieces that the data can be divided into. Most data parallel computations fall into three categories which we refer to us simple, domain, and transform parallelism.

Simple (embarrassingly) parallel behavior is characterized by completely independent operations that can be applied to different blocks of data. For example, if the entire data set needs to be filtered, each processor can filter a group of truces (i.e., a shot or CMP gather) independently with no communication between processors. This model of computation is also referred to as "owner computes", since each processor performs computations only on the data stored on that processor. If parallel I/O is also available, data can be read and written by all processors at once, us illustrated in Figure 2. Given I/O resources that scale with compututional resources, great speedup can be achieved using this model.

Domain parallelism, also referred to as domain decomposition, assigns subblocks or domains of the data to different processors in the parallel machine. Domain parallelism is differentiated from simple parallelism by the need for communication of information between subdomains. The most common example occurs with finite difference calculutions. where the finite difference grid is divided across the processors in a parallel computer (Figure 3). In the interior of each domain, the computations are independent, and can proceed in parallel. At the boundaries between domains, however. data from more than one block are needed for the finite difference computations. This requires moving data from one processor to another, either through shared memory that both processors can access, or by sending a message over a communication link. If either is busy, the processors must wait for the resource. Thus, communications can be a limiting factor for scalability of programs that use domain parallelism. These programs are often characterized by the ratio of "conpure-to-communicate" time in the algorithm. The higher the ratio, the more scalable the program is likely to be.

The most extensive form of parallelism used in seismic data processing is transform parallelism. This could be euphemistically referred to as "The search for the parallel dimcnsion." With multidimensional seismic data, there are many possible parallel dimensions over which computations could be distributed. In many cases, the "best" parallel dimension might change for different parts of the algorithm. In transform parallelism, the data are arranged to provide distributions across processors that match what is required by the computing algorithm. If the data do not initially match the required distribution, or if the required distribution changes, the data are redistributed across the processors using sorting and transpose routines.

As you might expect, transform parallelism provides a convenient framework for implementing multidimensional transforms. An example if the 2-D Fourier transform. Consider a 2-D array of seismic traces (Figure 4). Initially, the space dimension has been spread across the processors so that each owns a block of traces. We refer to the space dimension as the parallel *dimension* in this case. The first transform over time is computed with simple parallelism, where each processor calls an FFT routine for each of the traces it owns. The second transform over space requires transposition of the data, so that frequency rather than space is the parallel dimension. On shared memory parallel computers, the transpose could be conducted by allowing one of the processors to transpose the data using conventional techniques. However, any time you see the term "one of the processors" in the description of a parallel algorithm, you can be sure that scalability will be limited. For parallel computers with distributed memories, efficiency of the transpose becomes even more critical. Clearly, a parallel transpose algorithm is called for.

Fortunately, efficient parallel transpose routines for both shared and distributed memory architecture are readily avail-

able. An example that uses tiling is illustrated in Figure 5. The strip of data on each processor is subdivided into tiles, so that the number of tiles on each processor is equal to the total number of processors. The transpose occurs in two stages. First, processors exchange tiles across the diagonal. After the tile exchange, each individual tile is transposed locally to complete the operation. Both stages operate in parallel. After the transpose, the final FFT in the space direction can be computed using simple parallelism.

Transform parallelism to arrange the data, combined with simple "owner computes" parallel computation, provides an effective approach for scalable parallel implementation of a wide range of seismic processing algorithms.

The ARCO Seismic Benchmark Suite. SBS was designed to provide portable, parallel seismic processing algorithms for analyzing the performance of computing systems. The suite contains algorithms that are representative of a production software system and is based on a processing model similar to that used in many modern seismic processing environments. Information about SBS is available via the SEG Internet site for the Computing Applications Subcommittee.

SBS consists of three major parts:

1) *Seismic* executive which manages the processing flow, memory and timing.
2) *Application* routines which represent various operations that are customarily applied to seismic data.
3) *Utility* routines which provide for data input/output, basic mathematical operations, message passing for distributed memory systems, and examination of the results.

Nearly all source codes in the current version are written in FORTRAN 77. The directory structure, makefiles, and visualization tools are based on UNIX and X Windows, and a loosely enforced object oriented programming style is used for most applications. Future work will focus on using a stricter object oriented framework.

Rather than deal with the issues of floating point format conversions for seismic data, SBS generates data in the native format of the target machine. Moreover, the benchmark suite allows for various combinations of processing flows and data sizes to provide meaningful measurements and metrics on a variety of high performance computing platforms.

SBS makes extensive use of transform parallelism to enhance scalability. Seismic data is presented to geophysical algorithms as *parallel distributed arrays*. High level parallel algorithms for I/O, data distribution, and transposes isolate the application programs from the details of the underlying parallel computer architecture. Rather than dealing with shared memory, semaphores, message passing, or other low level parallel constructs, the application codes in SBS use high level data distribution and transpose routines to arrange the data in a way that matches the requirements of the processing algorithm. This approach has several benefits for portability, scalability, and maintenance. Portability is enhanced, since changes for particular architectures can be isolated to a few low level routines. Scalability is improved by allowing multiple levels of parallelism to be exploited by an algorithm. Finally, maintenance and readability of codes is improved, since the application now deals with geophysical operations such as Fourier transforms, filtering, phase shifts, and wavefield extrapolation, rather than the details of message passing or shared memory synchronization.

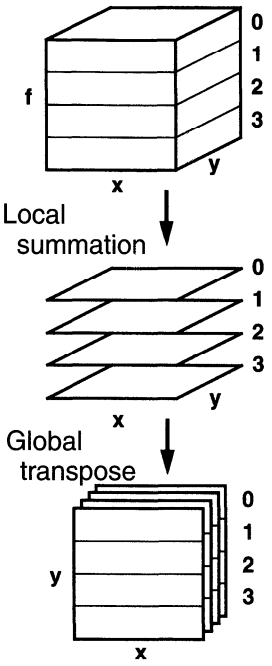
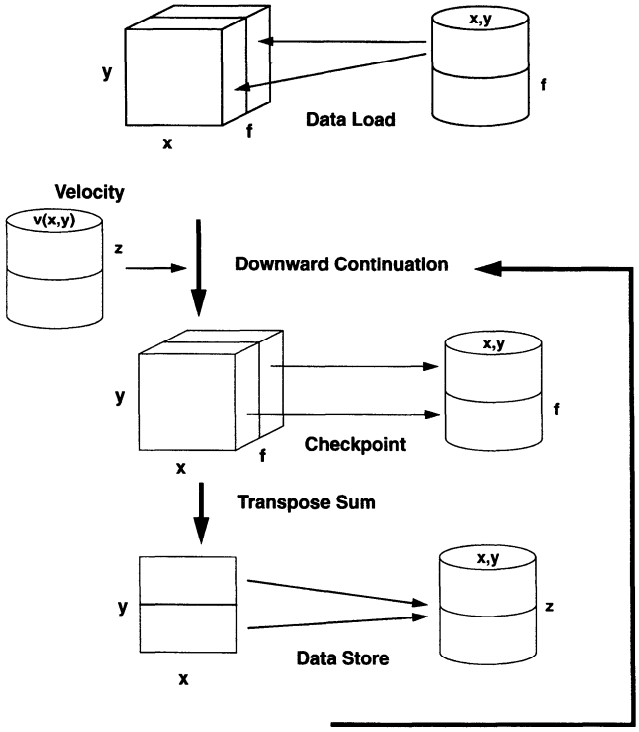


Figure 7. Parallel data reduction scheme for summing frequency slabs across processors. The x-y planes are first summed locally. A global transpose allows the final sum to also be performed in parallel. The final image plane is distributed across processors in the y direction.

Figure 8. (below) Full data flow for the parallel 3-D migration algorithm.



An example. We describe one of the SBS algorithms, 3-D poststack depth migration, as an example of a complete application.

The algorithm used in SBS is based on an implicit finite difference approach described by Zhiming Li (GEOPHYSICS, 1991). The algorithm begins with a 3-D seismic wavefield in time and space $P(t,x,y,z = 0)$ that was recorded on the surface of the earth. The data are Fourier transformed to the temporal frequency domain $P(f,x,y,z = 0)$. An equivalent wavefield one depth step deeper into the earth $P(f,x,y,z = zl)$ is obtained

from an implicit solution of a finite difference approximation to the scalar wave equation:

$$\frac{\partial P}{\partial z} = \frac{i\omega}{v(x,y,z)} \frac{\alpha S}{(1+\beta S)} P$$

where P is the seismic wavefield, ω is the angular frequency, v *(x,y,z)* is the propagation velocity, a and β are expansion coefficients, and

$$S = \frac{v^2(x,y,z)}{\omega^2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$$

The solution of these equations is referred to as "downward continuation." Given the downward continued wavefield P(f,x,y,z = zl), and image of the earth I(x,y,z = zl) is obtained by extracting the zero time (x,y) plane from P(f,x,y,z = zl), which is the sum of all frequencies. A complete 3-D image I(x,y,z) is built up one (x,y) plane at a time by recursively solving the above equations for each depth step in the output image space.

Parallel I/O and computation. SBS uses the concept of "parallel distributed arrays" to support both parallel I/O and computation. The seismic data are treated as a large regularly sampled array, with the last dimension (Fortran array indexing) spread across processors in the parallel machine. We illustrate this concept in Figure 6 for the initial step in the 3-D migration algorithm. The y dimension of the data array P(t,x,y) is spread across the processors, and each node manages a "slab" of t-x slices. Using the parallel I/O model in Figure 2, the data are read in parallel, and then Fourier transformed in parallel to yield P(f,x,y). For the downward continuation process, each frequency can be processed independently, so the parallel distributed dimension must be changed from y to f, yielding P(x,y,f). This is accomplished by transposing the data array across processors using the tiled transpose shown in Figure 5.

After the initial transform and transpose, the data are downward continued for each depth step. Since the image requires a sum of all frequencies. Data must again be moved between processors. A master-slave approach could be used, with one node collecting data from all the others, but this would create an I/O bottleneck, as in the fan-in/fan-out example in Figure 2. This bottleneck can be removed by using a parallel transpose operation, as shown in Figure 7. Each processor first performs a local summation P(x,y,f,n)->I(x,y,n) where n is the number of processors. The image array is then transposed across processors to yield I(x,n,y) with y as the parallel distributed dimension. A local sum completes the frequency summation, giving I(x,y), with y remaining the parallel distributed dimension. The data can now be written to disk in parallel, with each processor writing a strip of the image plane to disk.

Data flow for the complete algorithm is shown in Figure 8. The data are Fourier transformed, transposed, and written to disk. The imaging loop starts with the data P(x,y,f) on disk, which is read in parallel to load the data into memory. If there is not enough memory for the entire data volume, a partial image can be computed from a subset of the data, and then the partial images can be summed to form the final complete image.

In the downward continuation loop, the propagation velocity for the current depth step v(x,y,z = zl) is read from disk. Each processor requires a copy of the velocity slice. This can also create a bottleneck, so care should be taken to provide an efficient means for either interpolating a coarse representation or broadcasting a copy to all nodes. For long production runs, checkpoint volumes can be written at regular intervals to disk, allowing a restart capability. Note that all operations proceed in parallel. The speed of the application is determined by the relative mix of computer, internode communication, and disk I/O speeds of the hardware, rather than by serial bottlenecks in the algorithm.

The algorithm has been tested on both shared memory symmetric multiprocessors with 10s of processors, and on distributed memory parallel platforms with 100s of processors. Scaling tests show that intercode communication and disk I/O rates become increasingly important as the number of nodes increases. For the types of systems available today, acceptable speedups have been observed for up to 512 processors. As interconnect and external I/O rates increase, the number of processors that can be effectively used for this application also increases.

Conclusions. Scalable programs for seismic data processing require careful design of the underlying algorithm, software support for parallel operations, and balanced hardware resources for both computation and I/O. The underlying principle for parallel algorithm design should be "everything in parallel", eliminating master-slave and fan-in/fan-out approaches whenever possible. The ARCO Seismic Benchmark Suite provides an environment for testing and demonstrating these concepts. Fortunately, for seismic processing, the large amount of data involved offers many opportunities for parallelism, making scalable seismic processing an achievable goal. ■