ARCO Seismic Processing Performance Evaluation Suite

Seis 1.0 Users's Guide

October 1, 1993

Charles C. Mosher
ARCO Exploration and Technology

Siamak Hassanzadeh
Sun Microsystems, Incorporated

CHAPTER 1 Introduction

1.1 Introduction

Seismic data processing is ideally suited for parallel computation. Massively parallel computing systems are now available that can handle the demands of production seismic processing. These systems are evolving rapidly, however, and many different architectures are available. Benchmark programs that measure performance on key seismic processing tasks can be used to learn about new architectures and compare the performance of different systems. In the past, companies involved in seismic processing have used proprietary benchmark codes to perform these studies and influence decision making. In other industries, public domain codes provide standard measures of performance. As a result., individual companies can avoid the work involved in porting and analyze proprietary codes. Vendors cam also use the codes to design and test hardware and software, resulting in systems that perform better on the workload represented by the benchmark codes.

In this work, we have developed a suite of seismic processing benchmark codes for parallel processing computers. The codes are designed to be portable and offer opportunities for parallelism. Portability and parallelism are somewhat in conflict, given that there are at least 3 major parallel processing models: Fortran 77 loop parallelization, Fortran 90 array syntax, and message passing. In this release, we have consolidated the earlier F77 version into a single version using message passing as the model for parallel computing.

This current release provides Fortran 77 versions of pre-stack and post-stack seismic processing, for both 2D and 3D data examples. 3D finite difference modelling is also included. Note that benchmark metrics are incomplete in this release. The benchmark suite is being made available to vendors and other interested parties to begin the task of porting. As experience is gained in running the codes, appropriate benchmark metrics will be selected and adopted to provide a standard for measuring seismic processing performance.

CHAPTER 2 Programming Environment

2.1 Introduction

Seis1.0 is designed to be a portable, parallel environment for developing, benchmarking, and sharing seismic application codes. The underlying model for parallel computation is message passing, using a system independent message passing layer. Sample implementations for PVM (Parallel Virtual Machine) from Oak Ridge National Laboratory and NX message passing on the Intel iPSC series are included. The following sections describe source code organization, installation procedures, seismic data format, utility routines, and the parallel programming model.

2.2 Source Code Organization

All source codes in Seis version 1.0 are written in Fortran 77. The directory structure, makefiles, and visualization tools are based on the UNIX operating system and X11R3. An overview of the directory structure is shown in Figure 1.

The path to the top level directory, ‘Seis1.0’, must be set as environment variable ‘BENCH’. Binary and library archive files are managed via architecture strings stored in environment variables ‘ARCH’ and ‘TARGET_ARCH’, which should be set to a unique string representing the machine architecture. Examples are:

```
sun4            Sun SPARC
rs6000          IBM RS/6000
cray            Cray YMP
ipsc860         Intel iPSC/860
SUN4            Sun workstation cluster using PVM
```

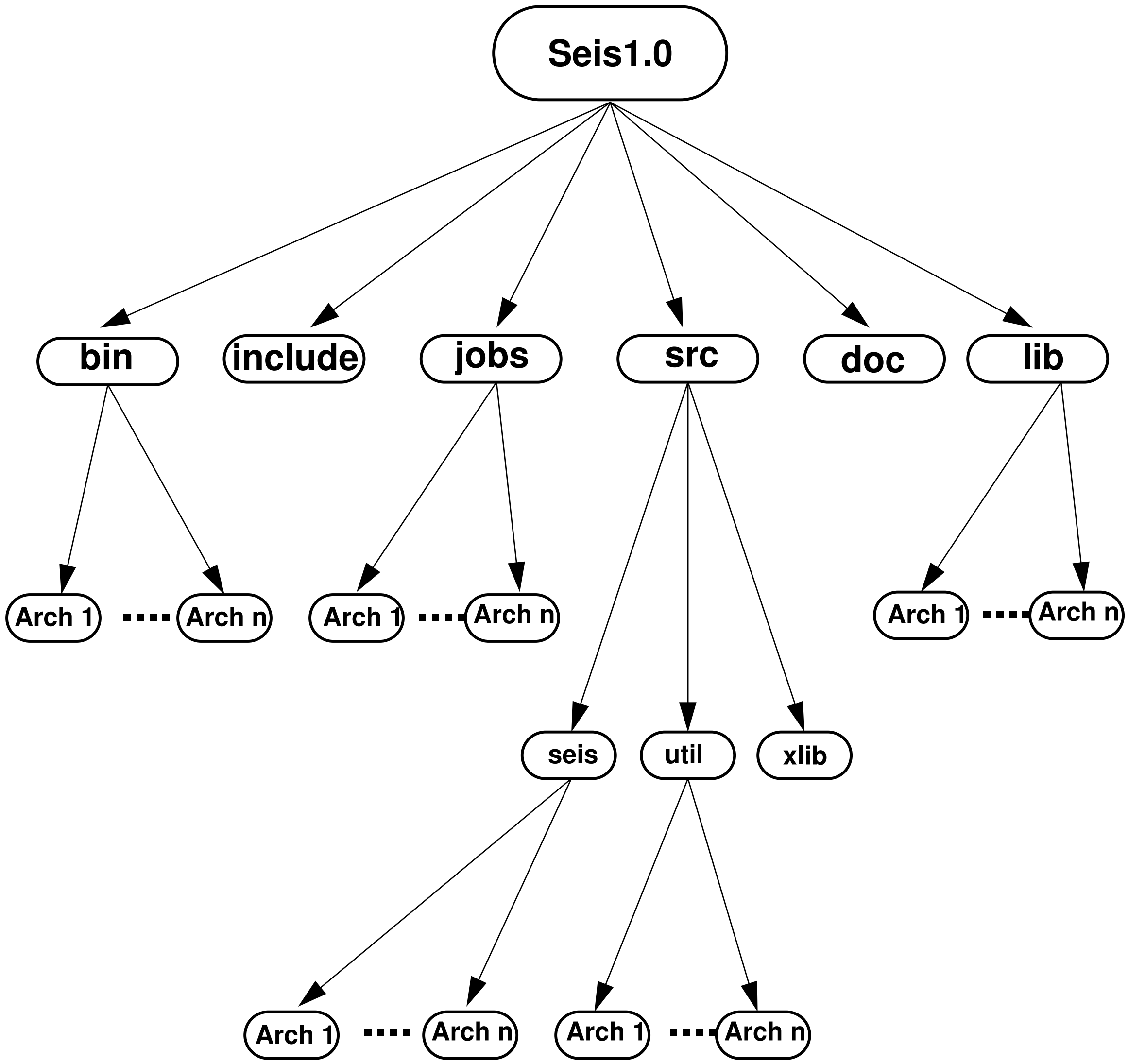
Two architecture strings are required to support cross-compilation and parallel architectures with a host/node structure. ‘ARCH’ should be set to the host architecture, and ‘TARGET_ARCH’ should be set to the node architecture. For uniprocessors, the strings should be equal. The ‘TARGET_ARCH’ variable is used to create appropriate subdirectories below ‘$BENCH/bin’ and ‘$BENCH/lib’, for customization in the ‘$BENCH/src’ directories, and to define default ‘make’ rules in the ‘$BENCH/include’ subdirectory. Rules for ‘make’ are stored in the file ‘$BENCH/include/makedef.$TARGET_ARCH’, and are included as the first line of all ‘makefiles’. This file should define ‘make’ macros for Fortran and C compilation. The following example shows rules for the IBM RS/6000.

```
#
# ARCO Seismic Benchmarks
# Default make rules for IBM RS6000 running AIX
#
FC = xlf
```

```
FFLAGS = -g -qextname
CC = cc
CFLAGS = -O -D$(ARCH)
OTHER = -lbsd
XLIB = -lX11
```

Note the use of the architecture string ‘$(ARCH)’ in the flags for C compilation, and the macros `OTHER` and `XLIB`, which are used to specify additional libraries for linking. For a more complex example, see ‘makedef.ipsc860’, which illustrates the use of ‘ARCH’ and ‘TARGET_ARCH’ for cross-compilation.

FIGURE 1 Directory Structure



Subdirectories '$BENCH/bin/$TARGET_ARCH' and '$BENCH/lib/$TARGET_ARCH' are created by the install script, '$BENCH/install'.

The 'jobs' directory contains sample run scripts and output for a few systems.

Source code is stored in the 'src' directory. Directory 'seis' contains the seismic application code, 'util' contains utility subroutines for I/O, message passing, and computation, and 'xlib' contains source for a data visualization tool. 'TARGET_ARCH' subdirectories are provided in the 'seis' and 'util' directories to allow customization for a particular architecture. 'Makefile's are provided in each source subdirectory to allow for incremental modification and re-compilation.

2.3 Installation

The source for Seis1.0 can be obtained via anonymous ftp from:

```
ftp.arco.com
```

in the file '/pub/SEG/Seis1.0.tar.Z'. The file is a compressed tar file, created using the Unix 'tar' and 'compress' commands. To install the source, move the file 'Seis1.0.tar.Z' to the location where you want the top directory 'Seis1.0' to be located, and then extract the files using the command:

```
zcat Seis1.0.tar.Z | tar xvf -
```

To compile and link the programs, move to the Seis1.0 directory, set the required environment variables, and run the install script. For example, on a Sun SPARC workstation:

```
cd Seis1.0
setenv BENCH `pwd`
setenv ARCH sun4
./install
```

The install script '$BENCH/install' creates architecture specific directories, and then traverses the source tree, executing 'make' in each subdirectory. The file '$BENCH/README' contains installation notes for the example architectures, and describes the environment variables that are required.

A small test example is provided that runs in a few minutes, and requires 2 megabytes of space in the /tmp directory on your system. To run the test example on a Sun workstation and visualize the output:

```
cd $BENCH/jobs/sun4
run.small
setenv DISPLAY unix:0
$BENCH/bin/sun4/seilook /tmp/seis/stest2
```

See the notes for 'seilook' below for instructions on operating the visualization program.

2.4 Seismic Data Format

Rather than dealing with the issues of floating point number format and tape handling, the ARCO benchmarks use a set of simple, standard I/O routines to generate and process data in the native format of the target machine. Seismic data sets consist of three files:

- An ASCII header file describing the data (path.HDR)

- A binary coordinate file containing trace coordinates (path.XYZ)

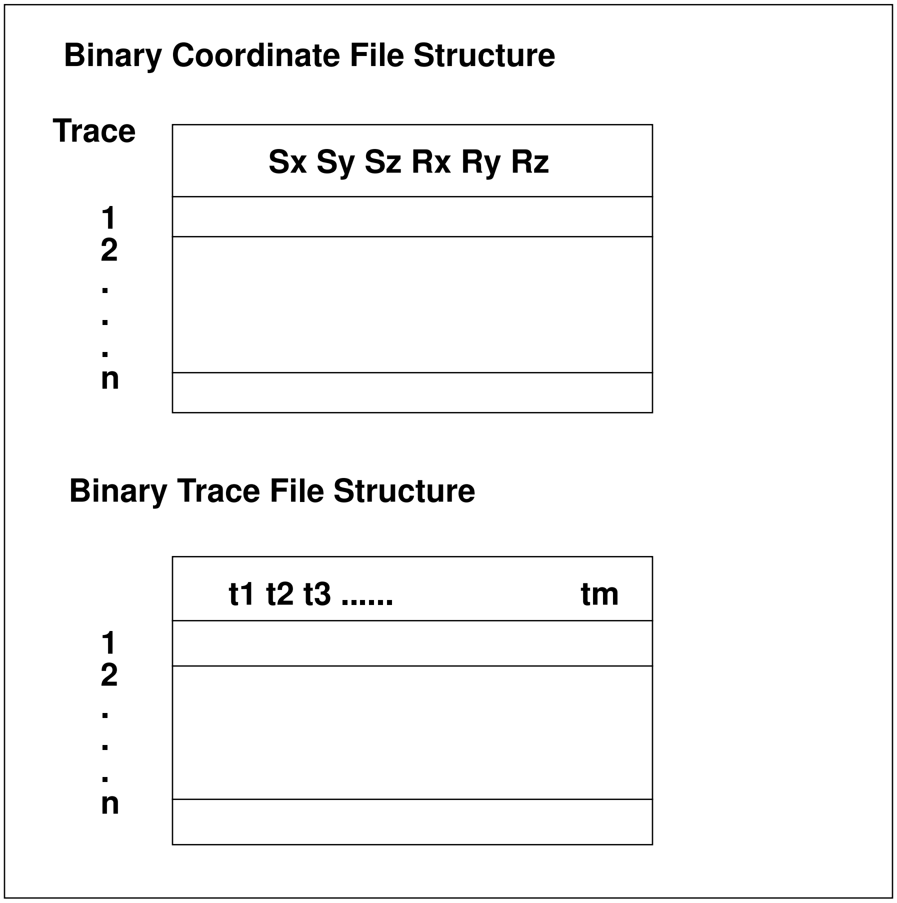
- A binary trace file (path.TRC)

The file formats are illustrated in Figures 2-3. The ‘path.HDR’ file consists of a mnemonic name followed by a floating point number on the following line (except for the group type, which is a 4 character string).The binary files are written using standard Fortran indexed I/O. The data are written one record per trace for both the coordinate and trace files.

FIGURE 2 ASCII Header File Contents

Field	Mnemonic	Description
1	SmpPrTrc	Samples per trace
2	TrcPrGrp	Number of traces per group
3	GrpPrLin	Number of groups per line
4	NLines	Number of lines
5	SampRate	Sample rate in milliseconds
6	GridX0	CDP Grid X coordinate origin
7	GridY0	CDP Grid Y coordinate origin
8	CMPSep	Separation between CDP’s on a line
9	LineSep	Separation between lines
10	GridRot	CDP Grid rotation
11	GrpType	Group type (4 characters)

FIGURE 3 Trace and Coordinate File Structure



2.5 Utility Subroutines

Utility routines are provided for seismic dataset I/O, basic math operations, and message passing. The routines are summarized below. See the source code in '$BENCH/src/util' for arguments and calling syntax.

Seismic dataset I/O ('seisio.f'):

```
seicrat - Create Seismic Benchmark File
seiopen - Open existing SBF data set
seiread - Read 2D array of traces and coordinates
          from SBF dataset
seiwrit - Write 2D array of traces and coordinates
          to SBF dataset
seiclos - close SBF dataset
seiinfo - return information about SBF dataset
```

The following basic math routines are defined in the files ‘vecsubs.f’ and ‘seifft.f’. These routines can be customized for specific architectures. See the ‘cray’ and ‘ipsc860’ subdirectories in ‘util’ for examples.

```
scopy - BLAS copy real vector
ccopy - BLAS copy complex vector
vfill - SEG vector fill
vsmul - SEG scalar-vector multiply
vsincos - vector sine/cosine
seiftm - return length and magnitude for FFT
seicft - 1D in-place complex FFT
seircft - 1D in-place real-to-complex FFT
seicrft - 1D in-place complex-to-real FFT
seimrcf - Multiple 1D real-to-complex FFT
seimcrf - Multiple 1D complex-to-real FFT
seif2rc - 2D real-to-complex FFT
seif2cr - 2D complex-to-real FFT
```

Seis1.0 includes Yet Another Message Passing Layer (YAMPL) that insulates application codes from the underlying message passing system. The following routines, defined in ‘message.c’, are used to access message passing services:

```
jpinit - parallel initialization
jmaster - return master node number
jnodes - return number of nodes in system
jnode - return node number
jsync - global sync
jkill - kill all processes
jsend - send message (synchronous)
jrecv - receive message (synchronous)
jasend - send message (async)
jarecv - receive message (async)
```

```
jamsgwt - async message wait
jbrdcst - broadcast message
jtrace - toggle tracing
jdcomp - set simple decomposition
jrange - return index range for simple decomposition
jtran23 - tiled distributed transpose primitive
```

Transpose operations on local and distributed memory arrays are defined in ‘trans.f’. The distributed transpose operations call the message passing primitive ‘jtran23’ to build higher-order transpose operations:

```
tran21 - transpose local 2D array
tran213 - transpose first 2 dimensions local 3D array
tran132 - transpose last 2 dimensions local 3D array
dtran21 - transpose distributed 2D array
dtran132 - transpose last two dimensions of 3D
          distributed array
```

2.6 Parallel Programming Model

Seismic data is inherently parallel, so many seismic computing tasks are trivial to parallelize. The ‘READ’ process simply reads a different 2D array of seismic traces on each process. The data are processed in-place, and then written back to disk in parallel. For routines that need to access the data in a different order, data can be read in transposed order, or the distributed transpose routines described above can be used to transpose memory resident data. All of the algorithms in Seis1.0 use data parallelism in conjunction with transpose operations to implement scalable parallel applications, without any direct calls to message passing routines.

Using this model, only one program is written, which is then loaded onto all the available processors. Synchronization and Master/Slave operation are only required to support file creation and standard output.

2.7 Utility Programs

Two utility programs are provided for examining SBF data sets:

```
seidump
```

Prompts for a SBF path name, then dumps ASCII, XYZ, and TRC information to standard output.

```
seilook path
```

Program 'seilook' is written in 'c' and X-windows (Xlib). Source code is located in $BENCH/src/xlib. The program loops and displays each group from the SBF file. The program has a simple event loop that looks for mouse button and key press events. The following actions are taken:

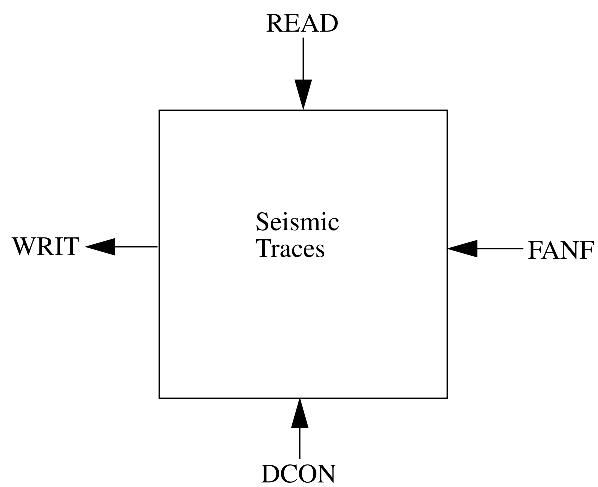
```
mouse button 1 - display next frame and halt

mouse button 2 - display previous frame and halt

mouse button 3 - display x,y coords of cursor
          and value of SBF .TRC file at that location

key 'f' - loop forward

key 'r' - loop reverse

key 'c' - cycle color scale from spectral to grey
          scale to 16 color grey scale (suitable for
          X windows dumps to PostScript printers)

key 'q' - quit

resize - When the window is resized by the user,
          simple pixel replication is used to produce
          an image that fits within the new size.
```

CHAPTER 3 Applications

3.1 The Seismic Executive

Traditional seismic processing systems operate as a pipeline. An originating process reads seismic traces, which are then passed through a chain of data processing routines. A final process writes the processed traces out to disk or tape. In this benchmark suite, a seismic processing executive is provided that manages the flow of data through a chain of processes. Data are passed through the system as a 2-dimensional array of seismic traces, or 'group' from the Seismic Benchmark file. The flow is illustrated in Figure 4. The size of the trace array is determined in a preparatory phase, and is set to hold the largest trace array that each process will manipulate. The 'READ' process reads traces and stores them in the trace array. Each process in the flow is called in turn to act on the data in-place. In the example shown, process 'FANF' applies a 2D spatial filter, process 'DCON' deconvolves each trace in the array, and process ‘WRIT’ copies the data from the trace array to disk in Seismic Benchmark Format.

FIGURE 4 Pre-Stack Seismic Processing Flow



3.1.1 Seismic Process Format

Each process consists of 4 subroutines with a fixed naming and calling convention. For a generic process ‘XXXX’, the four subroutines are the process name followed by the letters p, a, b, and c:

```
xxxxp, xxxxa, xxxxb, xxxxc
```

The ‘p’ routine is called to read input parameters and set memory requirements, the ‘a’ routine is called to initialize processing variables, the ‘b’ routine is called to act on the trace array, and the ‘c’ routine is called at termination.

The seismic executive, ‘SEIS’, manages the processing flow, memory, and timing. The calling syntax of the process subroutines is given below.

```
SUBROUTINE XXXXP( LDIM, MAXTRC, NRA, NSA, NPARM,
           ABORT, IPR )



INTEGER LDIM - Leading dimension of the 2D trace
          array required by this process.
INTEGER MAXTRC - Maximum number of traces to be
          output by this process.
INTEGER NRA - Number of words of reserved storage
          required by this process.
INTEGER NSA - Number of words of working storage
          required by this process.
LOGICAL ABORT - Abort code, set to .TRUE. to signal
          fatal errors.
INTEGER IPR - FORTRAN unit number for process
          print output.
```

A typical process reads parameters from the seismic parameter file, checks parameter syntax, and sets memory requirements. The seismic executive calls the ‘p’ routine for each process in the flow, and sets the size of the trace array to the maximum dimensions requested by each process. Reserved storage is allocated for each process, where processes store large arrays of process-dependent information. Other process-dependent information is stored in named common, ‘common /xxxx/’. One large working array is shared by all processes, whose length is the maximum length requested by all processes.

```
SUBROUTINE XXXXA (LDIM, MAXTRC, OTR, NRA, RA, NSA, SA,
          ABORT, IPR)
```

```
INTEGER LDIM - input leading dimension of the
          trace array
INTEGER MAXTRC - input second dimension of the
          trace array
REAL OTR(LDIM,MAXTRC) - input/output trace array.
          Contents are undefined for unitization step
A.
INTEGER NRA - length of permanent storage array
REAL RA - permanent storage for this process
INTEGER*4 NRA - length of working storage array SA
REAL SA - working storage for this process
LOGICAL ABORT - output abort flag, set to .TRUE.
          to abort
```

In a typical process, the ‘a’ routine is used to initialize counters, operators, and other parameters that will be fixed for the processing flow.

```
SUBROUTINE XXXXB( LDIM, MAXTRC, OTR, NRA, RA, NSA, SA,
          NTRI, NTRO, ABORT, IPR )

INTEGER LDIM - input leading dimension of
          trace array
INTEGER MAXTRC - input second dimension of trace array
REAL OTR(LDIM,MAXTRC) - input/output trace array
INTEGER NRA - length of permanent storage array RA
REAL RA - permanent storage for this process
INTEGER NSA - length of working storage array SA
REAL SA - working storage for this process
INTEGER NTRI - input number of traces
INTEGER NTRO - output number of traces, defaults to
          NTRI. A process may output a different num-
ber.
LOGICAL ABORT - output abort flag
INTEGER IPR - FORTRAN print unit number
```

The ‘b’ routine is where most of a processes work is done. A typical routine processes the traces in place, and returns the same number of traces out as were passed in. The SEIS executive uses the number of traces output as a flag. If ‘ntro = 0’, subsequent processes are skipped, and the first process in the flow is called.

```
SUBROUTIME XXXXC(LDIM, MAXTRC, OTR, NRA, RA, NSA, SA,
          NTRO, ABORT, IPR)

INTEGER LDIM - input leading dimension of trace array
INTEGER MAXTRC - input second dimension of trace array
REAL OTR(LDIM,MAXTRC) - output trace array
INTEGER NRA - length of permanent storage array RA
REAL RA - permanent storage for this process
INTEGER NSA - length of working storage array SA
REAL SA - working storage for this process
INTEGER NTRO - output number of traces, default 0
LOGICAL ABORT - output abort flag
INTEGER IPR - FORTRAN print unit number
```

Routine ‘xxxxc’ is called to terminate processing, and optionally to output traces that may have been stored in the ‘a’ or ‘b’ steps. The executive calls the ‘c’ routine when all other processes prior to the current one have completed. If a process sets the number of output traces to a non-zero value, the ‘b’ entry of subsequent processes are called to process the traces. For example, process ‘READ’ reads traces from disk in the ‘c’ subroutine. When the number of output traces is set to zero, the process will not be called again.

3.1.2 Include FIles

In addition to the named common ‘xxxx’ where a process stores parameters, include files are provided that allow access to coordinates and data attributes. The include files are:

‘coord.inc’ - contains source and receiver coordinates for each trace

‘group.inc’ - contains values from the HDR file and the current group and line number.

'para.inc' - parallel instance variables for number of nodes in system and the data decomposition.

3.1.3 Process Parameter Format

Processes use a simple format for reading parameters. A file containing ASCII characters defines the processing flow and parameters for each process. Each line in the parameter file consists of a 4 character process name, followed by fixed width fields containing parameter values. Processes call subroutine 'getparm' to return a parameter record from the parameter file, and then use Fortran internal read facilities to decode the parameter values. The processing flow is specified on a parameter record beginning with the 4 characters 'PROC'. For example, a parameter file to execute processes 'READ', 'NMOC', and 'WRIT' would look like:

```
----+----1----+----2----+----3----+----4----+----5
PROC READ NMOC WRIT
#              path
READ           /hpc/bench/data/shot
#              vnmo      stretch
NMOC           8000.0    30.0
#              path
WRIT           test
```

The first line is used for convenience to be sure parameters are justified properly. Comment lines ('#') are also used for convenience to identify parameters. To recover normal moveout velocity and stretch factor, process NMOC would execute the following code in subroutine 'nmocp':

```
integer index, ier
real vnmo, stretch
character*80 buf
```

```
index = 1
call getparm( 'NMOC', index, buf, ier )
read (buf,'(10x,2f10.0)') vnmo, stretch
```

Routine ‘getparm’ returns the ‘index’ line with leading characters ‘NMOC’ into character array ‘buf’. The error code ‘ier’ is non-zero on read errors and end-of-file conditions.

3.2 Benchmark Processes

For an overview of seismic processing, the user is referred to Yilmaz, 1987.

The following processes are provided in the initial release of SEIS:

```
DCON - seismic trace deconvolution
DGEN - synthetic data generation
DMOC - dip moveout correction
FANF - 2D spatial filtering by Fourier transform
GEOM - seismic geometry specification
M3FK - 3D Fourier domain migration
MG3D - 3D 1-pass depth migration
NMOC - normal moveout correction
RATE - processing rate computation
READ - read seismic benchmark file
FXMG - F-X domain finite difference migration
STAK - stack seismic traces
WRIT - write seismic benchmark file
XSUM - print checksum table
```

Detailed formats for process parameters are described in the file ‘$BENCH/doc/seisparm.doc.

The parallel behavior of a process can be of 3 types. A passive data parallel process simply processes the traces as passed in, and then passes out the processed traces.

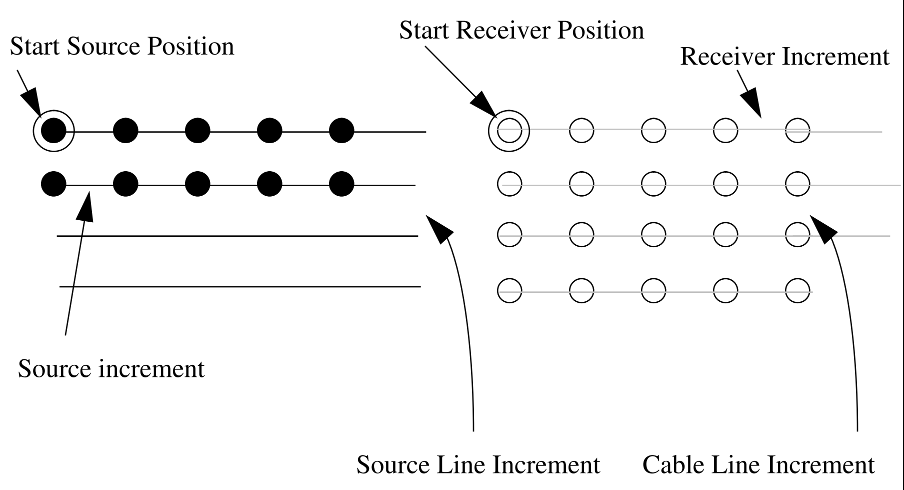
An active data parallel process serves as an originating point for data.Proecss READ is a typical example, where traces are inserted into the processing stream after being read from disk.

A complex data parallel process stores traces, operates in parallel on the stored data, and then outputs the processed data. Processes that perform seismic imaging, such as FXMG, are implemented as complex parallel processes.

3.2.1 Process GEOM

Process GEOM generates source and receiver XYZ coordinates. The basic geometry is illustrated in Figure 5. A source is moved along a rectilinear grid. The parameters for GEOM specify the start position of the source, and the movement of the source from shot to shot. When the number of shots per line is reached, the source is moved to the next line defined by the source line increment. Receivers are placed on a cable that is moved each time the source is moved. Receivers are spaced along the cable based on the receiver increment. The cable is moved between lines based on the cable line increment. Process GEOM can be used to generate pre-stack and post-stack 2D and 3D geometries. CDP stacking geometry is specified by GEOM, defined by the grid origin and delta values in CDP and line directions.

FIGURE 5 Shooting Geometry



GEOM is an active data parallel process. The data can be generated in two different modes. The default mode is to use ‘round-robin’ indexing. In this mode, shot indices are assigned to consecutive nodes. For example, on 4 nodes, node 0 will create shots (1,5,9, ...), node 1 creates shots (2,6,10, ...), and so on. In ‘BLK’ mode, shots are created in contiguous blocks. For example, on 4 nodes with 64 shots, node 0 would create shots (1,2,3, ..., 16), node 1 creates (17,18, ..., 32), and so on.

3.2.2 Process DGEN

Process DGEN generates synthetic seismic data based on the geometry from process GEOM. A band-pass wavelet is used in a constant velocity medium. A ghost pulse may be included in the wavelet for testing deconvolution. Up to 100 events may be included in the synthetics. Events may be direct arrivals, point diffractors, or planar reflectors.

For a diffracting point, travel time is computed from:

$$t = \frac{(|R - P| + |S - P|)}{V}$$

where t is the travel time, R is the receiver (x,y,z) location, P is the diffractor location, S is the source location, and V is the velocity.

Amplitude weighting is computed from:

$$a = \frac{r}{(|R - P| + |S - P|)}$$

where a is the amplitude weight, and r is the reflectivity.

For a planar reflector, the travel time and amplitude are computed by first finding the virtual source, which is the reflection of the true source across the reflecting plane:

$$t = \frac{|R - V|}{v}$$

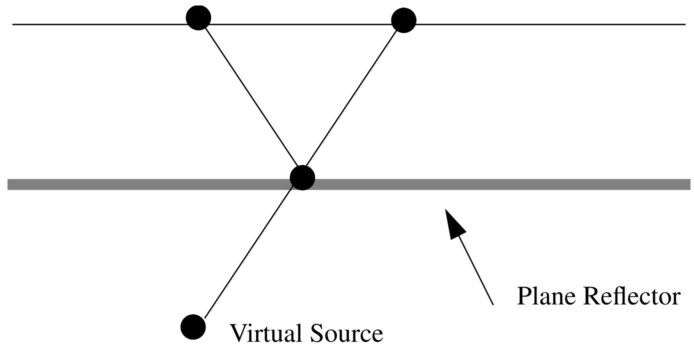
$$a = \frac{r}{|R - V|}$$

where V is the virtual source location and is given by:

$$V = S - 2\langle N, S - A\rangle N$$

where N is a normal to the reflecting plane and A is known point on the plane given by the Z axis intercept. (See FIGURE 6)

FIGURE 6 Travel Time Computation



Given the travel times and amplitudes of M events, the Fourier transform of the synthetic trace is given by:

$$f(w) = \sum_{j=1}^{M} A(w)\, r(j)\, exp(-iwt(j))$$

The output trace in the time domain is obtained by inverse Fourier transformation.

The pseudo code of DGEN can be stated as follows:

```
For each group {
For each trace{
For each reflector {
Compute reflection point, travel time, and amplitude
Add phase-shifted, weighted wavelet to output trace
}
Inverse Fourier transform trace from frequency to time
}
Pass group of traces to back to SEIS executive
}
```

DGEN is a passive data parallel process, and thus requires no special program for parallel operation.

3.2.3 Process DCON

Process DCON applies predictive deconvolution to remove ghost pulses introduced into the seismic wavelet by process DGEN. For information on deconvolution, see Yilmaz (1987), section 2.6.4. This process determines a least squares operator, which involves solving a set of Toeplitz equations. The operator is convolved with each trace to remove the ghost pulse.

DGEN is a passive data parallel process.

3.2.4 Process DMOC

Process DMOC applies residual moveout corrections to dipping events. See Hale (1988). Dip moveout is applied to common offset sections. Each input trace is integrated over elliptical travel time trajectories to produce an output dip-corrected common offset section. The elliptical trajectories are defined by:

$$t_0 = t_n \sqrt{1 - \frac{x^2}{h^2}}$$

where t_0 is the zero offset travel time, t_n is the non-zero offset time, x is the midpoint location, and h is the offset value of the current group. The range of integration on the offset section is determined by:

$$|x| \le \frac{|h|}{\sqrt{1 + v^2 \frac{t_n^2}{4h^2}}}$$

In process DMOC, the range of midpoint locations x is first determined, and then the input trace is interpolated and summed over the appropriate elliptical trajectory.

DMOC is a passive data parallel process.

3.2.5 Process FANF

Spatial filters are used to remove events with linear trajectories. Process DGEN can be used to generate linear direct arrivals that interfere with reflected arrivals. Process FANF can then be used to remove the direct arrivals. The range of events to be removed is specified as a low cut, low pass, high pass, and high cut dips in microseconds per foot. Process FANF performs a 2D Fourier transform of the input data, converting from space and time to wavenumber and frequency. In the FK domain, the time dip of an event can be computed from:

$$p = \frac{\omega}{k}$$

where p is the dip in seconds per foot, ω is temple frequency in radians per second, and k is wavenumber in radians per foot. Process FANF zeros data at dips less than the low cut and greater than the high cut. Data are tapered linearly between the cut and pass dips. The data are then returned to the space-time domain by inverse Fourier transformation, and passed back to the SEIS executive.

FANF is a passive data parallel process.

3.2.6 Process NMOC

Process NMOC applies normal moveout corrections. The parameters are the normal moveout velocity and stretch limit. A seismic trace at non-zero offset is converted to zero offset by stretching the trace in time according to:

$$t_0 = \sqrt{t_n^2 + \frac{x^2}{v}}$$

The stretched trace replaces the input trace. Output time samples stretched greater than the input stretch percentage are set to zero.

NMOC is a passive data parallel process.

3.2.7 Process M3FK

Fourier domain migration illustrates the use of multi-dimensional Fourier transforms to obtain images of geologic structure from recordings of pressure on the surface of the earth. Fourier domain migration in 3 dimensions can be accomplished by (Stolt and Benson, 1980, p. 91):

- Forward transform over (t, x, y) to (ω, k_x, k_y)
- Change of variables from $\omega \rightarrow k_z$ using the relation:

$$\omega = k_z \frac{c}{2} \sqrt{1 + \frac{(k_x^2 + k_y^2)}{k_z^2}}$$

- Inverse Fourier transform from (k_z, k_x, k_y) to (z, x, y)

The ‘M3FK’ process uses 1-D Fourier transforms to perform the 3D fft, then uses convolutional interpolation operators to perform the change of variables, followed by inverse Fourier transformation.

M3FK is a complex parallel process. Data flow is illustrated in Figure 7. Input data are required to be common offset sections in block mode (see process READ). Each common offset section is stored in memory, until a complete 3D volume is stored. The incoming data have dimensions (t,x,y). Routine ‘m3fkb’ transforms each (t,x) section to (kx,f) prior to storing in memory. The first time routine ‘m3fkc’ is called, the data are stored as (kx,f,y).

3.2.8 Processes READ and WRIT

Processes READ and WRIT read and create Seismic Benchmark files that can be browsed by program ‘seilook’. READ will read groups as specified in the HDR file, or in transposed mode to provide common offset sections for DMOC and STAK. Process WRIT creates output SBF datasets, but will not overwrite existing datasets. On parallel systems, data are read and written in parallel, so a single file system must be visible to all processors.

3.2.9 Processes FXMG and MG3D

Process FXMG performs finite difference migration on zero offset stacked sections from process STAK. A finite difference approximation to the wave equation is used to extrapolate the acoustic wavefield into the earth. The result is a depth section representing subsurface geology (Claerbout, 1986). MG3D applies an approximate version of FXMG, by applying the FXMG operator in the X and Y directions at each depth step.

3.2.10 Process STAK

Process STAK sums input traces into a zero offset section whose dimensions are determined by the CDP grid defined by process GEOM. For each trace, the output midpoint location is determined by averaging the source and receiver coordinates.

3.3 Test Examples

Parameter files are provided for 3 examples:

Small test, 2Mb output, <10 sec Cray YMP run-time:

```
small1.prm - GEOM DGEN FANF DCON NMOC WRIT
          Create and pre-process small data set
          output file '/tmp/seis/stest1.*'
small2.prm - READ DMOC STAK FXMG WRIT
          Produce 2D image of small data set
          output file '/tmp/seis/stest2.*'
```

Medium test, 400 Mb output, 1 hour Cray YMP run time:

```
medium1.prm - Same as small test
          output file '/tmp/seis/mtest1.*'
medium2.prm - Same as small test
          output file '/tmp/seis/stest2.*'
```

Small 3D test, 300 Mb output, 1 hour Cray YMP run time:

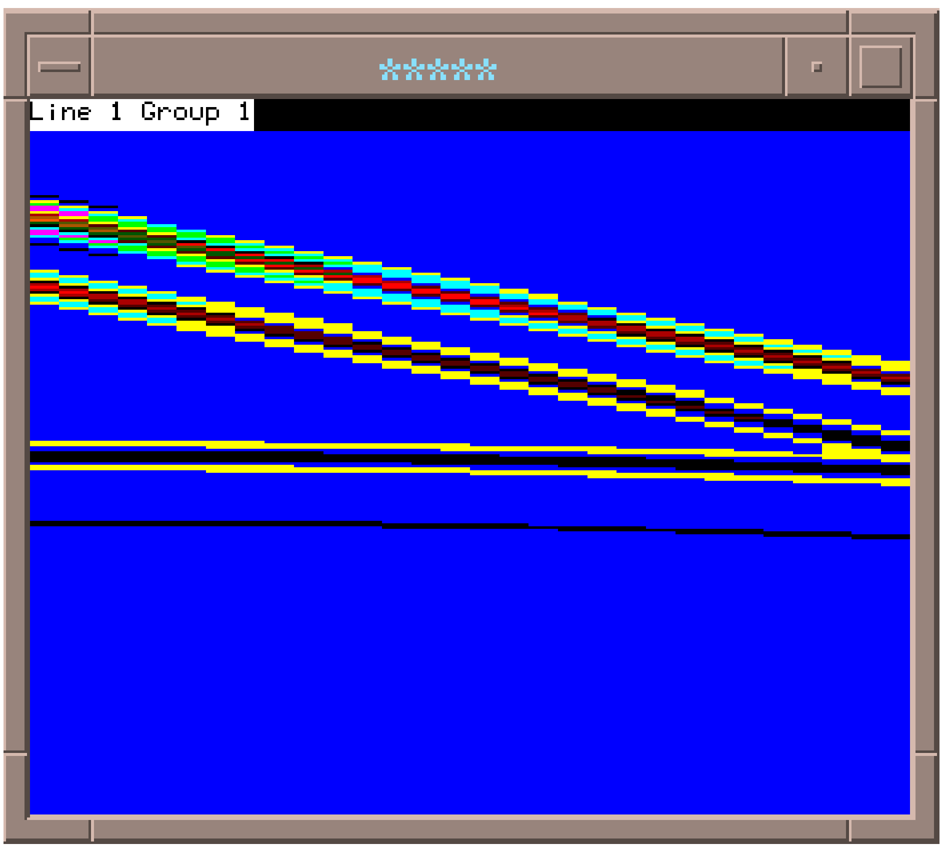
```
small3d1.prm - GEOM DGEN NMOC WRIT
          Generate small 3D data set
          output file '/tmp/seis/test3d1.*'
small3d2.prm - READ DMOC STAK WRIT
          Process and stack 3D data
          output file '/tmp/seis/test3d2.*'
```

```
small3d3.prm - READ M3FK WRIT
          FK migration
          output file '/tmp/seis/test3d3.*'
small3d4.prm - READ MG3D WRIT
          Depth migration
          output file '/tmp/seis/test3d4.*'
```

Run scripts and sample output on supported architectures for the tests above can be found in '$BENCH/jobs/$ARCH'. To run the small test on a workstation, execute the script '$BENCH/bin/sh/run.small'.

Figure 7 shows sample screen output for the output file '/tmp/seis/stest1' from 'small1.prm' using the 'seilook' utility.

FIGURE 7 Sample 'seilook' output for 'stest1' from 'run.small't

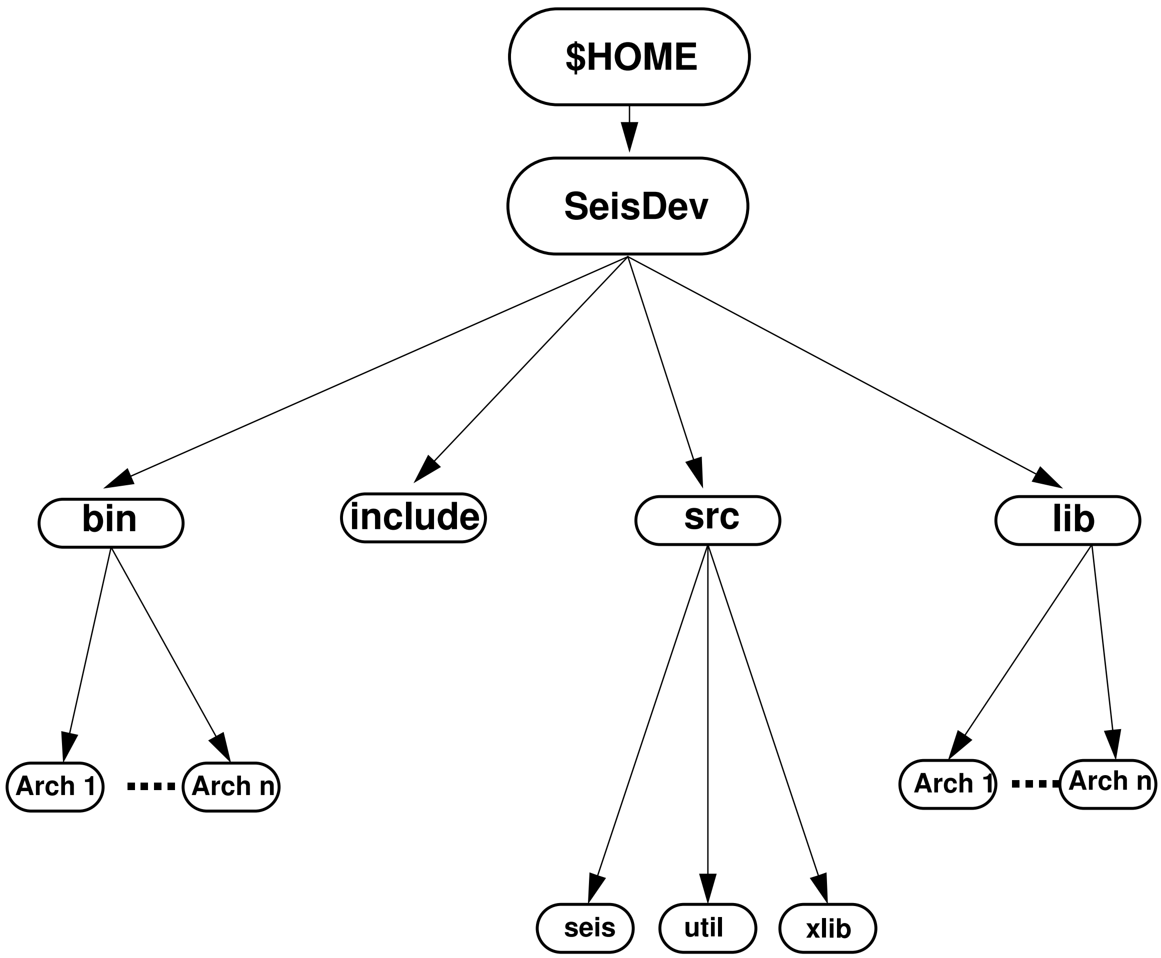


CHAPTER 4 Development Environment

4.1 Development Directory Structure

The directory structure for supporting new process development is a duplicate of the Seis1.0 directory structure, typically rooted in your home directory, referred to here as 'SeisDev' (Figure 8). The environment variable 'UBENCH' should be set to the path to the development directories (i.e $HOME/SeisDev). Architecture specific libraries are added below the 'bin' and 'lib' directories. A minimal set of include and make files are placed in the development directory structure that refer to $BENCH/src for most of the source. An example process, 'xxxx', is included in the '$UBENCH/src/seis' directory to illustrate adding a new process. The 'Makefile' is somewhat complex, since it has to link a version of 'seis' that includes both the system and user processes. Any of the source in $BENCH/src can be copied to the $UBENCH/src directories and modified.

FIGURE 8 Development Directory Structure



During the link process, the $UBENCH libraries precede the $BENCH libraries, so that any duplicate source is picked up from the $UBENCH libraries. The linked executable is stored as '$UBENCH/bin/$TARGET_ARCH/useis'.

To install the development environment, set environment variable 'UBENCH' to the path where you want the files installed, and then execute the script '$BENCH/installdev'. For example:

```
setenv BENCH (path to system files)
setenv UBENCH $HOME/SeisDev
setenv ARCH (architecture string)
setenv TARGET_ARCH (architecture string)
$BENCH/installdev
```

The install script copies the files, and then compiles and links a sample process. Process ‘XXXX’ computes the maximum amplitude in the data, and then finds the global maximum if more than one processor is used. To run the test example:

```
cd $UBENCH/src/seis
cat seis.prm
$UBENCH/bin/$TARGET_ARCH/useis
```

The test example uses GEOM and DGEN to generate 10 shot gathers. Process XXXX reports the amplitude of each shot, and then prints the maximum amplitude for all shots.

4.2 Adding Additional Processes

To add a new process ‘YYYY’, the following files must be created:

```
yyyy.f
yyyy.inc
```

The easiest way to do this is simply to copy ‘xxxx.f’ and ‘xxxx.inc’ and change occurrences of ‘xxxx’ and ‘XXXX’ to ‘yyyy’ and ‘YYYY’ using your favorite text editor. The important Fortran names are:

```
subroutine xxxxp - parameter initialization
subroutine xxxxa - run-time initialization
subroutine xxxxb - gather processing
subroutine xxxxc - cleanup processing
common /xxxx/ - process parameters
```

See Chapters 2 and 3 and source code for explanations and examples.

Prior to linking, the following files must then be modified:

```
Makefile
process.list
```

The Makefile contains the following lines:

```
#-------------------------------------------------------
# The following must have an entry for each new process
#-------------------------------------------------------

# Add user processes here
$(USLIB): \
          $(USLIB)(xxxx.o)

# Describe dependencies here
$(USLIB)(xxxx.o): xxxx.f xxxx.inc group.inc coord.inc
para.inc

# Add paramter include file for each process here too.
$(UBIN)/ucall.o: ucall.f \
          xxxx.inc
          $(FC) $(FFLAGS) -c -o $(UBIN)/ucall.o ucall.f

#-------------------------------------------------------
```

Add lines for process ‘yyyy’ as directed. Note the back-slashes in the example below, they are required as continuation line indicators::

```
#-------------------------------------------------------
# The following must have an entry for each new process
#-------------------------------------------------------

# Add user processes here
$(USLIB): \
          $(USLIB)(yyyy.o) \
          $(USLIB)(xxxx.o)

# Describe dependencies here
$(USLIB)(yyyy.o): yyyy.f yyyy.inc group.inc coord.inc
para.inc
$(USLIB)(xxxx.o): xxxx.f xxxx.inc group.inc coord.inc
para.inc

# Add paramter include file for each process here too.
$(UBIN)/ucall.o: ucall.f \
```

```
                yyyy.inc \
                xxxx.inc
                $(FC) $(FFLAGS) -c -o $(UBIN)/ucall.o ucall.f

#-------------------------------------------------------
```

4.3 Keyword Parameter I/O

Future versions of Seis will move from fixed field formats for parameters to a keyword=value format. New processes should use the new style for parameter I/O. The general form is as follows:

```
XXXX parm1=value1, parm2=value2
XXXX parm3="string with white space"
```

Records in 'seis.prm' should continue to have the process name as the first 4 characters. Parmeter=value pairs should be separated by commas if more than one pair appears on the same line. Strings containing white space should be enclosed in quotes. The following routine is used in process initialization code to retrieve parameters:

```
NAME:
sysparm - get parameter=keyword value from parameter file

SYNOPSIS:
character*(*) keyword
character*1 type
(character | integer | real) value
integer ier
call sysparm( keyword, type, value, ier )

DESCRIPTION:
Searches for seismic parameter file records with first four
characters that match process name, then attempts to locate
'keyword=value' string. Value is decoded according to type,
'I'=integer, 'F'=float or real, 'C'=character string.
Returns ier=0 on success, non-zero for no match.
```