Stanford Exploration Project, Report 111, June 9, 2002, pages 381–392

WEI: Wave-Equation Imaging Library

Paul Sava and Robert G. Clapp[1]

ABSTRACT

This paper introduces `WEI`, a new library in the SEP library of programs (`SEPlib`). The `WEI` library implements a `Fortran90` imaging engine for mixed-domain downward-continuation operators. The main imaging operators are broken into functional operators which can be modified by the user without explicit contact with I/O, parallelization etc. The code is parallelized using a combination of the Open MP and MPI standards, and can run on both shared-memory and cluster computers.

INTRODUCTION

Over the recent years, the continued increase in computer power backed by a sharp decrease in prices, and coupled with the advancements of computer cluster technology, have brought downward-continuation imaging methods, commonly referred to as wave-equation techniques, into the mainstream of seismic data processing (SEG Workshop, 2001).

Many research projects at SEP can be included in the wave-equation imaging category. All of these projects share a substantial part of their theoretical foundations, and a large part of their software infrastructure. It appears, therefore, redundant for each researcher to develop individually a substantial part of similar code.

`WEI` represents a `Fortran90` library of programs designed for wave-equation imaging using mixed-domain ($f-k, f-x$) downward-continuation operators. The main goal of this library is to provide a program engine that enables the users to develop various imaging operators without having to deal with I/O or parallelization issues. Parallelization is done over frequencies using a combination of the Open MP [2] and MPI [3] standards. The code is modular and reusable, in the sense that the main imaging operator is divided in blocks that perform specific tasks.

This paper is a brief summary of `WEI`. We present some of the theoretical background as well as the main function interfaces and parameters.

[1]**email:** paul@sep.stanford.edu, bob@sep.stanford.edu

[2]Open MP `http:www.openmp.org`

[3]Message Passing Interface `http:www.mpi-forum.org`

OPERATOR OVERVIEW

In migration by downward-continuation, the wavefield at depth $z+\Delta z$ is obtained by phase-shift from the wavefield at depth z (Claerbout, 1985)

$$\mathcal{W}(z+\Delta z) = \mathcal{W}(z)\, e^{-ik_z \Delta z}. \tag{1}$$

where the depth wavenumber k_z depends linearly through a Taylor series expansion on its value in the reference medium (k_{z_o}) and the slowness difference in the depth interval from z to $z+\Delta z$, $s(\mathbf{x},z)-s_o(z)$:

$$k_z \approx k_{z_o} + \left.\frac{\partial k_z}{\partial s}\right|_{s=s_o} (s-s_o), \tag{2}$$

where, by definition, $\Delta s = s(\mathbf{x},z)-s_o(z)$, and $\mathbf{x}$ denotes spatial position at depth z. The expression for $\left.\frac{\partial k_z}{\partial s}\right|_{s=s_o}$ can take many different forms, summarized in (Sava, 2000).

From Equations (1) and (2), we can write that

$$\begin{aligned} \mathcal{W}(z+\Delta z) &= \mathcal{W}(z)\, e^{-i\left(k_{z_o} + \left.\frac{\partial k_z}{\partial s}\right|_{s=s_o} \Delta s\right)\Delta z} && (3)\\ &= \mathcal{W}(z)\, e^{-ik_{z_o}\Delta z} e^{-i\left.\frac{\partial k_z}{\partial s}\right|_{s=s_o} \Delta s \Delta z}. && (4) \end{aligned}$$

Equation (3) represents a general form of the *main* mixed-domain downward-continuation operator. This operator can be broken up into a group of *functional* operators as follows:

- Wavefield continuation using mixed-domain phase-shift:

$$\mathcal{W}(z+\Delta z) = \mathcal{W}(z) \underbrace{\underbrace{e^{-i\,\mathtt{DSR}(s_o)\Delta z}}_{\mathtt{FK\ op}} \underbrace{e^{-i\left.\frac{\partial k_z}{\partial s}\right|_{s=s_o} \overbrace{[s_o - s(\mathbf{x})]}^{\mathtt{SL\ op}} \Delta z}}_{\mathtt{FX\ op}}}_{\mathtt{WC\ op}}; \tag{5}$$

- Imaging condition which transforms the wavefield into an image at any given depth level:

$$\mathcal{W}(z+\Delta z) \overbrace{\rightarrow}^{\mathtt{IG\ op}} R(z+\Delta z). \tag{6}$$

$\mathcal{W}(z+\Delta z)$ and $\mathcal{W}(z)$ are the wavefields at depths $z+\Delta z$ and z respectively, Δz is the depth step, s_o is the constant reference slowness in the slab from z to Δz, $s(\mathbf{x})$ is the variable slowness in the same depth slab, and $\mathtt{DSR}(s_o)$ represents the depth wavenumber expressed using the double-square root equation, and which is a function of the reference slowness (s_o).

For Equations (5) and (6), we can distinguish 5 functional operators. Each operator is initialized with a call to a function (`XXin`) and executed with a call to another function (`XXop`). In a typical example, the functional operators perform the following tasks:

1. **Wavefield continuation operator** (`WCin` & `WCop`)

   Continues the wavefield between two depth levels, using one or more reference slownesses.

   Interface: `integer function WCop(wfld,iws,izs,ith,FKop,FXop) result(st)`

   - `complex, dimension(:,:,:,:), pointer :: wfld` (wavefield slice)
   - `integer :: iws` (index of the frequency slice)
   - `integer :: izs` (index of the depth slice)
   - `integer :: ith` (thread number)
   - `FKop` :: $f-k$ operator
   - `FXop` :: $f-x$ operator

   Implemented examples:

   - `weimwc1` (mixed-domain wavefield continuation operator for 1 reference slowness)
   - `weimwcN` mixed-domain wavefield continuation operator for N reference slownesses)

2. **Slowness operator** (`SLin` & `SLop`)

   Selects the number and values of the reference slownesses (s_o), and sets-up the interpolation map between the wavefields continued using the various reference slownesses.

   Interface: `integer function SLop() result(st)`

   Implemented examples:

   - `weislo1` (slowness selector for 1 reference slowness)
   - `weisloN` (slowness selector for N reference slownesses)

3. $f-k$ **operator** (`FKin` & `FKop`)

   Performs phase-shift using the full $3-D$ DSR equation (Claerbout, 1985), the common-azimuth equation (Biondi and Palacharla, 1996), or the offset plane-waves equation (Mosher and Foster, 2000).

   Interface: `integer function FXop(iws,izs,ifk,ith,wfld) result(st)`

   - `integer :: iws` (index of the frequency slice)
   - `integer :: izs` (index of the depth slice)
   - `integer :: ifk` (index of the reference velocity)
   - `integer :: ith` (thread number)
   - `complex, dimension(:,:,:,:), pointer :: wfld` (wavefield slice)

   Implemented examples:

   - `weiwem` ($3-D$ prestack or $3-D$ offset plane-waves phase-shift)

- `weicam` ($3-D$ common-azimuth phase-shift)

4. $f-x$ **operator** (`FXin` & `FXop`)

   Performs phase shift that accounts for lateral slowness variation. Examples of ($f-x$) operators include but are not limited to split-step Fourier (Stoffa et al., 1990), local Born Fourier or local Rytov Fourier (Huang et al., 1999), Fourier Finite-Difference (Ristow and Ruhl, 1994), generalized screen propagators (Le Rousseau and de Hoop, 1998), etc.
   Interface: `integer function FXop(iws,izs,ifk,ith,wfld) result(st)`

   - `integer :: iws` (index of the frequency slice)
   - `integer :: izs` (index of the depth slice)
   - `integer :: ifk` (index of the reference velocity)
   - `integer :: ith` (thread number)
   - `complex, dimension(:,:,:,:,:), pointer :: wfld` (wavefield slice)

   Implemented example:

   - `weissf` (Split-step correction)

5. **Imaging operator** (`IGin` & `IGop`)

   Performs imaging in the offset-domain or the offset ray-parameter domain. This operator can also incorporate amplitude-preserving corrections.

   Interface: `integer function IGop(wfld,iws,ith) result(st)`

   - `complex, dimension(:,:,:,:,:), pointer :: wfld` (wavefield slice)
   - `integer :: iws` (index of the frequency slice)
   - `integer :: ith` (thread number)

   Implemented examples:

   - `weihcig` (Offset-domain common image-gathers)
   - `weipcig` (Offset ray-parameter p_h common image-gathers)

PARALLELIZATION

As it is common-practice for downward-continuation imaging methods, parallelization is done over frequencies. The library is designed for cluster computers, although it can also be used on shared-memory (SMP) machines.

In our implementation, the objects used for imaging, typically the slowness, the data and the image are spread to the compute nodes before any computation takes place. The main reasons for choosing this solution is speed and robustness, since no network traffic or remote file access happens during computation. This design is also more advantageous for inversion

where the same operation is repeatedly done on the same data which would otherwise have to be repeatedly transferred over the network.

The drawback of this design is that certain objects, for example the slowness and the image, get duplicated on all the nodes. This may be a problem for extremely big datasets, although large local storage is not so expensive and likely to further decrease in price (**?**).

Once the data is distributed to the nodes, each one of them operates independently of the other nodes until done. Typically, the local data is further broken into blocks (over depth and frequency) that can be loaded in memory.

There are at least two possibilities of distribution for the wavefield (Figure 1): in strategy 1, the node which distributes the wavefield acts as a compute node; in strategy 2, a host distributes the wavefield to the compute nodes and does not participate in the actual computation. Strategy 2 is slightly more efficient (i.e. completes the entire computation faster) than strategy 1, but it is characterized by poorer usage of the hardware (i.e. the master node is most of the time idle). Our implementation uses strategy 1.

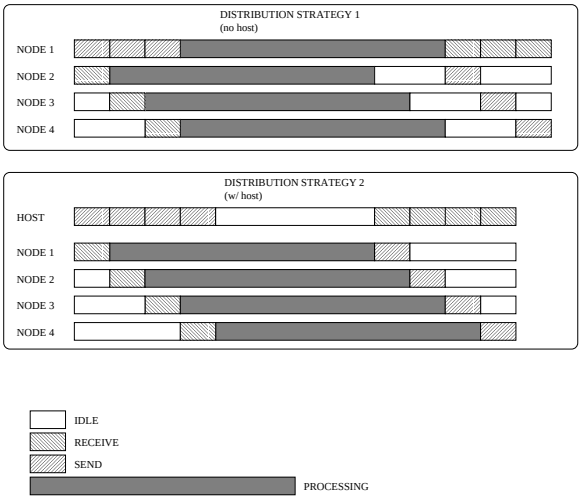

Figure 1: Two strategies for data distribution on cluster computers. paul2-weimpi [NR]

EXAMPLES

Datuming is a simple example of an operator that can be written using `WEI`. The wavefield at a given surface is recursively downward-continued in depth using a mixed-domain ($f-k, f-x$) operator using a relation like

$$\mathcal{W}(z+\Delta z) = \mathcal{W}(z)\, e^{-ik_{z_o}\Delta z} e^{-i\left.\frac{\partial k_z}{\partial s}\right|_{s=s_o}\Delta s \Delta z}. \tag{7}$$

The library is first called to initialize the functional operators

```
call weidtm_init(SLin=weislxN_init &
,                WCin=weimwcN_init &
```

```
,                FKin=weiwem3_init &
,                FXin=weissf3_init ),
```

where

- `SLin` represents the slowness operator;
- `WCin` represents the mixed-domain wavefield-continuation operator, a multi-reference slowness operator in this example;
- `FKin` represents the ($f-k$) operator, full $3-D$ prestack in this example;
- `FXin` represents the ($f-x$) operator, split-step Fourier in this example.

After initialization, the main datuming operator is called:

```
stat = weidtm(adj,add,D,U &
,                SLop=weislxN &
,                WCop=weimwcN &
,                FKop=weiwem3 &
,                FXop=weissf3 ),
```

where D and U are tags to the files storing the wavefield at the surface and at depth, respectively. Other parameters from the command line control the execution of the program (Appendix A).

A second example of operator that can be written in `WEI` is a modeling/migration pair. In this case, the main operator requires, in addition to the 4 functional operators used for datuming, a 5^{th} operator for the imaging condition. As we have done before, the `WEI` operator is first initialized

```
call weimig_init(SLin=weislsN_init &
,                WCin=weimwcN_init &
,                FKin=weiwem3_init &
,                FXin=weissf3_init &
,                IGin=weihcig_init ),
```

and then executed

```
stat = weimig(adj,add,R,D &
,                SLop=weislsN &
,                WCop=weimwcN &
,                FKop=weiwem3 &
,                FXop=weissf3 &
,                IGop=weihcig ).
```

Figure 2 shows the result of migrating the Marmousi synthetic using seven reference velocities. The main program, makefile, and parameter file are listed in Appendix A.

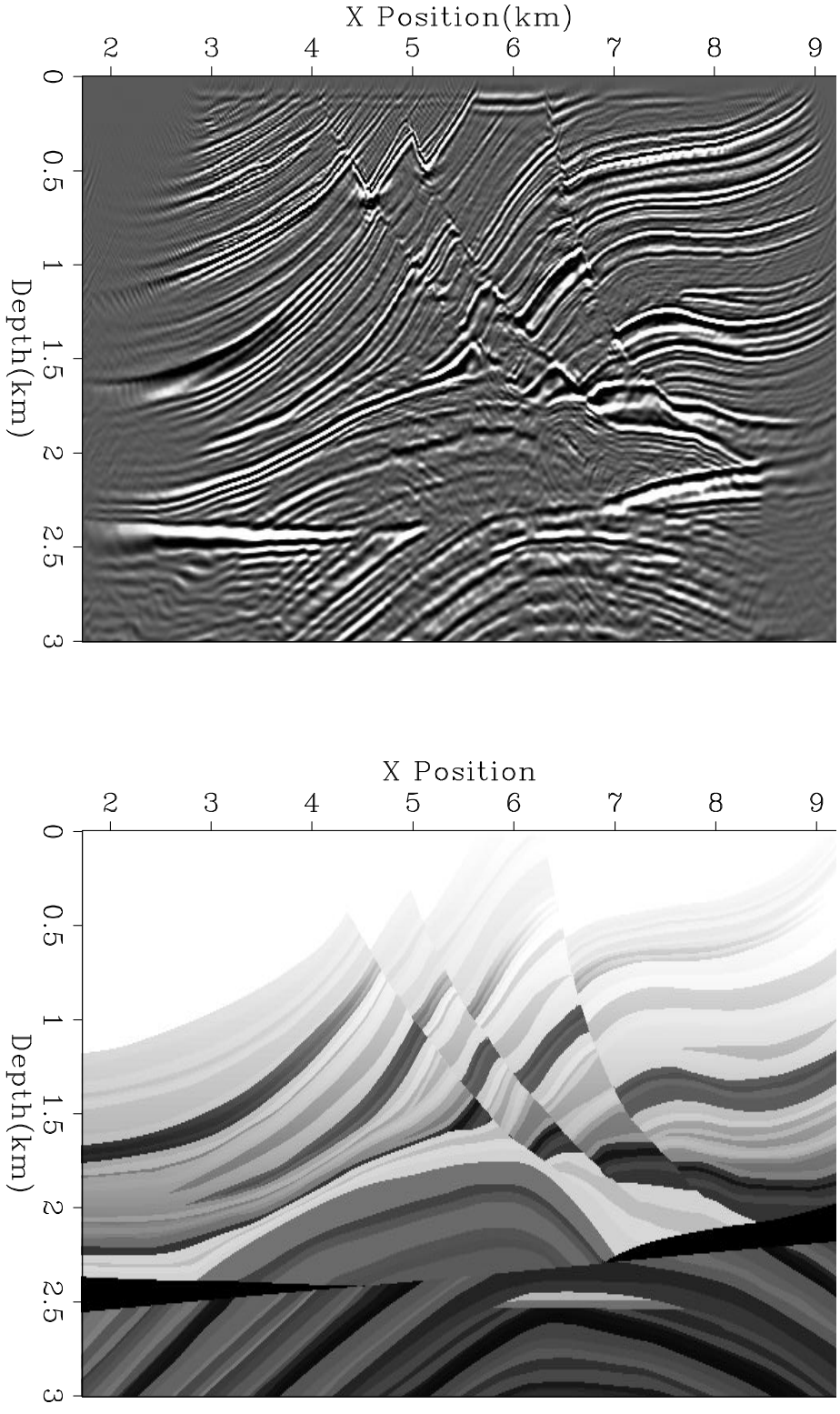

Figure 2: The top panel shows the Marmousi velocity model. The bottom panel shows the migrated image using `WEI`. The main program, makefile, and parameter file are listed in Appendix A. paul2-marm [CR,M]

REFERENCES

Biondi, B., and Palacharla, G., 1996, 3-D prestack migration of common-azimuth data: Geophysics, **61**, no. 6, 1822–1832.

Claerbout, J. F., 1985, Imaging the Earth's Interior: Blackwell Scientific Publications.

Huang, L., Fehler, M. C., and Wu, R. S., 1999, Extended local Born Fourier migration method: Geophysics, **64**, no. 5, 1524–1534.

Le Rousseau, J., and de Hoop, M., 1998, Modeling and imaging with the generalized screen algorithm: 68th Ann. Internat. Meeting, Soc. Expl. Geophys., 1937–1940.

Mosher, C., and Foster, D., 2000, Common angle imaging conditions for prestack depth migration: 70th Annual Internat. Mtg., Society of Exploration Geophysicists, Expanded Abstracts, 830–833.

Ristow, D., and Ruhl, T., 1994, Fourier finite-difference migration: Geophysics, **59**, no. 12, 1882–1893.

Sava, P., 2000, A tutorial on mixed-domain wave-equation migration and migration velocity analysis: SEP–**105**, 139–156.

SEG Workshop, 2001, Seismic imaging beyond Kirchhoff: Society of Exploration Geophysicists.

Stoffa, P. L., Fokkema, J. T., de Luna Freire, R. M., and Kessinger, W. P., 1990, Split-step Fourier migration: Geophysics, **55**, no. 4, 410–421.

APPENDIX A

Main program

```
# mini WEI example code
# Performs modeling and migration by downward continuation
# using Split-Step Fourier with N reference velocities
# Authors: Paul Sava (paul@sep) and Bob Clapp (bob@sep)
program WEImini{
  use sep+wei_util+wei_process+wei_mig+wei_cig+wei_kxmig+wei_slo+wei_ssf+wei_wem
  implicit none
  logical :: verb
  integer :: stat,ierr,iverb,n,i

#ifdef SEP_MPI
  call MPI_INIT(stat)
  call  set_no_putch()
#endif
  call sep_init(SOURCE)

#ifdef SEP_MPI
  call MPI_COMM_SIZE(MPI_COMM_WORLD, n,ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, i,ierr)
  if(i==0) {
     call from_param("verb",verb,.false.)
     iverb=0;if(verb) iverb=1
     call MPI_SEP_SEND_ARGS(n,10,iverb)
  } else call MPI_SEP_RECEIVE_ARGS()
#endif

  pro%operator=".R_.D"
  call weimig_init( SLin=weisloN_init, &
                    WCin=weimwcN_init, &
                    FKin=weiwem_init,  &
                    FXin=weissf_init,  &
                    IGin=weihcig_init  )

  stat =weimig(pro%adj,pro%add,pro%R,pro%D, &
                    SLop=weisloN, &
                    WCop=weimwcN, &
                    FKop=weiwem,  &
                    FXop=weissf,  &
                    IGop=weihcig  )
#ifdef SEP_MPI
  call MPI_FINALIZE(ierr)
#endif
  call sep_close();call exit(0)
}
```

makefile

```
#LOCATION OF DATA
DIR= /net/koko/data/data_syn/2d/marmousi/
DATA_0=${DIR}/marmcmp.H
```

```
DATA=data.H

#TRANSFORM THE DATA INTO TIME,CMPX,CMPY,OFFX,OFFY
#SWITCH TO KM FOR CONVENIENCE
${DATA}:  ${DATA_0}
        Transp plane=23 < ${DATA_0} >a.H
        Pad < a.H >b.H n3out=52 n2out=600
        Window3d < b.H n3=51 |Reverse >c.H which=4
        Cat axis=3 b.H c.H  |Pad n3out=108 >$@
        echo o2=1.725 d2=.0125 o3=-1.275 d3=.025 >>$@

#SWITCH TO KM FOR CONVENIENCE
vel_cor.view.H: ${DIR}/marmvel.H
        Window <  ${DIR}/marmvel.H j1=2 j2=2 |Scale dscale=.001 >b.H
        Cp b.H $@
        echo d1=.008 d2=.008 >>$@

#TRANSFORM DATA TO CMPX,CMPY,OFFX,OFFY,FREQ
freq.H: ${DATA}
        Transf f_min=1 f_max=40 shift2=1 wei=y < ${DATA} >$@

#TRANSFORM THE VELOCITY INTO CMPX,CMPY,CMPZ
vel.mig.H:      vel_cor.view.H
        Transp < vel_cor.view.H plane=12 |Transp plane=23 >$@

#MIGRATE THE DATA IMAGE=cmpx,cmpy,offx,offy,depth
image.H:        vel.mig.H       freq.H mig.P ${BINDIR}/WEImini.x
        ${BINDIR}/WEImini.x D=freq.H S=vel.mig.H R=$@ par=mig.P

#THE ZERO OFFSET IMAGE
image.zero.H:   image.H
        Window3d < image.H min3=0. n3=1 |Transp plane=12 >$@

#MAKE THE PICTURE
${RESDIR}/marm.v3 ${RESDIR}/marm.v:     image.zero.H vel_cor.view.H
        Grey < vel_cor.view.H allpos=y bias=1.5 >a.V label1="Depth(km)" \
   label2="X Position(km)" title=" "
        Grey < image.zero.H  >b.V label1="Depth(km)" label2="X Position(km)" title=" "
        Vppen gridnum=1,2 vpstyle=n < a.V b.V >c.V out=marm.v
        Vppen  vpstyle=n < a.V b.V >c.V out=marm.v3
```

Migration parameter file

```
operation='migration.' operator=".R_.D" #WE ARE TRANSFORMING BETWEEN IMAGE AND DATA
adj=y  #MIGRATION IS THE ADJOINT OPERATION


amy_n=1   amy_o=0.       amy_d=1.       #CMPY AXIS
amx_n=600 amx_o=1.725    amx_d=.0125    #CMPX AXIS
az__n=376 az__o=0.       az__d=.008     #DEPTH AXIS
ahx_n=64  ahx_o=-.175    ahx_d=.025     #OFFSET X AXIS
ahy_n=1   ahy_o=0.       ahy_d=1.       #OFFSET Y AXIS
aw__n=114 aw__o=0.689655 aw__d=0.344828 #FREQUENCY AXIS
```

```
velocity=y #WE ARE USING VELOCITY RATHER THAN SLOWNESS
image_real=y  #WE WANT THE IMAGE TO BE A REAL CUBE
nfk=7 #WE ARE USING 7 VELOCITIES
nzs=47 nfk=15 #NUMBER OF DEPTH AND FREQUENCIES TO HOLD IN MEMORY
```