Seismic inversion through operator overloading

Felix J. Herrmann

joint work with

C. Brown, H. Modzelewski, G. Hennenfent, S. Ross Ross

Seismic Laboratory for Imaging and Modeling

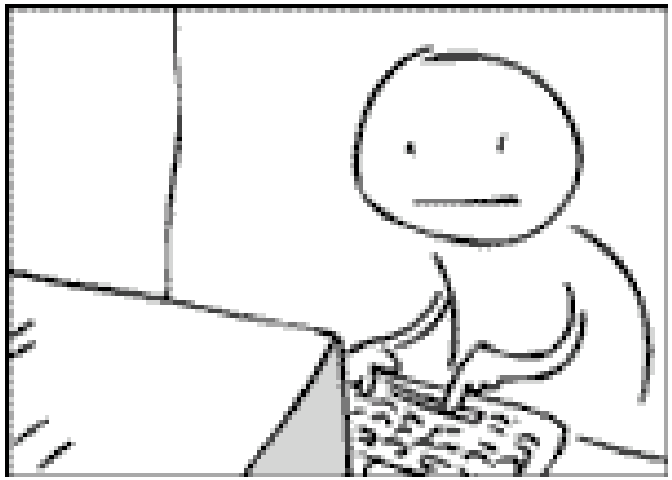
http://slim.eos.ubc.ca

AIP 2007, Vancouver, June 29

SLIM
Seismic Laboratory for Imaging and Modeling

Synopsis

- Challenge: integrate & scale "Pipe"-based software with object-oriented abstraction
  - fine-grained efficiency
  - coarse-grained abstraction
- Opportunity:
  - Create an object-oriented interface
  - Implement algorithms modeled directly from the math
  - Independent of the lower level software.
- (p)SLIMpy:
  - A Collection of Python classes:
    - vector, linear operator, R/T operation
- Benefits:
  - Reusable
  - End-Users
    - Mathematicians
    - Geophysicists



Motivation

- Inverse problems in (exploration) seismology are
  - large scale
  - # of unknowns exceeds 2^{30}
  - matrix-free implementation of operators
  - matrix-vector operations take hours, days, weeks
- Software development
  - highly technical coding <=> reusable OO programming
  - little code reuse
  - emphasis on flows <=> iterations as part of nonlinear matrix-free optimization

Motivation cont'd

- Interested in solving large-scale (nonlinear) optimization
  - element-wise reduction/transformation operations
  - linear matrix-free operators
  - nonlinear operators (future)

- Design criteria
  - code reuse & clarity
  - seamless upscaling
  - no overhead & possible speedup

Opportunity

- Create an interface for the user to implement algorithms
  - Object-oriented
  - Interfaces the ANA (coordinate-free abstract numerical algorithms) with lower level software (Madagascar)
  - Independent of the lower level software that can be
    - in-core
    - out-of-core (pipe-based)
    - serial or MPI

Our solution

- Operator overloading
  - OO abstraction of vectors and operators
  - coordinate free
- SLIMpy: a compiler for coarse grained "instruction set" for ANA's
  - concrete vector & operator class in Python
  - Madagascar = instruction set
- Interactive scripting in Python
  - math "beauty" at no performance sacrifice
  - scalable

(p)SLIMpy

- Provides a collection of Python classes and functions, to represent algebraic expressions
- Is a tool used to design and implement algorithms
  - clean code; No more code than is necessary
  - Designed to facilitate the transfer of knowledge between the end user and the algorithm designer

Context

- SLIMpy is based on ideas list by:
  - William Symes' Rice Vector Library.
    - http://www.trip.caam.rice.edu/txt/tripinfo/rvl.html
    - http://www.trip.caam.rice.edu/txt/tripinfo/rvl2006.pdf
  - Ross Bartlett's C++ object-oriented interface, Thyra.
    - http://trilinos.sandia.gov/packages/thyra/index.html
  - Reduction/Transformation operators (both part of the Trilinos software package).
    - http://trilinos.sandia.gov/
  - PyTrilinos by William Spotz.
    - http://trilinos.sandia.gov/packages/pytrilinos/

Who should use SLIMpy?

- Anyone working on large-to-extremely large scale optimization:
  - NumPy, Matlab etc. etc.
  - unix pipe-based (Madagascar, SU, SEPlib etc.)
  - seamless migration from in-core to out-of-core to parallel
- Anyone who would like to produce code that is:
  - readable & reusable
  - deployable in different environments
  - integrable with existing OO solver libraries

- *Write solver once, deploy "everywhere" ...*

Abstraction

Let data be a vector $y \in \mathbb{R}^n$.
Let $\mathbf{A}_1 := \mathbf{C}^T \in \mathbb{C}^{n \times M}$ be the inverse curvelet transform and $\mathbf{A}_2 := \mathbf{F}^H \in \mathbb{C}^{n \times n}$ the inverse Fourier transform.

Define $\mathbf{A} := \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1^T & \mathbf{x}_2^T \end{bmatrix}^T$

Solve

$$\tilde{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{x}\|_1 \quad \text{s.t.} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2 \leq \epsilon$$

```
y = vector('data.rsf')

A1 = fdct2(domain=y.space).adj()

A2 = fft2(domain=y.space).adj()

A = aug_oper([A1, A2])

solver = GenThreshLandweber(10,5,thresh=None)

x=solver.solve(A,y)
```

Vector & linear operator definition

Math	SLIMpy	Matlab	RSF
y=data	y=vector('data.rsf')	y=load('data')	<y.rsf
$A=C^T$	C=linop(domain,range).adj()	defined as function	sffdct inv=y

Instruction set for ANA's

- Solvers consist of
  - reduction/transformation operations that include
    - element-wise addition, subtraction, multiplication
    - vector inner products
    - norms l1, l2 etc
  - application of matrix-free matrix vector multiplications including adjoints
  - composition of linear operators
  - augmentation of linear operators

Reduction transformation operations

Math	SLIMpy	Matlab	RSF
y=a+b	y=a+b	y=a+b	<a.rsf sfmath b=b.rsf output=input+b > y.rsf
$y=a^T b$	y=inner(a,b)	y=a'*b	?
y=diag(a)*b	y=a*b	y=a.*b	<a.rsf sfmath b=b.rsf output=input*b > y.rsf

Linear operators

Math	**SLIMpy**	**Matlab**	**RSF**
y=Ax	y=A*x	y=A(x)	<x.rsf sffft2 > y.rsf
$z=A^H y$	y=A.adj()*y	z=A(y,'transp')	<y.rsf sffft2 inv=y > z.rsf
$\mathbf{A} = [\mathbf{A}_1 \quad \mathbf{A}_2]$	A=aug_oper ([A1,A2])	new function	complicated
$\mathbf{A} = \mathbf{B}\mathbf{C}^T$	A=comp ([B,C.adj()])	new function	complicated

Example - Define Linear Operator

- Define and apply a Linear Operator

```
#User defined fft operator
class fft_user(newlinop):
    command = "fft1"
    params  = ()

    def __init__(self,space,opt='y',inv='n'):
        self.inSpace = space
        self.kparams = dict(opt=opt,inv=inv)
        newlinop.__init__(self)

#Initialize/Define the User created Linear
Operator
F = fft_user(vec1.getSpace())
final_create = F * vec1
```

- Use predefined operators through a plug-in system.
- Create and import your own operators.
- Reuse linear operator def's.

Example - Compound Operator

- Define a compound operator:
  - form new operators by combining predefined linear operators.
  - adjoints automatically created.

```
RM = weightoper(wvec=remove,inSpace=data.getSpace())
C = fdct3(data.getSpace(),cpxIn=True,*transparams)
REMV=comp([C.adj(),RM,C])

PAD = Pad_wdl(data.getSpace(),padlen=padlen,winlen=winlen)
F = fft3_wdl(PAD.range(),axis=1,sym='y',pad=1)
T = transpoper(F.range(),memsize=2000,plane=[1,3])
PFT=comp([T,F,PAD])

P = Multpred_wdl(T.range(),filt=1,input=filterf)
PEF = comp([PAD.adj(),F.adj(),T.adj(),P,PFT])
```

interp3d_remv.py
by Deli Wang

Example - Augmented Matrix

- Define an augmented matrix:

```
#Creating vector space of a 10 by 10
vecSpace = space(n1=10,n2=10, d...

#Creating a vector of zeros from th...
x = vecSpace.generateNoisyData()
y = vecSpace.generateNoisyData()

A = fft1(vecSpace)
D = dwt(vecSpace)

# Define the Augmented Matrix
V = aug_vec( [[x],
              [y]])
L = aug_oper([[ D,A ],
              [ A,D ]])
# Multiple the two matrix's
RES = L*V
```

- Set up augmented linear systems.
- Close to visual representation of the matrix system.

Produce testable code

- Automatic domain-range checking on linear operators.
- Automatic dot-test checking on all linear operators with optional flag.
- Allow dry-run for program testing.

Dot-test check

```
OK
lsqr:~/research/tools/SLIMpy/core/SLIMpy/doc/tut cbrown$ py us
running dottests:
Dottest:
   +Comp:[SLIMpy: fft1 object, SLIMpy: fft object]
   +fft
   +dipfilter
   +Comp:[SLIMpy: fft1 object, SLIMpy: fft object]
   +Comp:[SLIMpy: fft object, SLIMpy: fft1 object]
   +fft1
   +linearop_r
   +Comp:[SLIMpy: conjoper object, SLIMpy: dipfilter object]
-building test suite
-running test suite
----------------------------------------------
Comp:[SLIMpy: fft1 object, SLIMpy: fft object]
sfmath: d1 mismatch: need 0.1
sfmath: d2 mismatch: need 0.1
sfmath: o2 mismatch: need -0.5
sfmath: d1 mismatch: need 1
sfmath: d2 mismatch: need 1
sfmath: o2 mismatch: need 0
F----------------------------------------------
fft
sfmath: d2 mismatch: need 0.1
sfmath: o2 mismatch: need -0.5
sfmath: d2 mismatch: need 1
sfmath: o2 mismatch: need 0
(-20.5373382568+9.54271221161j) == (-20.5373382568+9.542712211
1.0
.----------------------------------------------
dipfilter
F----------------------------------------------
Comp:[SLIMpy: fft1 object, SLIMpy: fft object]
sfmath: d1 mismatch: need 0.1
sfmath: d2 mismatch: need 0.1
sfmath: o2 mismatch: need -0.5
sfmath: d1 mismatch: need 1
sfmath: d2 mismatch: need 1
sfmath: o2 mismatch: need 0
F----------------------------------------------
Comp:[SLIMpy: fft object, SLIMpy: fft1 object]
sfmath: d1 mismatch: need 0.1
sfmath: d2 mismatch: need 0.1
sfmath: o2 mismatch: need -0.5
sfmath: d1 mismatch: need 1
sfmath: d2 mismatch: need 1
sfmath: o2 mismatch: need 0
```

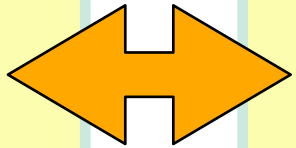
- Use --dottest flag at command prompt.
  - separate from the application at no overhead.
  - Parses the script for the linear operators.

Dot-test check

- Clean code

SLIMpy Dot-Test Code

```
domain = self.oper.domain()
range = self.oper.range()

domainNoise = self.generateNoisyData(domain)
rangeNoise = self.generateNoisyData(range)

self.operinv = self.oper.adj()

trans = self.oper * domainNoise
inv = self.operinv * rangeNoise

tmp2 = rangeNoise * trans.conj()
tmp1 = inv.conj() * domainNoise

import numpy
norm1 = numpy.sum(tmp1[:])
norm2 = numpy.sum(tmp2[:])

ratio = abs(norm1/norm2)

self.assertAlmostEqual
(norm1,norm2,places=11,msg=msg)
print ratio
```



Some of Madagascar's 200 Line Dot-Test Code

```
    if (0 == pid[4]) {
        /* reads from p[2], runs the adjoint, and
writes to p[3] */
        close(p[2][1]);
        close(STDIN_FILENO);
        dup(p[2][0]);
        close(p[3][0]);
        close(STDOUT_FILENO);
        dup(p[3][1]);
        argv[argc-1][4]='1';
        execvp(argv[0],argv);
        _exit(4);
    }
    if (0 == pid[5]) {
        /* reads from p[3] and multiplies it with
random model */
        close(p[3][1]);
        close(STDIN_FILENO);
        dup(p[3][0]);
        pip = sf_input("in");
        init_genrand(mseed);
        dp = 0.;
        for (msiz=nm, mbuf=nbuf; msiz > 0; msiz -=
mbuf) {
            if (msiz < mbuf) mbuf=msiz;

            sf_floatread(buf,mbuf,pip);
            for (im=0; im < mbuf; im++) {
                dp += buf[im]*genrand_real1 ();
            }
        }
        sf_warning("L'[d]*m=%g",dp);
```

Observations

Python's operators: +,-,*,/ are overloaded

- operator: * overloaded for linear op's

Linear operator constructor

- automatically generates adjoints & dot-tests
- keeps track of number type and domain & range

Constructors for

- compound operators
- augmented systems

Lurking problem w.r.t. efficiency ...

- y=A*x+A*z not as fast as y=A*(x+z)

Abstract Syntax Tree (AST)

- Operator overloading allows us to create an Abstract Syntax Tree (AST)

- Abstract Syntax Tree allows for
  - analysis compound coarse-grained statements in ANA's
  - remove inefficiencies
  - translate statements into a concrete instruction set
  - do optimization

Abstract Syntax Tree (AST)

- An AST is a finite, labeled, directed tree where:
  - Internal nodes are labeled by operators
  - Leaf nodes represent variables/vectors

- AST is used as an intermediate between a parse tree and a data structure.

Piped-based optimization

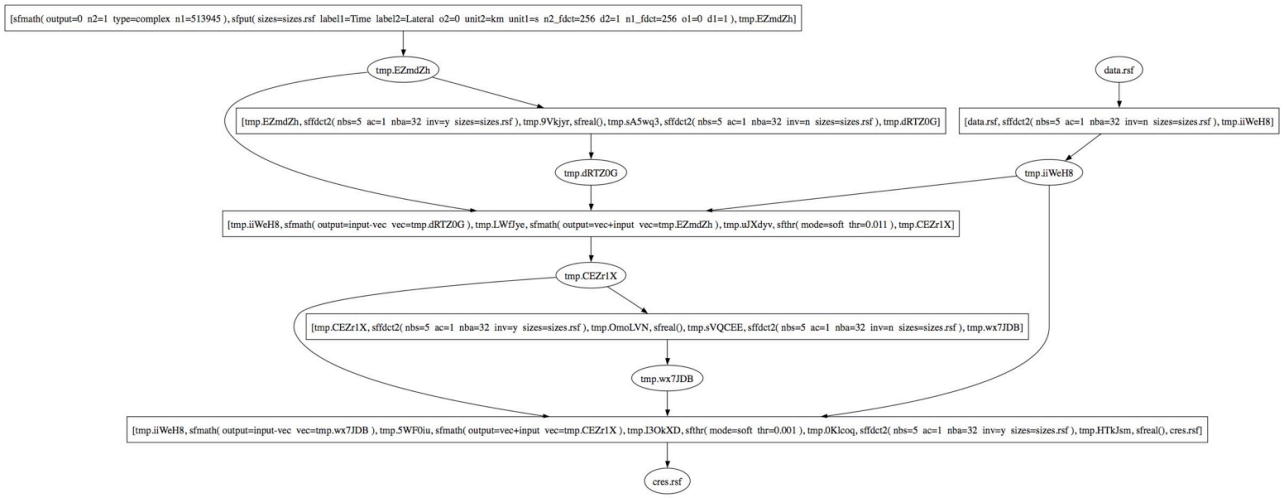
- Executing single unix pipe-based commands is inefficient
  - better to chain together
  - reduce disk IO
- becomes complex when iterating

```
/Volumes/Users/dwang/tools/python/2.5/bin/python ./interp3d_remv.py -n data=shot256.rsf filter=srme256.rsf
remove=wac3D_256.rsf transparams=6,8,0 solverparams=2,1 eigenvalue=40.2 output=interp80_fcrsi.rsf
mask=mask256_80.rsf flag=1
sfmath n1=256 n2=256 n3=256 output=0 | sffdct3 nbs=6 nbd=8 ac=0 maponly=y sizes=sizes256_256_256_6_8_0.rsf
None < sfmath  output="0" n2="256" n3="256" type="float" n1="256" | sfput  head="tfile" o3="0" o2="0" d2="15" d3="15"
o1="0" d1=".00" > tmp.walacs  ()
shot256.rsf < sfcostaper  nw1="15" nw2="15" > tmp.uNaFyB  ()
None < sfmath  output="0" n2="256" n3="256" type="float" n1="256" | sfput  head="tfile.rsf" d2="15" o2="0" o3="0"
d3="15" o1="0" d1=".00" > tmp.hyqCQa  ()
None < sfcmplx tmp.uNaFyB tmp.hyqCQa  | sfheadercut  mask="mask256_80.rsf" | sffdct3  nbs="6" ac="0" nbd="8" inv="n"
sizes="sizes256_256_256_6_8_0.rsf" > tmp.XiJ7Xs  ()
wac3D_256.rsf < sfmath  output="vec*input" vec="tmp.XiJ7Xs" | sffdct3  nbs="6" ac="0"
sizes="sizes256_256_256_6_8_0.rsf" inv="y" nbd="8" > tmp.KdkI78  ()
srme256.rsf < sfcostaper  nw1="15" nw2="15" > tmp.C1Qrmt  ()
None < sfcmplx tmp.C1Qrmt tmp.walacs  | sffdct3  nbs="6" ac="0" nbd="8" inv="n" sizes="sizes256_256_256_6_8_0.rsf" >
tmp.TvVRiJ  ()
wac3D_256.rsf < sfmath  output="vec*input" vec="tmp.TvVRiJ" | sffdct3  nbs="6" ac="0"
sizes="sizes256_256_256_6_8_0.rsf" inv="y" nbd="8" > tmp.DZjYCe  ()
```

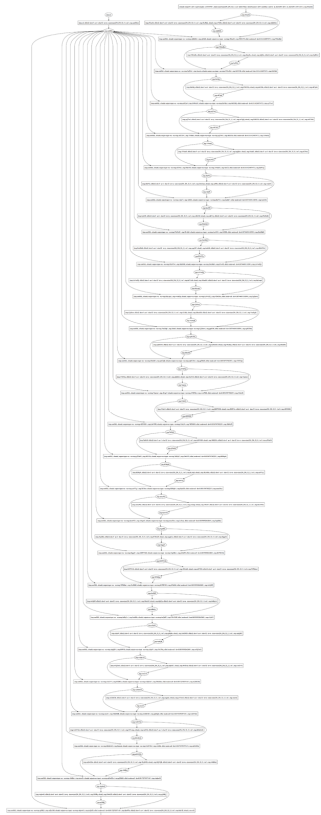
Visualization

dnoise.py OuterN=5 InnerN=5 --dot > dnoise_5x5.dot

3 iterations



25 iterations



Optimization

- Currently implemented:
  - Unix pipe-based optimization
    - Unique to SLIMpy
    - Assembles commands into longest possible pipe structures
  - "Language" Specific Optimization
    - Madagascar
- Goals within reach:
  - Symbolic Optimization
    - eg. A(x) + A(y) = A(x+y)
  - Parallel Optimization
    - Load balancing

Optimization

- Optimization of the AST is modular.
- Users can
  - daisy chain each optimization function together
  - specify which optimizations to perform

```
#get the current AST
Tree = getGraph()
# perform Optimizations
O1 =  symbolicOptim( Tree )       # Perform Symbolic Optimizations
O2 =  pipe_Optim( O1 )            # Optimize for Unix pipe Structure
O3 =  language_rsf_Optim( O2 )  # Optimize for RSF
```

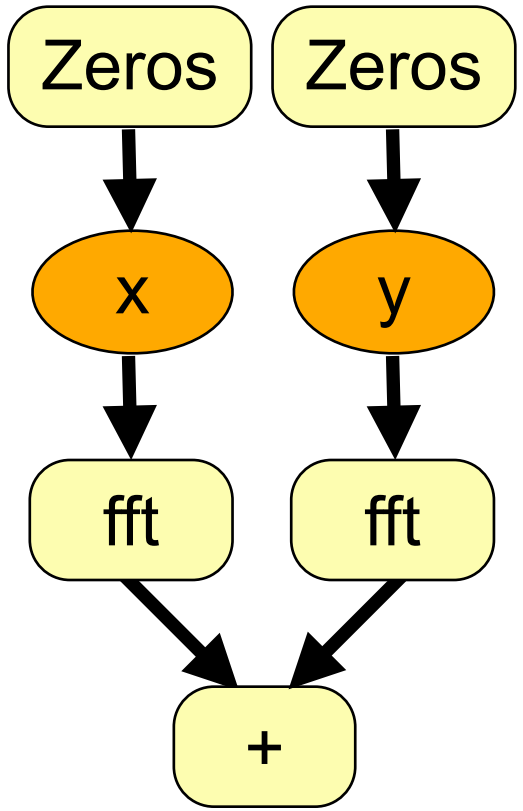
Example

- Calculate tree
- Walkthrough
  - Simple three step optimization

Code

```
# Create a Vector Space of a 10 x 10 image
vecspace = VectorSpace(n1=10,n2=10,plugin="rsf")
x = vecspace.zeros()      # Vector of zeros
y = vecspace.zeros()      # Vector of zeros
A = fft( vecspace )       # fft operator
V = aug_vec( [[x],[y]] )  # augmented vector system
M = aug_oper([[ A, 0 ],   # augmented operator
              [ 0, A ] ) # system

ans = M(V)      # apply the operator to the vector
```

AST

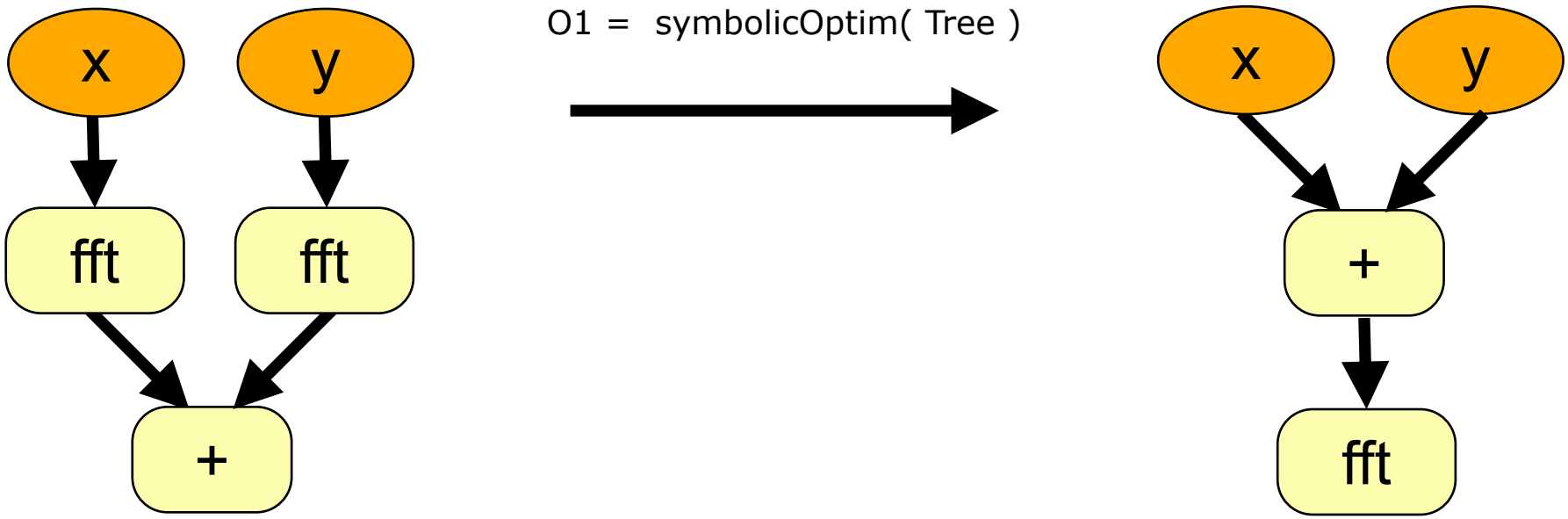


Symbolic optimization

- More efficient to add the vectors first. If we know **A** its a Linear Operator.
  - only do one FFT computation

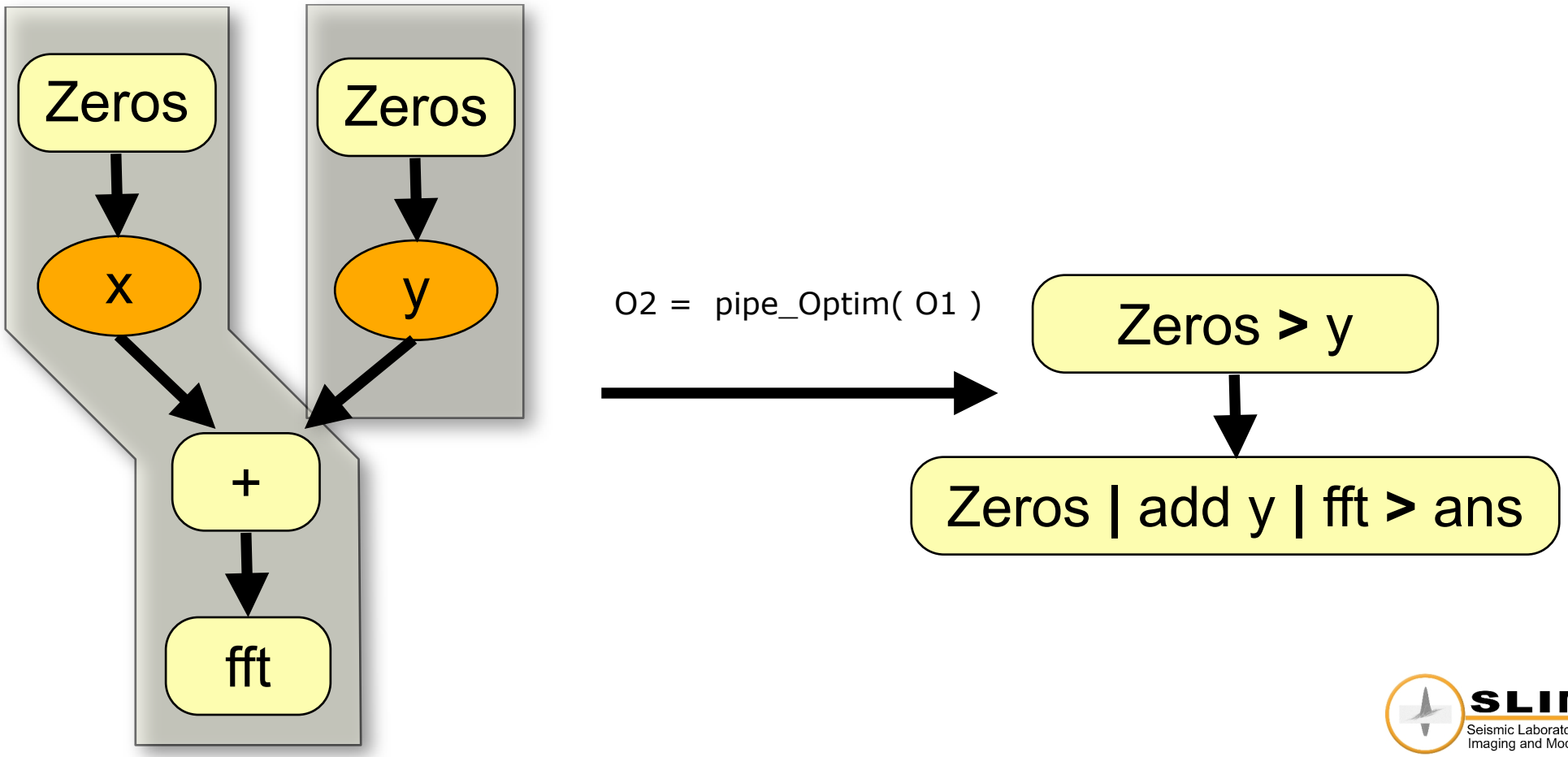
Ax + Ay

A(x + y)



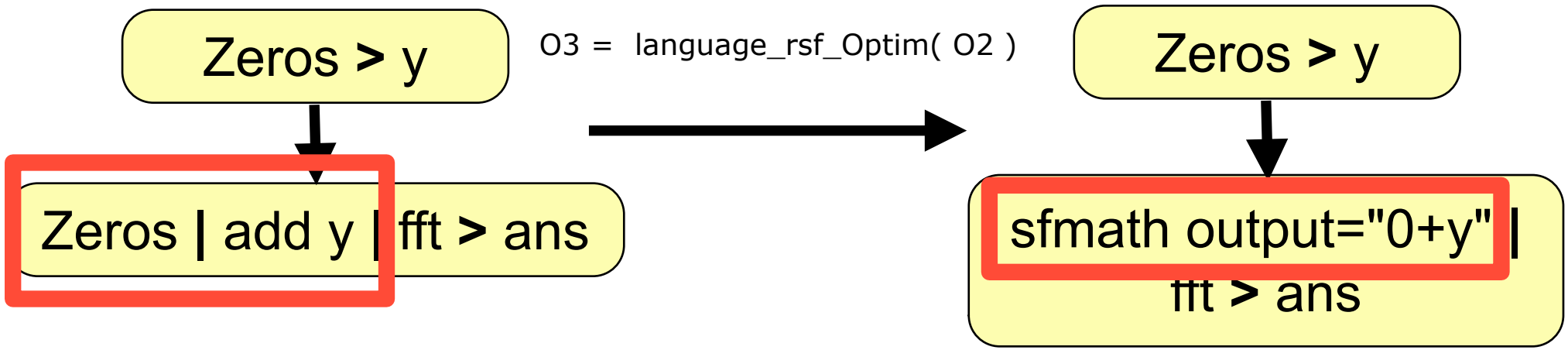
Unix Pipe optimization

- Compresses individual commands into minimal number of command pipes.



Language specific

- find shortcuts to reduce workload



At What Cost

- What are the performance costs?
  - linear time - with respect to the number of operations
  - depends on the optimize functions used

Performance cost

- 100 iterations of the solver

dnoise.py OuterN=10 InnerN=10 --debug=display ...

Display:

Code ran in : **1.09** seconds
Complexity : **1212** nodes
Ran : **302** commands

- 900 iterations of the solver

dnoise.py OuterN=30 InnerN=30 --debug=display ...

Display:

Code ran in : **6.51** seconds
Complexity : **10812** nodes
Ran : **2702** commands

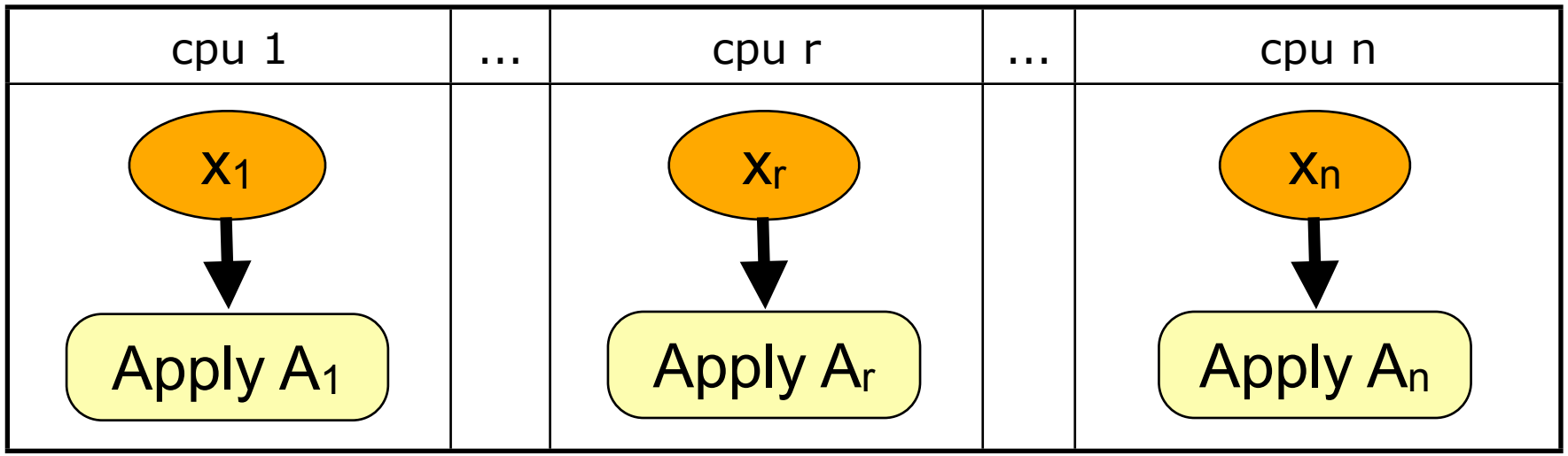
Pathway to parallel

- SLIMpy is scalable
  - Includes parallellization
- Embarrassingly parallel
  - Separate different branches of the AST
- Domain decomposition
  - Slice the data-set into more manageable peaces
  - Domain decomposition with partition of unity

Embarrassingly parallel

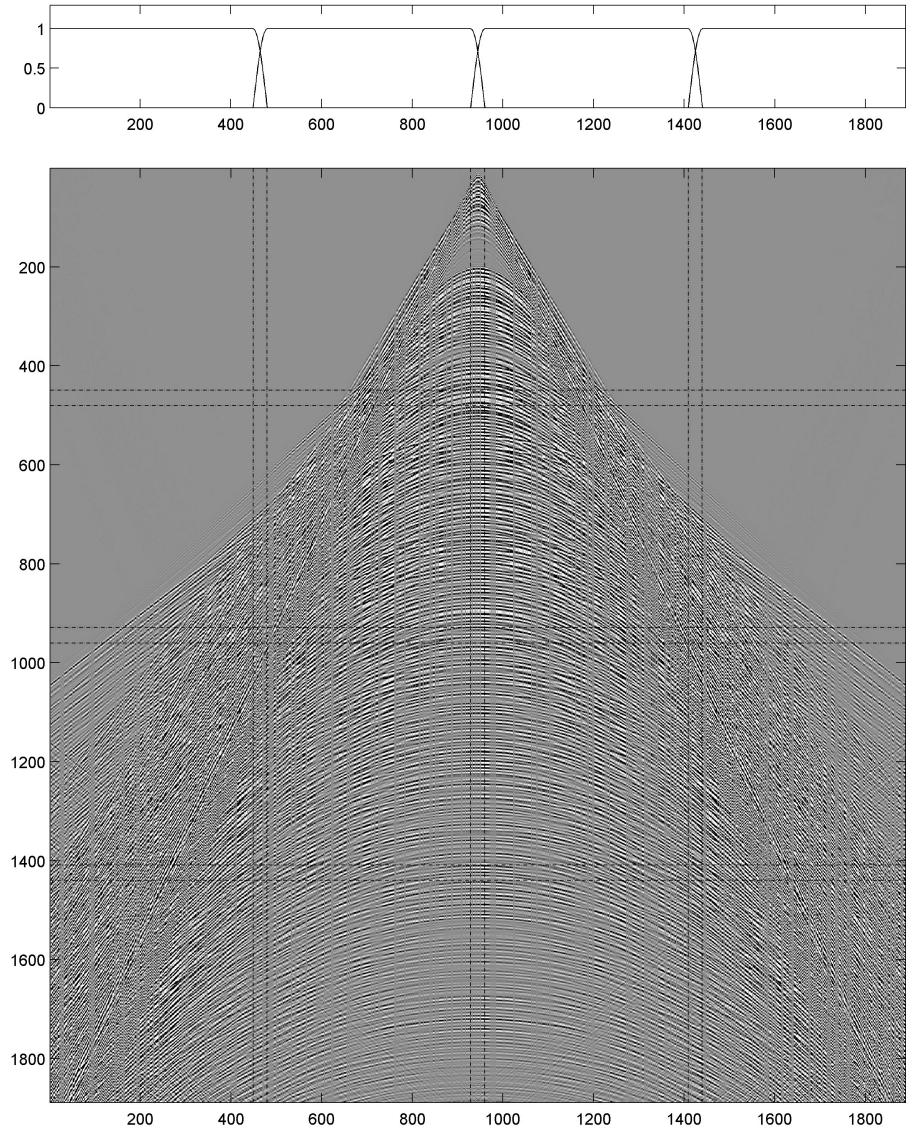
- Separate branches of the AST can be easily distributed to different processors.

```
V = aug_vec( [[x1], ..., [xn] )  # augmented vector system
M = aug_oper([[ A1,     ...,      0 ],   # augmented operator
              [  0, ..., Ar, ..., 0 ]
              [ 0,      ...,      An ] )
ans = M * V
```



Domain decomposition

- 4 windows along each dimension
- Taper functions shown on top
- Partition of unity
- 2D/3D Overlap arrays in-core with PETSc
- Dashed lines show window edges
  - space in between lines is tapered overlap



Domain decomposition

- dnoise.py
  - serial

```
dnoise.py data=data.rsf output=res.rsf [pSLIMpy options]
```

  - parallel

```
mpirun [options]
  dnoise.py data=data.rsf output=res.rsf [pSLIMpy options]
```

  - presto!

In Mathematical Terms...

- Linear operator $\mathbf{A}$
  - communicates overlap
  - converts non-overlapping windows to overlapping ones
- Linear Operator $\mathbf{B}$
  - applies taper at edges of windows
- Combined operator $\mathbf{T} = \mathbf{BA}$ satisfies:
  - $\langle \mathbf{Tx}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{T}^H \mathbf{y} \rangle$
  - $\mathbf{T^H T} = \mathbf{I}$

Performance

- Simple benchmark test
  - 10 forward/transpose PWFDCT's

Decomposition	Global Size	Local Size	Overlap	Execution Time
2x2	1024x1024	544x544	16	51.97
		630x630	64	66.94
	2048x2048	1056x1056	16	201.85
		1152x1152	64	237.83
4x4	2048x2048	544x544	16	52.67
		630x630	64	67.26
	4096x4096	1056x1056	16	202.53
		1152x1152	64	240.25
8x8	4096x4096	544x544	16	61.89
		630x630	64	87.84
	8192x8192	1056x1056	16	220.68
		1152x1152	64	257.50

Future Benefits of AST

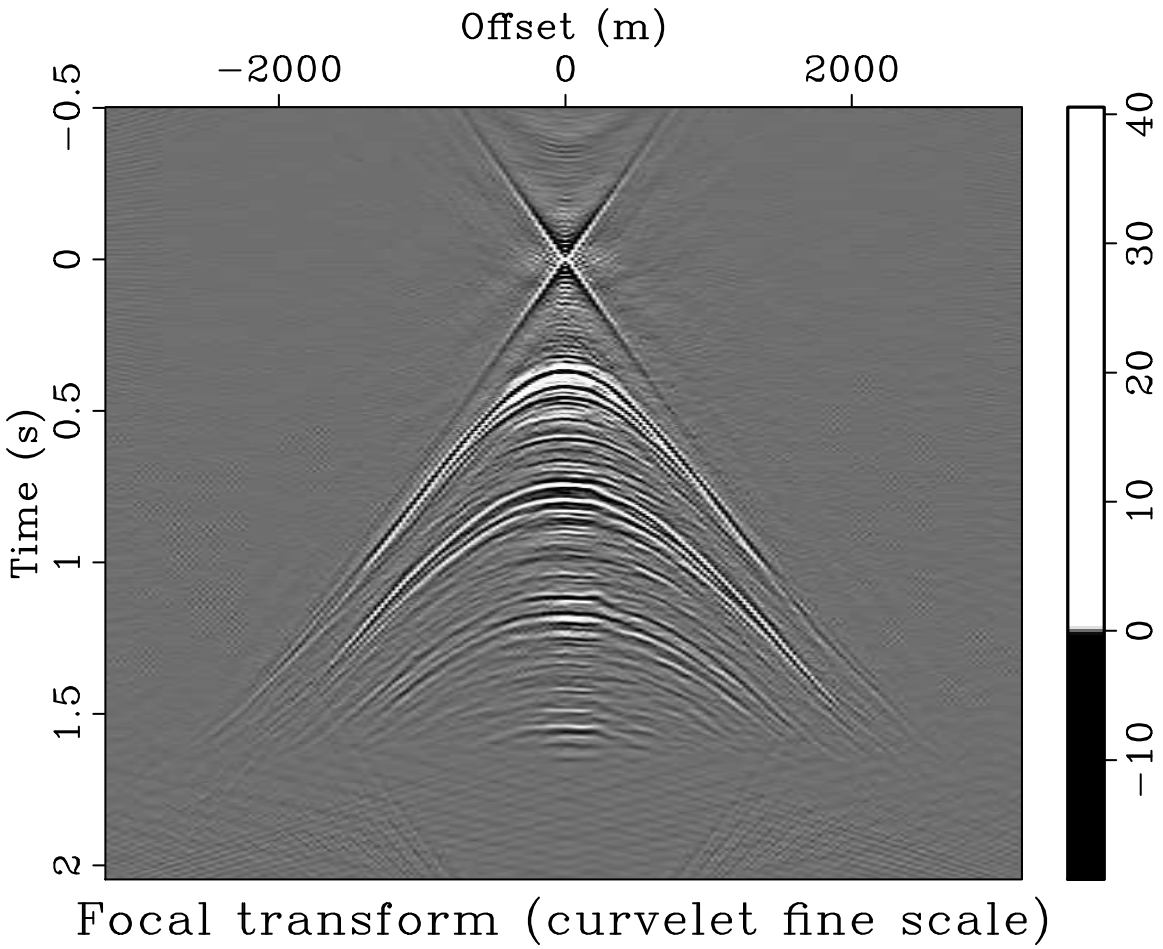
- Integrate with the many software tools for AST analysis.
  - Algorithm efficiency
- Advanced optimization techniques.
- Easily adapt AST optimizations to other platforms like Matlab.

Observations

- End Users
  - Mathematician:
    - Implementing coordinate-free algorithms
    - Easy to go from theory to practical applications
    - Possible to compound and augment linear systems
  - Geophysicist:
    - Simplifying a large process flow to a single line
    - No compromising of functionality
    - Easier to control
    - Parameter and domain-range tests
    - Pathway to Reproducible Research
  - Reuse SLIMpy code for a number of applications.
  - Concrete implementation with overloading in Python.

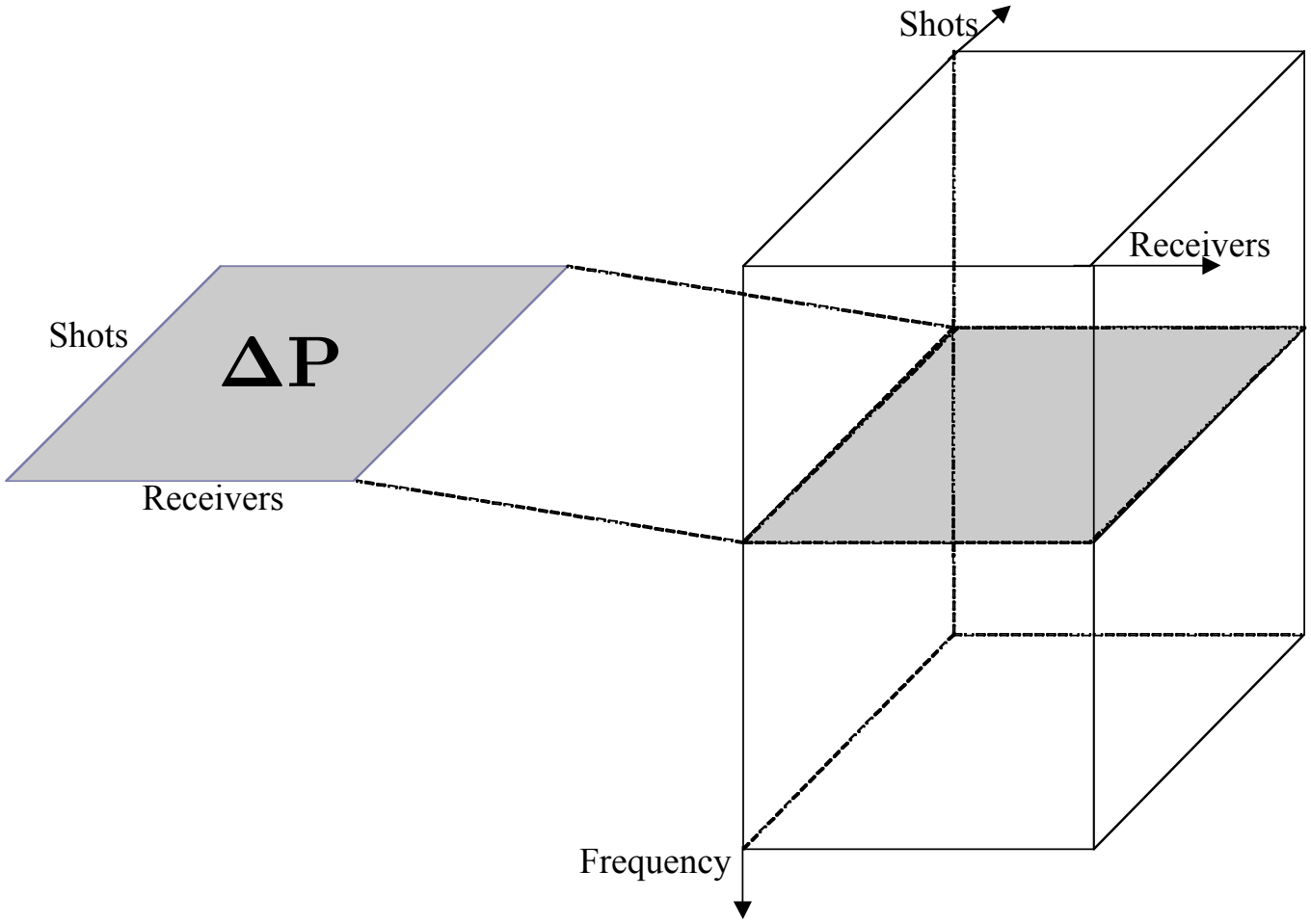
Focal Transform Interpolation

- SLIMpy App by Deli Wang
- SLIMpy code very compact.
  - Interpolation around 25 lines.
- Easily readable.
- Wrapped in a SConstruct file for easy use of Reproducible Research.
- Set parameters in the SConstruct and run the application with one command.
- Uses reusable ANA.



Picture by Deli Wang

Primary operator



Frequency slice from data matrix with dominant primaries.

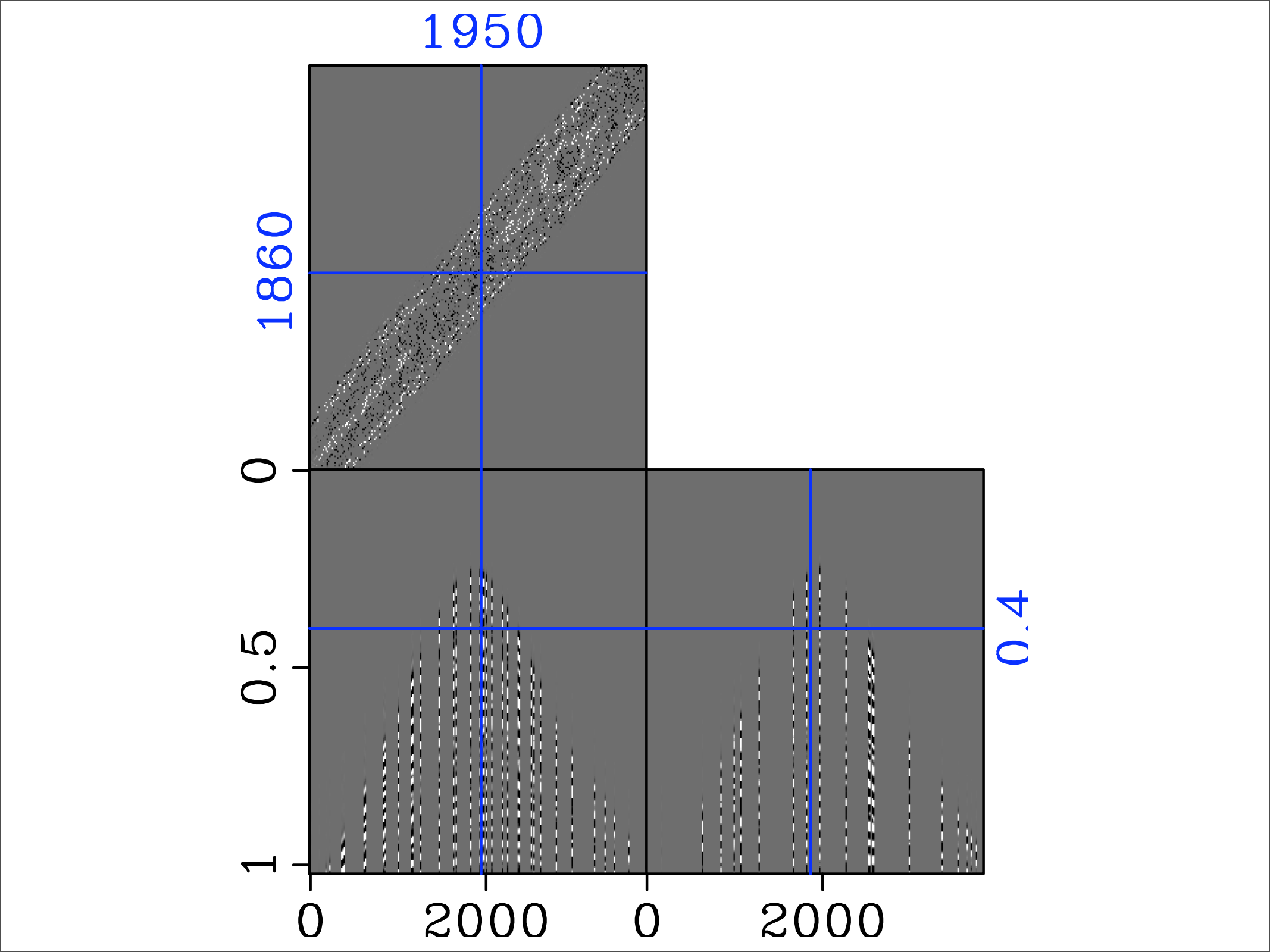
Recovery with focussing

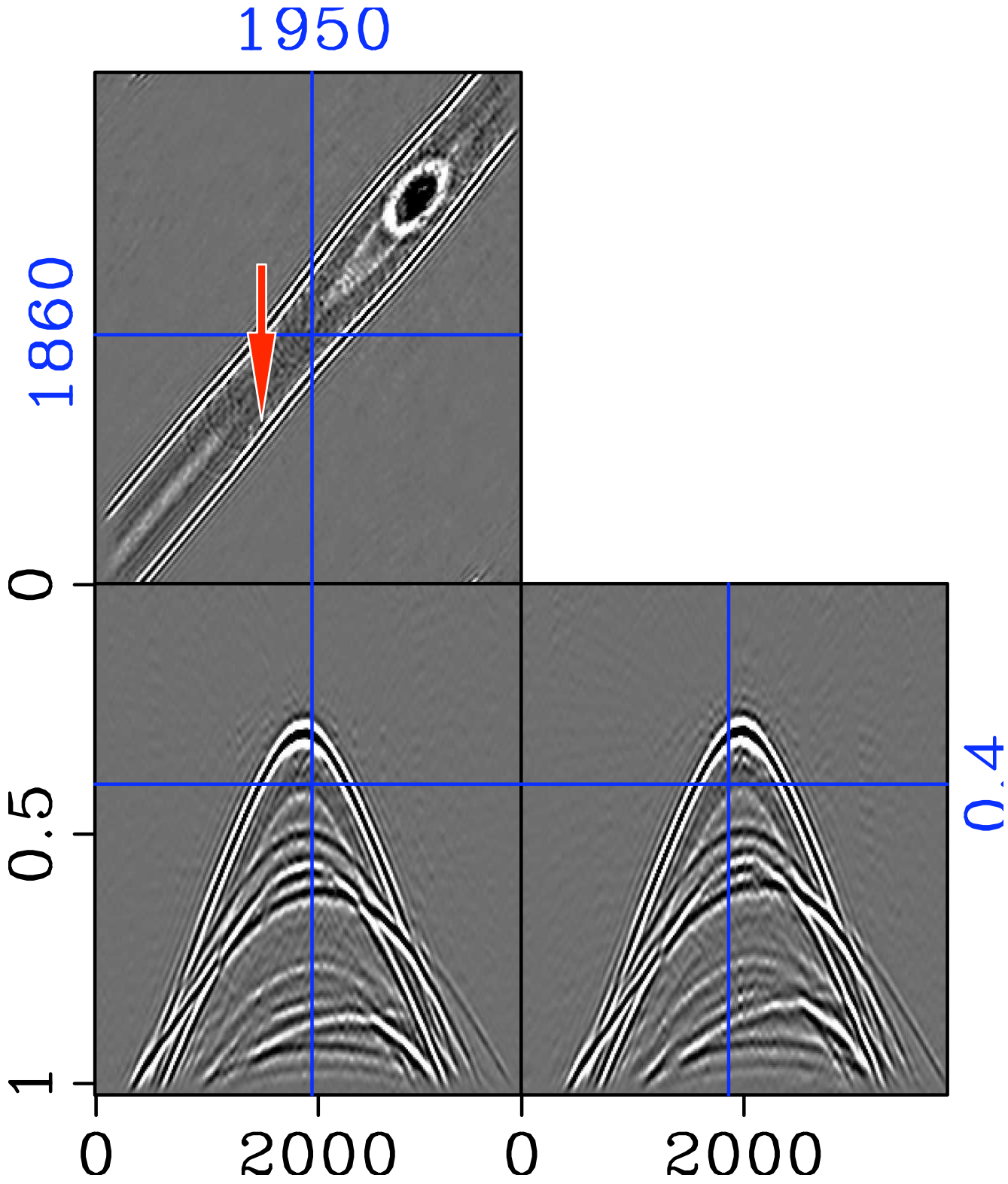
Solve

$$\mathbf{P}_\epsilon : \quad \begin{cases} \widetilde{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{x}\|_1 & \text{s.t.} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2 \leq \epsilon \\ \widetilde{\mathbf{f}} = \mathbf{S}^T \widetilde{\mathbf{x}} \end{cases}$$

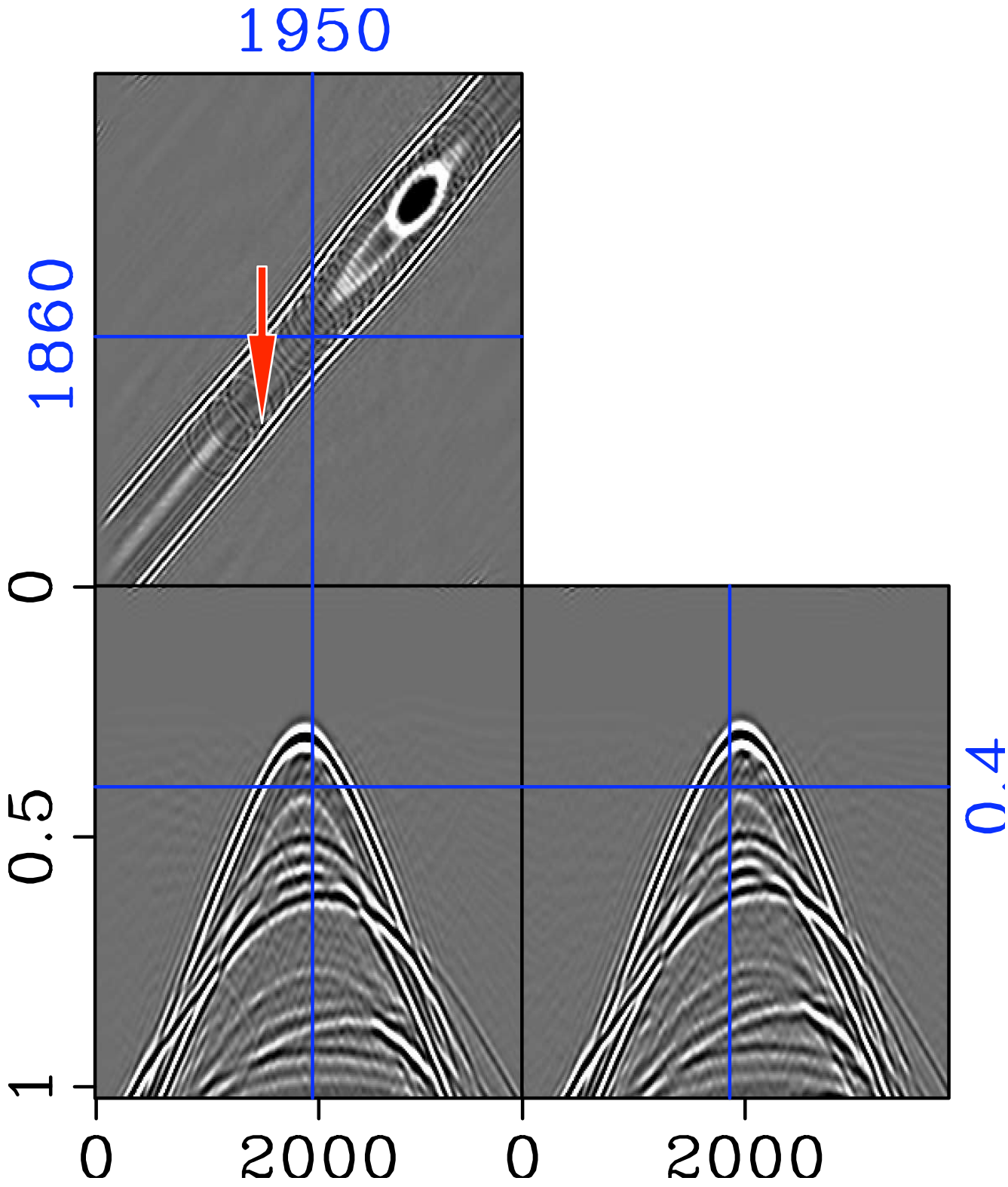
with

$$\begin{aligned} \mathbf{A} &:= \mathbf{R}\boldsymbol{\Delta}\mathbf{P}\mathbf{C}^T \\ \mathbf{S}^T &:= \boldsymbol{\Delta}\mathbf{P}\mathbf{C}^T \\ \mathbf{y} &= \mathbf{R}\mathbf{P}(:) \\ \mathbf{R} &= \text{picking operator.} \end{aligned}$$


1950
1860
0
0.5
1
0.4
0
2000
0
2000



SLIM
Seismic Laboratory for Imaging and Modeling



SLIM
Seismic Laboratory for Imaging and Modeling

Focal Transform Interpolation

- SConstruct file used to generated result.
  - Efficient flow system only re-calculates what has changed.

```
Flow('wac3D_256',None,
    '''
    math n1=55521587 output="1" |sfput d1=0.004 o1=0|
    pad beg1=16777216>rwac3D.rsf &&
    sfmath <rwac3D.rsf output="input*0" >iwac3D.rsf &&
    sfcmplx rwac3D.rsf iwac3D.rsf|sfput d1=0.004 o1=0 o2=0 o3=0 n2=1 n3=1 d2=15 d3=15>$TARGET &&
    sfrm rwac3D.rsf iwac3D.rsf
    ''',stdout=-1)

Flow('interp80_fcrsi',['shot256','srme256','wac3D_256','mask256_80',interp_focal_remv],
       python+' '+interp_focal_remv + ' -n ' + """
                        data=${SOURCES[0]}
                        filter=${SOURCES[1]}
                        remove=${SOURCES[2]}
                        transparams=6,8,0
                        solverparams=2,1
                        eigenvalue=40.2
                        output=$TARGET
                        mask=${SOURCES[3]}
                        flag=1
                    """,stdin=0,stdout=-1)

Result('interp80','interp80_fcrsi.rsf',cube('Interp80% missing'))
Result('cubshot','shot256.rsf',cube('shot'))
End()
```

Focal Transform Interpolation

- This small Focal Transform Interpolation application by Deli Wang produces...

```
TAP = Taper_wdl(data.getSpace(),taplen=15)
data = TAP*data
filter = TAP*filter

data = cmplx(data,data.getSpace().zeros())
filter = cmplx(filter,filter.getSpace().zeros())

#Remove fine scale wavelet operator
RM = weightoper
(wvec=remove,inSpace=data.getSpace())

#2D/3D, complex/real, curvelet transform
C = fdct3(data.getSpace(),cpxIn=True,*transparams)
PAD = Pad_wdl(data.getSpace
(),padlen=padlen,winlen=winlen)

#1D complex in/out FFT
F = fft3_wdl(PAD.range(),axis=1,sym='y',pad=1)

#Data cube transpose
T = transpoper(F.range(),memsize=2000,plane=[1,3])
D = weightoper
(wvec=eigenvalue,inSpace=filter.getSpace())
R = pickingoper(data.getSpace(),mask)
REAL = Real_wdl(data.getSpace())
PFT = comp([T,F,PAD])
```

```
#Remove fine scale wavelet operator
REMV = comp([C.adj(),RM,C])
filter = REMV*filter
filterf = PFT*(D*filter)
P = Multpred_wdl(T.range(),filt=1,input=filterf)

#WCC in frequency domain
PEF = comp([PAD.adj(),F.adj(),T.adj(),P,T,F,PAD])

#Define global linear operator
if (flag==1):
   A = comp([R,PEF.adj(),C.adj()])
else:
   A = comp([R,C.adj()])
data = R*data
data = REMV*data


#Define and run the solver
thresh = logcooling(thrparams[0],thrparams[1])
solver = GenThreshLandweber(solverparams
[0],solverparams[1],thresh=thresh)
x = solver.solve(A,data)
```

interp3d_remv.py
by Deli Wang

Focal Transform Interpolation

- ...over 30 pipes utilizing more than 90 commands to obtain its result.

```
/Volumes/Users/dwang/tools/python/2.5/bin/python ./interp3d_remv.py -n data=shot256.rsf filter=srme256.rsf
remove=wac3D_256.rsf transparams=6,8,0 solverparams=2,1 eigenvalue=40.2 output=interp80_fcrsi.rsf
mask=mask256_80.rsf flag=1
sfmath n1=256 n2=256 n3=256 output=0 | sffdct3 nbs=6 nbd=8 ac=0 maponly=y sizes=sizes256_256_256_6_8_0.rsf
None < sfmath  output="0" n2="256" n3="256" type="float" n1="256" | sfput  head="tfile" o3="0" o2="0" d2="15"
d3="15" o1="0" d1=".00" > tmp.walacs  ()
shot256.rsf < sfcostaper  nw1="15" nw2="15" > tmp.uNaFyB  ()
None < sfmath  output="0" n2="256" n3="256" type="float" n1="256" | sfput  head="tfile.rsf" d2="15" o2="0" o3="0"
d3="15" o1="0" d1=".00" > tmp.hyqCQa  ()
None < sfcmplx tmp.uNaFyB tmp.hyqCQa  | sfheadercut  mask="mask256_80.rsf" | sffdct3  nbs="6" ac="0" nbd="8"
inv="n" sizes="sizes256_256_256_6_8_0.rsf" > tmp.XiJ7Xs  ()
wac3D_256.rsf < sfmath  output="vec*input" vec="tmp.XiJ7Xs" | sffdct3  nbs="6" ac="0"
sizes="sizes256_256_256_6_8_0.rsf" inv="y" nbd="8" > tmp.KdkI78  ()
srme256.rsf < sfcostaper  nw1="15" nw2="15" > tmp.C1Qrmt  ()
None < sfcmplx tmp.C1Qrmt tmp.walacs  | sffdct3  nbs="6" ac="0" nbd="8" inv="n"
sizes="sizes256_256_256_6_8_0.rsf" > tmp.TvVRiJ  ()
wac3D_256.rsf < sfmath  output="vec*input" vec="tmp.TvVRiJ" | sffdct3  nbs="6" ac="0"
sizes="sizes256_256_256_6_8_0.rsf" inv="y" nbd="8" > tmp.DZjYCe  ()
tmp.DZjYCe < sfmath  output="0.0248756218905*input" | sfpad  n1="512" | sffft3  inv="n" pad="1" sym="y" axis="1"
| sftransp  plane="13" memsize="2000" > tmp.F9Y0IZ  ()
tmp.KdkI78 < sfheadercut  mask="mask256_80.rsf" | sfpad  n1="512" | sffft3  inv="n" pad="1" sym="y" axis="1" |
sftransp  plane="13" memsize="2000" | sfmultpred  input="tmp.F9Y0IZ" adj="1" filt="1" | sftransp  plane="13"
memsize="2000" | sffft3  inv="y" pad="1" sym="y" axis="1" | sfwindow  n1="256" | sffdct3  nbs="6" ac="0" nbd="8"
inv="n" sizes="sizes256_256_256_6_8_0.rsf" > tmp.brima0  ()
tmp.brima0 < sfsort  ascmode="False" memsize="500" > tmp.8rOLD8  ()
${SCALAR01} = getitem(tmp.8rOLD8, 72299, )
${SCALAR02} = getitem(tmp.8rOLD8, 71575815, )
```

Future goals

- MPI Interfacing through SLIMpy
  - Allows for a user defined MPI definition.
  - Will work with a number of different MPI platforms (Mpich, LAM).
  - Easily call MPI features through SLIMpy script.
- Integration with existing OO solver frameworks such as Trilinos
- Automatic Differentiation

Conclusions

Use a scripting language to access low-level implementations of (linear) operators (seismic processing tools).

Easy to use automatic checking tools such as domain-range checks and dot-test.

Overloading and abstraction with small overhead.

AST allows for optimization.

Reusable ANAs and Applications.

Is growing into a “compiler” for ANA’s ....

SLIMpy Web Pages

- More information about SLIMpy can be found at the SLIM Homepage:

  http://slim.eos.ubc.ca

- Auto-books and tutorials can be found at the SLIMpy Generated Websites:

  http://slim.eos.ubc.ca/SLIMpy

Acknowledgments

- Madagascar Development Team
  - Sergey Fomel
- CurveLab 2.0.2 Developers
  - Emmanuel Candes, Laurent Demanet, David Donoho and Lexing Ying
- SINBAD project with financial support
  - BG Group, BP, Chevron, ExxonMobil, and Shell
- SINBAD is part of a collaborative research and development grant (CRD) number 334810-05 funded by the Natural Science and Engineering Research Council (NSERC)