THE UNIVERSITY OF BRITISH COLUMBIA | VANCOUVER

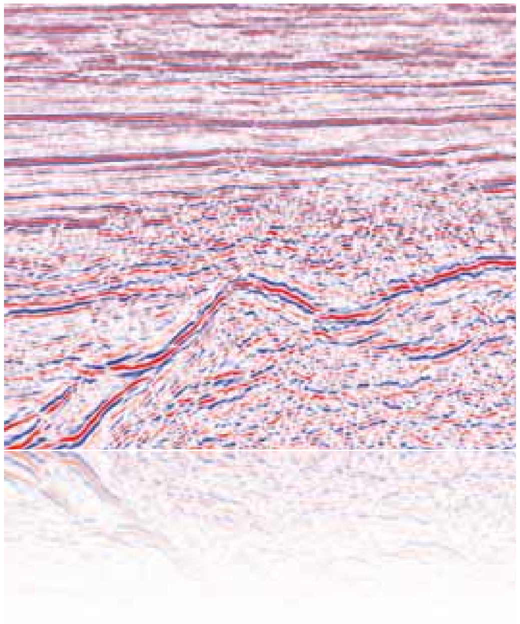
Highlights of SLIM software release to SINBAD sponsors

Cody R. Brown
cbrown@eos.ubc.ca

Gilles Hennenfent
ghennenfent@eos.ubc.ca

Felix Herrmann
fherrmann@eos.ubc.ca

Seismic Laboratory for Imaging & Modeling
Department of Earth & Ocean Sciences
The University of British Columbia



BP
On-site software installation visit
Tuesday, October 2nd, 2007

What is SLIMpy2

- Basically an interpreter for piped-based applications.
  - our application being Madagascar

Users

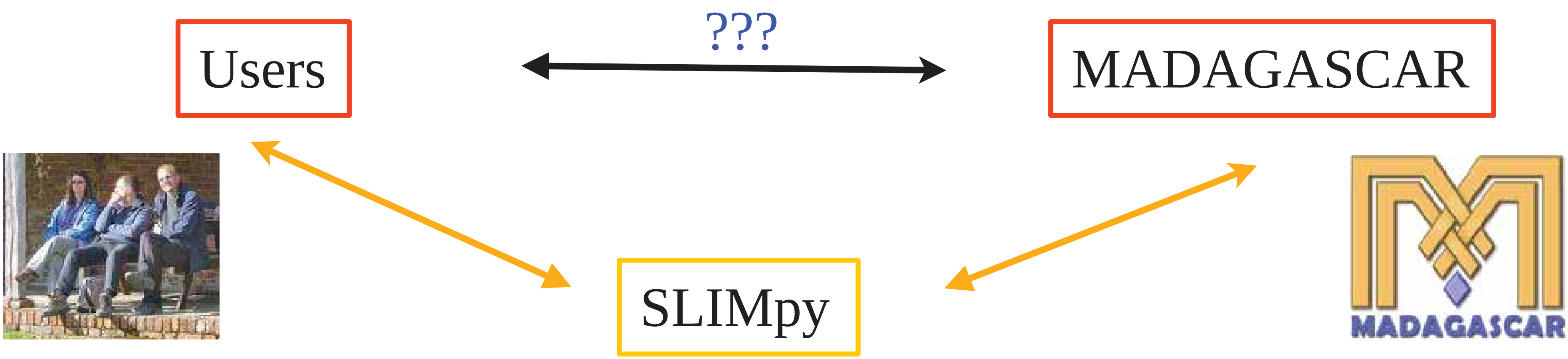
???

MADAGASCAR

MADAGASCAR

What is SLIMpy2

- Basically an interpreter for piped-based applications.
  - our application being Madagascar



Why SLIMpy2

- Code reusability:
  - We can use the same ANA's for multiple applications.
- Code clarity:
  - Easy to follow and easy to program for the non-programmer.
- Code minimization:
  - Performs extremely complicated tasks with little code. SLIMpy automates many checks and features of operators.

Advanced Features of SLIMpy2

- Core
  - Data Structure
  - Plug-In System, Domain-Range Tracking
  - Operators: Linear Operators, Compound Operators, Augmented Matrices
    - adjoints pre-defined for linear operators
  - **Abstract Syntax Tree**
    - optimizations
- ANAs
  - Overview of the Landweber ANA
- Apps/Demos
  - dnoise SLIMpy script from scratch

SLIMpy2 - Data Structure

- SLIMpy is an out-of-core interpreter.
  - Currently uses Madagascar.
  - Any piped-based applications can be adapted to SLIMpy.

SLIMpy2 - Data Structure

- SLIMpy is an out-of-core interpreter.
  - Currently uses Madagascar.
  - Any piped-based applications can be adapted to SLIMpy.

SLIMpy2 - Data Structure

- SLIMpy is an out-of-core interpreter.
  - Currently uses Madagascar.
  - Any piped-based applications can be adapted to SLIMpy.

- All data imported with SLIMpy are stored through spaces.
  - Spaces are special header information of the data.
  - SLIMpy uses these spaces to calculate information on the transform before it is applied.
  - Can easily get information such as L2-norm directly from any vector space.

Show Tutorial One - spaces

SLIMpy2 - Plug-In System

- Information about each linear operator is stored in the plug-in class.
  - Currently only Madagascar operators are indexed.
  - Very objected orientated.

SLIMpy2 - Plug-In System

- Information about each linear operator is stored in the plug-in class.
  - Currently only Madagascar operators are indexed.
  - Very objected orientated.

SLIMpy2 - Plug-In System

- Information about each linear operator is stored in the plug-in class.
  - Currently only Madagascar operators are indexed.
  - Very objected orientated.
- Potentially integrate multiple applications in one file.
  - Use SU, SEP and Madagascar operators together in one script.

SLIMpy2 - Domain-Range Tracking

- Accesses information from the plug-in system.

- Using this information it can predict transformed data spaces.
  - This information can be used for Domain-Range Tracking.
  - Can assist in debugging and corning problems.
  - Allows us to work with pseudo-data without performing any transformations!

SLIMpy2 - Operators

- The Meat of SLIMpy
  - Will demonstrate how to use and apply these operators, but some information first.

SLIMpy2 - Operators

- The Meat of SLIMpy
  - Will demonstrate how to use and apply these operators, but some information first.

SLIMpy2 - Operators

- The Meat of SLIMpy
  - Will demonstrate how to use and apply these operators, but some information first.

- Linear operators are pre-defined with adjoint transformation.

SLIMpy2 - Operators

- The Meat of SLIMpy
  - Will demonstrate how to use and apply these operators, but some information first.

- Linear operators are pre-defined with adjoint transformation.
- User-defined linear operators:
  - Will automatically generate most code that is not specified.
  - Applies generic adjoint information to the operator if adjoint not defined

SLIMpy2 - Operators

- The Meat of SLIMpy
  - Will demonstrate how to use and apply these operators, but some information first.

- Linear operators are pre-defined with adjoint transformation.
- User-defined linear operators:
  - Will automatically generate most code that is not specified.
  - Applies generic adjoint information to the operator if adjoint not defined
- Compound Operators
  - Build complex operators from simple building blocks.
  - SLIMpy will calculate adjoints automatically from the smaller building blocks.

SLIMpy2 - Operators

- The Meat of SLIMpy
  - Will demonstrate how to use and apply these operators, but some information first.

- Linear operators are pre-defined with adjoint transformation.
- User-defined linear operators:
  - Will automatically generate most code that is not specified.
  - Applies generic adjoint information to the operator if adjoint not defined
- Compound Operators
  - Build complex operators from simple building blocks.
  - SLIMpy will calculate adjoints automatically from the smaller building blocks.
- Augmented Matrices
  - Define augmented matrices visually.

Abstraction

Let data be a vector $y \in \mathbb{C}^n$.
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(y.space).adj()

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

A = aug_oper([A1, A2])

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

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

Vector and 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

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
$y=a^Tb$	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

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
$y=a^Tb$	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

- a and b are data vectors.

Linear Operators

Math	SLIMpy	Matlab	RSF
$y=Ax$	y=A*x	y=A(x)	<x.rsf sffft2 >y.rsf
$z=A^Ty$	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])	not easy	complicated
$\mathbf{A} = \mathbf{B}\mathbf{C}^T$	A=CompoundOperator ([B,C.adj()])	define new function	complicated

Linear Operators

Math	**SLIMpy**	**Matlab**	**RSF**
y=Ax	y=A*x	y=A(x)	<x.rsf sfft2 >y.rsf
$z=A^Ty$	y=A.adj()*y	z=A(y,'transp')	<y.rsf sfft2 inv=y >z.rsf
$\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}$	A=aug_oper([A1,A2])	not easy	complicated
$\mathbf{A} = \mathbf{B}\mathbf{C}^T$	A=CompoundOperator ([B,C.adj()])	define new function	complicated

Show Tutorial Two - operators

Abstract Syntax Tree (AST)

- The Brains of SLIMpy
- Abstract Syntax Tree allows:
  - analysis of compounded commands
  - removal of inefficiencies
  - translation of statements into a concrete instruction set
- Stores commands as nodes and optimizes through the AST
  - analyzes dependancies and generates an optimal tree

Abstract Syntax Tree (AST)

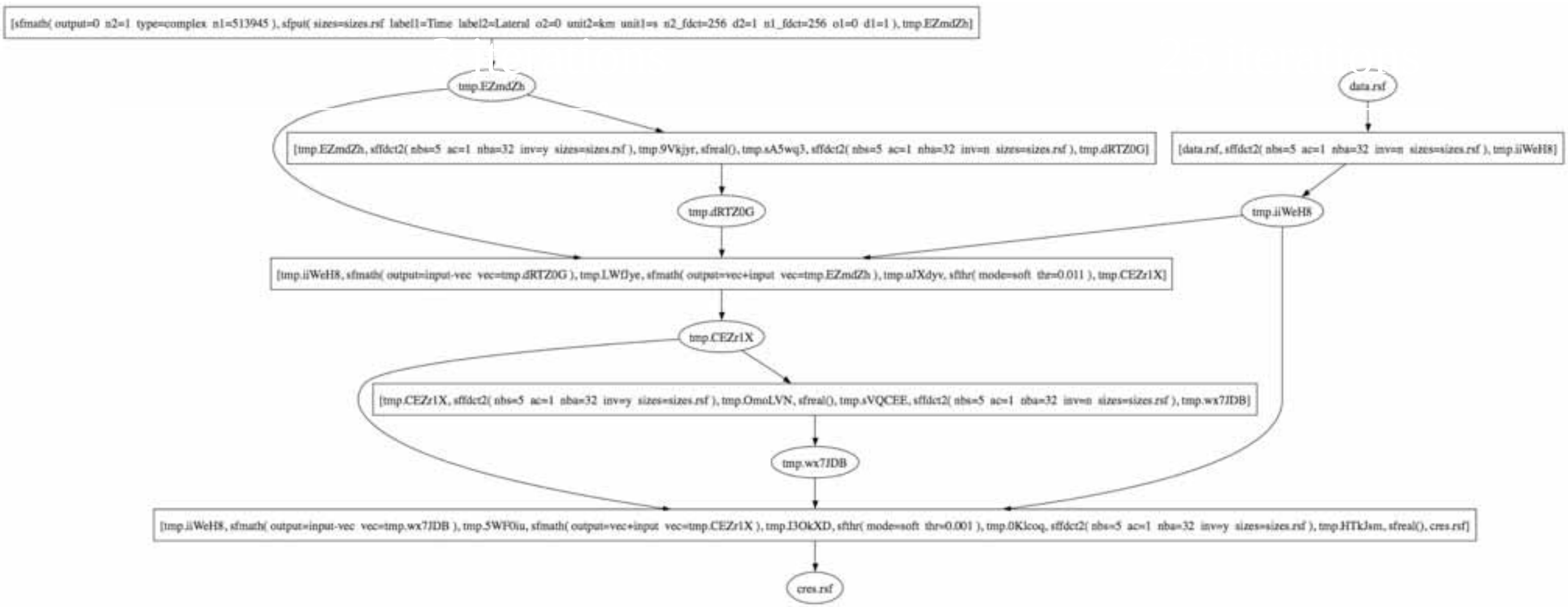
- An AST is a finite, labeled, directed tree where:
  - internal nodes are labeled by operators
  - leaf nodes represent variables

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

Piped-Based Difficulties

- Executing single unix commands is inefficient.
  - better to chain together to reduce IO traffic
  - reduce number and size of intermediate data files
- Difficulties with large iterations.
  - many intermediate data files

Visualization



Optimization

- Currently we have:
  - Unix pipe-based optimization
    - unique to SLIMpy2
    - assembles commands into longest possible pipe structure
  - “Language” Specific Optimization
    - Madagascar operators

- Goals within reach:
  - Symbolic Optimization for certain operators
    - eg. A(x) + A(y) = A(x+y)
  - Parallel Optimization
    - load balancing, distributing expensive commands

Optimization

- Optimization of the AST can be done in a modular fashion.
  - done internally

- In the future users can:
  - 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
```

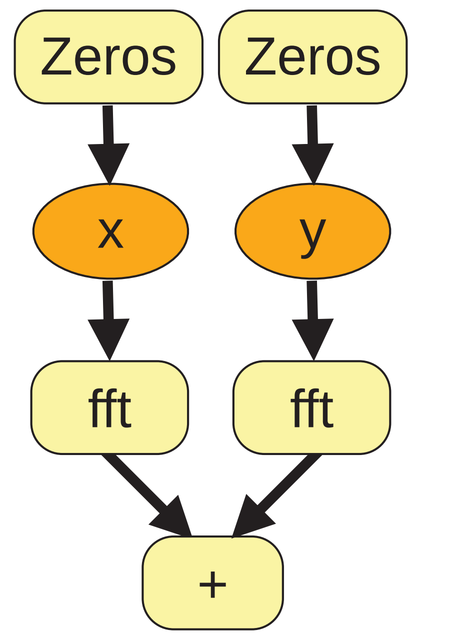
Example

- Resulting AST from this SLIMpy application.
- Walkthrough
  - Simple three step optimization

Code

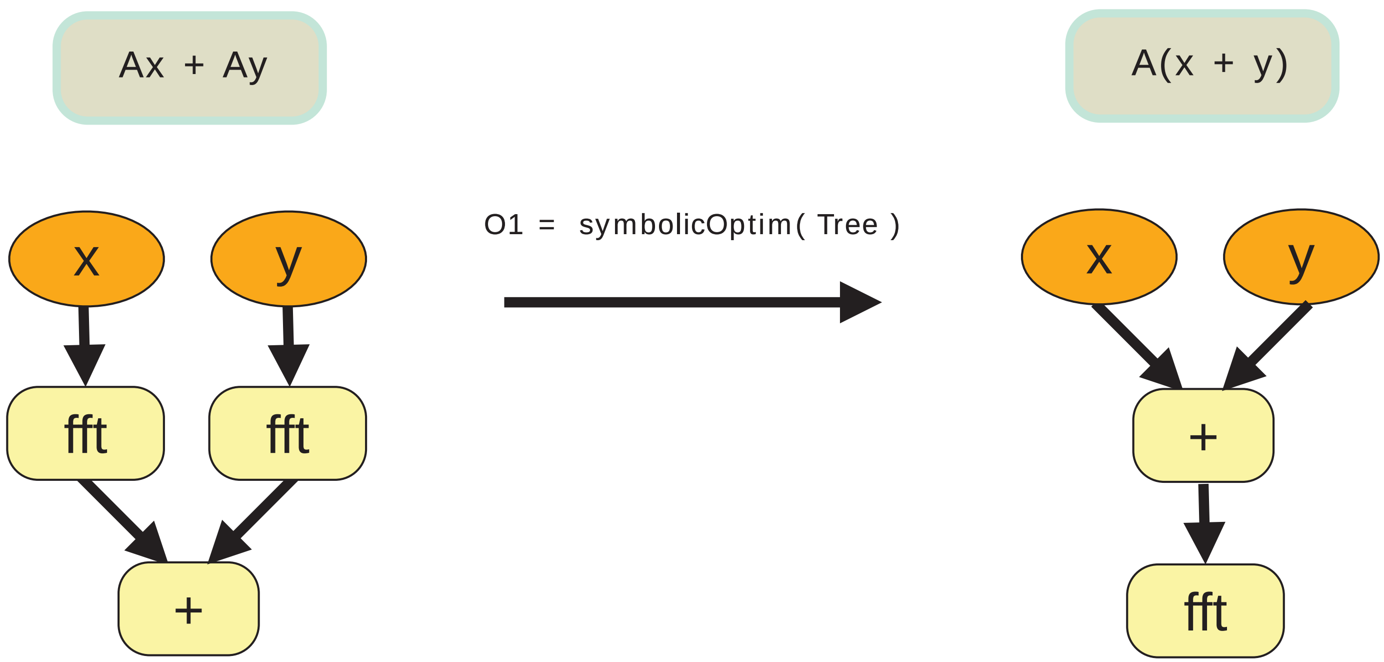
```
# Create a Vector Space of a 10 x 10 image
vecspace = Space(n1=10,n2=10,plugin='slim2rsf')
x = vecspace.zeros()      # Vector of zeros
y = vecspace.zeros()      # Vector of zeros
A = fft( vecspace )        # fft operator
V = aug_vec( [[x],[y]] )  # augmented vector system
M = aug_oper([[ A, 0 ],   # augmented operator
              [ 0, A ]] ) # system

ans = M * V      # apply the operator to the vector
```

AST



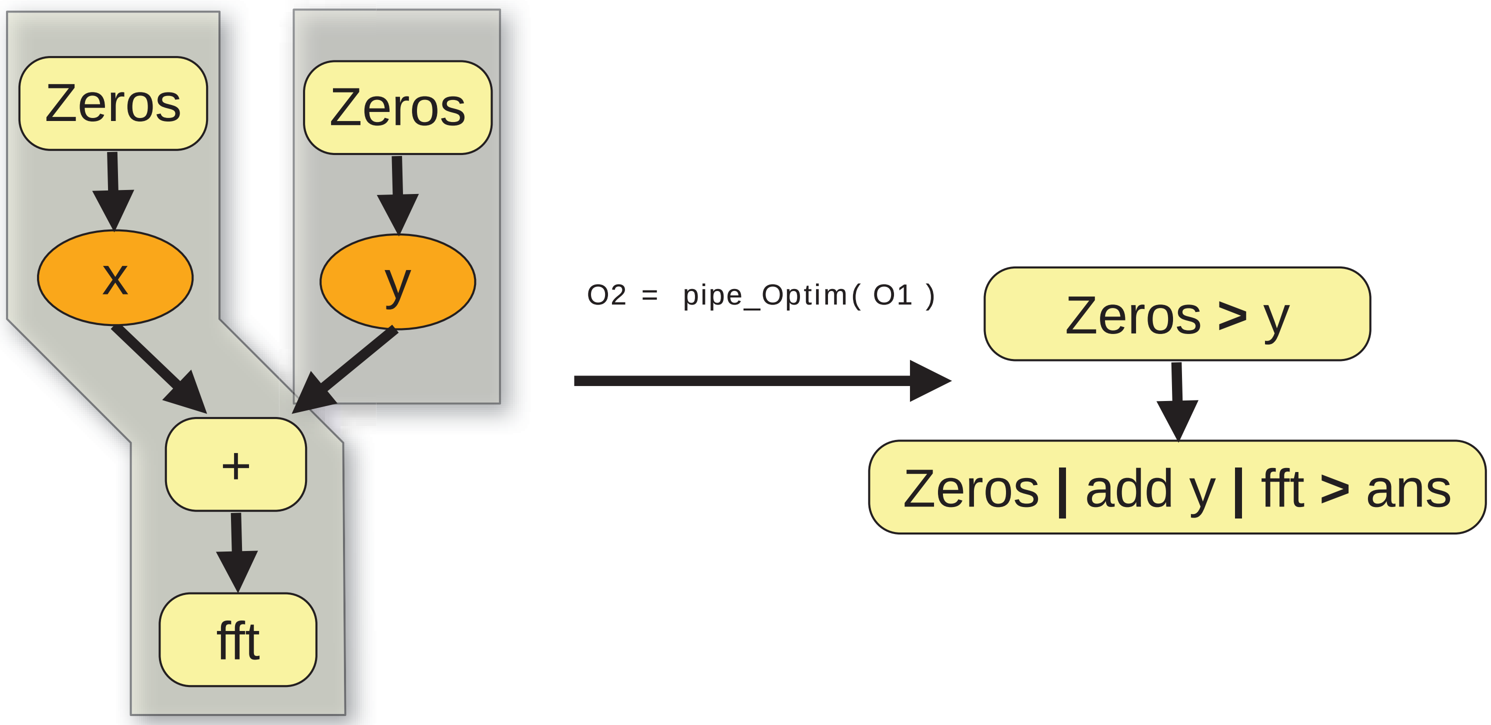
Symbolic Optimization

- If we know it is a linear operator then it is more efficient to add the vectors first.
  - only do one FFT computation



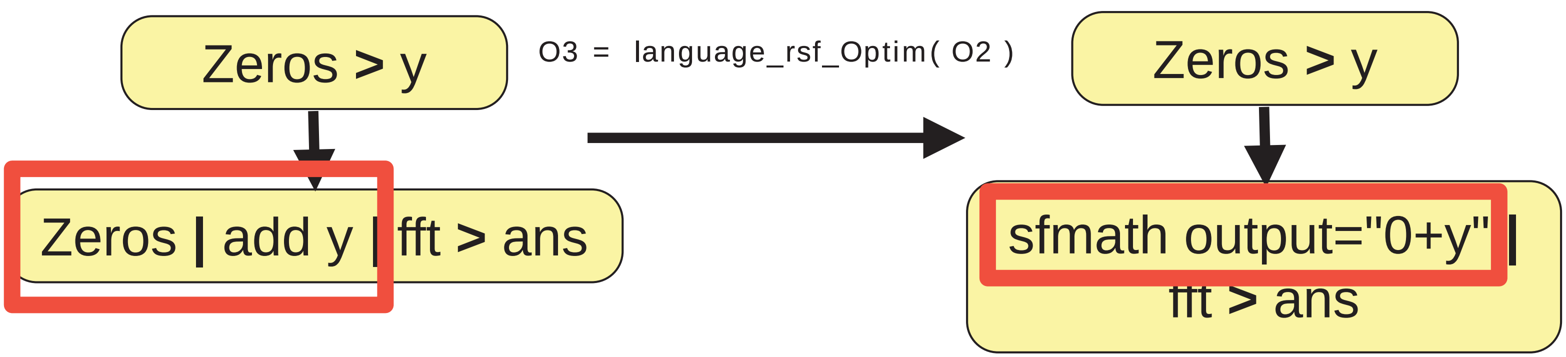
Unix-Pipe Optimization

- Compress individual commands into minimal number of command pipes.
  - dramatically reduce IO throughout the script



Language Specific

- Find shortcuts to reduce workload.
- All shortcuts are defined in the plug-in system.



Why We Need an AST

- A goal of SLIMpy2 is to create an efficient interface from Abstract Syntax Trees to low level software such as Madagascar.
- Pre-processing allows for control over the tree structure.
  - Generate the AST with iterative algorithms.
  - This opens the door for future types of pre-processing.
  - One of the most expandable features of SLIMpy.

At What Cost

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

Performance Cost of the AST

- 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
```

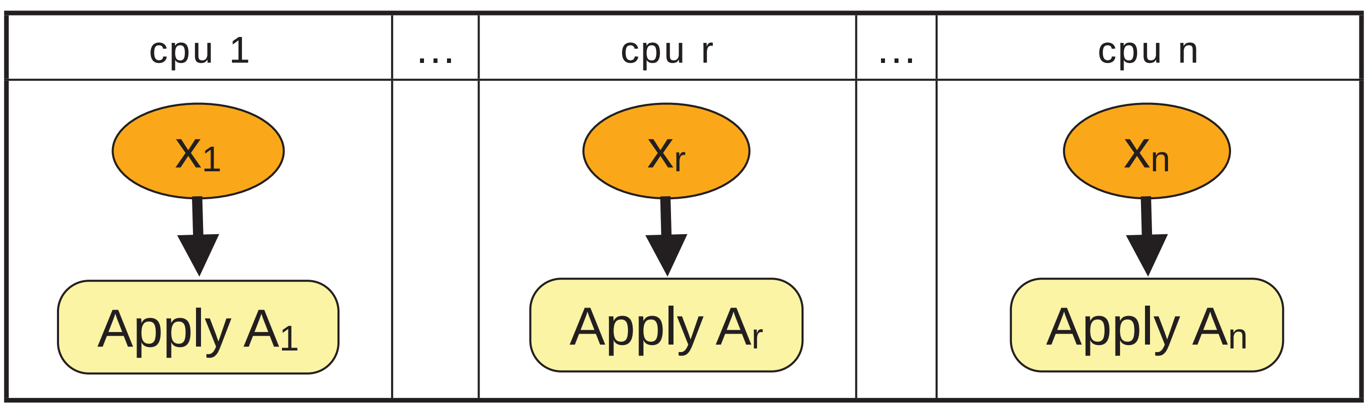
Pathway to Parallel SLIMpy2

- Embarrassingly parallel through AST
  - The AST already contains and formats the tree with information about dependancies.
  - Can easily separate different branches of the AST for different nodes.
- Domain decomposition
  - slice the data-set into more manageable pieces
- Future MPI integration
  - utilize embarrassing parallel branches through MPI
  - wrap proper MPI operators and launch with mpirun commands

Embarrassingly Parallel

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

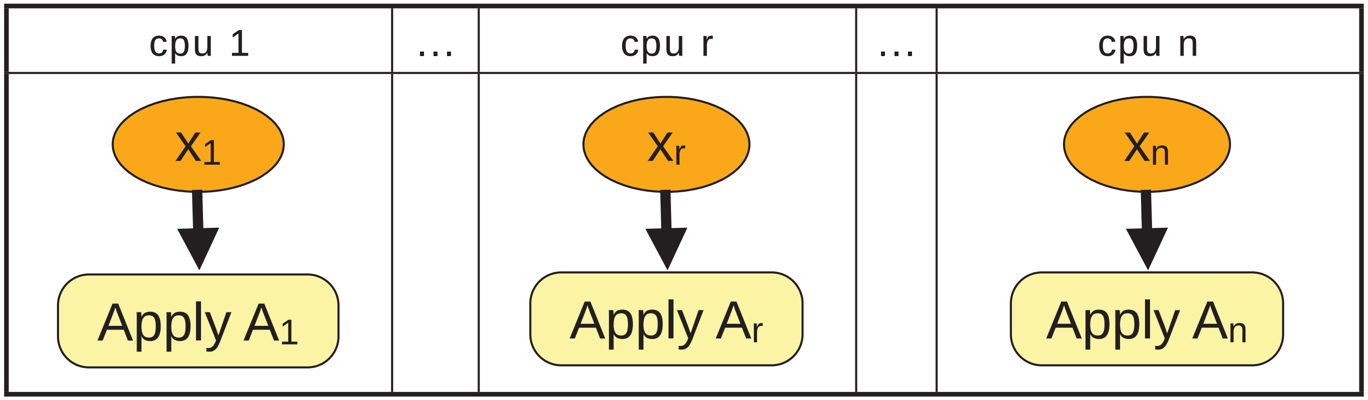
```
V = aug_vec( [[x1], ..., [xn]] )          # augmented vector system
M = aug_oper([[ A1,    ...,      0 ],  # augmented operator
               [  0, ..., Ar, ..., 0 ],
               [ 0,      ...,      An ]] )
ans = M * V
```



Embarrassingly Parallel

- Separate branches of the AST can easily be 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
```



Show Tutorial Three - multi-core option

Abstract Numerical Algorithms (ANAs)

- Pathway to reusable code.
- SLIMpy has a suite of solvers that can be used in a number of different applications.
- Easily experiment and test new solvers with very little code changes in the application.

Dissecting an ANA

- Generating an ANA is very simple.
  - Standard Python class.
  - Algorithms can be implemented in three parts.

Dissecting an ANA - Landweber

- Declare your functions to use.

Modules the ANA uses.

```
from SLIMpy.SLIMmath.Steppers import OneDimStep1
from SLIMpy.User.SolverUtils.AbstractSolver import solver
from SLIMpy.User.SolverUtils.thresholds import threshobj

# set default threshhol scheme
thresh = threshobj()

class GenThreshLandweber(solver):
```

Superclass the solver class.

Dissecting an ANA - Landweber

- Define the variables your ANA will have access too.
  - usually includes iterations and ANA variables

Take and store your variables

```
def __init__(self,OuterN,InnerN,thresh=thresh):
        self.OuterN = OuterN
        self.InnerN = InnerN
        self.thresh = thresh
```

Dissecting an ANA - Landweber

- Declare a solve class.
  - This class is called by the user.
  - Takes an abstract operator and data
  - Solves the algorithm, returns the result.

abstract operator

```
def solve(self,A,data):
        log = self.log(2,'solver')
        print >> log ,'\n%%%%%% into GenThreshLandweber'
        # pre-computation of matched filter
        Coefs = A.adj() * data
        # prepare first guess
        print >> log , 'Initial guess:'
        vctorSpace = Coefs.getSpace()
        x = vctorSpace.zeros()
        # execute loops
        print >> log ,'Executing loops:'
        for i in OneDimStep1(1,self.OuterN):
                for j in OneDimStep1(1,self.InnerN):
...
```

usually includes loops

Apps/Demo - dnoise from scratch

- Now that we have all the tools necessary we can implement our own dnoise application.
  - everything is already defined
  - wrap everything up and apply it as an application

- Only a couple more lines to code.

Apps/Demo - dnoise from scratch

- Now that we have all the tools necessary we can implement our own dnoise application.
  - everything is already defined
  - wrap everything up and apply it as an application

- Only a couple more lines to code.

Show Tutorial Four - dnoise demo

Conclusions

- Use SLIMpy as an interpreter to Madagascar.
  - allow SLIMpy to do the background work for you

- AST allows for optimization.
- Reusable ANAs and Applications.

- Can use SLIMpy as a bridge for using different pipe-based together in one universal language.

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
- CurveLab Developers
- 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)