Seismic inversion through operator overloading

Sean Ross-Ross

Synopsis

- Challenge:
  - integrate & scale IO intensive “Pipe”-based software
  - does not facilitate transfer of knowledge
  - extend flow-based processing to iterative processing
- Opportunity:
  - Create an object-oriented layer
    - Implement algorithms modeled directly from math
    - Independent of the lower-level software.
- (p)SLIMpy:
  - A Collection of Python classes
  - vector, linear operator, R/T operation
- Benefits:
  - Reusable
  - Scalable (with parallelization including domain decomposition)

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
  - little code reuse
  - emphasis on processing flows
  - no environment to do iterations as part of optimization

Opportunity

- Create a abstraction layer for the user to implement algorithms
  - Object-oriented
  - Interfaces ANAs (coordinate-free abstract numerical algorithms) with lower level flow-based seismic processing software (e.g. Madagascar)
  - Independent of the lower level software that can be
    - in-core
    - out-of-core (pipe-based)
    - serial or parallel

(p)SLIMpy

- Provides a collection of Python classes and functions, to represent abstract numerical algorithms

- Is a tool used to design and implement algorithms
  - ***clean*** code
  - ***no*** overhead
  - designed to facilitate the **knowledge transfer** between *end users* and the *algorithm designers* (scientists)

- Easily scaleable
  - transparent parallel IO service

(p)SLIMpy

- Industry uses scalable in-core solutions which are not applicable for non-separable transforms in dimensions higher than two (e.g. 3-D curvelet transform)
- SLIM’s technology is based on **nonseparable** transforms that take 100.000’s of traces as input for a single transform
- SLIMPy is designed to handle large IO intensive non-separable transforms

Context

- SLIMpy is a IO-intensive adaptation of exisiting ideas:
  - 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/

Methodology

SLIMpy package is divided into four distinct parts

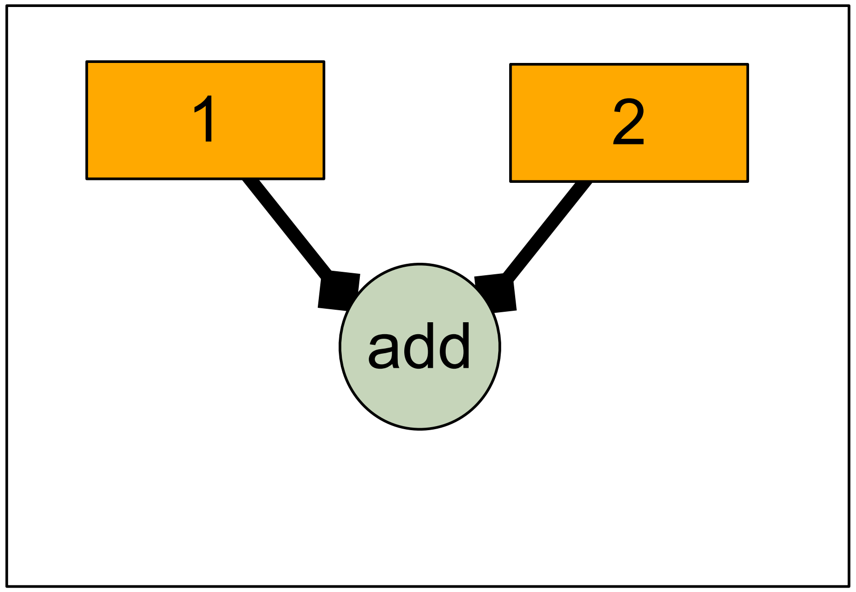
- Vector/Operator interface
  - For the algorithm designer
  - Enforce consistency & enable code reuse
- Abstract Syntax Tree
  - For SLIMpy developers
  - Allows for pipe optimization & parallelization
- Compiler
  - for developers porting to another lower level software
  - Interface to low-level pipe-based commands (e.g. SU, RSF)
- Reproducible research interface
  - For end users
  - Allows for access from scons & integration with papers

Operator Overloading

- Operators like +, -, or * have different implementations depending on the types of their arguments
- SLIMpy uses operator overloading on
    - vectors
        - vec1 + vec2
    - linear operators
        - Oper * vec
- adds nodes to the Abstract Syntax Tree

Abstract Syntax Tree (AST)

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



= 3

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

Example

- Apply Operator to two data-sets
- add the transformed data
- apply soft thresholding
- apply adjoint operator

Example SLIMpy

```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

C = fft( domain=vec.space )

coeffs1 = C * vec1

coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3

End()
```

SLIM
Seismic Laboratory for
Imaging and Modeling

Example SLIMpy

```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

C = fft( domain=vec.space )
```

} Define Vectors & Linear Operator

```
coeffs1 = C * vec1

coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3

End()
```

Example SLIMpy

```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

C = fft( domain=vec.space )
```

```
coeffs1 = C * vec1

coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3
```

} Perform operations on vectors

```
End()
```

Example SLIMpy

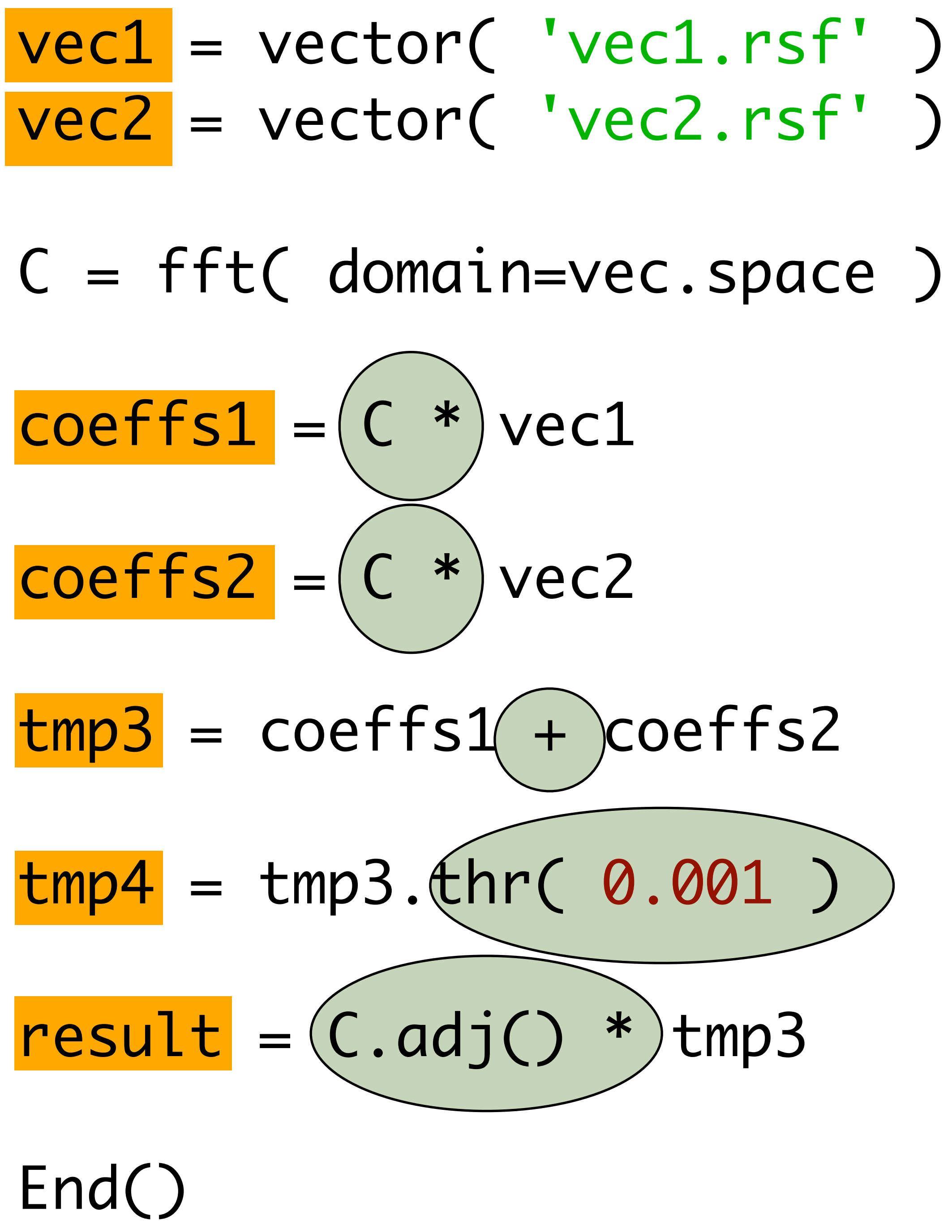
```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

C = fft( domain=vec.space )

coeffs1 = C * vec1

coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3

End()
```

Compile AST

Example SLIMpy

```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

C = fft( domain=vec.space )

coeffs1 = C * vec1

coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3

End()
```

Highlight data and commands



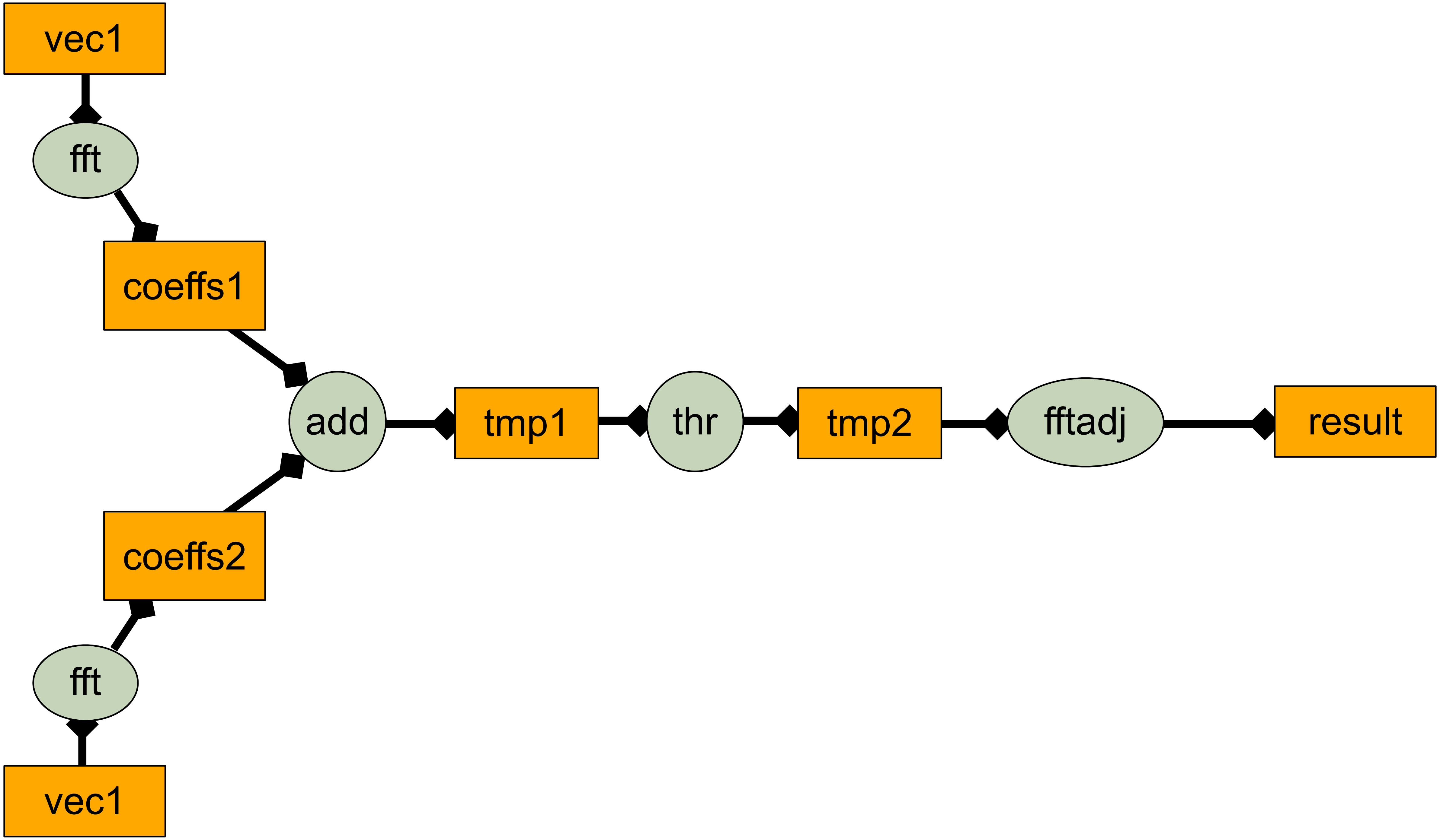
```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

C = fft( domain=vec.space )

coeffs1 = C * vec1

coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3

End()
```

Can be split into different nodes:

commands

data

Abstract Syntax Tree



Madagascar

Could work for any Lower Level software

```
< ./data2.rsf sffft1 opt="n" inv="n" sym="y" | DATAPATH=/
Tools/toolboxes/rsf_stuff/tmp_datapath/ sffft3 opt="n"
inv="n" sym="y" pad="1" axis="2" > /Tools/toolboxes/
rsf_stuff/tmp_datapath/slim.12648.env1.FakeMo.fft.
00004.rsf
< ./data1.rsf sffft1 opt="n" inv="n" sym="y" | sffft3 opt="n"
inv="n" sym="y" pad="1" axis="2" | sfmath output="vec
+input" vec="/Tools/toolboxes/rsf_stuff/tmp_datapath/slim.
12648.env1.FakeMo.fft.00004.rsf" | sfthr mode="soft"
thr="0.001" | sffft3 opt="n" inv="y" sym="y" pad="1"
axis="2" | DATAPATH=/Tools/toolboxes/rsf_stuff/datapath/
sffft1 opt="n" inv="y" sym="y" > ./result.rsf
```

SLIMpy Compiles

- Most Compilers build an AST
  - c,c++ build and AST with processor instruction set
- SLIMpy builds coarse-grained AST with Linear Algebra commands.
  - reduction/transformation operations that include
    - element-wise addition, subtraction, multiplication
    - vector inner products
    - norms l1, l2 etc
  - Matrix/Vector operations that include
    - implicit linear operators
    - explicit linear operators

Parallelization

- Different Branches of the AST can be run on different hosts.
  - handles copying and organizing data
  - exploits parallel extension of RSF (parallel “file system”)
- SLIMpy can also utilize existing MPI programs
- Look at the previous example again
  - Can use domain decomposition

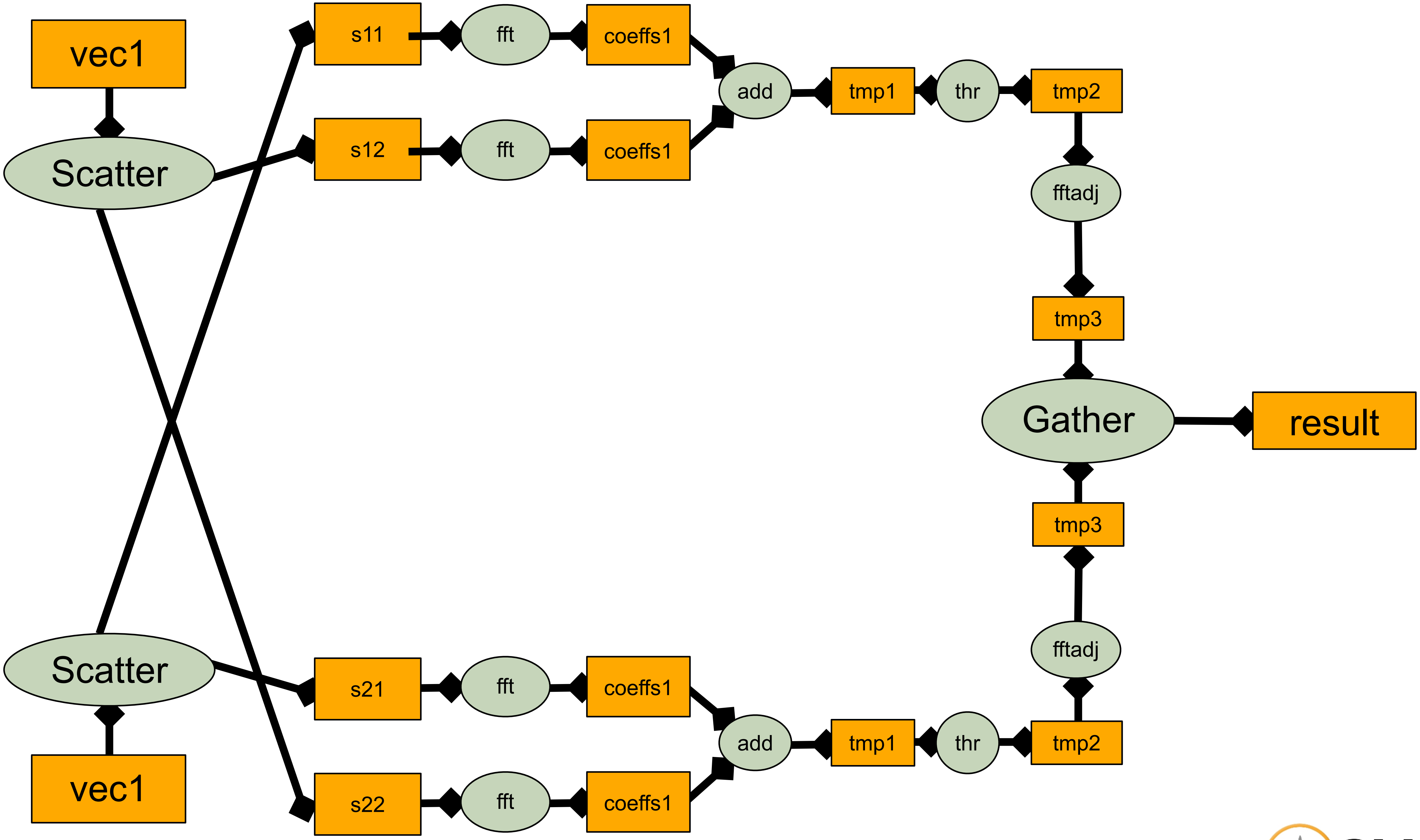
Parallel:
Domain Decomposition

```
vec1 = vector( 'vec1.rsf' )
vec2 = vector( 'vec2.rsf' )

P = Scatter( domain=vec.space , [2,1] )
F = fft( domain=P.range )
C = CompoundOperator( [F,P] )

coeffs1 = C * vec1
coeffs2 = C * vec2

tmp3 = coeffs1 + coeffs2

tmp4 = tmp3.thr( 0.001 )

result = C.adj() * tmp3

End()
```

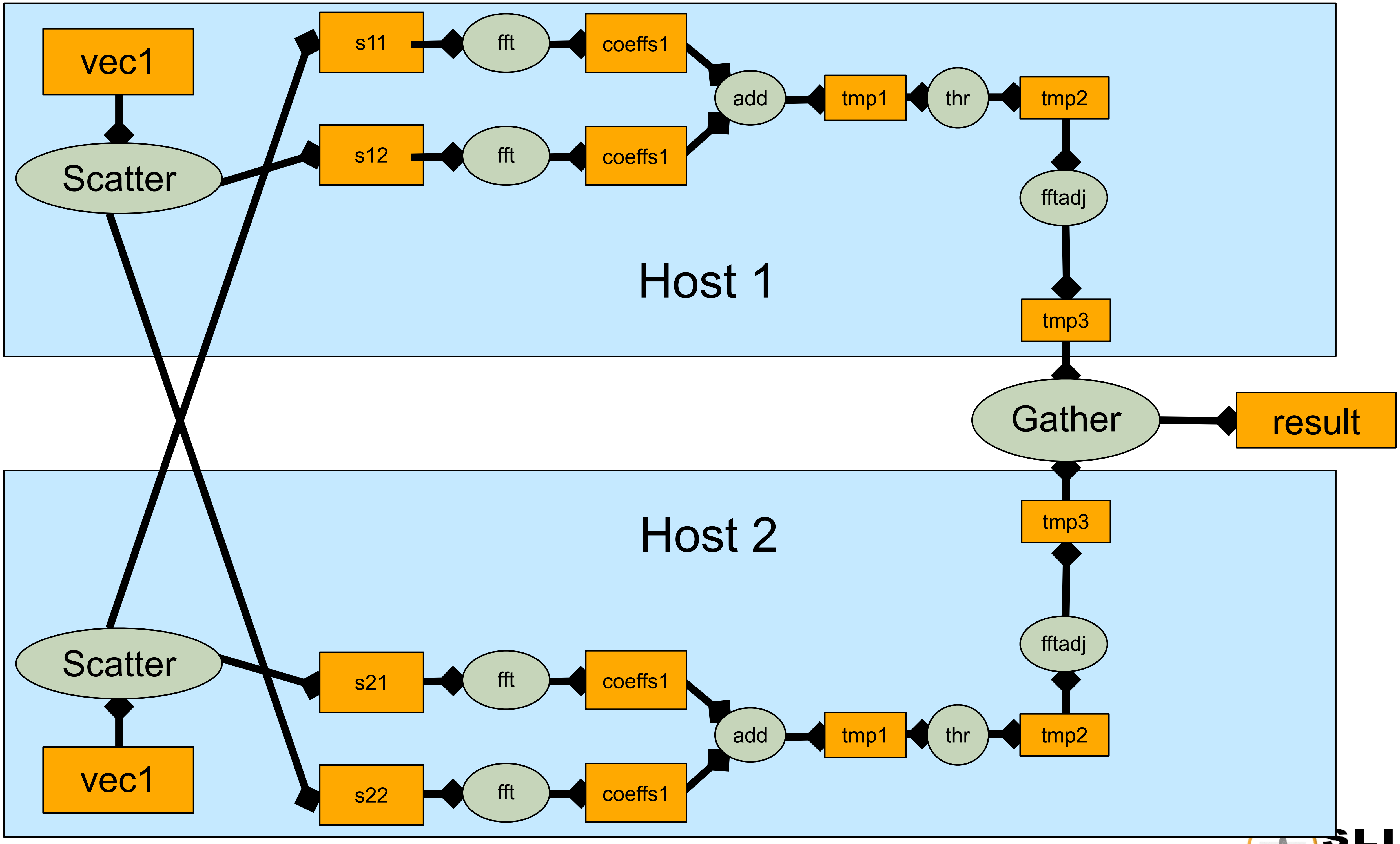
Redefine operator C as:
F(P(X))

C <=> Scatter -> fft
C.adj() <=> fft inv -> Gather

Abstract Syntax Tree



Abstract Syntax Tree



Output

```
< ./s2.rsf sfwindow f1="0" f2="0" n1="5" n2="10" | sffft1 opt="False" inv="n" sym="True" |
DATAPATH=/Tools/toolboxes/rsf_stuff/tmp_datapath/ sffft3 opt="False" inv="n" sym="True" pad="1"
axis="2" > /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft.00011.rsf
< ./s2.rsf sfwindow f1="5" f2="0" n1="5" n2="10" | sffft1 opt="False" inv="n" sym="True" |
DATAPATH=/Tools/toolboxes/rsf_stuff/tmp_datapath/ sffft3 opt="False" inv="n" sym="True" pad="1"
axis="2" > /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft.00012.rsf
< ./s1.rsf sfwindow f1="5" f2="0" n1="5" n2="10" | sffft1 opt="False" inv="n" sym="True" | sffft3
opt="False" inv="n" sym="True" pad="1" axis="2" | sfmath output="vec+input" vec="/Tools/toolboxes/
rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft.00012.rsf" | sfthr mode="soft" thr="0.001" | sffft3
opt="False" inv="y" sym="True" pad="1" axis="2" | DATAPATH=/Tools/toolboxes/rsf_stuff/
tmp_datapath/ sffft1 opt="False" inv="y" sym="True" > /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.
14381.env1.FakeMo.fft1.00020.rsf
< ./s1.rsf sfwindow f1="0" f2="0" n1="5" n2="10" | sffft1 opt="False" inv="n" sym="True" | sffft3
opt="False" inv="n" sym="True" pad="1" axis="2" | sfmath output="vec+input" vec="/Tools/toolboxes/
rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft.00011.rsf" | sfthr mode="soft" thr="0.001" | sffft3
opt="False" inv="y" sym="True" pad="1" axis="2" | sffft1 opt="False" inv="y" sym="True" |
DATAPATH=/Tools/toolboxes/rsf_stuff/datapath/ sfcat /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.
14381.env1.FakeMo.fft1.00020.rsf axis="1" > ./res.rsf
sfrm /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft1.00020.rsf
sfrm /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft.00012.rsf
sfrm /Tools/toolboxes/rsf_stuff/tmp_datapath/slim.14381.env1.FakeMo.fft.00011.rsf
```

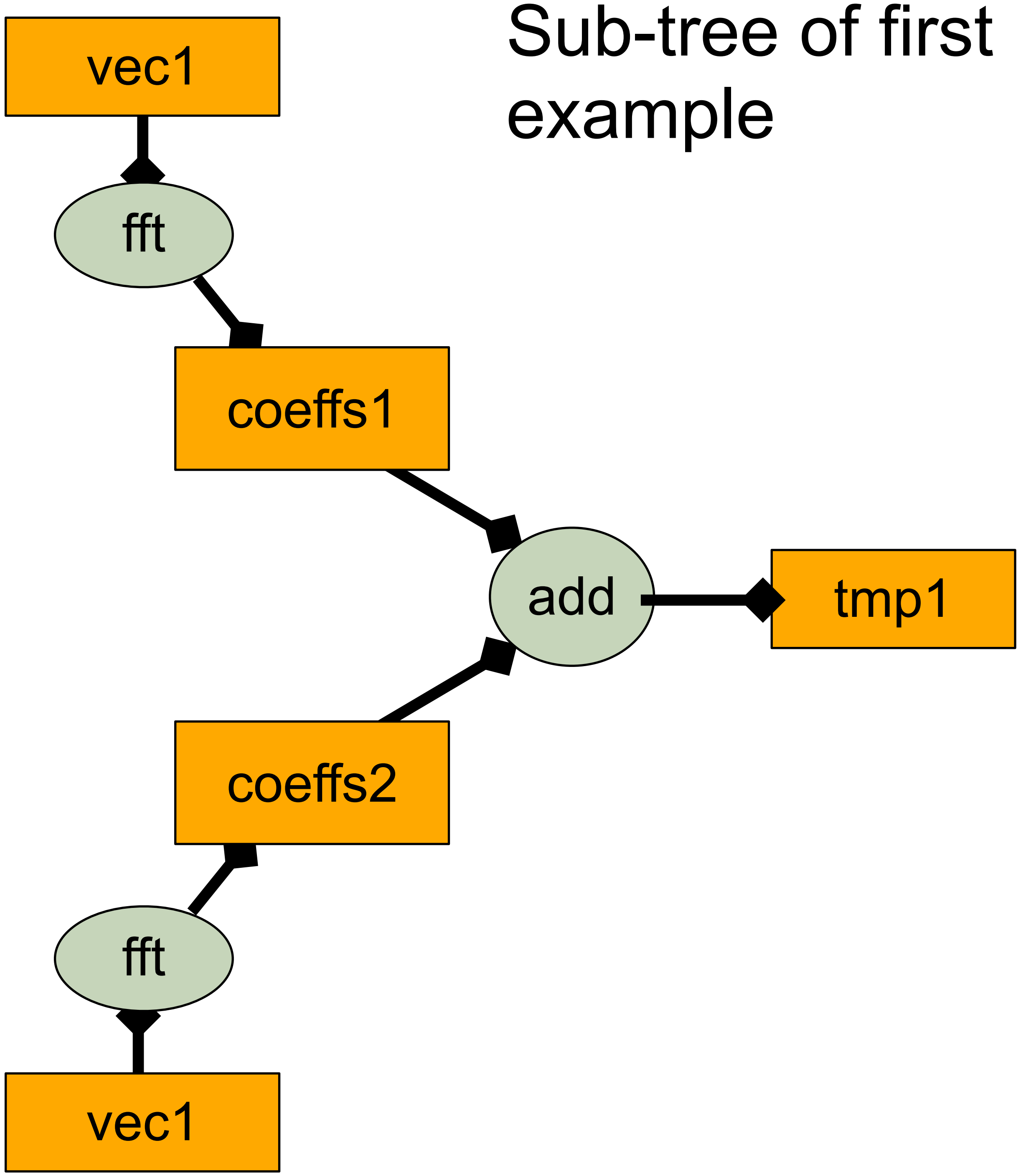
Other Benefits of AST

- Lots of existing software use AST

- Potential for optimization
  - C++: has “-O2” flag

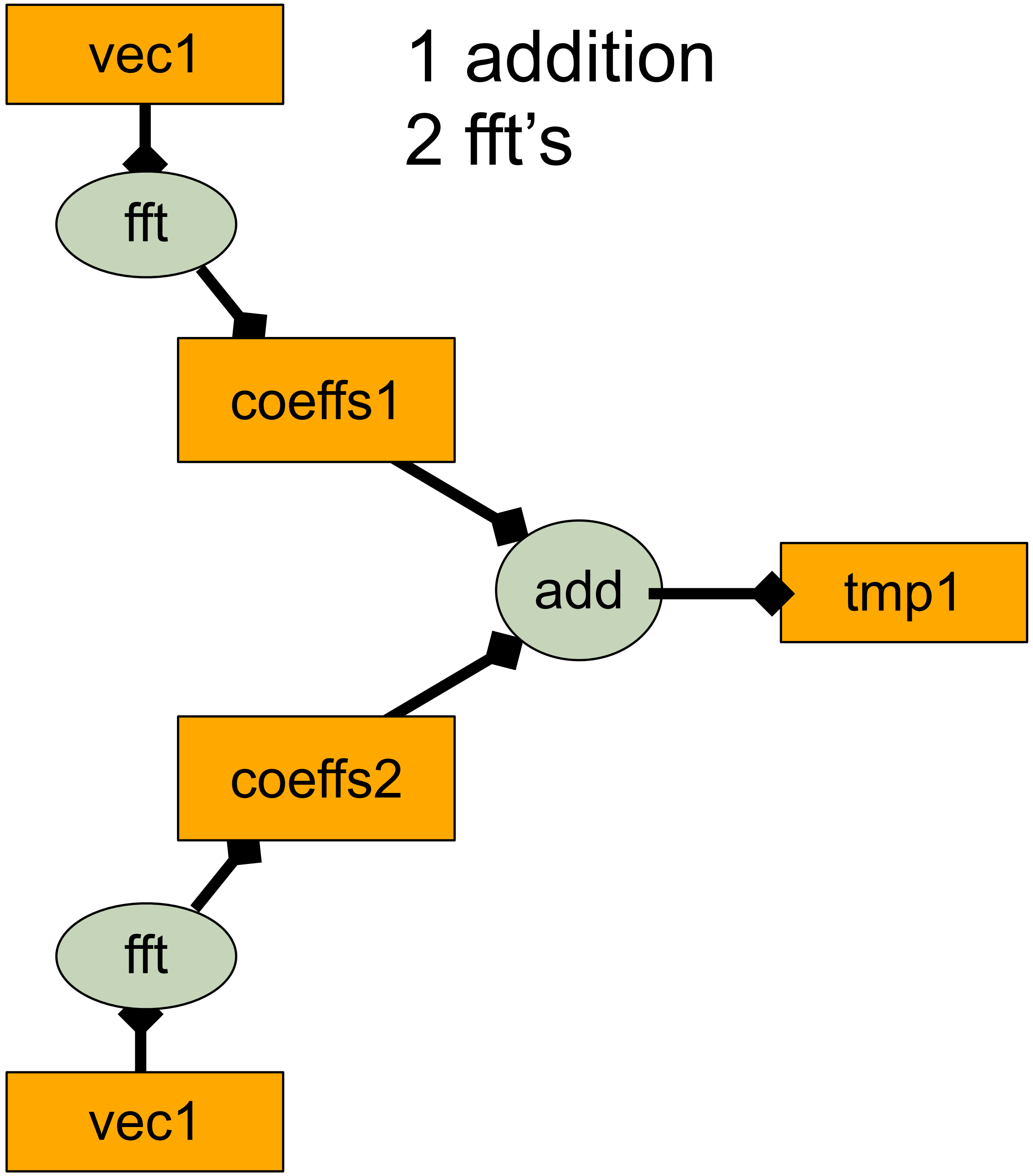
- SLIMpy: Symbolic optimization
  - F(x) + F(y) <=> F(x+y)
- Useful for augmented system of equations
  - $[F\ F] * [x\ y]^T$

AST Optimization

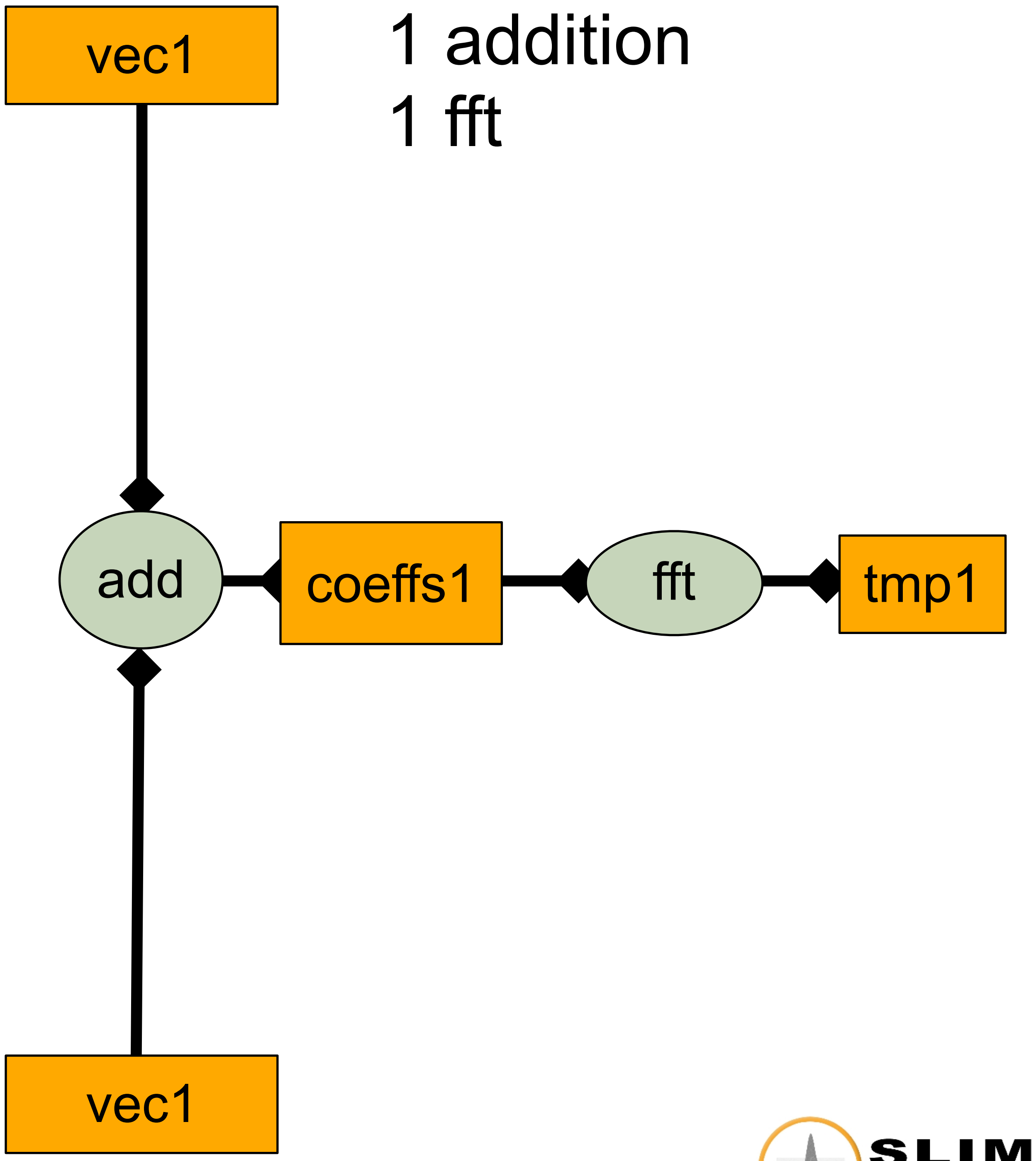


Sub-tree of first example

AST Optimization



1 addition
2 fft’s



1 addition
1 fft

Features

- Automatic Domain/Range checking of Operators
- Automatic dot-test for linear operators
  - <F(x), F(y)> == <x,y>
- Plugin system to add definitions of lower level software

SLIM
Seismic Laboratory for Imaging and Modeling

Demo

- Using an l1-solver

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

- For 2d and 3d de-noise:
  - where A == ST is the inverse curvelet transform
- For interpolation:
  - A = [R C^H]
  - $S^T = C^H$
- For interpolation with parallel and MPI
  - A = ...
  - x = ...

SLIM Seismic Laboratory for Imaging and Modeling

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" ...*

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 ....

Future plans

Improve stability of parallel extension

Prepare a public-domain release

Extend the functionality of the AST

- symbolic optimization

Implement the Kronecker product

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)