InSAR Scientific Computing Environment

Advanced Information Systems Technology Task AIST-08-0023

Paul A. Rosen, JPL (PI)
Howard Zebker, SU (Co-I)
Eric Gurrola, JPL (Co-I)
Giangi Sacco, JPL (Co-I)
Mark Simons, Caltech (Co-I)
Scott Hensley, JPL (Collaborator)
David Sandwell, SIO (Collaborator)

December 14, 2009
Fall AGU, San Francisco, CA 2009
# InSAR Scientific Computing Environment

**PI:** Rosen, Paul, Jet Propulsion Laboratory

## Objective
- Develop an open-source, modular, extensible InSAR computing environment for the research community
- Incorporate state-of-the-art, highly accurate algorithms to automate InSAR processing for non-experts and experts alike
- Document algorithms, formats and interfaces to facilitate community involvement in continuing development beyond the AIST horizon

## Approach
- Develop community-based requirements for InSAR processing methods and generalized data models
- Develop a modular, extensible, object-oriented processing framework
- Develop modules for the ISCE architecture
- Test and document ISCE framework

## Co-Is/Partners
- Howard Zebker, Stanford University
- Mark Simons, Caltech
- Eric Gurrola, Giangi Sacco, Scott Hensley, JPL
- David Sandwell, Scripps Institution of Oceanography

## Key Milestones
- First draft of requirements document (8/09→10/09)
- First draft of architecture description (1/10)
- Algorithmic accuracy improvements (4/10)
- Framework recasting of processing engines (12/10)
- Principal code elements complete (4/11)
- Completion of testing (2/12)
- Deliver final documentation and software (3/12)

## TRL\textsubscript{in} = 3
InSAR SCE and ROI_PAC background

- InSAR SCE was proposed and accepted as an open source package.
- InSAR SCE was founded on a science community workshop funded by NASA.
- The science community accepted the proposal based on this notion; the community looks forward to an open source solution.
  - Without open source, this development becomes much less relevant for scientists.

- ROI_PAC was released through EAR and contains similar algorithms. The algorithms are published and in the public domain. Its license has expired.
- Upgrades to ROI_PAC to be included in InSAR SCE are published and just need to be implemented.
The Standard ROI_PAC Flow

- Standard ROI_PAC recipe creates geocoded unwrapped interferograms from two raw data files, an orbit file, and other ancillary meta-data.
- Allows the use of a DEM for terrain correction and topography removal from interferogram.
- Allows the use of an a priori deformation model for the purpose of enhancing baseline re-estimation or unwrapping.
- Prescription for processing is somewhat inflexible.
InSAR SCE Framework Requirements

Note: zoom to 400% to read

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Validation Method</th>
<th>Responsible Parties</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7.5.1</td>
<td>The SCE Framework should provide a user interface that is intuitive and easy to use.</td>
<td>Observation</td>
<td>User, Stakeholder</td>
<td>2/1/11</td>
</tr>
<tr>
<td>5.7.5.2</td>
<td>The SCE Framework should provide a user interface that is adaptable to different devices.</td>
<td>Observation</td>
<td>User, Stakeholder</td>
<td>2/1/11</td>
</tr>
<tr>
<td>5.7.5.3</td>
<td>The SCE Framework should provide a user interface that is consistent across all platforms.</td>
<td>Observation</td>
<td>User, Stakeholder</td>
<td>2/1/11</td>
</tr>
<tr>
<td>5.7.5.4</td>
<td>The SCE Framework should provide a user interface that is customizable.</td>
<td>Observation</td>
<td>User, Stakeholder</td>
<td>2/1/11</td>
</tr>
<tr>
<td>5.7.5.5</td>
<td>The SCE Framework should provide a user interface that is responsive to user feedback.</td>
<td>Observation</td>
<td>User, Stakeholder</td>
<td>2/1/11</td>
</tr>
</tbody>
</table>

Note: This table is illustrative and not comprehensive.
The ROI_PAC framework

PERL

Input data from previous stage
- Process input data
- Create initialization file
- System call to run Fortran code
- Retrieve and process results
- Output data for next stage

FORTRAN

- Parse initialization file and initialize parameters
- Compute
- Save results to file or print to screen
The mroipac framework

**PYTHON MAIN**
- Input data from previous stage
  - Process input data
    - Initialization components
  - Call python wrapper around Fortran
    - Retrieve and process results
      - Output data for next stage

**PYTHON WRAPPER**
- Extract parameters from components
  - Initialize Fortran data module and Image API
    - Run Fortran compute engine
      - Retrieve data from Fortran data module

**FORTRAN**
- Image API
  - Compute engine
- Data module
The mroipac motto

**do one thing, but do it right**

**ROI_PAC**
- Receive input data and initialize
- Handle file and std I/O
- Operate on input data

**mroipac**
- Receive input data and initialize
- Handle file and std I/O
- Operate on input data

- Control API
- StdIO and Image API
- Computational Engine
Image API Example

Example: interleaving scheme translation.

Input image: PIL

<table>
<thead>
<tr>
<th>R₀₀</th>
<th>I₀₀</th>
<th>R₀₁</th>
<th>I₀₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁₀</td>
<td>I₁₀</td>
<td>R₁₁</td>
<td>I₁₁</td>
</tr>
<tr>
<td>R₂₀</td>
<td>I₂₀</td>
<td>R₂₁</td>
<td>I₂₁</td>
</tr>
</tbody>
</table>

Image API

Output image: BSQ

<table>
<thead>
<tr>
<th>R₀₀</th>
<th>R₀₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁₀</td>
<td>R₁₁</td>
</tr>
<tr>
<td>R₂₀</td>
<td>R₂₁</td>
</tr>
<tr>
<td>I₀₀</td>
<td>I₀₁</td>
</tr>
<tr>
<td>I₁₀</td>
<td>I₁₁</td>
</tr>
<tr>
<td>I₂₀</td>
<td>I₂₁</td>
</tr>
</tbody>
</table>

R = Real
I = Imaginary
PIL = pixel inter-leaved
BIL = band inter-leaved
BSQ = band sequential

Engine format: BIL
Example: FormSLC – the core radar processor

The Driver

```python
import FormSLCPy
import obj ...

#initialize the objects from a file
objects = obj.createObject(file)

#run the module by passing the objects
FormSLCPy.run(objects)

#get results after computation
results = FormSLCPy.getResults()
```

```plaintext
image.filename = filename
image.width = width ...

radar.frequency = frequency
radar.chirp = chirp ...

platform.name = name
platform.height= height ...

planet.radius = radius
planert.angularSpeed= angularSpeed ...
```
Restructuring of I/O in Fortran

Create image: old way

```fortran
program OldWay
integer*4 FD
character*80 filename
integer*4 lineWidth
......
! open file from inside the FORTRAN code
open(FD,file=filename,access='direct',recl=lineWidth,form='unformatted')
```
Restructuring of I/O in Fortran

Create image: new way

```fortran
program NewWay
integer*8 imagePointer
character*80 filename
integer*4 lineWidth, tileHeight
......
! get the image pointer
call initImage(imagePointer,filename,'read','l','complex',lineWidth,tileHeight)
```

! initialize image object associated with pointer
call initImage(imagePointer,filename,'read','l','complex',lineWidth,tileHeight)
Restructuring of I/O in Fortran

Set or get data: old way

```fortran
program OldWay
complex*8,dimension(:),allocatable:: data
integer*4 lineNumber
......
! read or write data directly from or to file
read(FD, rec = lineNumber) (data(k), k = 1, lineWidth)
(or)
write(FD, rec = lineNumber) (data(k), k = 1, lineWidth)
```
Restructuring of I/O in Fortran

Set or get data: new way

program NewWay
   complex*8,dimension(:),allocatable:: data
   integer*4 lineNumber

   ! get one line of data by passing the associated image pointer
   call getLine(imagePointer,data,lineNumber)
   (or)
   ! set one line of data
   call setLine(imagePointer,data,lineNumber)
Image API is available outside the framework

<table>
<thead>
<tr>
<th>DRIVER</th>
<th>PROGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>Fortran</td>
</tr>
<tr>
<td>C or C++</td>
<td>Fortran</td>
</tr>
<tr>
<td>Python</td>
<td>Fortran (mroipac)</td>
</tr>
<tr>
<td>C or C++</td>
<td>C or C++</td>
</tr>
<tr>
<td>Fortran</td>
<td>C or C++ (not tested)</td>
</tr>
<tr>
<td>Python</td>
<td>C or C++ (mroipac)</td>
</tr>
<tr>
<td>Python</td>
<td>Python (not implemented, but easy)</td>
</tr>
</tbody>
</table>

**DRIVER** = who initializes the Image API  
**PROGRAM** = who uses the Image API methods
Image API is Documented

install:LineAccessorPy

install:LineAccessorPy Namespace Reference

```
def install:LineAccessorPy:getLineAccessoPointer ( self )
Provides pointer associated with the "C" accessor object.
Returns:
int pointer to the "C" LineAccessor object.


def install:LineAccessorPy:getMachineEndianness ( self )
Returns the endianness of the machine running the code.
Does not require that initLineAccessor be called before execution.
Returns:
character 'B' for big endian and 'L' for little endian.


def install:LineAccessorPy:getTypeSize ( self, type )
Returns the size of the data type "type".
Parameters:
type data type.
Returns:
int size of type.


def install:LineAccessorPy:initLineAccessor ( self, filename, filemode, endian, type, row, col )
Initializes the accessor object.
Parameters:
filename string name of the file to be accessed.
filemode string access mode of the file.
e endian character endianness of the data stored in the file. Values are 'B' or 'L' for big endian and 'T' or 'L' for little endian.
type string file data type.
row int number of rows of the buffer file. Set it to one if no tiling is desired.
col int number of columns of the buffer file; it must be equal to the number of columns of the associated file.
See also:
printAvailableDataTypesAndSizes().
```

Classes

class LineAccessor
This Class provides a set of convenient method.

Functions

- `def initLineAccessor ()` initializes the accessor object.
- `def createLineAccessorObject ()` creates a LineAccessor object.
- `def createFile ( )` creates a file object opened in write or w/extend.
- `def getMachineEndianness ()` returns the endianness of the machine running the code.
- `def finalizeLineAccessor ( LineAccessor obj )` always call this function if a LineAccessor obj.
- `def changeBanScheme ( )` changes the file format from BandScheme to.
- `def convertFileEndianness ( )` changes the file endianness.
- `def getTypeSize ( )` returns the size of the data type "type".
- `def getFileLength ( )` provides the number of lines of the file associated.
- `def getFileWidth ( )` provides the number of columns of the file associated.
- `def getLineAccessoPointer ( )` provides pointer associated with the "C" accessor.
- `def printObjectInfo ( )` prints a series of information related to the file.
- `def printAvailableDataTypesAndSizes ( )` prints the available data types and their sizes.

Variables

- `LineAccessoObj` pointer to the C++ LineAccessoLineAccesso.
Summary and Outlook

• InSAR Scientific Computing Environment is being designed according to (our interpretation of) community request for a more modular modern InSAR framework.
  – Requirements are responsive to community workshop recommendations
  – Ability to perform simple operations through complex scripts

• Community desire to have open software is a goal not without obstacles.
  – Security concerns and misconceptions exist with regard to spaceborne SAR data and processing capability
  – InSAR SCE is breaking ground in pushing for open distribution from an FFRDC
  – Some form of licensing scheme is inevitable (GPL, Apache, etc.)

• We are hoping to release beta code for core framework elements by Summer 2010 for the community to test and contribute to.