PyStella

Python bindings for the STELLA library

Lucas Benedicic[1]

[1]CSCS - ETH Zürich

July 7, 2014

Outline

1. Using STELLA
   - Accessing the C++ backend
2. Defining new stencils
   - Just-In-Time compilation

Outline

Accessing the C++ backend

- Expose STELLA's *DataFields* and *Stencils* to Python.

```
~:> ipython
In [1]: from stella import DataField
In [2]: from stella.stencil import Coriolis
```

Accessing the C++ backend

- Expose STELLA's *DataFields* and *Stencils* to Python.

```
~:> ipython
In [1]: from stella import DataField
In [2]: from stella.stencil import Coriolis
```

NumPy compatibility

- *DataFields* use *NumPy* arrays as their data storage structure.

```
In [3]: u = DataField ( )
In [4]: u.storage = np.arange(6000, dtype=np.float64).reshape((100,60,1))
In [5]: print (u.domain)
Out [5]: (100,60,1)
```

NumPy compatibility

- *DataFields* use *NumPy* arrays as their data storage structure.

```
In [3]: u = DataField ( )
In [4]: u.storage = np.arange(6000, dtype=np.float64).reshape((100,60,1))
In [5]: print (u.domain)
Out [5]: (100,60,1)
```

NumPy compatibility

- Built-in support for data I/O and plotting.

```
In [6]: f=open ('/tmp/my_data.csv', 'r')
In [7]: u.storage = np.fromfile (f, dtype=np.float64))
In [8]: v.storage = np.fromfile (f, dtype=np.float64))
In [9]: f.close ( )
```

NumPy compatibility

- Built-in support for data I/O and plotting.

```
In [6]: f=open ('/tmp/my_data.csv', 'r')
In [7]: u.storage = np.fromfile (f, dtype=np.float64))
In [8]: v.storage = np.fromfile (f, dtype=np.float64))
In [9]: f.close ( )
```

Documentation

- Interactive documentation available from Python.

```
In [9]: Coriolis?
Type:        builtin_function_or_method
String form: <built-in function fromfile>
Docstring: Coriolis (in_u, in_v, in_fc, out_utens, out_vtens)
Applies the Coriolis stencil using the given force over the input data
fields, generating two independent output fields.
Parameters:
in_u  : input data field;
in_v  : input data field;
in_fc : a scalar representing the force applied;
...
```

Documentation

- Interactive documentation available from Python.

```
In [9]: Coriolis?
Type:        builtin_function_or_method
String form: <built-in function fromfile>
Docstring: Coriolis (in_u, in_v, in_fc, out_utens, out_vtens)
Applies the Coriolis stencil using the given force over the input data
fields, generating two independent output fields.
Parameters:
in_u  : input data field;
in_v  : input data field;
in_fc : a scalar representing the force applied;
...
```

Execution

- *Stencil* execution from Python.

```
In [10]: c = Coriolis (u, v, 3.5, utens, vtens)
In [11]: c.do ( )
In [12]: print (utens)
Out[12]:
array([[[  0.00000000e+00],
        [  1.00000000e+00],
        [  2.00000000e+00],
        ...,
        [  5.70000000e+01],
        [  5.80000000e+01],
        [  5.90000000e+01]],
...
```

Execution

- *Stencil* execution from Python.

```
In [10]: c = Coriolis (u, v, 3.5, utens, vtens)
In [11]: c.do ( )
In [12]: print (utens)
Out[12]:
array([[[  0.00000000e+00],
        [  1.00000000e+00],
        [  2.00000000e+00],
        ...,
        [  5.70000000e+01],
        [  5.80000000e+01],
        [  5.90000000e+01]],
...
```

Complete example

```
# -----
# Solving the solution of the heat equation by using the Laplace operator with a time steppi
# -----
import numpy as np
from stella.data import SwapField
from stella.stencil import LaplaceStencil

# set number of time steps to run the simulation
timeSteps = 20000
# boundary condition values
westValue = 0.0
eastValue = 1.0
# initialize the sizes of the domain of our fields
fieldDomain = np.array ((100, 60, 1))
# instantiate a swap field, containing input and output data fields
data = SwapField (name='MyDataField', domain=fieldDomain)
# initialize values of the input field to 0.5
data.input.storage = np.array ([0.5] * in_data.domain.size)
# instantiate the Laplace stencil object, implemented in C++ Stella
laplace = LaplaceStencil (data.input, data.output, westValue, eastValue)
# perform the simulation
for t in range (timeSteps):
    # apply the boundary condition and the Laplace stencil
    laplace.do ( )
    # show some progress information every 20 steps
    if (t % 20) == 0:
        print ("Step %d/%d" % (t, timeSteps))
    # swap the input and output data fields
    data.swap ( )
```

Outline

New stencils in Python

- *Stencil* definition in pure Python using *NumPy*.
- Executable in pure Python mode.
- Eases design, prototyping and debugging.
- Automatic (JIT) compilation to native C++.

New stencils in Python

- *Stencil* definition in pure Python using *NumPy.*
- Executable in pure Python mode.
- Eases design, prototyping and debugging.
- Automatic (JIT) compilation to native C++.

New stencils in Python

- *Stencil* definition in pure Python using *NumPy.*
- Executable in pure Python mode.
- Eases design, prototyping and debugging.
- Automatic (JIT) compilation to native C++.

New stencils in Python

- *Stencil* definition in pure Python using *NumPy.*
- Executable in pure Python mode.
- Eases design, prototyping and debugging.
- Automatic (JIT) compilation to native C++.

New stencils in Python - example

```
# --- DEFINITION of the Coriolis stencil object
class CoriolisKernel (StencilKernel):
    """Class definition of the Coriolis stencil.-"""
    def __init__ (self, utens, vtens):
        super.__init__ (self)
        # output fields
        self.utens = utens
        self.vtens = vtens
    def _USlowTensStage (self, ctr, in_v, in_fc):
        """The 'Do' function of the U stage.-"""
        return ( in_fc * np.average ((in_v, in_v[1, 0])) +
                 in_fc * np.average ((in_v[0, -1], in_v[1, -1])) ) / 2.0
    def _VSlowTensStage (self, ctr, in_u, in_fc):
        """The 'Do' function of the V stage.-"""
        return ( in_fc * np.average (in_u[0, 0], in_u[0, 1]) +
                 in_fc * np.average (in_u[-1, 0], in_u[-1, 1]) ) / 2.0
    def kernel (self, in_u, in_v, in_fc):
        """This stencil comprises two independent stages.-"""
        for p in out_utens.interior_points (sweep='cKIncrement'):
            self.out_utens[p] += self._USlowTensStage (p, in_v, in_fc)
        for p in out_vtens.interior_points (sweep='cKIncrement'):
            self.out_vtens[p] -= self._VSlowTensStage (p, in_u, in_fc)
# --- USAGE of the Coriolis stencil object
kernel = CoriolisKernel (utens, vtens)
kernel.compilation.should_unroll = False
kernel.compilation.backend       = 'cxx'
kernel.kernel (u, v, 3.5)
```

New stencils in Python - another example

```
# -- DEFINITION of the FastWavesSCTridiag stencil object
class FastWavesSCTridiag (StencilKernel):
    """Class definition of the FastWavesSCTridiag stencil.-"""
    def __init__ (self, y, bet):
        super.__init__ (self)
        # output fields
        self.y   = y
        self.bet = bet
        self.tmp = DataField (self.y.get_domain ( ))
    def _ForwardStage (self, ctr, in_b, in_rhs):
        """The 'Do' function of the Forward stage, applied over the KMinimunCenter.-"""
        self.bet = in_b
        self.y   = in_rhs / self.bet
    def _ForwardStageFull (self, ctr, in_a, in_b, in_c, in_rhs):
        """The 'Do' function of the Forward stage, applied over the FullDomain.-"""
        self.tmp = in_c[0, 0, -1] / self.bet
        self.bet = in_b - in_a * self.tmp
        self.y   = (in_rhs - in_a * self.y[0, 0, -1] / self.bet
    def _BackwardStage (self, ctr, in_tmp):
        """The 'Do' function of the Backward stage.-"""
       self.y -= in_tmp[0, 0, 1] * self.y[0, 0, 1]
    def kernel (self, in_a, in_b, in_c, in_rhs):
        """Compulsory function definition.-"""
        for p in self.y.interior_points (sweep='cKIncrement', height='KMinimumCenter'):
            self._ForwardStage (p, in_b, in_rhs)
        for p in self.y.interior_points (sweep='cKIncrement'):
            self._ForwardStageFull (p, in_a, in_b, in_c, in_rhs)
        for p in self.y.interior_points (sweep='cKDecrement'):
            self._BackwardStage (p, self.tmp)
```

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.

Summary

- Follows well-known, standard interfaces, e.g., *SciPy/NumPy*.
- Takes advantage of a rich set of existing tools for:
  - interactive prototyping and documentation, e.g., *ipython*;
  - plotting, e.g., *matplotlib*;
  - debugging, e.g., *ipdb*;
  - data I/O, e.g., *NumPy*.
- Outlook
  - first version of Stella stencils from Python ready for the PASC GridTools meeting.