phyfun_jupyter_notebook_demo_02.23.18

April 19, 2018

1 Phylanx in Python: A Tool that Sort-Of Works

In [3]:
```
import phylanx
from phylanx.ast import *
from phylanx.ast.utils import printout
import numpy as np

et = phylanx.execution_tree
```

In [4]:
```
# Step 1: Write PhySL code in PhySL
# but execute it with Python
result = et.eval("""
block(
    define(fib,n,
    if(n<2,n,
        fib(n-1)+fib(n-2))),
    fib)""",et.var(10))
print(result)
```

```
55
```

In [5]:
```
# Step 2: Write Python code that
# gets magically turned into PhySL code.

@Phylanx() # biuld with phylanx!
def foo():
    print("hello")    # normal code
    return 1          # At the moment, it needs a valid return type
```

In [6]:
```
foo() # This worked, but the output looks obscure
```

Out[6]:
```
<_phylanx.execution_tree.primitive at 0x7fe73c0abd50>
```

In [7]:
```
foo()[0] # Extract the Python value
```

Out[7]:
```
1.0
```

```
In [8]: printout(foo()) # This also works.

1.0


In [9]: import numpy as np
        f = np.linspace(0,10,11) # create a simple numpy array

In [10]: @Phylanx
         def pr(a):
             print(a) # This prints to the console, so we don't see it.
             return a[1:5] # Slicing works
         print("output")
         print(f) # print the numpy array, we can pass it to phylanx
         #print(pr(f)) # phy_print should know how to print arrays

output
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]


In [11]: # About the console issue... the problem is stdout doesn't go to the notebook.
         import os
         os.write(1,b'Hello, Console')

Out[11]: 14

In [12]: # You can access the source code like this.
         # We have $ in the symbol names to identify line and column number.
         print(foo.__physl_src__)

         # Using regexes, we can make it more human readable
         import re
         print(re.sub(r'\$\d+','',foo.__physl_src__))

block$1$0(define$1$0(foo$1$0, block$1$0(cout("hello"),  1)), foo$1$0)

block(define(foo, block(cout("hello"),  1)), foo)


In [13]: @Phylanx()
         def fib(n):
             if n < 2:
                 return n
             f1 = fib(n-1)
             f2 = fib(n-2)
             return f1+f2 # Doesn't work, the underlying PhySL only supports a single return
```

```
---------------------------------------------------------------------------

Exception                                 Traceback (most recent call last)

<ipython-input-13-115beb1bb6a9> in <module>()
----> 1 @Phylanx()
      2 def fib(n):
      3     if n < 2:
      4         return n
      5     f1 = fib(n-1)


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
436
437             self.__physl_src__ = '%s(%s)\n' % (
--> 438                 full_node_name(tree.body[0], 'block'), physl.recompile(tree))
439
440         def __call__(self, *args):


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
376                 return self.nodes[nm](self, a, allowreturn)
377             else:
--> 378                 return self.nodes[nm](self, a)
379         except NotImplementedError:
380             raise Exception('unsupported AST node type: %s' % nm)


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
207     def _Module(self, a):
208         args = [arg for arg in ast.iter_child_nodes(a)]
--> 209         return self.recompile(args[0])
210
211     def _Return(self, a, allowreturn=False):


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
376                 return self.nodes[nm](self, a, allowreturn)
377             else:
--> 378                 return self.nodes[nm](self, a)
379         except NotImplementedError:
380             raise Exception('unsupported AST node type: %s' % nm)


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
154                     s += self.recompile(aa, True)
155                 else:
```

```
--> 156                     s += self.recompile(aa, False)
    157                     s += ", "
    158             s += ")"


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
376                 return self.nodes[nm](self, a, allowreturn)
377             else:
--> 378                 return self.nodes[nm](self, a)
379         except NotImplementedError:
380             raise Exception('unsupported AST node type: %s' % nm)


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
265             s += ")"
266         else:
--> 267             s += self.recompile(a.body[0], allowreturn)
268         s += ","
269         if len(a.orelse) > 1:


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
376                 return self.nodes[nm](self, a, allowreturn)
377             else:
--> 378                 return self.nodes[nm](self, a)
379         except NotImplementedError:
380             raise Exception('unsupported AST node type: %s' % nm)


~/.local/lib/python3.5/site-packages/phylanx-0.0.1-py3.5-linux-x86_64.egg/phylanx/ast/tr
212         if not allowreturn:
213             raise Exception(
--> 214                 "Return only allowed at end of function: line=%d\n" % a.lineno)
215         args = [arg for arg in ast.iter_child_nodes(a)]
216         return " " + self.recompile(args[0])


Exception: Return only allowed at end of function: line=3
```

In [14]:
```
@Phylanx()
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1)+fib(n-2) # OK, if every switch in an if/elif/else block
                                 # does a return, that counts as a single return
```

```
In [15]: fib(8)[0]
```

```
Out[15]: 21.0
```

```
In [16]: @Phylanx()
         def sw(s):
             # if / elif /else works
             if s == "hello":
                 return "hi"
             elif s == "goodbye":
                 return "bye"
             else:
                 return "?"
         print(sw("goodbye"))
         print(re.sub(r'\$\d+','',sw.__physl_src__))
```

```
bye
block(define(sw, s, if(s == "hello", "hi",if(s == "goodbye", "bye", "?"))), sw)
```

```
In [17]: @Phylanx()
         def floop():
             sum = 0
             n = -1
             for i in range(10,1,n): # Basic for loops work
                 sum += i
             return sum
         print(re.sub(r'\$\d+','',floop.__physl_src__))
         print(floop())
```

```
block(define(floop, block(define(sum,0), define(n,-(1)), if(n > 0,for(define(i,10),i < 1,store(i
```

```
54
```

```
In [18]: @Phylanx()
         def wloop():
             sum = 0
             i = 0
             while i < 10: # basic while loops work
                 sum += i
                 i += 1
             return sum
         print(wloop())
```

```
45
```

```
In [19]: import numpy as np

         @Phylanx()
         def ar():
             return constant(0,10) # creation of phylanx arrays
         print(ar())
         assert ar() == np.zeros(10) # compare with numpy
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
In [20]: @Phylanx()
         def ar2(a):
             return a[3:8]
         print(ar2(np.linspace(0,10,11))) # you can pass numpy arrays
```

```
[3, 4, 5, 6, 7]
```

```
In [21]: # %load ./examples/algorithms/lra_csv_phyfun.py
         #  Copyright (c) 2018 Steven R. Brandt
         #  Copyright (c) 2018 R. Tohid
         #
         #  Distributed under the Boost Software License, Version 1.0. (See accompanying
         #  file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

         import phylanx
         from phylanx.ast import *
         from phylanx.ast.utils import printout


         @Phylanx("PhySL")
         def lra(file_name, xlo1, xhi1, ylo1, yhi1, xlo2, xhi2, ylo2, yhi2, alpha,
                 iterations, enable_output):
             data = file_read_csv(file_name)
             x = data[xlo1:xhi1, ylo1:yhi1]
             y = data[xlo2:xhi2, ylo2:yhi2]
             weights = constant(0.0, shape(x, 1))
             transx = transpose(x)
             pred = constant(0.0, shape(x, 0))
             error = constant(0.0, shape(x, 0))
             gradient = constant(0.0, shape(x, 1))
             step = 0
             while step < iterations:
                 if enable_output:
                     print("step: ", step, ", ", weights)
                 pred = 1.0 / (1.0 + exp(-dot(x, weights)))
                 error = pred - y
                 gradient = dot(transx, error)
```

```
            weights = weights - (alpha * gradient)
            step += 1
        return weights


res = lra("breast_cancer.csv", 0, 569, 0, 30, 0, 569, 30, 31, 1e-5, 750, 0)
printout(res)
```

```
1.4660777870530044
2.1753973169666887
8.607506009599426
3.8683427321006554
0.014103556383654771
-0.0034904356047104775
-0.023108908034423236
-0.00996550404834849
0.02679073480864044
0.011034305003479293
0.00570457098114057
0.1465402590294279
-0.031519040099807126
-3.7435783680219563
0.0008180767053176969
-0.010092561810918235
-0.0024513652094121327
-0.00020053639122219802
0.0024897987887794112
0.00022004547317046596
1.5497531958808144
2.795818676641823
8.783537220330045
-5.050962633585487
0.01803008202668555
-0.018165348612708906
-0.0451929407737975
-0.010102790322770713
0.03786452009149631
0.01086804591972698
```

```
In [22]: # %load ./examples/algorithms/lra_csv_phyfun_np.py
         #  Copyright (c) 2018 Steven R. Brandt
         #  Copyright (c) 2018 R. Tohid
         #
         #  Distributed under the Boost Software License, Version 1.0. (See accompanying
         #  file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
         import phylanx
         import numpy as np
```

```
from phylanx.ast import *
from phylanx.ast.utils import printout


@Phylanx("PhySL")
def lra(x, y, alpha, iterations, enable_output):
    weights = constant(0.0, shape(x, 1))
    transx = transpose(x)
    pred = constant(0.0, shape(x, 0))
    error = constant(0.0, shape(x, 0))
    gradient = constant(0.0, shape(x, 1))
    step = 0
    while step < iterations:
        if (enable_output):
            print("step: ", step, ", ", weights)
        pred = 1.0 / (1.0 + exp(-dot(x, weights)))
        error = pred - y
        gradient = dot(transx, error)
        weights = weights - (alpha * gradient)
        step += 1
    return weights


file_name = "breast_cancer.csv"

data = np.genfromtxt(file_name, skip_header=1, delimiter=",")
x = data[:, :-1]
y = data[:, -1:]
y = y.reshape((y.shape[0], ))
res = lra(x, y, 1e-5, 750, 0)
printout(res)
```

```
1.4660777870530044
2.1753973169666887
8.607506009599426
3.8683427321006554
0.014103556383654771
-0.0034904356047104775
-0.023108908034423236
-0.00996550404834849
0.02679073480864044
0.011034305003479293
0.00570457098114057
0.1465402590294279
-0.031519040099807126
-3.7435783680219563
0.0008180767053176969
-0.001092561810918235
```

```
-0.0024513652094121327
-0.00020053639122219802
0.0024897987887794112
0.00022004547317046596
1.5497531958808144
2.795818676641823
8.783537220330045
-5.050962633585487
0.01803008202668555
-0.018165348612708906
-0.0451929407737975
-0.010102790322770713
0.03786452009149631
0.01086804591972698
```

Partial list of things that don't work yet. * Using one phyfun function from within another * direct creation of data structures, i.e. [3,4,2,9] * Automatic conversion of return types to python types * Step arguments to Python slice operators * Nonetype as a return value * Jupyter magics for phylanx * parallel blocks / asyncs lambdas * Fix print function for Jupyter

```
In [1]: %%phylanx_cell
        print("No function!")
        for i in range(10):
            print("i=",i+1)
```

```
No function!
i= 1
i= 2
i= 3
i= 4
i= 5
i= 6
i= 7
i= 8
i= 9
i= 10
```

```
In [ ]: # %load /home/sbrandt/.ipython/profile_default/startup/magics.py
        from __future__ import print_function
        from IPython.core.magic import (Magics, magics_class, line_magic,
                                        cell_magic, line_cell_magic)

        import ast,re,os
        from threading import Thread

        from phylanx.ast import *
```

```
def readfds(fd):
    while True:
        result=os.read(fd,1000)
        print(result.decode('utf-8'),end='')

class IORedirecter:
    def __init__(self):
        self.fds = os.pipe()
        self.threadReadFDs = Thread(target=readfds,args=(self.fds[0],))
        self.threadReadFDs.start()
        self.saveStdout = os.dup(1)
    def __enter__(self):
        os.close(1)
        os.dup(self.fds[1])
    def __exit__(self,type,value,traceback):
        os.close(1)
        os.dup(self.saveStdout)

iod = IORedirecter()

# The class MUST call this class decorator at creation time
@magics_class
class MyMagics(Magics):

    @cell_magic
    def phylanx_cell(self,line,cell=None):
        tcell = 'def _cell_():\n  '+re.sub('\n','\n  ',cell)+'\n  return 1'
        tree = ast.parse(tcell)
        physl = PhySL()
        assert len(tree.body) == 1

        self.__physl_src__ = '%s(%s)\n' % (
            full_node_name(tree.body[0], 'block'), physl.recompile(tree))
        with iod:
            et.eval(self.__physl_src__)

ip = get_ipython()
ip.register_magics(MyMagics)
```