Parallel Cartographic Modeling Language : A 30 Minute Introduction to Spatial Data Processing in Python

Eric Shook
High-Performance Computing and GIS Lab
Department of Geography
Kent State University
eshook@kent.edu

KENT STATE UNIVERSITY

High-Performance Computing and GIS
HPCGIS Laboratory

What we will cover

Review Cartographic Modeling

Introduce Parallel Cartographic Modeling Language

Create our own Local Sum Operation

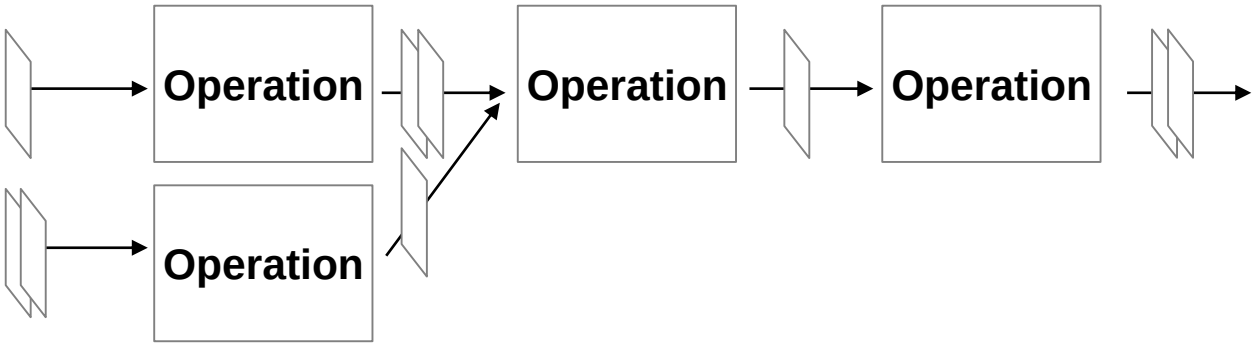
- You will join the ranks of awesomeness by becoming a parallel programmer

Explore spatial data-processing using PCML

Cartographic Modeling

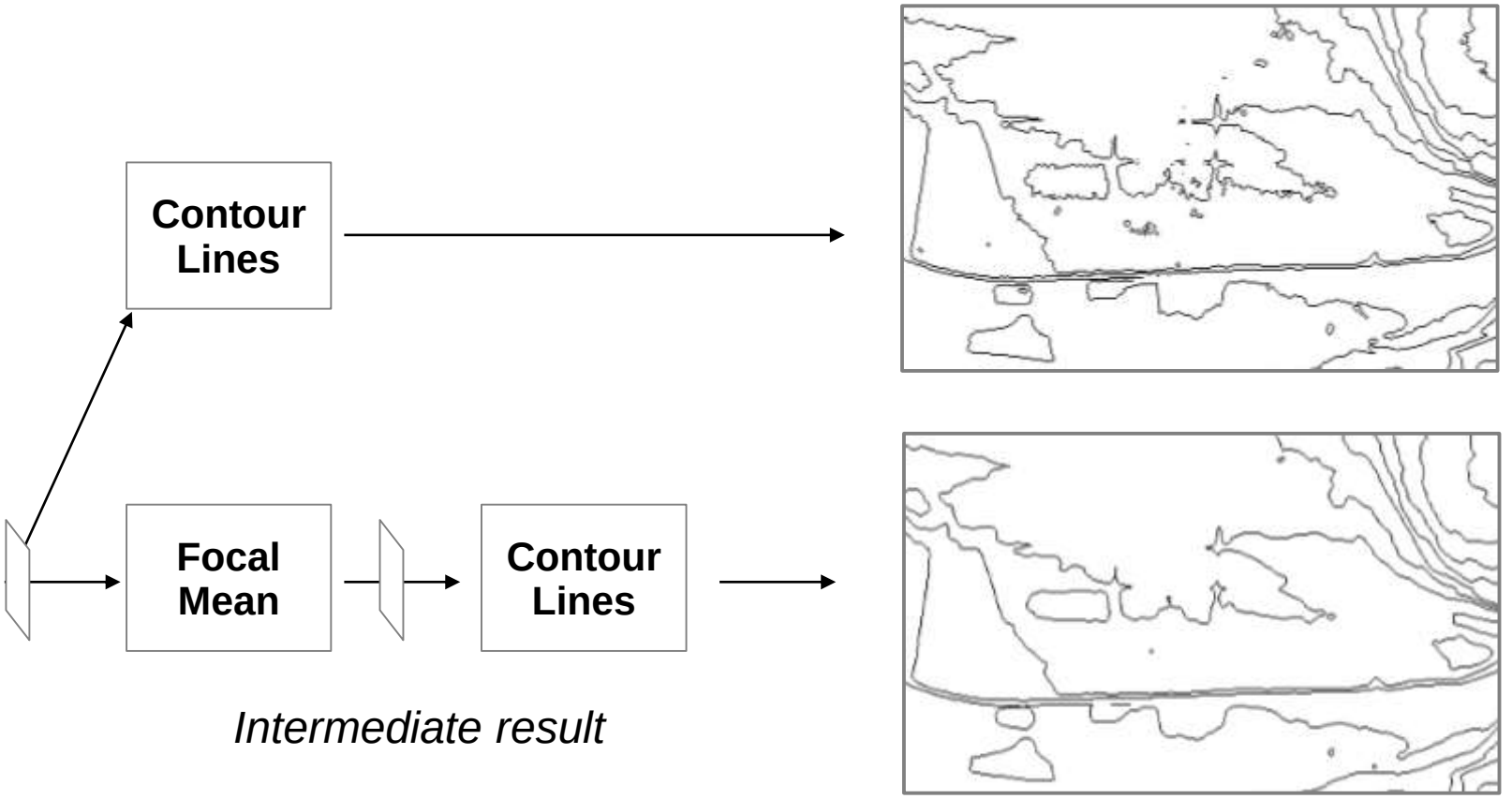
"Cartographic Modeling is a general methodology for the analysis and synthesis of geographical data" (Tomlin 1990)

- Clearly and consistently decomposes spatial data processing into elementary components
- Often expressed as a flowchart



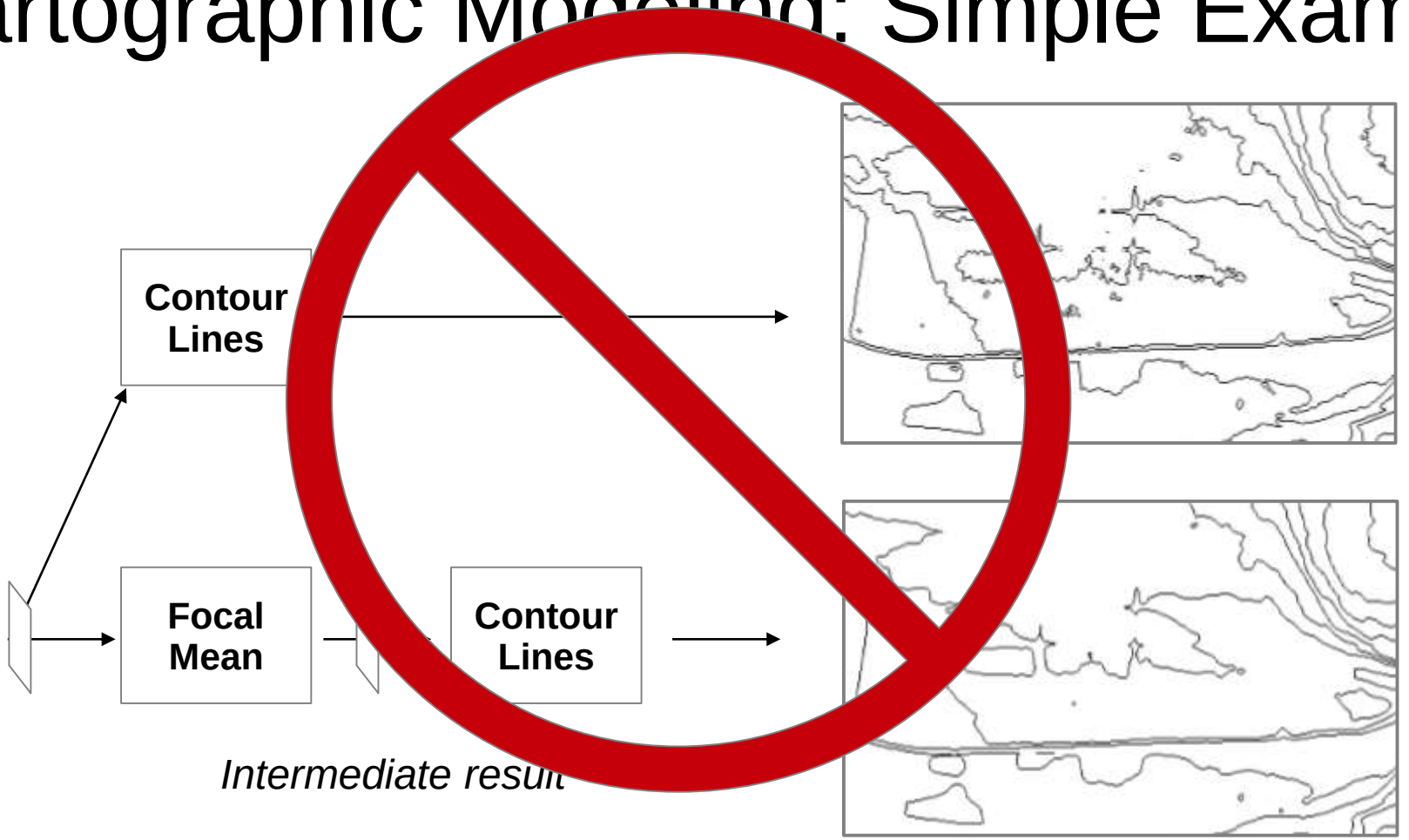
Joseph K. Berry (1987), C. Dana Tomlin (1990, 2012), Kemp (1993); Takeyama, and Couclelis (1997); Pullar (2001); Haklay (2004); Mennis, Viger, and Tomlin (2005); Mennis (2010); ...

Cartographic Modeling: Simple Example



Many operations exist from buffers to calculating hill slopes to reclassifications

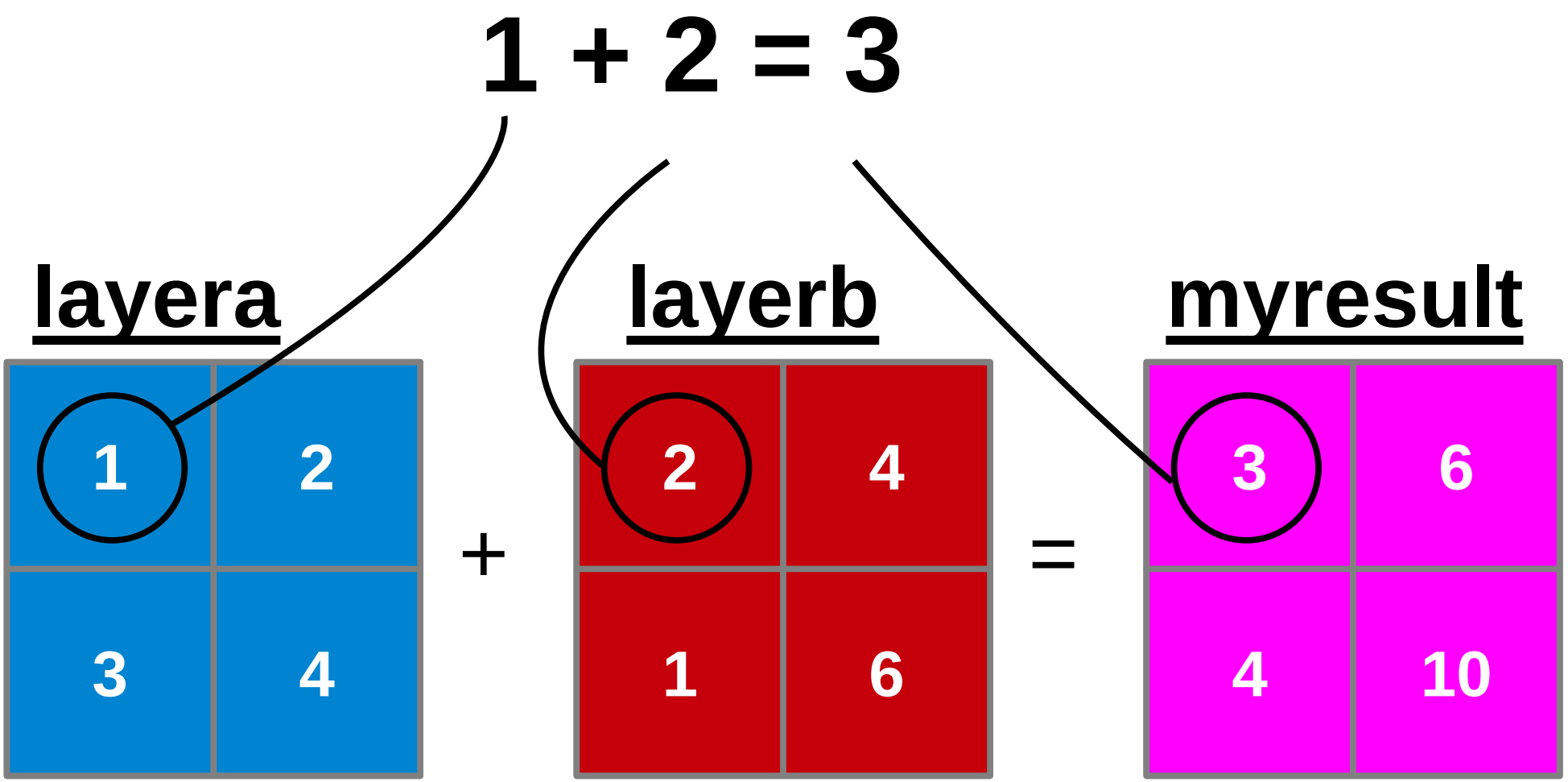
Cartographic Modeling: Simple Example



Rather than just connecting operations together

We will create our own operation and join the data-intensive computing party (yay!)

myLocalSum – What we are going to write



Our own Local Sum Operation

PCML

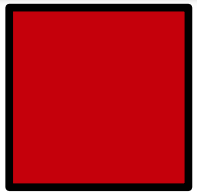
Parallel Cartographic Modeling Language (PCML) was created based on 3 design goals:

1. Usability *(Easy to use)*
2. Programmability *(Easy to develop)*
3. Scalability *(Able to handle big data)*

PCML supports automatic parallelization

Cartographic Modeling

Layer



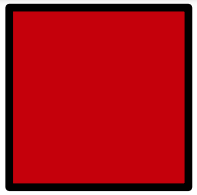
Operation

Location

Function

Cartographic Modeling

Layer



Operation

?

How to support parallel spatial data-processing?

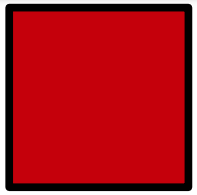
Location

2

Function

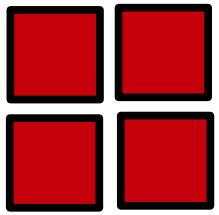
Subdomains

Layer

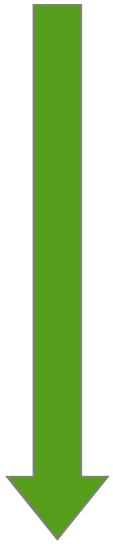


Operation

Subdomains



Executors

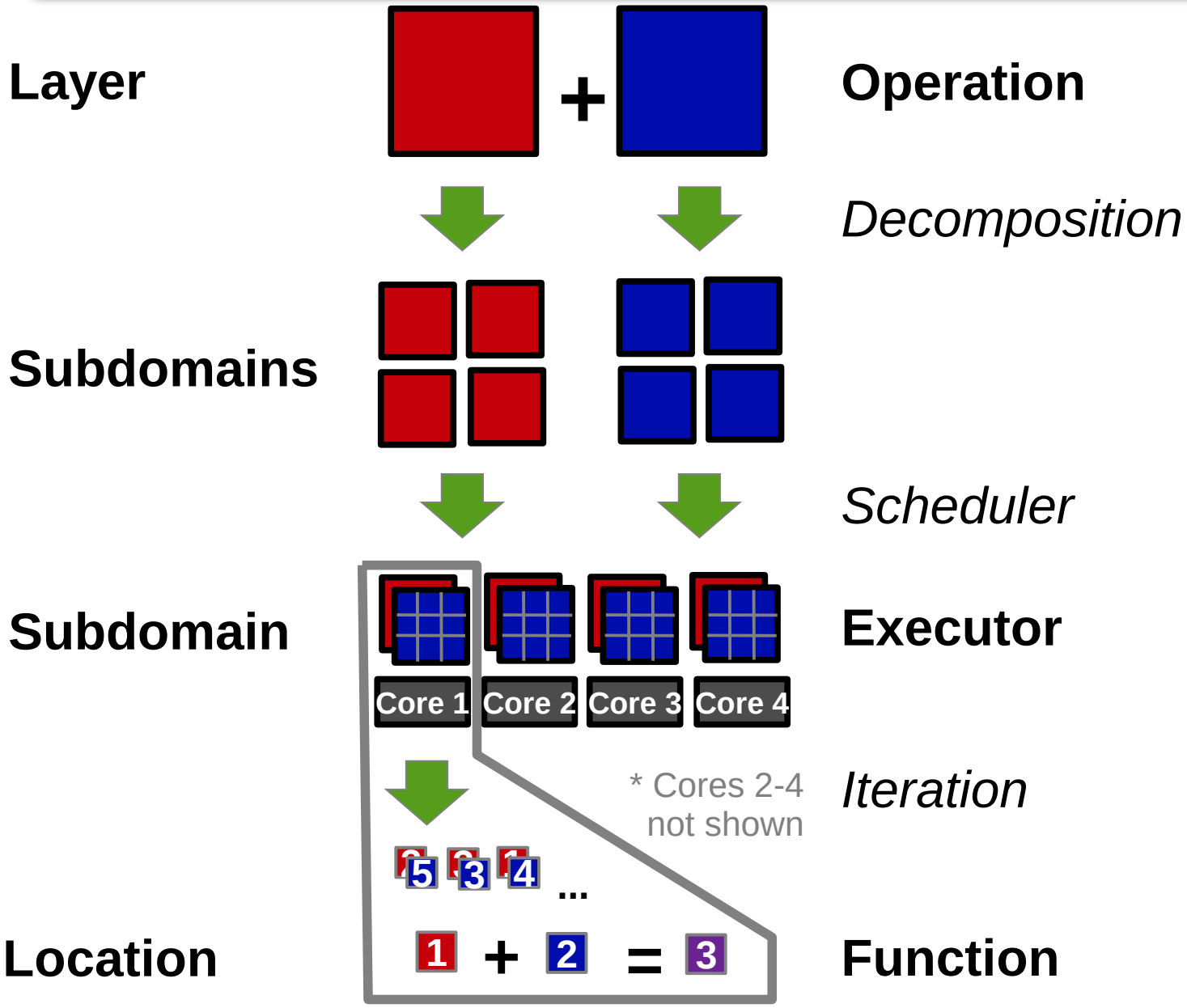


Subdomains are portions of a layer that can be processed simultaneously (or in parallel) in PCML

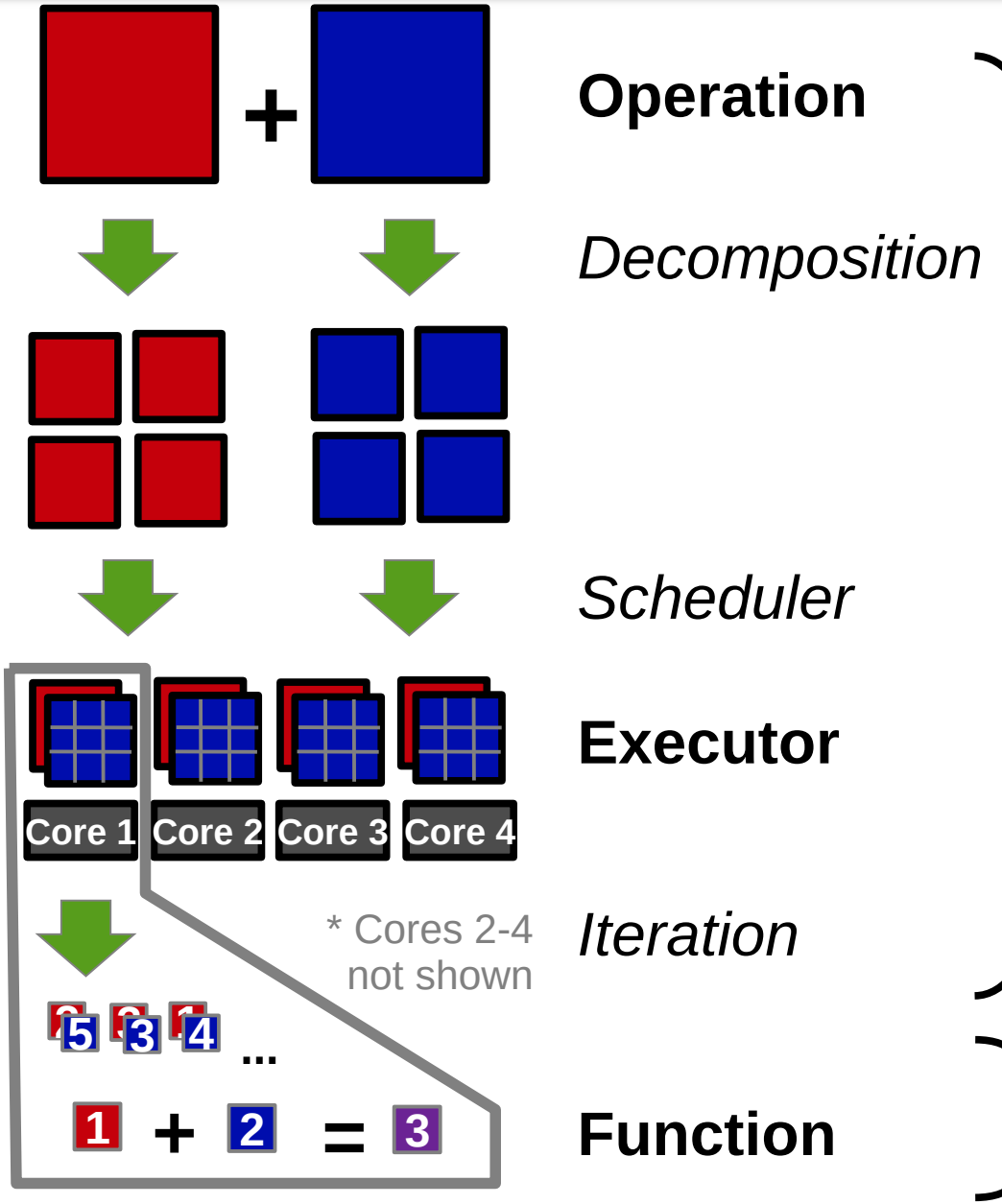
Location

Function

PCML Conceptual Design



PCML Conceptual Design

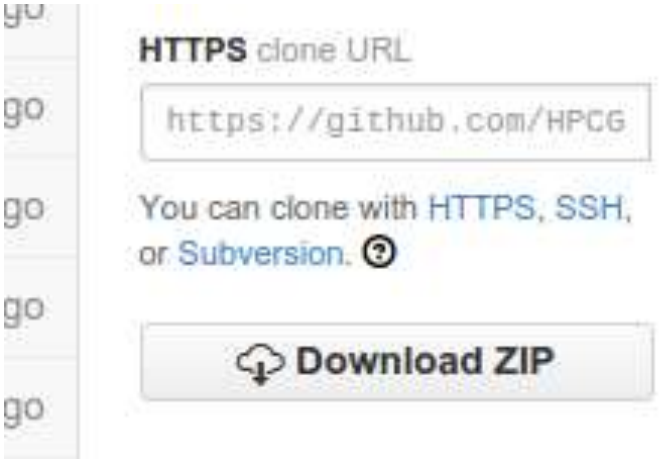


Step 0 – Download PCML

https://github.com/hpcgislab/pcml

- Click "Download ZIP"
- Unzip on your computer
- Open:

  demo_30min.py



Okay let's try it!

30 Minute Demo Overview (demo_30min.py)

Read 2 data layers (4 x 6 cell raster files)

Print the values out to the screen

Add them together

You will write a myLocalSum Operation

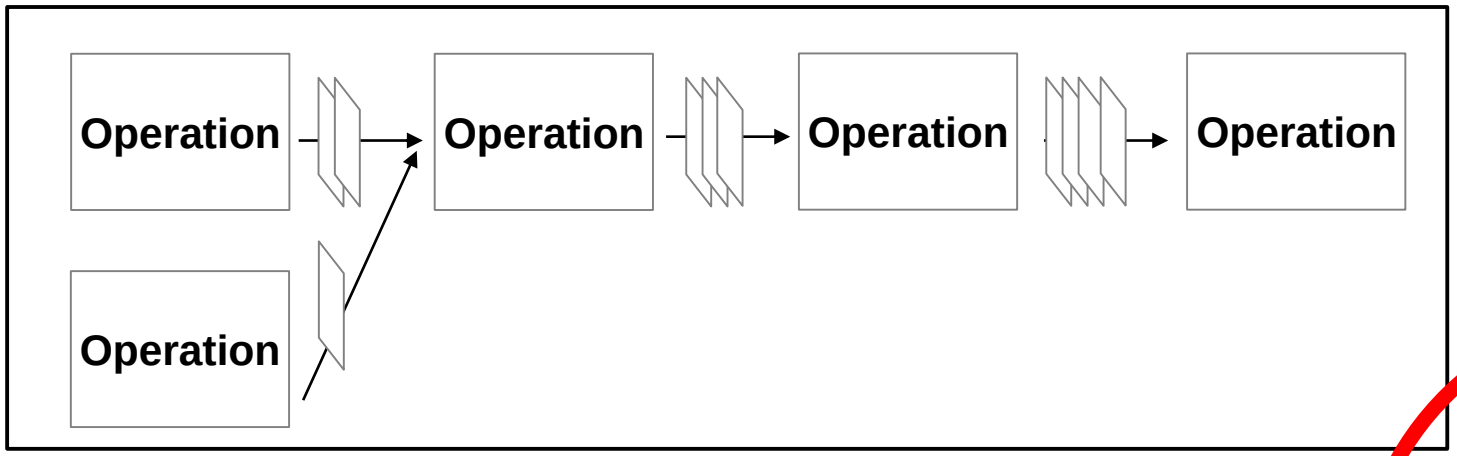
You will use your operation to sum them

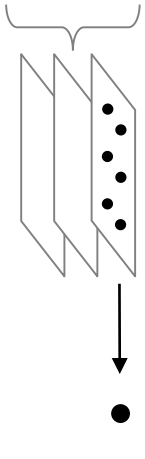
You will compare the results

You will then be an awesome PCML programmer!

Review: Cartographic Modeling

Procedure is a sequence of Operations





Layer

Location

Operation

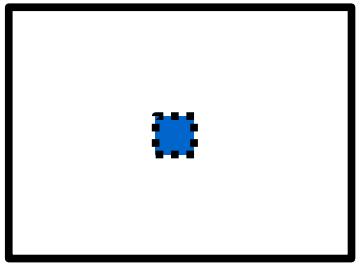
Function

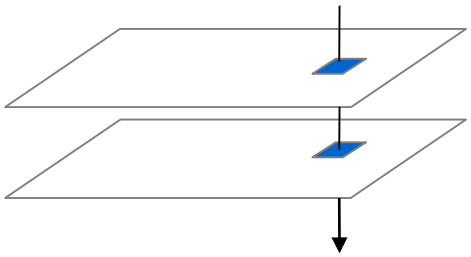
Classification:
Local
Focal
Zonal
Global

Joseph K. Berry (1987), C. Dana Tomlin (1990, 2012), Kemp (1993); Takeyama, and Couclelis (1997); Pullar (2001); Haklay (2004); Mennis, Viger, and Tomlin (2005); Mennis (2010); ...

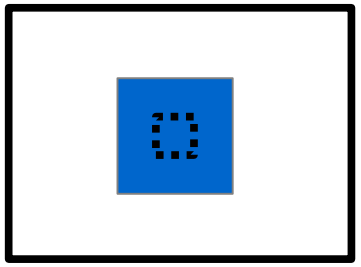
Local, Focal, Zonal, Global Operations

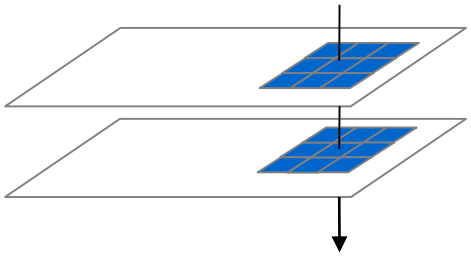
Local Operation



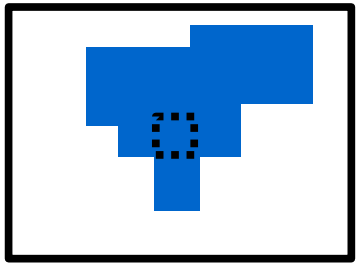


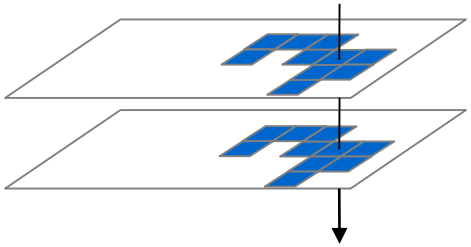
Focal Operation



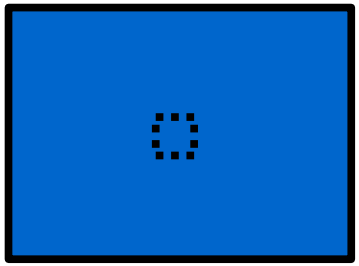


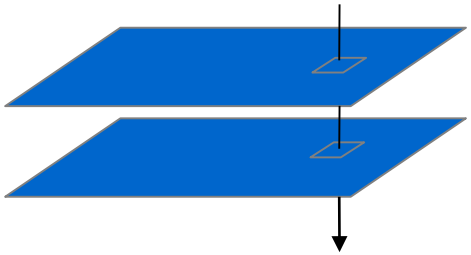
Zonal Operation





Global Operation





1a – Classify your Operation

```
####################################
#
# 1. In this section, you will create your own myLocalSum operation
#
####################################
```

@localoperation

Decorator

Decorators are easy to use. They allow us extend the functionality of Python. We will extend the functionality of a *local operation,* which is a built-in component of PCML, which provides automatic parallelization.

1b – Define your Function

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
#
# 1. In this section, you will create your own myLocalSum operation
#
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

@localoperation
def myLocalSum(self, locations, subdomains):

Function

A function is defined using the *def* keyword and can accept arguments and execute code statements. Statements within functions must be indented.

def function_name*(parameters):*
    Python code

1b – Define your Function

All PCML functions accept the same parameters:

self – (Python standard) To access Operation-level information
locations – List of locations to process using this function
subdomains – List of subdomains associated with the locations

```
@localoperation
def myLocalSum(self, locations, subdomains):
```

Function

A function is defined using the *def* keyword and can accept arguments and execute code statements. Statements within functions must be indented.

def function_name*(parameters):*
    Python code

1c – Create your location variables

```
####################################
#
# 1. In this section, you will create your own myLocalSum operation
#
####################################

@localoperation
def myLocalSum(self, locations, subdomains):
    sum = 0
```

Variables

Variables can by assigned values using the equal sign (just like in math class).

1c – Create your location variables

```
###################################
#
# 1. In this section, you will create your own myLocalSum operation
#
###################################

@localoperation
def myLocalSum(self, locations, subdomains):
    sum = 0
    for loc in locations:
```

For loops

This for loop will go over each location (loc) one by one.

Dictionary

```
adictionary = { 'x': 41.1, 'y': -81.3, 'v': 10 }

print adictionary['x']    # Print value for this key
                          # which would print 41.1
```

Dictionaries

Dictionaries store multiple elements
in a single data structure.

Elements consist of a **name**:
‘x’, ‘y’, or ‘v’ and a **value**: 41.1, -81.3, 10.

1d – Add each location value to sum

All PCML locations are dictionaries with 3 keys:

x – X coordinate
y – Y coordinate
v – Value at the location

```
@localoperation
def myLocalSum(self, locations, subdomains):
    sum = 0
    for loc in locations:
        sum = sum + loc['v']
```

Add up each value (‘v’) from all the locations (loc)

Python Statement Breakdown

sum = sum + loc[‘v’]

Value

Access the location’s value using the name **‘v’**. This is the standard way to access location values in PCML.

Python Statement Breakdown

sum = sum + loc['v']

Add Sum to Value

The sum variable holds the current sum.

Add the current sum and the location's value together

Python Statement Breakdown

sum = sum + loc[‘v’]

New Sum Variable

Now sum will be assigned the result from adding the current sum and the location’s value

1e – Return your result

```
####################################
#
# 1. In this section, you will create your own myLocalSum operation
#
####################################

@localoperation
def myLocalSum(self, locations, subdomains):
    sum = 0
    for loc in locations:
        sum = sum + loc['v']
    return sum
```

Function return

A function can return a value to the code that called it.

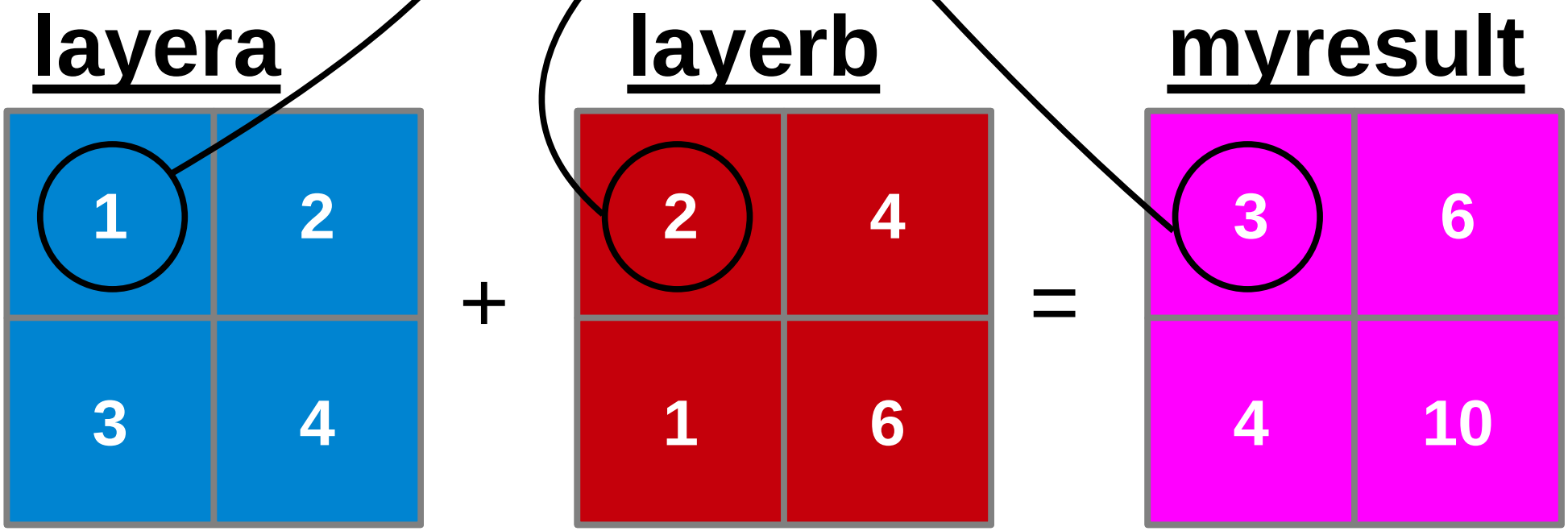
Awesomeness (Almost)

You did it!

```
@localoperation
def myLocalSum(self, locations, subdomains):
    sum = 0
    for loc in locations:
        sum = sum + loc['v']
    return sum
```

myLocalSum – What you wrote

loc['v'] **1 + 2 = 3** return sum



Your Local Sum Operation

Call your new Operation

```
#############################################################
#
# 2. In this section, call your myLocalSum operation
#     and save the results to a 'myresult' layer
#
#############################################################

myresult=myLocalSum(layera,layerb)
```

Indent 4 spaces to Align with comments

Next, we need to look for bugs

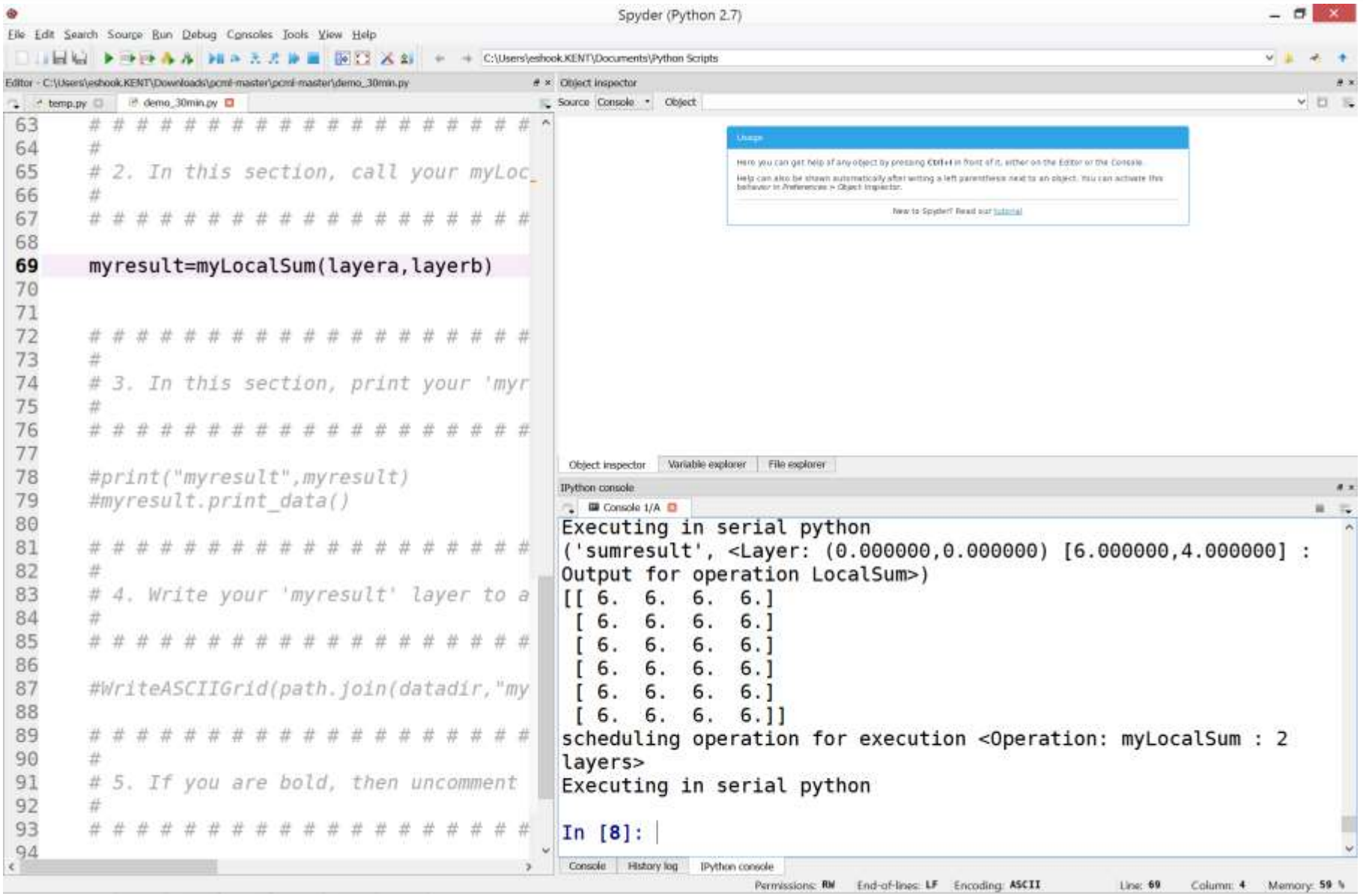
Not these bugs

These are bugs and errors that you may see

```
$ python demo_30min.py
  File "demo_30min.py", line 4

     ^
SyntaxError: invalid syntax
```

Run the Demo Program



Spyder editor shown (any editor will do)

Now Print and Write the Output

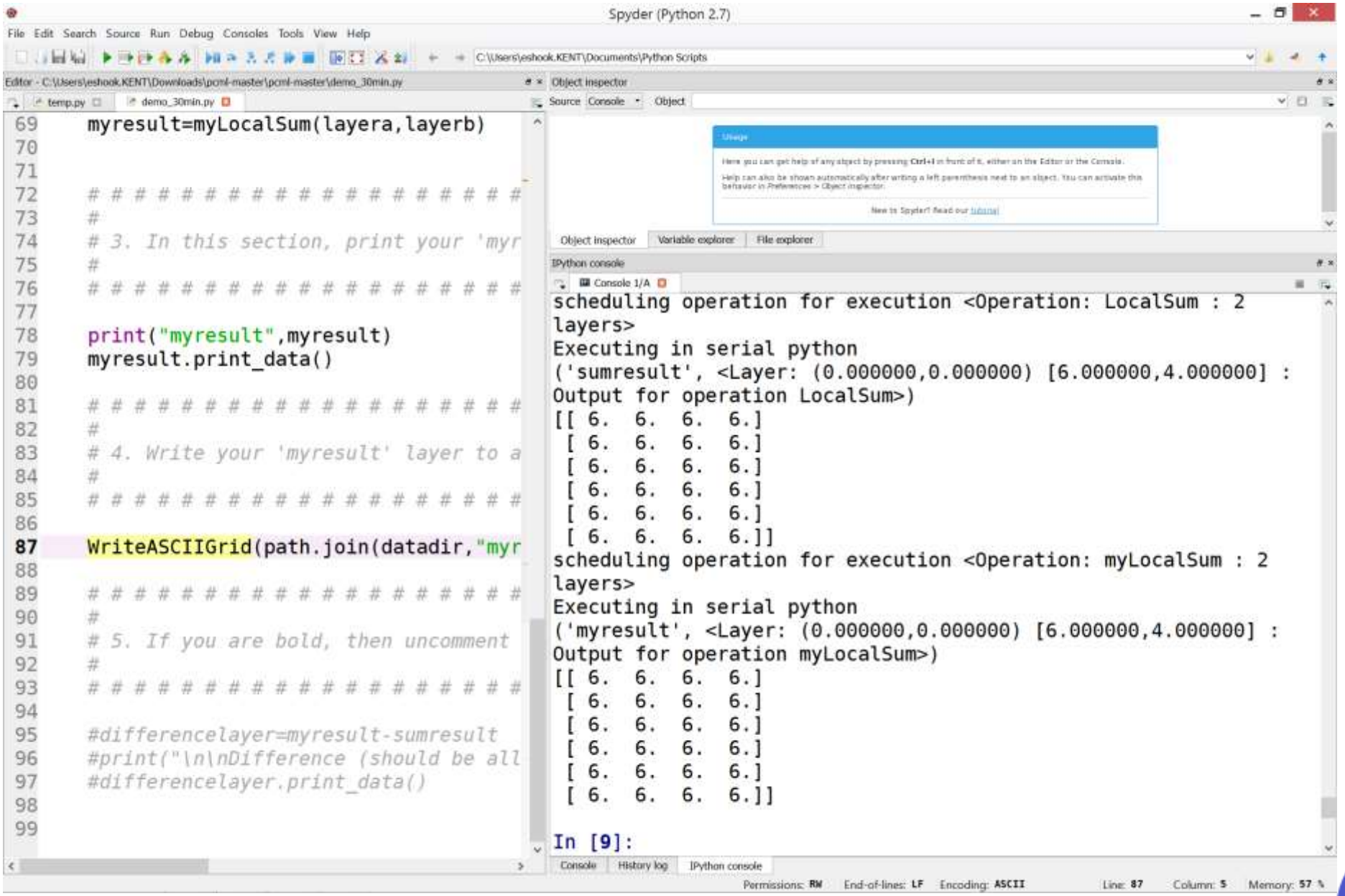
```
##########################################
# 3. In this section, print your 'myresult' layer
#
##########################################

print("myresult",myresult)
myresult.print_data()


##########################################
# 4. Write your 'myresult' layer to a file named 'myresult.asc'
#
##########################################

WriteASCIIGrid(path.join(datadir,"myresult.asc"), myresult)
```

Now Print and Write the Output

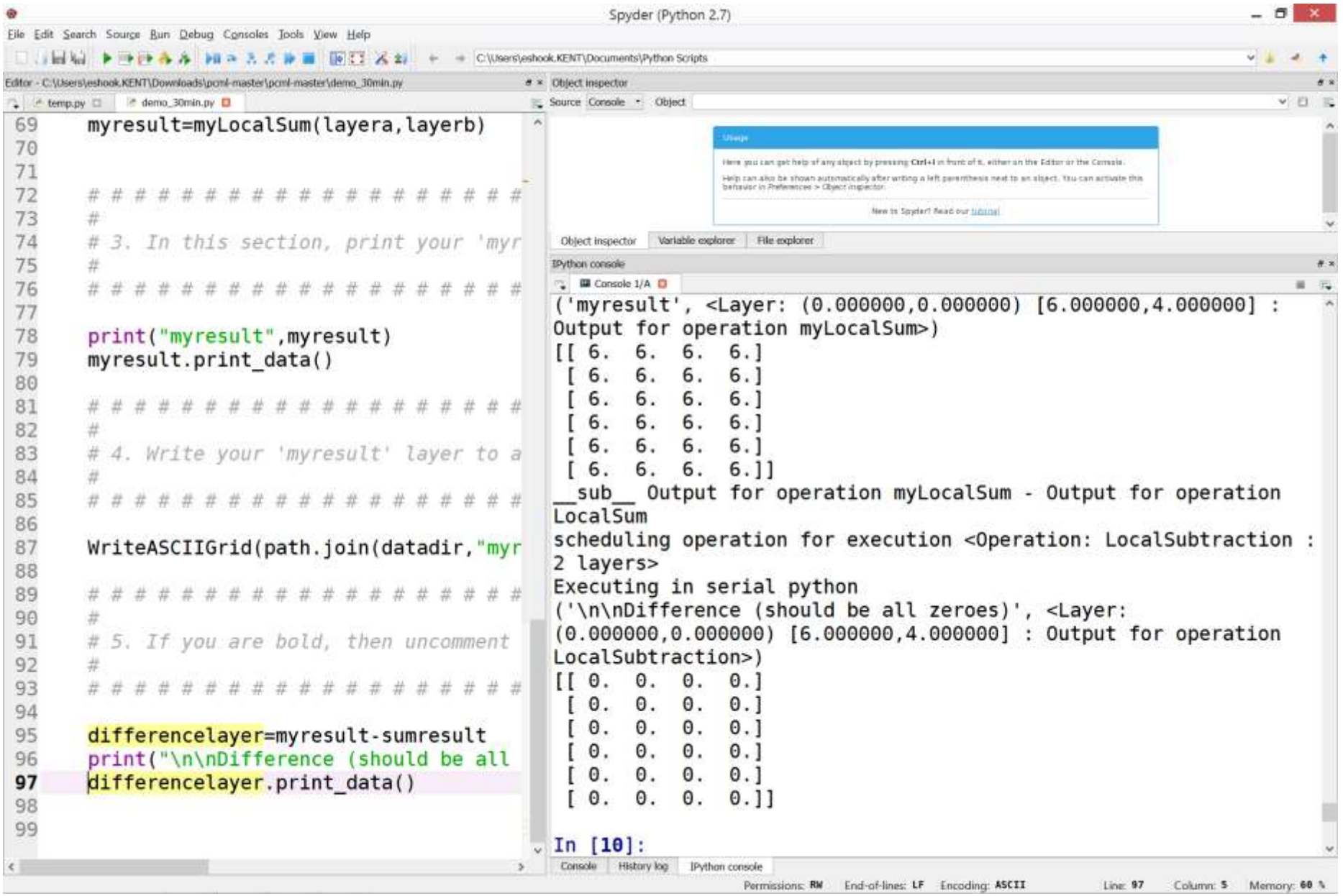


The Ultimate Test: The Difference

```
############################################
#
# 5. If you are bold, then uncomment the next three lines of code
#       to see if the difference is zero!
#
############################################

differencelayer=myresult-sumresult
print("\n\nDifference (should be all zeroes)",differencelayer)
differencelayer.print_data()
```

Success!


Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Tools View Help
C:\Users\eshook.KENT\Documents\Python Scripts
Editor - C:\Users\eshook.KENT\Downloads\pcml-master\pcml-master\demo_30min.py
temp.py
demo_30min.py
69 myresult=myLocalSum(layera,layerb)
70
71
72 # # # # # # # # # # # # # # # # # # #
73 #
74 # 3. In this section, print your 'myr
75 #
76 # # # # # # # # # # # # # # # # # # #
77
78 print("myresult",myresult)
79 myresult.print_data()
80
81 # # # # # # # # # # # # # # # # # # #
82 #
83 # 4. Write your 'myresult' layer to a
84 #
85 # # # # # # # # # # # # # # # # # # #
86
87 WriteASCIIGrid(path.join(datadir,"myr
88
89 # # # # # # # # # # # # # # # # # # #
90 #
91 # 5. If you are bold, then uncomment
92 #
93 # # # # # # # # # # # # # # # # # # #
94
95 differencelayer=myresult-sumresult
96 print("\n\nDifference (should be all
97 differencelayer.print_data()
98
99
Object inspector
Source Console Object
Object inspector Variable explorer File explorer
IPython console
Console 1/A
('myresult', <Layer: (0.000000,0.000000) [6.000000,4.000000] :
Output for operation myLocalSum>)
[[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]]
__sub__ Output for operation myLocalSum - Output for operation
LocalSum
scheduling operation for execution <Operation: LocalSubtraction :
2 layers>
Executing in serial python
('\n\nDifference (should be all zeroes)', <Layer:
(0.000000,0.000000) [6.000000,4.000000] : Output for operation
LocalSubtraction>)
[[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]]
In [10]:
Console History log IPython console
Permissions: RW End-of-lines: LF Encoding: ASCII Line: 97 Column: 5 Memory: 60 %

Other Things to Try

Change the Iteration strategy *

LocalSum(layera,layerb,iteration=columnmajoriteration)

Change the Decomposition strategy *

LocalSum(layera,layerb,decomposition=columndecomposition)

Change the Executor Type from Parallel to Serial

PCMLConfig.exectype = ExecutorType.serialpython

Change the number of cores

PCMLConfig.num_procs = 1

Use a different operation

Take a look at the library of operations (in **pcml/lib/**)

* add **print(loc)** in the for loop to see the changes in Iteration and Decomposition

Welcome to the (awesome) group of PCML developers!

* PCML is an open-source project and we welcome your contributions

Acknowledgements

A huge thanks to (the growing number of) PCML contributors:

Michael Hodgson

Shaowen Wang

Babak Behzad

Kiumars Soltani

April Hiscox

Zhengliang Feng

Emil Shirima

Jayakrishnan Ajayakumar

Sandeep Vutla

Gowtham Kukkadapu

Suman Jindam

This work was in-part funded by the CyberGIS Fellows program

This work used the Kent State University Arts and Sciences Cluster

Thank you!

Questions?

Please contact
Eric Shook
eshook@gmail.com