Comparing Map Algebra Implementations for Python: Rasterio and ArcPy

Mary Marek-Spartz
Department of Resource Analysis, Saint Mary's University of Minnesota, Minneapolis, MN 55404

Keywords: Map Algebra, ArcPy, Rasterio, Python, Raster Processing, Geographic Information Systems (GIS), Free and Open Source Software (FOSS), Environmental Systems Research Institute (ESRI)

Abstract

As Geographic Information Systems (GIS) expand, tools for spatial analysis and raster processing are in high demand. Open source solutions for GIS can provide users with low-cost, generic, and interoperable alternatives to proprietary software. Map algebra is uniquely situated to benefit from open source implementations. This study compares map algebra tools of the proprietary ESRI ArcPy library and the open source Rasterio library. The analysis assesses performance of both libraries in terms of time and memory usage. Based on these performance metrics, Rasterio should be considered a suitable alternative to ArcPy for some GIS workloads.

Introduction

Advancements in satellite imagery and remote sensing technology have given rise to large raster datasets that are increasingly accessible to GIS users (Clewley, Bunting, Shepherd, Gillingham, Flood, Dymond, and Moghaddam, 2014). As GIS datasets expand, tools for spatial analysis and raster processing are in high demand. Proprietary desktop software has long been the standard in GIS analysis. Though these systems offer powerful analysis tools and algorithms, they are not immune to programming, processing, and licensing limitations. As a result of rapidly changing spatial analysis requirements, GIS programs need to be adaptive and transparent. Software must evolve with the field to produce intuitive and dynamic applications that accommodate modern GIS workloads.

Free and Open Source Software (FOSS) for GIS can provide users with low-cost, generic, and interoperable alternatives to proprietary software. Open source GIS software is multiplying due to the Internet and increasingly code-literate users, meanwhile the need for highly extensible GIS tools has led to an increase in the amount of GIS software and libraries being developed under open source licenses (Steiniger and Bocher, 2008). As the open source movement grows, it is important to examine the implications for proprietary software. How does an open source GIS solution compare to proprietary analysis tools in terms of performance? What are the relative advantages and disadvantages of choosing an open source solution?

This study provides a comparative analysis of two GIS libraries: Environmental Systems Research Institute (ESRI) ArcPy and Rasterio. ArcPy is a proprietary, closed

Marek-Spartz, Mary. 2015. Comparing Map Algebra Implementations for Python: Rasterio and ArcPy. Volume 18, Papers in Resource Analysis. 14pp. Saint Mary's University of Minnesota Central Services Press. Winona, MN. Retrieved (date) from http://gis.smumn.edu

source, general-purpose GIS library. Rasterio is an open source raster-processing library. This study examines the quantitative performance metrics of the map algebra tools provided by ArcPy and Rasterio. Tests were implemented in each library to compare the time and memory usage required to complete a variety of map algebra operations.

Background

Free and Open Source Software (FOSS)

FOSS is a transparent model of software development where source code is freely shared and can be read, modified, and redistributed (Deek and McHugh, 2008). This free exchange allows for collaboration among programmers and researchers and aides in the advancement of software (Deek and McHugh). This is in contrast to proprietary software development in which the creation of the source code is privatized and licensing restrictions usually forbid redistribution or modification of code (Deek and McHugh).

Deek and McHugh (2008) cite several major advantages of open source models over proprietary systems. Open software often has the following characteristics:

- Lower direct costs as modifications are usually distributed among developers via the Internet
- High portability and lower resource utilization
- More eyes on the code for spotting bugs, increasing reliability and security
- Fast-paced development and free updates for new software

These advantages of the open source model can provide benefits to the diverse community of GIS users, so it is not surprising there has been an increase in FOSS GIS software downloads in recent years (Steiniger and Bocher, 2008). However, there is still significant demand for specialized proprietary GIS software (Donnelly, 2010). ESRI products, like ArcGIS, corner a large portion of the GIS market with 350,000 clients worldwide (ESRI, 2015). Due to the sophistication and longevity of ArcGIS tools, it is often used as a standard for new GIS analysis software (Donnelly, 2010).

Map Algebra

Map algebra accounts for much of the raster analysis capabilities of GIS software (Mennis, 2010). Map algebra is a form of raster analysis that uses operators to combine and create raster layers (Bruns and Egenhofer, 1997). Map algebra implementations consist of a set of functions that can take one or more raster datasets and apply arithmetical and logical operators to output a modified raster grid. Figure 1 shows two examples of map algebra operations. Operators such as these are common in GIS analysis. For example, the units of a raster grid can be converted using a formula, such as changing the values of a raster from degrees Fahrenheit to degrees Celsius.

Boolean rasters can also be used for analysis. Boolean rasters store two values, *True* and *False*, or *one* and *zero*, respectively. They help visualize where an element exists and where it does not. Figure 2 shows an example of a map algebra operation that outputs a Boolean raster with values equal to one where the slope is greater than ten.

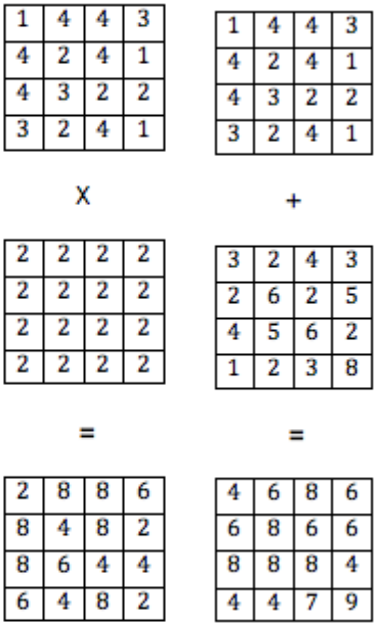

Figure 1. Multiplying by a constant grid (left) and adding two grids (right).

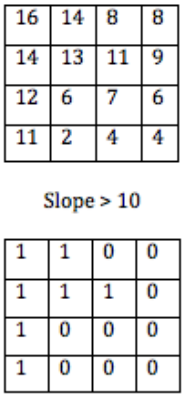

Figure 2. A Boolean grid where areas that are *True* (=1) have a slope greater than 10 and all other cells are *False* (=0).

There is also a set of operators to combine Boolean rasters. These include *and* (&), *or* (|), *not* (~), and *exclusive or* (^). For example, it may be necessary to find the areas where two habitats overlap. This would involve using a '&' operator on two Boolean rasters to determine where the values are *True* for both rasters and essentially multiplies the two rasters together. Figure 3 shows an example of this type of operator.

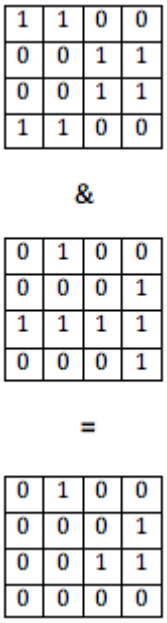

Figure 3. A Boolean operation.where areas that are shared between both grids are *True* (=1).

The cells that are equal to one show where the two habitat areas overlap.

As illustrated, map algebra is a simple yet powerful way of analyzing raster matrices and provides many solutions to GIS problems. Map algebra is not exclusive to a single domain; it is a common GIS task that can benefit from an open source implementation (Mennis, 2010). Research in map algebra thrives under environments where new implementations can build upon existing algorithms (Mennis). The increasing amount of large spatio-temporal datasets available for GIS creates a demand for non-domain specific, intuitive, and interoperable map algebra processing tools. Consequently, Map algebra is uniquely situated to benefit from the advantages of FOSS GIS tools (Mennis). Understanding these advantages guides development of raster processing tools and warrants research that analyzes functionality and

performance with respect to existing proprietary software packages that dominate the field of raster analysis (Donnelley, 2010).

Libraries

ArcPy

ESRI's ArcGIS exposes many of its tools to Python through the ArcPy library, including map algebra operations. ArcPy contains operators for implementing map algebra functions through the ArcGIS Spatial Analyst extension (ESRI, 2015). ArcPy is a Python package that is included with ArcGIS proprietary software (ESRI, 2015). Because the source code is closed, it is hard to know what ArcPy is built upon, but it requires NumPy. NumPy is an open source Python data analysis library for performing matrix operations (Van Der Walt, Colbert, and Varoquaux, 2011). The NumPy package provides an array data structure that stores elements of fixed size to allow for efficient numerical computing (Van Der Walt *et al.*, 2011). This makes NumPy arrays ideal for calculating raster surfaces for map algebra implementations.

Rasterio

Rasterio is an open source library developed by Mapbox, Inc. for reading and writing geospatial raster datasets in Python (Mapbox, 2015). Rasterio provides a NumPy interface to GDAL rasters. The NumPy array can be modified and subsequently exported using GDAL (Mapbox). GDAL, or the Geospatial Data Abstraction Library, is a library used for the conversion and modification of a variety of geospatial data formats (gdal.org, 2015). GDAL is licensed by the Open Source Geospatial Foundation and can be used to parse and process raster data in a variety of formats including ArcInfo GRID files (gdal.org).

Methods

Implementation of Python Map Algebra Scripts

Rasterio and ArcPy were installed on a Dell Inspiron 3000 Series i3847 Desktop with a 3.2 GHz Intel Core i5-4460 Processor and 8 GB RAM, running Windows 7. ArcGIS for Desktop 10.3 was installed, including ArcPy, Python 2.7.8 (32-bit), and NumPy 1.7.1. To minimize differences between the Rasterio and ArcPy environment, Rasterio was installed against the Python installation included with ArcGIS. However, Rasterio relies on a more recent version of NumPy (version 1.9.2) than ArcPy, so NumPy was upgraded and downgraded when switching between libraries.

Equivalent Python scripts were written for ArcPy and Rasterio using the same raster dataset. The raster dataset used for all tests was a TIFF sample file provided by Rasterio (tests/data/ RGB.byte.tif from Mapbox, 2015). The functions written for both the Rasterio and ArcPy implementation were as follows:

1. Open and read a raster file. (reading_one_raster.py)
2. Write a raster to a file. (writing_one_raster.py)
3. Add by a constant. (adding_one_raster_by_a_ constant.py)

4. Subtract by a constant. (subtracting_one_raster_by_a_constant.py)
5. Multiply by a constant. (multiplying_one_raster_by_a_constant.py)
6. Divide by a constant. (dividing_one_raster_by_a_constant.py)
7. A formula involving multiple constants. (converting_celsius_to_fahrenheit_one_raster.py)
8. Add two rasters. (adding_two_rasters.py)
9. Subtract two rasters. (subtracting_two_rasters.py)
10. Multiply two rasters. (multiplying_two_rasters.py)
11. Divide one raster by another. (dividing_two_rasters.py)
12. A formula involving multiple rasters. (two_raster_formula.py)
13. Convert to a Boolean raster. (converting_to_a_boolean_one_raster.py)
14. Use the '*not'* Boolean operator. (boolean_not_one_raster.py)
15. Use the '*and'* Boolean operator. (booleanAND.py)
16. Use the '*or'* Boolean operator. (booleanOR.py)
17. Use a combination of Boolean operators. (boolean_formula.py)

Each program was scripted as simply and as similar as possible between both libraries. The same raster files were used for both implementations. The Rasterio program for converting raster values from Celsius to Fahrenheit is as follows:

```
import rasterio
# ---
@profile
def benchmark():
    with rasterio.open('R.byte.tif') as src:
        arr = src.read()

    arr = (arr * 9) / 5 + 32
    with rasterio.open('output.tif', 'w',
**src.meta) as dst:
        dst.write_band(1, arr[0])

benchmark()
```

The ArcPy script for the same program is similar:

```
import arcpy
from arcpy.sa import ApplyEnvironment

arcpy.CheckOutExtension("Spatial")
arcpy.env.overwriteOutput=True

# ---

@profile
def benchmark():
    src = arcpy.Raster('R.byte.tif')
    arcpy.env.snapRaster = src
    dst = ((src * 9) / 5 + 32) % 255
    ApplyEnvironment(dst).save('output.tif')

benchmark()
```

The '#---' acts as a separator between the setup and the statement. The setup portion of the code exists above '#---' and includes imports and environment settings. The code below the '#---' is the map algebra function, or the part of the code that constitutes the benchmark statement.

Benchmarks

When considering software performance, efficiency is defined as the amount of time and resources used to complete a task (Wagner and Scott, 1995). ArcPy and Rasterio scripts were tested for performance by assessing their execution time and memory usage.

Execution Time

Execution time is the amount of time it takes for a block of code to run. The program that runs the benchmarks loaded the files as strings and split them into setup and benchmark statements. A helper function was written to analyze the time performance of each map algebra function for both ArcPy and Rasterio. This function takes three

arguments, a statement block (for the ArcPy and Rasterio benchmark), a setup block (for imports needed by the benchmarks), and whether to enable or disable garbage collection.

Garbage collection is used by some programming languages to de-allocate unused memory. While this keeps memory usage from growing indefinitely, it can occasionally pause execution of the program, which can affect time performance. For this reason, the benchmark programs were evaluated both with garbage collection on and garbage collection disabled.

The helper function used the 'timeit' Python module to time the execution of the benchmark. To minimize the impact of the import time, and instead emphasizing actual processing time, timeit can be parameterized to execute the statement multiple times per setup (Python Software Foundation, 2014). This study emphasizes running the statement rather than the setup because the statement performance is more important for the majority of users. The helper function tells timeit to run the benchmark forty times per setup block (n=40). Timeit returns the result of executing the setup and statements as a single time in seconds. To get the average execution time of the benchmark, this was averaged, and converted to milliseconds for easier understanding. Timeit was also configured to repeat both the setup and the statements (n=40), to returns a list of such times. In total, the setup blocks were ran 40 times, and the benchmark statements for each program were ran 1,600 times, (40 per setup block execution).

```
def milliseconds(t):
    return t * 1000

def profile(stmt='pass', setup='pass',
            run_gc=False):
    n = 40
    setup = 'gc.collect();' + setup
    if run_gc:
        setup = 'gc.enable();' + setup
    t = timeit.Timer(stmt, setup)
    return [
         milliseconds(x / n)
         for x
         in t.repeat(repeat=n, number=n)
    ]
```

Memory

The same seventeen benchmarks were analyzed using 'memory_profiler', a Python package for such analysis. It outputs a report with line-by-line memory consumption, including both total memory consumed and incremental (how much memory was consumed relative to the previous line of execution).

Results

Rasterio used less time for all seventeen map algebra tasks. The average execution time for the ArcPy scripts was observed to be significantly slower than Rasterio using a pairwise Wilcoxon signed rank test ($p < .0001$). Figure 4 compares the averaged execution time (in milliseconds) of all of seventeen map algebra operations combined for each implementation. Garbage collection did not make a discernable difference for either ArcPy or Rasterio aside from having a few more outliers for the cycles when garbage collection ran.

The average execution time for Rasterio was overall much shorter than ArcPy with the exception of reading one raster file, which was similar for both libraries. In addition to faster execution times, the Rasterio programs remained consistent in their time usage with little variation about the mean. ArcPy's programs varied more with some tasks taking much longer than others, namely, the Boolean operations (Appendix A).

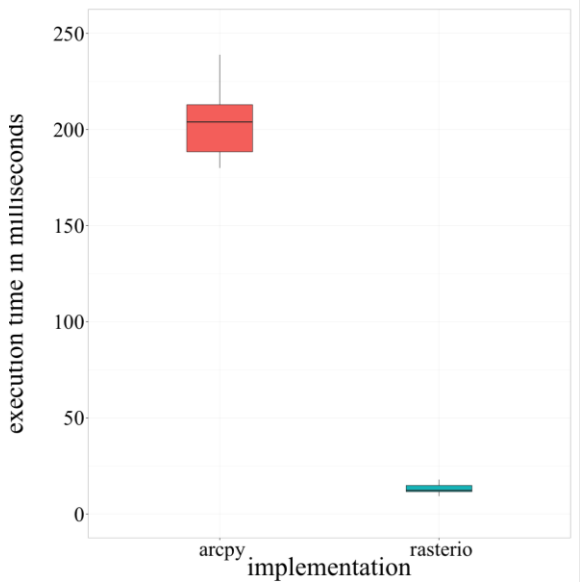

Figure 4. Summary statistics for average execution time of ArcPy (left) and Rasterio (right) aggregating all seventeen map algebra functions for each implementation.

Figure 5 is a visualization of every execution time report generated by each implementation. Each point on the scatter plot represents one trial.

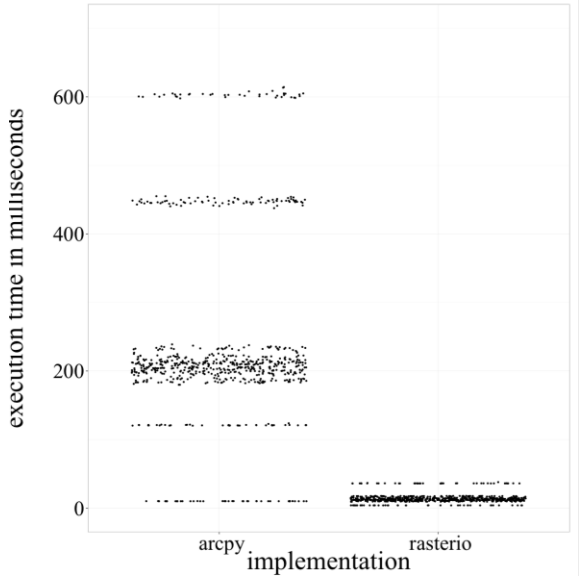

Figure 5. Execution times of all trials for ArcPy (left) and Rasterio (right).

There were 40 trials for each of the seventeen map algebra programs for both ArcPy and Rasterio, therefore, there are 680 points plotted for ArcPy and 680 points plotted for Rasterio.

The areas where the points are most dense roughly represent the average execution time for the implementation. ArcPy shows a broader range of execution times with some programs' trials falling well above 400 and even 600 milliseconds. The only ArcPy program that was comparable to Rasterio (reading a raster file) is visible as a cluster of 40 points at around 10 milliseconds. Figure 5 shows Rasterio trials were not only faster than those of ArcPy, but they were more consistent as well.

Additionally, Rasterio used considerably less memory than ArcPy for all seventeen of the map algebra tasks (Appendix B). For the programs that were tested, Rasterio appears to use roughly one quarter of the memory that ArcPy used. The incremental memory usage of ArcPy was higher for every task, except reading one raster file. ArcPy may be using a software design feature known as lazy loading where the program waits to load the image until the data is needed for an operation. However, without access to the code, this cannot be confirmed.

Discussion

An open source GIS solution may be preferred over proprietary software for certain tasks. Although ArcPy offers a greater breadth of GIS analysis tools, it is not necessarily the most powerful or efficient analysis option. This study suggests Rasterio is superior in execution time and memory usage for map algebra. There were also additional advantages to Rasterio that were encountered during the implementation.

Originally, a three-band imagery raster was being used in testing. ArcPy was not able to process the three bands without additional lines of code, but this would make the ArcPy scripts use different logic than the Rasterio scripts. The three-band raster was modified to a single-band raster in order to make the scripts for both libraries comparable, but it is important to note Rasterio was able to operate on three bands without additional clarification. Additionally, Rasterio supports multiple platforms and was readily installed for Mac OS X, which is advantageous for work environments with a heterogeneous IT infrastructure.

Does it Scale?

An important advantage of Rasterio and many open source technologies is their ability to scale costs efficiently. This is especially important for large raster processing tasks. Many of the algorithms used to implement map algebra are embarrassingly parallel, that is, the workload can easily be distributed into multiple processing tasks (Wagner and Scott, 1995). The large workloads associated with GIS data often encounter performance problems and hardware limitations (Wagner and Scott, 1995). Large-scale raster processing implementations are optimized when the work can be distributed among several computers (cloud computing or parallel computing) (Clewley *et al*., 2014). This type of computing is becoming essential as GIS systems progress and multidimensional datasets expand (Wagner and Scott, 1995). Proprietary software can make parallelization difficult and expensive, therefore, modular, open source programs are well suited for these processing tasks (Clewley *et al*., 2014).

Implication for ArcGIS

Proprietary software, like ArcGIS, has much to gain from examining open source GIS platforms such as Rasterio. ArcGIS already utilizes many open source technologies including Python and NumPy. The future of ArcGIS software will depend on its ability to recognize the changing needs of GIS users. Adopting transparent and interoperable practices of open source technologies may help advance ArcGIS tools through collaboration and scalability. There is evidence ESRI is incorporating these concepts into their products. ArcGIS Pro is a recent addition to ArcGIS for Desktop that offers a modular application for viewing and analyzing ArcGIS content (ESRI, 2015). It features a simple user interface with a built-in Python console for interactive programming. It also encourages sharing of content within ArcGIS organizations. ArcGIS Pro depends on ArcGIS for Desktop for processing. However, the architecture of the applications suggests a less monolithic approach to GIS tools from ESRI going forward.

Why Open Source?

The success of Rasterio and other open source GIS libraries will depend on continued contributions from the community of open source developers. Mapbox is gaining popularity among GIS users and will likely continue to support its products. Rasterio is increasing functionality and has already improved many of its features during this study. Many will still prefer the

wide array of analysis options that ArcGIS for Desktop offers. However, when choosing analysis tools, GIS professionals should take a critical look at the requirements of their workload and be parsimonious about their selections.

It is important to understand that open source and proprietary approaches to software development lead to different design philosophies. Proprietary software development often leads to monolithic, large, cohesive programs that do a wide array of tasks. In contrast, open source software development typically leads to programs that are modular, interoperable, and designed for a specific task. Often, focusing on a specific task leads to higher quality. Extensibility and optimum performance are goals of many open source implementations. Both approaches have their advantages and disadvantages. Projects requiring scalable and portable tools benefit from open source solutions.

Rasterio is specifically used for raster analysis, but due to the diverse community of open-source contributors, there are a wide variety of tools available for most GIS tasks. For example, Fiona is an open source library that is similar to Rasterio, but built for processing vector data (including shapefiles) (Python Software Foundation, 2014).

This study debunks some of the myths surrounding open source GIS applications; performance, quality, user support, and ease of use can be just as good with open source software as with proprietary software. Depending on the task, an open source solution can save time and money and should be considered as an alternative to proprietary software.

Acknowledgments

I would like to thank John Ebert and the Department of Resource Analysis at Saint Mary's University of Minnesota for their help with this research and Kyle Marek-Spartz for his support and advice throughout this project.

References

Bruns, H. T., and Egenhofer, M. J. 1997. User interfaces for map algebra. *URISA-Washington DC*, *9*, 44-55.

Clewley, D., Bunting, P., Shepherd, J., Gillingham, S., Flood, N., Dymond, J., and Moghaddam, M. 2014. A Python-Based Open Source System for Geographic Object-Based Image Analysis (GEOBIA) Utilizing Raster Attribute Tables. *Remote Sensing*, *6*(7), 6111-6135. doi:10.3390/rs6076111

Deek, F. P., and McHugh, J. A. 2008. *Open source: technology and policy* (pp. 1-18). New York: Cambridge University Press.

Donnelly, F. P. 2010. Evaluating open source GIS for libraries. *Library Hi Tech*, *28*(1), 131-151.

ESRI. 2015. ArcGIS Platform. Retrieved February 10, 2015 from http://www. esri.com/ software/arcgis/.

Geospatial Data Abstraction Library (GDAL). Retrieved February 8, 2015 from www.gdal.org.

Mapbox. 2015. Rasterio. Retrieved February 10, 2015 from https://github.com/ mapbox/rasterio.

Mennis, J. 2010. Multidimensional Map Algebra: Design and Implementation of a Spatio-Temporal GIS Processing Language. *Transactions In GIS*, *14*(1), 1-21. doi:10.1111/j.1467-9671.2009.01179.x.

Python Software Foundation. 2014. 27.5 timeit: Measure Execution Time of

Small Code Snippets. Retrieved February 12, 2015 from https://docs.python.org /3/library/timeit.html.

Steiniger, S., and Bocher, E. 2009. An overview on current free and open source desktop GIS developments. *International Journal of Geographical Information Science*, *23*(10), 1345-1370.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22-30.

Wagner, D. F., and Scott, M. S. 1995. Improving the performance of raster GIS: a comparison of approaches to parallelization of cost volume algorithms. In *AUTOCARTO-CONFERENCE-* (pp. 164-176).

Appendix A. Execution Time Summary Statistics (in milliseconds).

Implementation	Minimum	First Quartile	Median Quartile	Mean	Third Quartile	Maximum
adding_one_raster_by_a_constant.py						
ArcPy	194.7	197.3	199.1	198.8	200.1	205.1
ArcPy with GC	194.5	197.3	198.7	198.7	200.4	206.7
Rasterio	9.3	9.4	9.7	9.8	10.1	11.8
Rasterio with GC	9.3	9.3	9.4	9.7	10.1	11.1
adding_two_rasters.py						
ArcPy	209.6	211.3	212.4	212.5	213.4	216.9
ArcPy with GC	209.2	211.5	213.0	213.0	214.4	217.7
Rasterio	11.5	11.6	11.8	11.8	12.0	13.0
Rasterio with GC	11.5	11.5	11.8	11.9	12.0	13.2
boolean_formula.py						
ArcPy	597.8	600.4	603.0	602.9	604.2	614.2
ArcPy with GC	595.7	599.8	602.3	602.9	605.1	612.7
Rasterio	17.5	17.6	17.6	17.6	17.7	17.9
Rasterio with GC	17.4	17.5	17.6	17.8	17.7	21.4
boolean_not_one_raster.py						
ArcPy	183.4	186.5	188.1	187.9	189.1	193.4
ArcPy with GC	182.6	186.6	188.3	188.2	189.6	196.8
Rasterio	11.6	11.8	12.0	11.9	12.0	12.6
Rasterio with GC	11.6	11.7	11.8	11.8	11.9	12.5
booleanAND.py						
ArcPy	441.0	445.1	447.2	447.4	448.6	455.0
ArcPy with GC	440.1	445.3	447.9	447.5	449.5	452.8
Rasterio	13.8	13.9	14.1	14.2	14.4	15.0
Rasterio with GC	13.8	14.0	14.2	14.3	14.4	17.0
booleanOR.py						
ArcPy	437.6	445.5	447.0	447.3	449.8	454.2
ArcPy with GC	440.1	446.6	448.4	449.1	451.2	460.6
Rasterio	13.8	14.2	14.5	14.5	14.7	15.5
Rasterio with GC	13.8	14.6	14.7	14.6	14.8	14.9
converting_celsius_to_fahrenheit_one_raster.py						
ArcPy	208.3	210.1	211.4	211.6	212.2	218.0
ArcPy with GC	207.9	209.5	210.9	211.3	213.0	217.7
Rasterio	16.5	16.6	16.7	16.8	16.8	17.6
Rasterio with GC	16.4	16.7	16.7	16.7	16.8	16.9
converting_to_a_boolean_one_raster.py						
ArcPy	180.0	183.2	183.8	184.2	185.0	189.4
ArcPy with GC	180.4	182.6	184.0	184.4	185.9	191.2
Rasterio	11.5	11.7	11.8	11.9	11.9	15.2
Rasterio with GC	11.5	11.6	11.7	11.8	11.9	12.7
dividing_one_raster_by_a_constant.py						
ArcPy	199.3	202.4	204.2	204.1	205.3	209.5
ArcPy with GC	199.3	201.5	202.9	202.7	203.6	209.7
Rasterio	14.1	14.1	14.2	14.2	14.3	14.8
Rasterio with GC	14.0	14.1	14.1	14.2	14.2	15.1
dividing_two_rasters.py						
ArcPy	200.4	201.7	203.5	203.2	204.5	206.4
ArcPy with GC	200.5	202.7	203.7	203.8	204.8	209.9
Rasterio	35.9	36.1	36.2	36.2	36.2	37.9

Rasterio with GC	35.8	36.0	36.1	36.2	36.1	39.0
multiplying_one_raster_by_a_constant.py						
ArcPy	192.6	195.7	197.1	197.0	198.3	203.6
ArcPy with GC	192.6	196.2	197.7	197.6	199.4	202.1
Rasterio	9.6	10.2	10.4	10.5	10.6	13.7
Rasterio with GC	9.7	10.2	10.4	10.4	10.6	11.8
multiplying_two_rasters.py						
ArcPy	211.7	213.8	215.5	216.2	216.8	231.7
ArcPy with GC	210.8	214.0	215.0	215.2	216.6	219.2
Rasterio	11.9	12.1	12.2	12.2	12.2	14.0
Rasterio with GC	11.9	12.0	12.1	12.1	12.1	12.5
reading_one_raster.py						
ArcPy	10.0	10.1	10.1	10.1	10.2	10.3
ArcPy with GC	10.0	10.1	10.1	10.1	10.1	10.2
Rasterio	4.0	4.0	4.1	4.1	4.1	4.1
Rasterio with GC	4.0	4.0	4.0	4.0	4.0	4.1
subtracting_one_raster_by_a_constant.py						
ArcPy	204.9	207.0	209.1	209.1	210.6	216.7
ArcPy with GC	204.4	207.2	209.6	211.0	212.8	229.3
Rasterio	9.7	10.0	10.3	10.4	10.6	11.2
Rasterio with GC	9.8	10.0	10.2	10.3	10.4	11.6
subtracting_two_rasters.py						
ArcPy	216.9	220.2	221.4	221.7	222.7	226.7
ArcPy with GC	218.1	221.0	221.8	223.9	224.4	255.0
Rasterio	11.7	11.9	12.0	12.0	12.1	12.7
Rasterio with GC	11.9	12.1	12.1	12.2	12.2	13.9
two_raster_formula.py						
ArcPy	229.3	232.2	233.1	233.5	234.9	238.9
ArcPy with GC	229.7	232.5	233.2	233.5	234.6	238.1
Rasterio	15.1	15.2	15.4	15.4	15.5	17.5
Rasterio with GC	15.0	15.2	15.3	15.3	15.5	15.8
writing_one_raster.py						
ArcPy	120.3	120.7	121.0	121.1	121.3	123.8
ArcPy with GC	120.2	120.6	121.1	121.4	121.6	126.5
Rasterio	12.8	13.1	13.3	13.5	13.4	17.5
Rasterio with GC	12.8	13.2	13.4	13.6	13.6	16.1

Appendix B. Memory Usage by Program in Mebibytes (MiB).

Implementation	Memory at Start	Memory at End	Memory Difference
adding_one_raster_by_a_constant.py			
ArcPy	236.789	247.387	10.598
Rasterio	34.543	37.191	2.648
adding_two_rasters.py			
ArcPy	240.324	253.477	13.153
Rasterio	34.660	38.453	3.793
boolean_formula.py			
ArcPy	240.266	256.973	16.707
Rasterio	34.648	39.016	4.368
boolean_not_one_raster.py			
ArcPy	238.523	252.328	13.805
Rasterio	34.621	37.246	2.625
booleanAND.py			
ArcPy	238.605	253.668	15.063
Rasterio	34.637	38.453	3.816
booleanOR.py			
ArcPy	240.074	255.188	15.114
Rasterio	34.648	38.461	3.813
converting_celsius_to_fahrenheit_one_raster.py			
ArcPy	238.699	249.441	10.742
Rasterio	34.609	37.246	2.637
converting_to_a_boolean_one_raster.py			
ArcPy	238.070	251.473	13.403
Rasterio	34.551	37.176	2.625
dividing_one_raster_by_a_constant.py			
ArcPy	237.734	248.156	10.422
Rasterio	34.523	37.156	2.633
dividing_two_rasters.py			
ArcPy	238.840	252.184	13.344
Rasterio	34.668	38.504	3.836
multiplying_one_raster_by_a_constant.py			
ArcPy	243.738	254.258	10.520
Rasterio	34.516	37.148	2.632
multiplying_two_rasters.py			
ArcPy	238.129	251.734	13.605
Rasterio	34.637	38.449	3.812
reading_one_raster.py			
ArcPy	238.652	240.273	1.621
Rasterio	34.609	37.836	3.227
subtracting_one_raster_by_a_constant.py			
ArcPy	238.484	250.742	12.258
Rasterio	34.520	37.152	2.632
subtracting_two_rasters.py			
ArcPy	238.648	253.254	14.606
Rasterio	34.668	38.465	3.797
two_raster_formula.py			

Rasterio	34.625	38.957	4.332
ArcPy	238.438	252.602	14.164
writing_one_raster.py			
ArcPy	238.676	247.355	8.679
Rasterio	34.625	39.332	4.707