Lab: Multiprocessing and ArcPy

Lab. Parallel Computation using Multiprocessing and Arcpy

ArcPy is a Python site package for ArcGIS 10+. ArcPy enables the automation of map creation and the conversion and management of data and provides access to a large number of Geoprocessing tools, functions, classes and modules which can be incorporated into GIS workflows. Multiprocessing is a site package which allows for the parallelization of a Python script. It can be used to parallel both locally and remotely. This package functions by creating sub-processes as opposed to utilizing threads. This lab will outline important parts of implementing the multiprocessing package to a script, as well as provide several examples of its use.

Installation and Set up

- To use ArcPy, ArcGIS must be installed on computer with a valid license.
- To use multiprocessing, Python language package version 2.6 or newer is needed.
- The Parallel Python module must be downloaded separately to your machine before use.
- Once the installations are finished, these modules can be imported directly to the Python script.

Multiprocessing Functions

1. Pool (**Processes = number**)

A process pool object which controls a pool of worker processes to which jobs can be submitted. It supports asynchronous results with timeouts and callbacks and has a parallel map implementation. (https://docs.python.org/2/library/multiprocessing.html) You can choose the number of processes to assign using this function.

2. Map(**function,iterable[,chunksize]**)

This blocks the function until the result is ready. Function is only executed in one of the workers of the pool and it supports only one iterable argument. The second argument can be a list; the list represents how you split tasks. Each element in the list will be passed to the function and run parallel.

3. Close

Prevents any more tasks from being submitted to the pool. Once all the tasks have been completed the worker processes will exit. (https://docs.python.org/2/library/multiprocessing.html)

4. Join

Wait for the worker processes to exit. One must call ***close()*** or ***terminate()*** before using ***join()*** .(https://docs.python.org/2/library/multiprocessing.html)

Examples of Multiprocessing

Prime number sum

This first example illustrates how the Multiprocessing module works on a simple mathematical operation (not using Arcpy). The script calculates the sum of prime numbers below a given integer in parallel.

```
import time,math,multiprocessing   # Import module

def isprime(n):  # Defines"isprime" function: Returns True if n is prime
    if not isinstance(n, int):
        raise TypeError("argument passed to is_prime is not of 'int' type")
    if n < 2:
        return False
    if n == 2:
        return True
    max = int(math.ceil(math.sqrt(n)))
    i = 2
    while i <= max:
        if n % i == 0:
            return False
        i += 1
    return True

# Calculates sum of all primes below given integer n
def sum_primes(n):
    return sum([x for x in xrange(2,n) if isprime(x)])

def main():
    # Define chunksize
    inputs = [100000, 100100, 100200, 100300, 100400, 100500, 100600, 100700]
    pool = multiprocessing.Pool()  # Initialize pool of workers

    # Use function name and list as arguments and map jobs to different workers,
    # and get the results object
    out = pool.map(sum_primes,inputs)
    for i in range(len(out)):
        # Format print out on screen
        print "Sum of primes below" , inputs[i] , 'is' , out[i]
    pool.close()   # Prevent processes to enter the pool
    pool.join()    # Wait until all the processes are done

# Calls main function: calculates time at process start, time at process end,
calculates elapsed time and prints it
if __name__ == '__main__':
    t1 = time.time()
    main()
```

Add Field and Calculate Field

This second example adds and calculates field contents to all shapefiles located in a specified directory. The application of Multiprocessing here is designed to speed up a repetitive process applied to a large number of datasets. This example uses Arcpy Multiprocessing and is an example of how multiprocessing can handle tasks related to vector GIS data.

```
import os,arcpy,multiprocessing,time     # Import modules

def update_shapefiles(shapefile):        # Define function for shapefile updating
    fieldList = arcpy.ListFields(shapefile)  # Create list of all fields in dataset
    for i in fieldList:
        if i == "TYPE" or i == "NAME":
           # Delete "TYPE" or "NAME" titled fields if they pre-exist
           arcpy.DeleteField_management(shapefile,[i])
           # Add field called "TYPE" in "TEXT" format
           arcpy.management.AddField(shapefile,'TYPE','TEXT')
           # Add field called "NAME" in "TEXT" format
           arcpy.management.AddField(shapefile,'NAME','TEXT')

    desc = arcpy.Describe(shapefile)

    type1 = str(desc.shapeType)   # Store string of shapefile type
    name = shapefile[:-4]         # Store portion of shapefile name

    arcpy.CalculateField_management(shapefile,'TYPE','"'+   type1 +'"')
    arcpy.CalculateField_management(shapefile,'NAME','"'+   name + '"')

def main():
    workspace = r'D:\YOUR_WORKSPACE_HERE! '# Define workspace
    arcpy.env.workspace  = workspace      # Set Arcpy environment workspace
    fcs = arcpy.ListFeatureClasses('*')  # List all feature classes

    # Create fully pathed workspace locations
    fc_list = [os.path.join(workspace,  fc) for fc in fcs]

    # Pool processes and multiprocess "update_shapefile" function on "fc_list"
    pool = multiprocessing.Pool()
    pool.map(update_shapefiles,  fc_list)
    pool.close()
    pool.join()

# Calls main function: calculates time at process start, time at process end,
calculates elapsed time and prints it
if __name__ == '__main__':
    t1 = time.time()
    main()
    t2 = time.time()
    print t2-t1
```

Download Word Document Copy of Lab Assignment

Creator Name	Jie Tian
Content Title	Multiprocessing and ArcPy
Content Type	Lab Assignment
Part Of	Module 3: High Performance Geospatial Computing
Learning Objectives	
Background Knowledge	
Resources Needed	
Work Mode	
Relation to Project	
Feedback Needed	