**Lab. High Performance Geocomputing on Amazon Virtual Machine**

The multiprocessing Python module enables the distribution of computation work between multiple processes, taking advantage of multiple CPU cores and larger amounts of available system memory. The goal of this lab is to develop a Python script for processing vector GIS datasets with many features and to run the script on an Amazon Virtual Machine. The task is to find identical features among 1 million points and the idea is to improve the performance by splitting data into groups to be processed simultaneously. A fishnet of tiles is pre-created to define the spatial extents for the groups.



**Python Script**

The script consists of two major functions: **find\_identical** and **main**. Find\_identical is basically a worker function to perform finding identical within a given tile and returning a numpy array as the result. In the main function, a pool object is created and the find\_identical function is mapped to a list of all the tiles to run the jobs in parallel.

1. **Import the modules and set the environment**

import multiprocessing

import numpy

import arcpy

arcpy.env.overwriteOutput = True

1. **Find\_identical function**

def find\_identical(oid):

    '''Worker function to perform Find Identical, and return'''

    '''a list of numpy arrays as the results.'''

    # Create a feature layer for the tile in the fishnet.

    tile = arcpy.management.MakeFeatureLayer(

                        'YOUR\_WORKSPACE/fishnet',

                        'layer{0}'.format(oid[0]),

                        """OID = {0}""".format((oid[0])))

    # Get the extent of the feature layer and set the environment

    # to use during Find Identical.

    tile\_row = arcpy.da.SearchCursor(tile, 'shape@')

    geometry = tile\_row.next()[0]

    arcpy.env.extent = geometry.extent

    # Execute Find Identical

    identical\_table = arcpy.management.FindIdentical(

                         'YOUR\_WORKSPACE/Points',

                         'in\_memory/identical', 'Shape')

    # Convert the resulting table into a numpy array and return

    result\_array = arcpy.da.TableToNumPyArray(identical\_table, ["\*"])

    return result\_array

    # End find\_identical

1. **main function**

def main():

    t1 = time.time()

  # Create a list of OID's used to chunk the inputs

    fishnet\_rows = arcpy.SearchCursor(

              'YOUR\_WORKSPACE/fishnet', '', '', 'OID')

    oids = [[row.getValue('OID')] for row in fishnet\_rows]

    # Create a pool class and run the jobs--

    # the number of jobs is equal to the length of the oids list

    pool = multiprocessing.Pool()

    result\_arrays = pool.map(find\_identical, oids)

    # Concatenate the resulting arrays and create an output table

    # reporting any identical records.

    result\_array = numpy.concatenate(result\_arrays,axis=0)

    arcpy.da.NumPyArrayToTable(result\_array,

                                'YOUR\_WORKSPACE/identical')

    # Synchronize the main process with the job processes to ensure

    # proper cleanup.

    pool.close()

    pool.join()

 t2 = time.time()

 # Print the time spent

 print t2 - t1

    # End main

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**AWS Virtual Machine**

Now you can log into your AWS account and launch four EC2 instances with 4, 8, 12, and 16 CPU cores, respectively. ArcGIS then needs to be installed on these EC2 virtual machines. The instruction for ArcGIS installation on Amazon EC2 can be found in this Youtube video. <https://www.youtube.com/watch?v=fX_Cdh6DzQM>. You can then run the script on these virtual machines and record the time spent on each. Plot out the time spent against the number of CPU cores to see how the performance has been improved.

**Credit:** the python script is modified from the example found at

http://blogs.esri.com/esri/arcgis/2011/08/29/multiprocessing/