Practical examples

• Parallel Processing Factor (PPF)
• Python multiprocessing module

Assuming some familiarity with geoprocessing in ArcGIS and Python scripting

ArcGIS for Desktop version 10.2.2

Testing resources

• Dell OptiPlex 790 - Win 7
  • 1 CPU; 4 cores; 8 logical processors
  • Network adapter: 1 Gbps
• Win VM - Win Server 2008 R2 Std.
  • 2 CPU; 4 cores/CPU; 4 logical processors/CPU
  • Network adapter: 1 Gbps & 10 Gbps
In a Nutshell

With geoprocessing

With multi-geoprocessing
Geoprocessing: taking advantage of multiple CPUs

  - Parallel Processing Factor
    - Divide and perform operations across multiple processes
    - Environment setting
      - Additional parameters that affect a tool's results
      - Not all tools honor all environment settings
Environment Settings

How to access

ArcCatalog - Folder Connections

ArcToolbox

Build Pyramids And Statistics

Input Data or Workspace

Include Sub-directories (optional)

Build Pyramids (optional)

Calculate Statistics (optional)

Skip Existing (optional)

Statistics Options

Pyramids Options
Environment Settings Dialog

Parallel Processing Factor

Tools that honor the Parallel Processing Factor environment will divide and perform operations across multiple processes.

Many modern computers include multiple-core CPUs. Spreading a geoprocessing operation across multiple processes can speed up performance by taking advantage of more than one core. The performance benefit of parallel processing varies from tool to tool.

Usage notes:

- The value of this environment determines the number of processes across which a tool spreads its operation. Those processes will be divided between hardware cores (processors) built into the machine. The number of hardware cores does not change based on this setting.
- Each tool that honors this environment has a built-in default for the number of processes given a particular machine. You can change this based on your data, operation, and available resources.
- If you specify a percent value (using the % symbol) then the number of processes used will be the specified percentage of the number of cores on the machine, rounded to the nearest integer. For example, on a 4-core machine:
  - Setting 50% means the operation will be spread over 2 processes (50% * 4 = 2).
  - Setting 66% means the operation will be spread over 3 processes (66% * 4 = 2.64 which rounds to 3).
  - Setting 100% means the operation will be spread over all 4 processes (100% * 4 = 4).
No simple listing
Check the tool help web page
ArcGIS Help Library (10.2)
  • Desktop > Geoprocessing > Tool reference
  • Select a toolbox, toolset, then a tool
    • Build pyramids and statistics

“In the first release of ArcGIS Pro 1.0, there are 25 tools that support parallel processing, including a number of geostatistics tools and Spatial Analyst tools”.
Note: Python scripting syntax: `arcpy.env.parallelProcessingFactor = string`
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty string (blank)</td>
<td>Let tool determine how many processes are used. This is the default.</td>
</tr>
<tr>
<td>0</td>
<td>Do not spread operations across multiple processes.</td>
</tr>
<tr>
<td>n</td>
<td>Use the specified number of processes.</td>
</tr>
<tr>
<td>n%</td>
<td>Calculate the number of processes using the specified percentage. (# of processes = # of system cores * n / 100).</td>
</tr>
</tbody>
</table>
100 GeoTIFFs on network drive*

*Watch network utilization as # of processes increases
Using Arcpy with the Multiprocessing (Mp) Module

- **The multiprocessing module**
  - Distribute work across multiple processes on a given machine
  - Script may run faster
  - Installed with ArcGIS for Desktop

- **Two approaches we will look at**
  - Processing many individual datasets
  - Processing datasets with many features
Be aware of file locks (e.g. *.lock)

At lot of factors at play to improve speed
- CPU/memory/disk speed/data location
- Number of files/features
- Data format
- Is your workflow conducive to multiprocessing?

Trial and error

Take advantage of storing data in memory
Can this code be split into sub-parts that can be run concurrently?

What ‘work’ are we doing?
Toward Multiprocessing

Place ‘work’ (define projection) into a function

```python
import os
import arcpy

# Define a function to perform work on a shapefile
def update_shapefiles(shapefilename):
    # Define the projection to wgs84. Factory code is 4326.
    arcpy.management.DefineProjection(shapefilename, 4326)

def main():
    # Set the current workspace
    inWs = r'c:\temp\data\'
    arcpy.env.workspace = inWs

    # Get a list of all the shapefiles in the current workspace
    fcs = arcpy.ListFeatureClasses('**')
    for fc in fcs:
        # Call function. Pass the name of the shapefile
        shapefilename = os.path.join(inWs, fc)
        update_shapefiles(shapefilename)

if __name__ == '__main__':
    main()
```
Use the multiprocessing module to run the function `update_shapefiles` on multiple files concurrently

- Divide the work among multiple workers
- See the following slide
Define the projection for all shapefiles in a directory using multiprocessing

```python
import os
import arcpy
import multiprocessing

# Define a function to perform work on a shapefile
def update_shapefiles(shapefilename):
    import arcpy
    # Define the projection to wgs84. Factory code is 4326.
    arcpy.management.DefineProjection(shapefilename, 4326)

def main():
    # Set the current workspace
    inWs = r'c:\temp\data'\n    arcpy.env.workspace = inWs

    # Get a list of all the shapefiles in the current workspace
    fcs = arcpy.ListFeatureClasses('*')
    fc_list = [os.path.join(inWs, fc) for fc in fcs]

    # Create a pool class and run the jobs. The number of jobs is equal to the number of shapefiles
    pool = multiprocessing.Pool(multiprocessing.cpu_count() -1)
    pool.map(update_shapefiles, fc_list)

    # Synchronize the main process with the job processes to ensure proper cleanup.
    pool.close()
    pool.join()

if __name__ == '__main__':
    main()
```
Task: define projection

<table>
<thead>
<tr>
<th>Number of shapefiles</th>
<th>Time (single)</th>
<th>Time (multi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1 sec.</td>
<td>25 sec.</td>
</tr>
<tr>
<td>20,000</td>
<td>5 min.</td>
<td>4 min.</td>
</tr>
<tr>
<td>100,000</td>
<td>40 min.</td>
<td>16 min.</td>
</tr>
</tbody>
</table>
Not always faster
  • Cost to start up processes
  • Many factors
  • More work > more benefit?
  • Trial and error
Try out the examples provided
Select groups of features by attribute(s) (e.g. OBJECTID or another field)

Example:

- Add and calculate two fields; write to .dbf
  - Data source: SDE feature class
  - Select by OBJECTID groups

<table>
<thead>
<tr>
<th>Number of features</th>
<th>Time (single)</th>
<th>Time (multi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,300</td>
<td>20 sec.</td>
<td>35 sec.</td>
</tr>
<tr>
<td>2.3 million</td>
<td>80 min.</td>
<td>20 min.</td>
</tr>
</tbody>
</table>

Let’s take a look at the example scripts
Let’s look at a few more

- Image service
  - Download rasters [8 processes]

<table>
<thead>
<tr>
<th>Number of features</th>
<th>Time (single)</th>
<th>Time (multi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>13 min.</td>
<td>10 min.</td>
</tr>
</tbody>
</table>
Let’s look at a few more
• Feature service
  • Esri zip codes

<table>
<thead>
<tr>
<th>Number of features</th>
<th>Time (single)</th>
<th>Time (multi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000</td>
<td>80 sec.</td>
<td>40 sec.</td>
</tr>
<tr>
<td>10,000</td>
<td>6.5 min.</td>
<td>1.5 min.</td>
</tr>
<tr>
<td>30,522</td>
<td>20 min.</td>
<td>4 min.</td>
</tr>
</tbody>
</table>

• Many variable can influence results
  • Complex geometry or not
  • Do you control service parameters?
    • Maximum number for records allowed to select at one time
    • Number of instances
    • Network link speed
Let’s look at a few more

- Image service
- Feature service
  - Esri zip codes

<table>
<thead>
<tr>
<th>Number of features</th>
<th>Time (single)</th>
<th>Time (multi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000</td>
<td>80 sec.</td>
<td>40 sec.</td>
</tr>
<tr>
<td>10,000</td>
<td>6.5 min.</td>
<td>1.5 min.</td>
</tr>
<tr>
<td>30,522</td>
<td>20 min.</td>
<td>4 min.</td>
</tr>
</tbody>
</table>
For multiprocessing be sure to uncheck

Run Python script in process
Multiprocessing is supported in ArcGIS for Server

- Many factors can influence results
  - Number of instances
  - Memory
  - Network link speed
Helpful Resources

- Python multiprocessing module
- Multiprocessing with ArcGIS – Approaches and Considerations (Part 1)
- Using Arcpy with multiprocessing – Part 1
- Using Arcpy with multiprocessing – Part 2
- Using Arcpy with multiprocessing – Part 3
- Multi-Core and Distributed Programming in Python
Do you have a script that might benefit from multiprocessing?
Contact Information

Bruce Godfrey
GIS Librarian
bgodfrey@uidaho.edu
208-292-1407