From Python Scripting to Parallel Spatial Modeling

Jesús Carabaño Bravo
jcarababan@abo.fi
PhD Candidate at ÅBO AKADEMI
Faculty of Science and Engineering
Introduction

We are building a **prototype** of a spatial **framework** / library / module that uses **compilers** techniques to automatically **optimize** sequential raster scripts in e.g. python and executes scripts in **parallel** on e.g. GPUs so that we can handle **very large** datasets

*Looks like raster processing in* **ArcPy / Matlab / Numpy**

**Applications:** land use, hydrology, air quality, land erosion, predictive analysis, geomorphology, ecology
Script 1: urban development

```
1 from map import * ## "Parallel Map Algebra" package
2
3 a = 6.4640  # Constant coefficient
4 b1 = 43.5404 # Elevation coefficient
5 b2 = 1.9150  # Slope coefficient
6 b3 = 41.3441 # Distance to city centers coefficients
7 b4 = 12.5878 # Distance to transportations coefficient
8 b5 = [0.0,-9.865,-8.746,-9.268,-8.032,-9.169,-8.942,-9.45] # {water,urban,barren,forest,shrub,woody,herb,crop,wetlad}
9 d = 5        # dispersion parameter
10 q = 16000   # max cells to become urban per year
11
12 x1 = read('dem')  # elevation layer
13 x2 = read('slope') # slope layer
14 x3 = read('center') # distance to centers layer
15 x4 = read('transp') # distance to transportations layer
16 x5 = read('landuse') # land use layer
17 e = read('excl')  # exclusion layer (e.g. water bodies)
18 s = read('urban')  # initial state: urban / not-urban
19 N = 50           # years of simulation i.e. time steps
20
21 for i in range(N) :
22    z = a + b1*x1 + b2*x2 + b3*x3 + b4*x4 + pick(x5,b5)
23    pg = exp(z) / (1 + exp(z))
24    pc = pg * !e * !s * focalSum(s) / (3^3-1)
25    pd = pc * exp(-d * (1 - pc / zonalMax(pc)))
26    ps = q * pd / zonalSum(pd)
27    s = s || ps > rand()
28
29 write(s,'output')
```

Ref: Wu 2002 “Calibration of stochastic cellular automata: the application to rural-urban land conversions”
Script 1: urban development

```python
1 import urban  # imports 'urban()' function, containing Listing 1
2
3 prob = zeros()  # urban probability map
4 M = 1000  # Monte Carlo iterations
5
6 for i in range(0,M) :  # Monte Carlo method
7    prob = prob + urban()  # urban() returns the urban layer
8    prob = prob / M  # urban() ∈ {0,1} ==> prob ∈ [0,1]
9
10 write(prob,'output')
```

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Baseline</th>
<th>GPU</th>
<th>GPU + Loc</th>
<th>GPU + Loc + Act</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>20 min</td>
<td>10 min</td>
<td>3 min</td>
<td>1 min 10 s</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1x</td>
<td>2x</td>
<td>6.66x</td>
<td><strong>17.14x</strong></td>
</tr>
<tr>
<td><strong>Baseline = CPU SIMD Threaded</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Loc = Locality Opt.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Act = Only Active Blocks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Monte Carlo | 1 iteration | 10 iter. | 100 iter. | 10000 iter. |
| Execution  | 1 min 14 s | 12 min | 121 min | 1214 min |

Ref 1 = 32s with 64 GPUs, We = 70s with just 1 GPU

Ref 1: Guan 2016 “A hybrid parallel cellular automata model for urban growth simulation over GPU/CPU…

Ref 2: Guan 2014 “pRPL 2.0: Improving the Parallel Raster Processing Library,” Trans. GIS, vol. 18, …
Script 1: urban development

California

Marin County

More 20 GB including all layers!

However the machine only has 16 GB of RAM
Script 2: flooding model

```python
from map import *  # "Parallel Map Algebra" package

h = read('dem')    # digital elevation layer
w = read('water')  # water depth layer
i = read('inflow') # inlets inflow layer
o = read('outflow')# outlets outflow layer
N = 1000           # number of time steps

def swap(x,i,j):
    x[i], x[j] = min(x[i], x[j]), max(x[i], x[j])

def netsort5(x):
    swap(x,0,1); swap(x,2,3); swap(x,0,2)
    swap(x,3,4); swap(x,0,3); swap(x,1,3)
    swap(x,2,4); swap(x,1,4); swap(x,1,2)

def avglevel(w,h,x):
    netsort5(x)  # ascending order
    s = w+x[0]  # sum variable
    n = 1       # count variable
    for i in range(1,5):
        b = (s >= x[i]*i)
        s += b*x[i]
        n += b
    return s / n

def gather(w,h):
    x = [0]*5  # neighborhood (NBH)
    x[0] = h   # central cell
    x[1] = h[0,-1] + w[0,-1]
    x[2] = h[-1,0] + w[-1,0]
    x[3] = h[+1,0] + w[+1,0]
    x[4] = h[0,+1] + w[0,+1]
    return avglevel(w,h,x)

def distri(w,h,l):
    wh = w+h  # prev water level
    c = max(0, l[0,-1] - wh)
    c += max(0, l[-1,0] - wh)
    c += max(0, l[+1,0] - wh)
    c += max(0, l[0,+1] - wh)
    cwh = max(c + wh, h)
    return cwh - h

for j in range(0,N):
    w = w + i  # fill inlet
    l = gather(w,h)  # gather avg
    w = distri(w,h,l) # distribute
    w = max(w-o,0)  # drain water

write(w,'output')
```

Ref: S. Di Gregorio and R. Serra, "An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata"

Ref: P. Topa, "Cellular Automata Model Tuned for Efficient Computation on GPU with Global Memory Cache"
The objective of our project, called SoDA (Soil Degradation Assessment), is therefore to develop and validate a dynamic simulation of soil erosion at the meter scale, keeping in mind a constant care for visualization.

To ensure the correctness of our model, we have a preoccupation with a validation by the images and numerical data that simulation will provide. For that purpose collaborations with experts in this field of research are essential and we do cooperate with an INRA laboratory (Agronomy Unit Laon-Reims-Mons, France).

2. Description of the processes involved in soil erosion and surface crusting

In order to simulate the soil structure evolution we need to have a description of the different processes involved in this phenomenon. In our literature review we have encountered three main kinds of soil erosion: wind erosion [20], thermal erosion [21] and hydraulic erosion [17].

Although thermal weathering, due to thermal shocks and gravity, influences the topological aspect of the soil and has been modeled for several visual simulations [15,19,21,22], we will concentrate on hydraulic erosion which is the main process responsible for the evolution of the soil surface of agricultural soil. Hydraulic erosion can be caused by rainfall or running water (Fig. 1). First, raindrops cause disaggregation: continuous matter or aggregates break down into smaller fragments [23]. Then, these fragments can be mobilized and transported by splash effect or by runoff [24,25].

Rainfall and runoff cause structural reorganization by the formation of crusts. Crusts are thin soil surface layers more compact and hard, when dry, than the material directly beneath. Generally, two main types of crust are distinguished by their mode of formation: structural crusts and sedimentary crusts [26]. A structural crust develops in situ and is the result of gradual coalescing of aggregates caused both by particle translocation and by raindrop compaction, whereas a sedimentary crust is formed by deposition of the particles suspended in overland flow [27].

Crusts hamper seedling emergence, reduce infiltration and favor runoff, puddling and thus erosion. The photographs of Fig. 2 show the visual perception of soil evolution and degradation with the generation of both types of crust from the initial soil shown in the first photograph: structural crust, in the second photograph, and sedimentary crust, in the last one. This evolution can be characterized by four main points:

1. aggregates outlines become less sharp and can even disappear,
2. filled orifices become more and more numerous,
3. surface roughness is decreasing,
4. the color of the soil and its reflectance properties are changing: the particle segregation results in either chemical (minerals) or physical (particle size) differentiation, which are both correlated with reflectance properties of soil [28].

It is worth noticing that these criteria are purely visual. For an agronomist the first and very important tool is direct visual observation. That is why in our project we have a constant care for visualization and we want to get images as well as numerical results from our simulator, in order to allow visual comparisons between simulation and reality. In particular, we aim to recreate the same visual evolution.

3. Related works

We present in this section a brief review of models found in the field of Computer Graphics and Soil Science followed by a few comments.

3.1. Computer graphics models

Musgrave et al. [19] demonstrate a new method for creating mountain fractal terrains with the use of height fields. They suggest two erosion algorithms which simulate hydraulic erosion by flowing water, ignoring evaporation and infiltration, and thermal weathering due to the thermal shocks which chips away steep inclines and forms talus slopes.

Kelley et al. [13] produce images of realistic-looking terrain with an algorithm consisting of two distinct steps.
Script 2: flooding model
More than magic

Python Script ➔ Compiler Magic ➔ Parallel Speedup

```python
27 def gather(w, h):
28     x = 0.5  # neighborhood (UMI)
29     x[0] = h  # central cell
30     x[1] = h[0], h[1]  # prev water level
33     x[4] = h[3], h[4]  # prev water level
34     return avglevel(w, h, x)
35
36 def distri(w, h, l):
37     wh = w[h]  # prev water level
38     c = max(0, l[0], l[1] - wh)
39     c = max(0, l[1], l[2] - wh)
40     c = max(0, l[2], l[3] - wh)
41     c = max(0, l[3], l[4] - wh)
43     wh = max(c + wh, h)
44     return cwh - wh
45
46 for j in range(0, H):
47     w = w + 1  # fill inlet
48     l = gather(w, h)  # gather avg
49     w = distri(w, h, l)  # distribute
50     w = max(w, 0)  # drain water
51
52 write(w, 'output')
```
Compiler Techniques

a) Python Script

IN1 = read(input1)
IN2 = read(input2)
L1 = LocalOp(IN1)
L2 = LocalOp(IN2)
R = RadialOp(L1)
F = FocalOp(R,L2)
L3 = LocalOp(R,F)
Z = ZonalOp(L3,F)
write(Z, output)

b) Dependency Graph

L1 → L2
R → F
L3 → Z

f) Tasks & Dataflow

Input

L1 → L2
R → F
L3 → Z
Output

c) Grouped Graph

Radial
c) Grouped Graph

R
L1
L2
F
L3 Z

g) Blocks, Jobs

IN1 = read(input1)
IN2 = read(input2)
L1 = LocalOp(IN1)
L2 = LocalOp(IN2)
R = RadialOp(L1)
F = FocalOp(R,L2)
L3 = LocalOp(R,F)
Z = ZonalOp(L3,F)
write(Z, output)

Radial

Kernel

In-L1-
R-Out

Focal

Kernel

In-L2-
F-Out

Zonal

Kernel

In-L3-Z-Out

d) GPU Code

e) Compilation

OpenCL

Compiler

Executable

Decomposition

Reordering

Time (task) dimension

Space (block) dimension

T R 0
R R 1
R R 2
F 2
Z 0
R 0
R 3
Struggling for locality

The #1 optimization is **locality**. Avoids idle CPU cycles waiting for the data

**CPU**
- ~100 GigaFlops
- Cache L1: 100 KB
- Cache L2: 1 MB
- Cache L3: 10 MB

**GPU**
- ~1 TeraFlops
- On-chip: 10 MB
- GPU mem: 1~10 GB

**Main Memory**: 10 ~ 100 GB
- 1 GB/s

**Solid State Drive**: 0.1 ~ 1 TB
- 100 MB/s

**Hard Disk Drive**: 1 ~ 10 TB (capacity)
- ~100 GigaFlops
- 10 GB/s

**Bandwidth**
- 100 GB/s

**Orders of Difference**
- 4x orders to GPU raw processing power
- 3x orders to GPU memory and CPU processing power
- 2x orders of difference to main memory
- 1x order of magnitude difference in bandwidth

GRASS, QGIS, ArcGIS *don’t exploit locality!*
Summary

A framework that automatically optimizes raster python scripts and executes them in parallel.

*Looks like* **ArcPy / Matlab / Numpy**

**Applications:** land use, hydrology, air quality, land erosion, predictive analysis, geomorphology, ecology

**Consequences**

- You *might* not need supercomputing power
- because your machine is still *underutilized*
- and workstations are *easier* to work with!
Thanks for your time!

Jesús Carabaño Bravo
jcaraban@abo.fi
PhD Candidate at ÅBO AKADEMI
Faculty of Science and Engineering