Åbo Akademi

From Python Scripting to Parallel Spatial Modeling

Jesús Carabaño Bravo
jcaraban@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

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

a  = 6.4640    # Constant coefficient
b1 = 43.5404   # Elevation coefficient
b2 = 1.9150    # Slope coefficient
b3 = 41.3441   # Distance to city centers coefficients
b4 = 12.5878   # Distance to transportations coefficient
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}
d  = 5         # dispersion parameter
q  = 16000     # max cells to become urban per year

x1 = read('dem')      # elevation layer
x2 = read('slope')    # slope layer
x3 = read('center')   # distance to centers layer
x4 = read('transp')   # distance to transportations layer
x5 = read('landuse')  # land use layer
e  = read('excl')     # exclusion layer (e.g. water bodies)
s  = read('urban')    # initial state: urban / not-urban
N  = 50               # years of simulation i.e. time steps

for i in range(N) :
    z  = a + b1*x1 + b2*x2 + b3*x3 + b4*x4 + pick(x5,b5)
    pg = exp(z) / (1 + exp(z))
    pc = pg * !e * !s * focalSum(s) / (3*3-1)
    pd = pc * exp(-d * (1 - pc / zonalMax(pc)))
    ps = q * pd / zonalSum(pd)
    s  = s || ps > rand()

write(s,'output')
```

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

Script 1: urban development

```
import urban # imports 'urban()' function, containing Listing 1

prob = zeros() # urban probability map
M = 1000       # Monte Carlo iterations

for i in range(0,M) :      # Monte Carlo method
    prob = prob + urban() # urban() returns the urban layer
prob = prob / M            # urban() ∊ {0,1} ==> prob ∊ [0,1]

write(prob,'output')
```

Optimizations	Baseline	GPU	GPU + Loc	GPU + Loc + Act
Execution	20 min	10 min	3 min	1 min 10 s
Speed Up	1x	2x	6.66x	**17.14x**

Baseline = CPU SIMD Threaded *Loc = Locality Opt.* *Act = Only Active Blocks*

Monte Carlo	1 iteration	10 iter.	100 iter.	1000 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

More **20 GB**
including all layers!

However the machine
only has **16 GB** of RAM

Script 2: flooding model

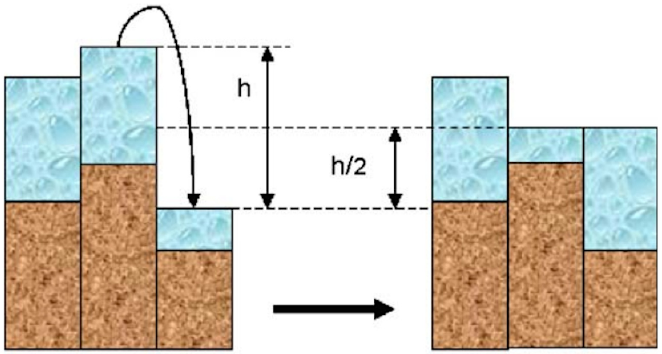
```
from map import * ## "Parallel Map Algebra" pac

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

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

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) :                    #   I
    netsort5(x) # ascending order        #   o
    s = w+x[0]  # sum variable           # Di
    n = 1       # count variable         #
    for i in range(1,5) :                # Ce
        b = (s >= x[i]*i)                # 7 I
        s += b*x[i]                      #
        n += b                           # Vo
    return s / n                         #
                                         # |
```

```
def gather(w,h) :
    x = [0]*5 # neigborhood (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)
    c += max(h, l) - 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 wate

write(w,'output')
```

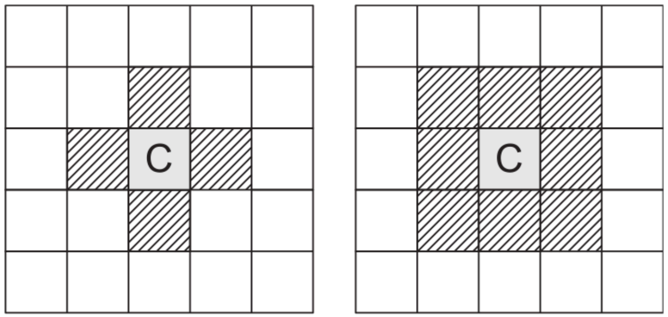
Ref: S. Di Gregorio and R. Serra, “An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata”

Ref: P. Topa, “Cellular Automata Model Tuned for Efficient Computation on GPU with Global Memory Cache”

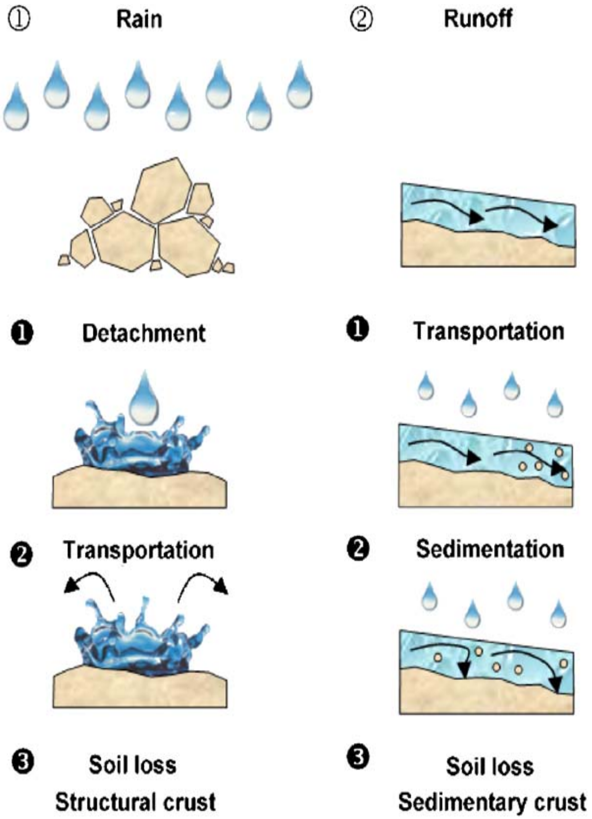
Script 2: flooding model



1-dimensional water balance



2-dimensional neighborhood

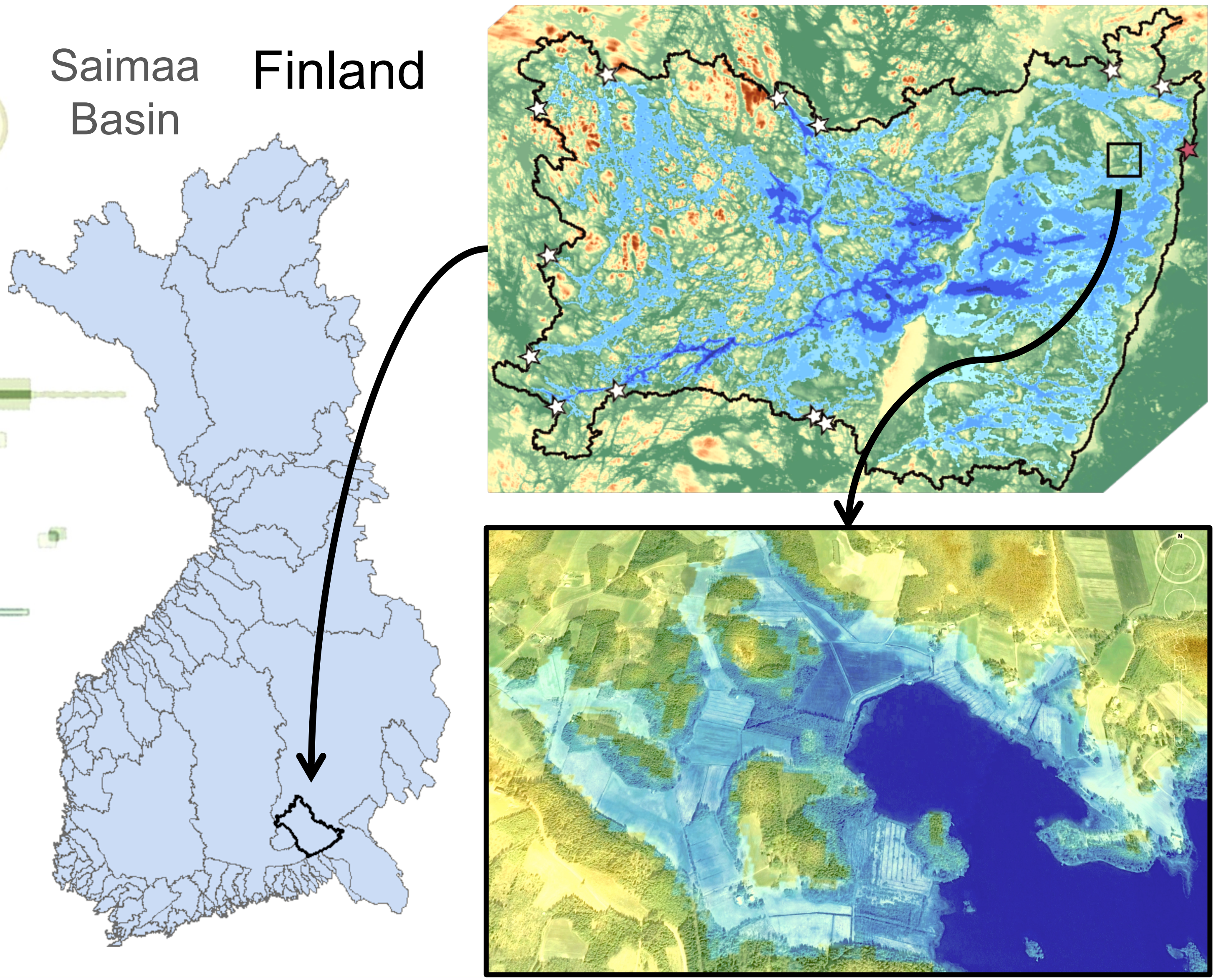


More advanced erosion model

Ref: Valette 2016 “SoDA project: A simulation of soil surface degradation by rainfall”

Ref: D. D’Ambrosio 2001 “A Cellular Automata model for soil erosion by water”

Script 2: flooding model

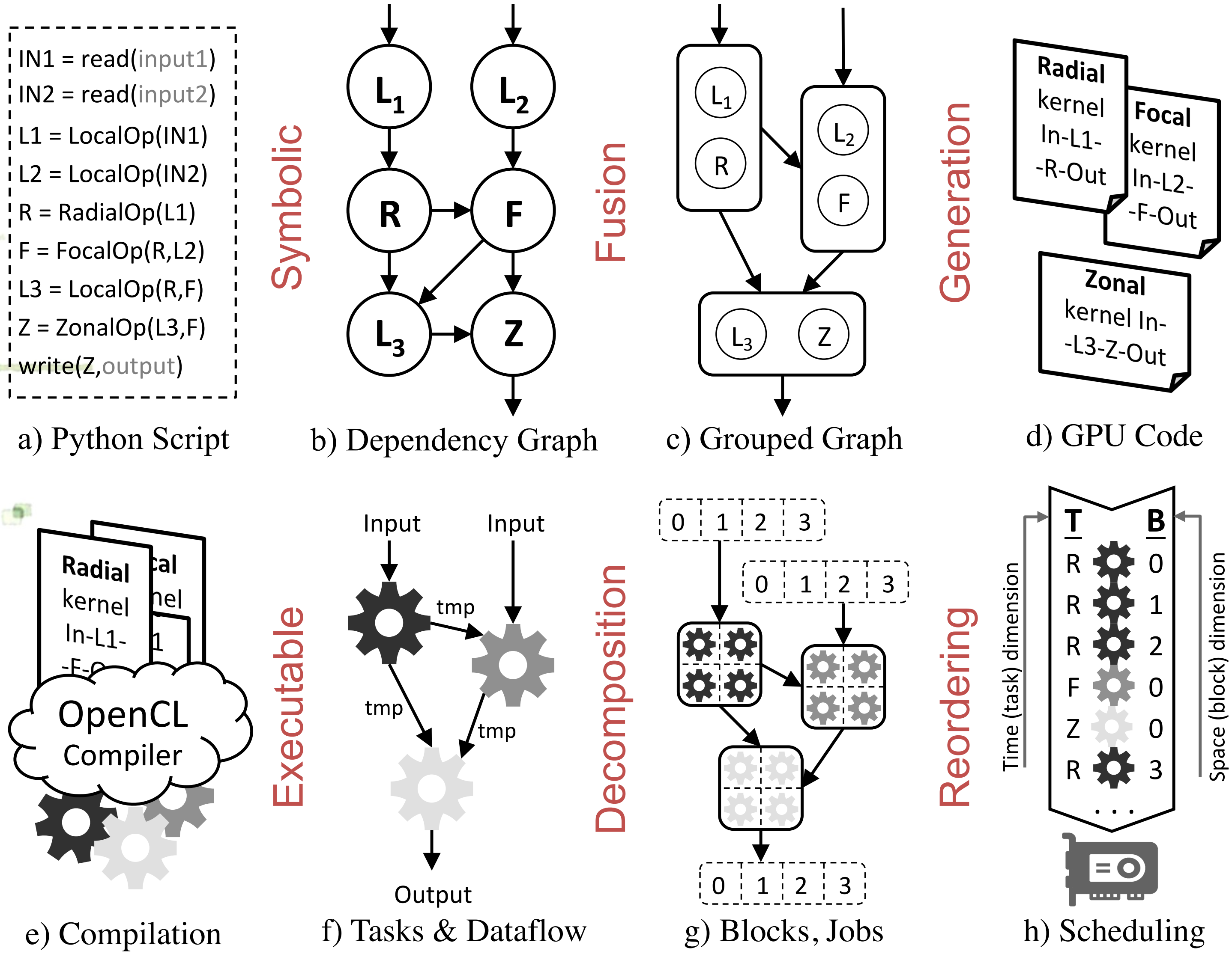


More than magic

Python Script → Compiler Magic → Parallel Speedup

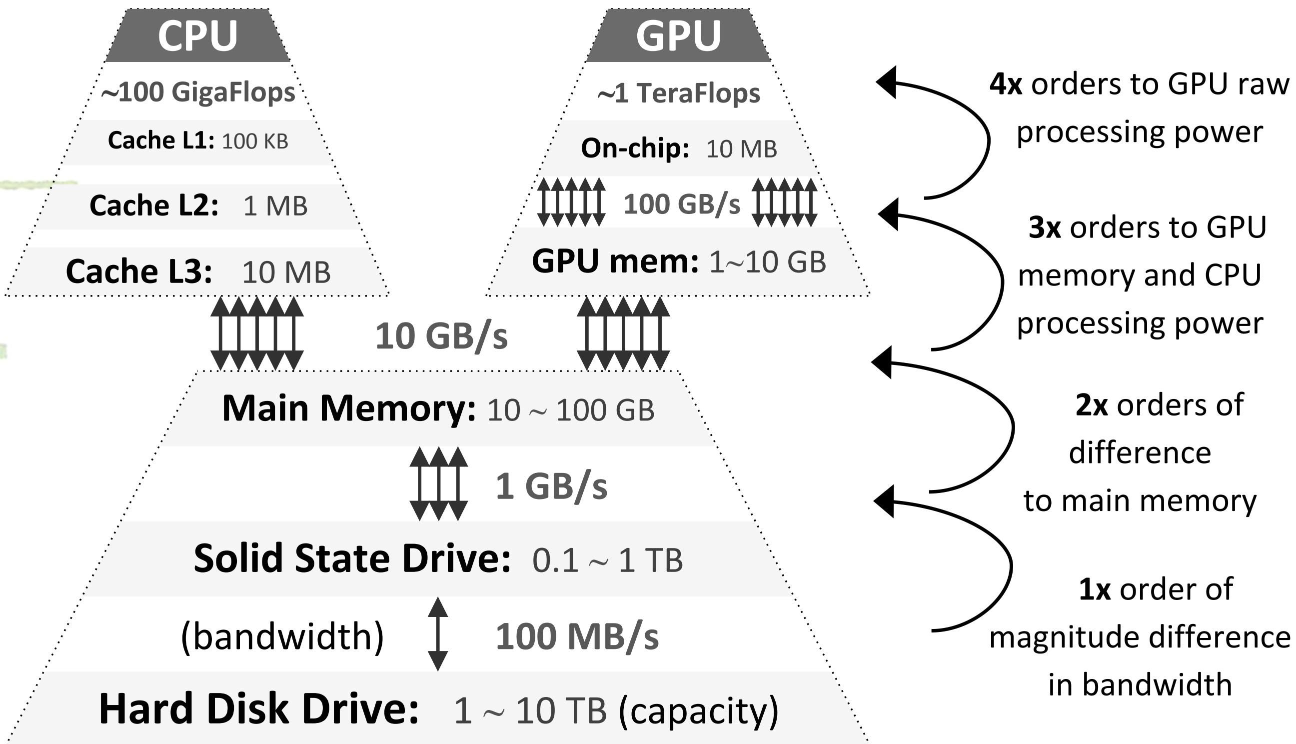
```
def gather(w,h) :
    x = [0]*5 # neigborhood (NBH)
    x[0] = h  # central cell
    x[1] = h[0,-1] + w[0,-1]
    x[2] = h[-1,0] + w[-1,0]
    x[3] = h[+1,0] + w[+1,0]
    x[4] = h[0,+1] + w[0,+1]
    return avglevel(w,h,x)

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)
    c += max(h, l) - 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-c,0)       # drain wate

write(w,'output')
```

Compiler Techniques



Struggling for locality

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



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