SMODERP2D SOIL EROSION MODEL ENTERING AN OPEN SOURCE ERA WITH GPU-BASED PARALLELIZATION

Landa, Jeřábek, Pešek, Kavka
Faculty of Civil Engineering
Czech Technical University in Prague

What is SMODERP2D?

- Runoff-soil erosion physically-based distributed episodic model
- Calculation and prediction processes at agricultural areas and small watersheds

- Written in Python
- Open Source project since 2018
- Code on GitHub https://github.com/storm-fsv-cvut/smoderp2d
- GNU GPL 3.0 licence

History

- Long-term project driven by the Department of Landscape Water Conservation at the Czech Technical University in Prague
- Originally developed as a surface runoff simulated by profile model (1D)
- Later redesigned using spatially distributed method → SMODERP2D
- Originally written in Fortran, later in Visual Basic, currently in Python
- Major refactoring / functionality improvements started in 2018

SMODERP2D Model

- GIS based model
- Cell by cell mass balance information
- Kinematic wave approach for sheet flow
- Explicit solution of time discretization
- Rill development implemented in the model
- See related paper for details

Model structure:

$$\frac{\mathrm{d}h}{\mathrm{d}t} = (q_{in} + ep) - (q_{out} + inf + ret)$$

$$q_{in} = \sum \quad \text{sheet and rill flow}$$

$$q_{out} = \quad \text{sheet and rill flow}$$

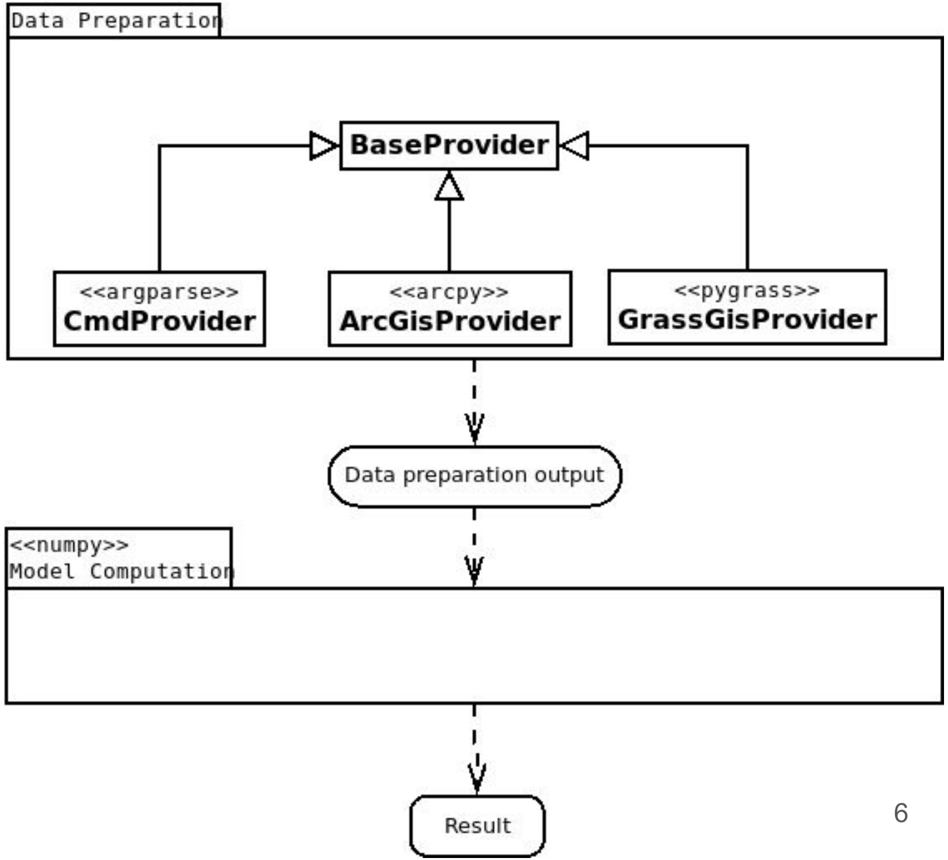
$$inf = \quad \text{Philips, Green-Ampt, Modified Green-Ampt, (Richars equation; future work)}$$

Workflow

1. **Data preparation (pre-processing)**
    a. DEM⟶calculate flow direction
    b. Soil map, land-use map ⟶ assign soil properties to each polygon
    c. Rainfall data ⟶ check the data for correctness
    d. Watercourse network (WN) ⟶check WN connection⟶ assign order to WN reaches
2. **Model computation**
    a. For each DEM cell and each time step:
        i. Calculate the current rainfall and infiltration
        ii. Calculate inflow from surrounding and outflow from current cell
        iii. Move to the next time step
3. **Data post-processing**
    a. Store rasters, vectors and text files with results

Ongoing development | Major goals

- Major refactoring: separate Data Preparation and Model Computation packages
- Data preparation provided by various software packages (ArcGIS 10.x/Pro, GRASS GIS, QGIS)
- Python 3 support
- Model computation parallelization experiments (ongoing, experimental)



Ongoing development | GIS tools

Options:

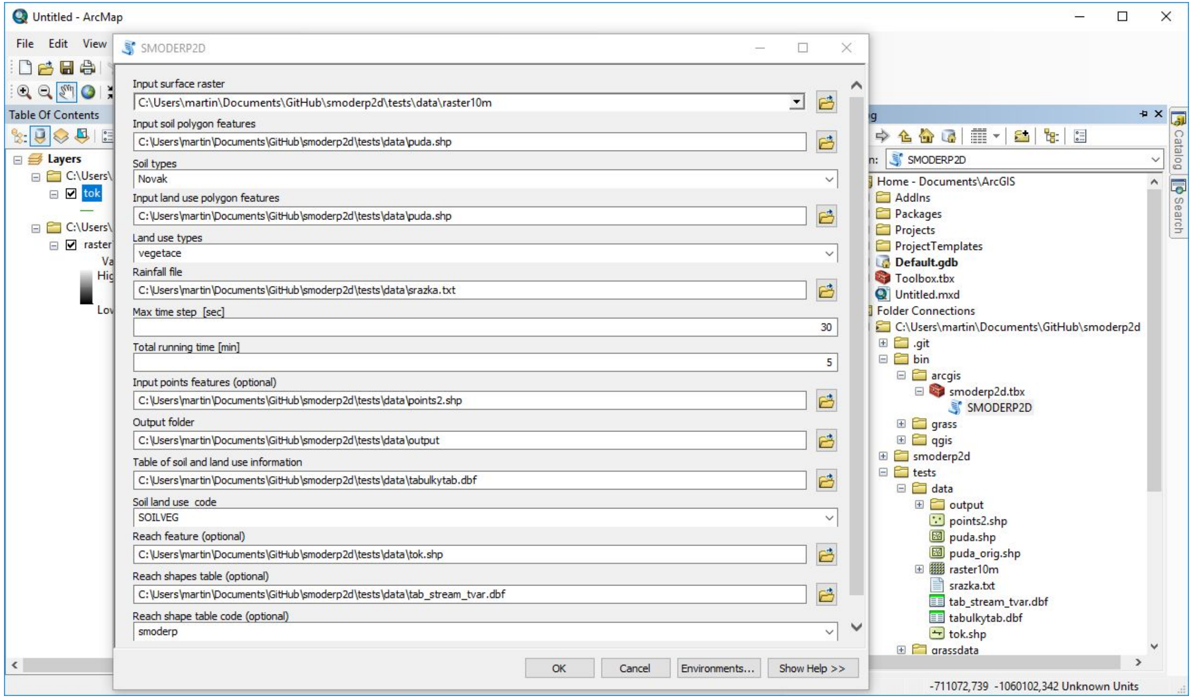
1. Perform data preparation (pre-processing) part only
2. Run model computation only
3. Perform full workflow (data preparation + model computation)

Supported platforms:

1. Esri ArcGIS 10.x (Pro)
2. GRASS GIS
3. QGIS

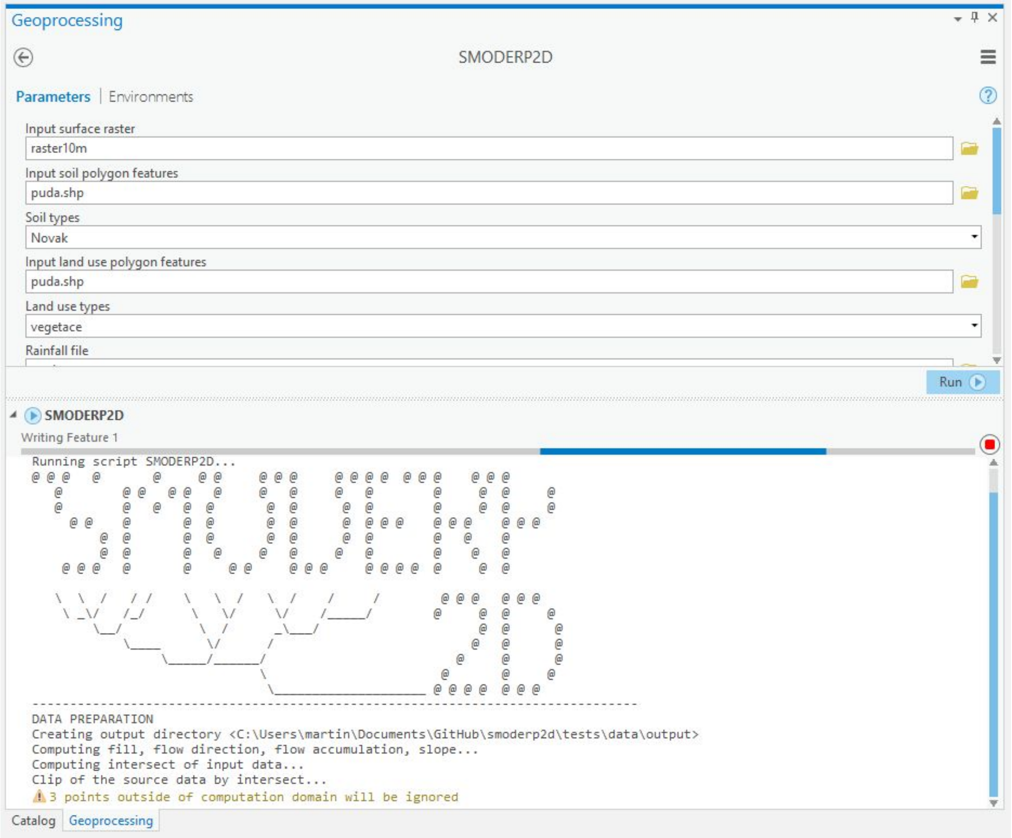
GIS tools | Esri ArcGIS

Toolbox for ArcGIS 10.x / Pro



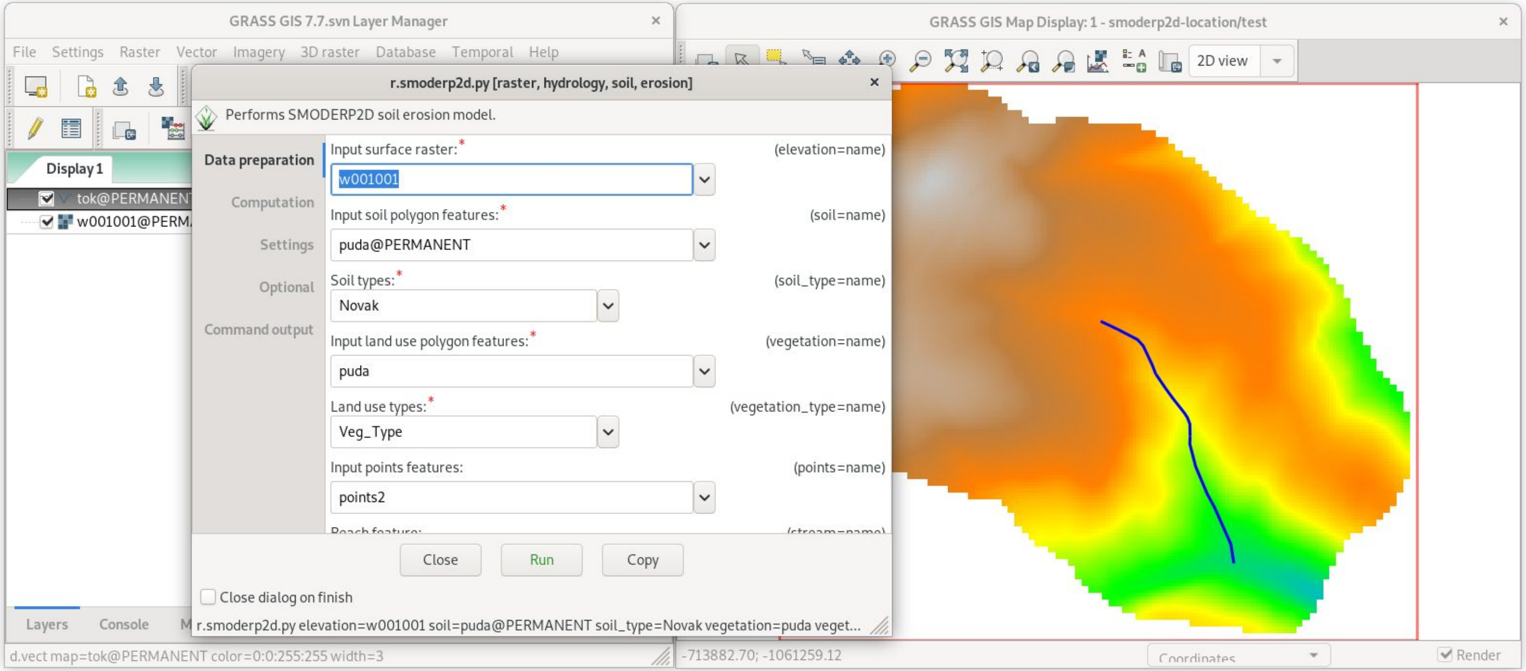
GIS tools | ArcGIS Toolbox

- Existing ArcGIS toolbox rewritten
- Data preparation done by ArcGIS GIS provider (arcpy)
- Added support for ArcGIS Pro (Python3)
- Available on GitHub



Ongoing development | GIS tools

GRASS GIS 7.8+ Addons



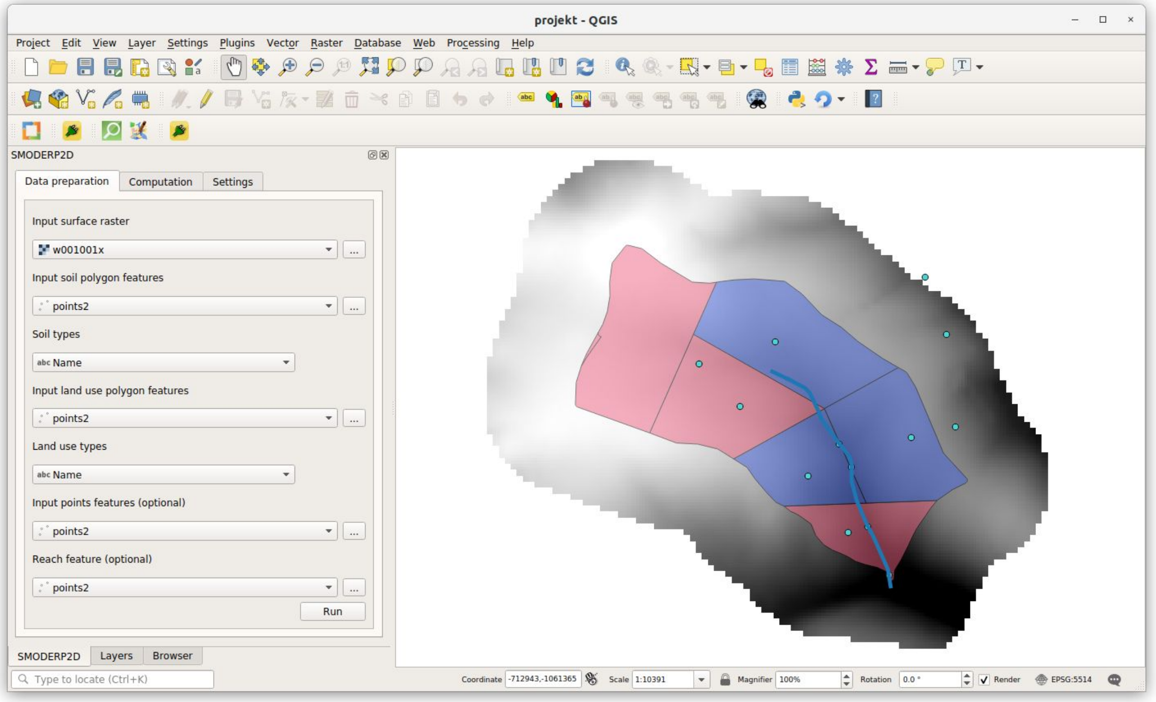
GIS tools | GRASS GIS Addons

- New GRASS Addons
  r.smoderp2d
- Available via g.extension
- Data preparation done
  using GRASS GIS provider
  (pygrass)
- GRASS GIS 7.8 (currently
  RC1) required (Python3)

```
# calculates to_node (fid of preceding segment)
self._add_field(stream, "to_node", "DOUBLE", -9999)
to_node = {}
with VectorTopo(stream) as stream_vect:
    for line in stream_vect:
        start, end = line.nodes()
        cat = line.cat
        for start_line in start.lines():
            if start_line.cat != cat:
                to_node[cat] = start_line.cat
```

Ongoing development | GIS tools

QGIS 3 Plugin



GIS tools | QGIS plugin

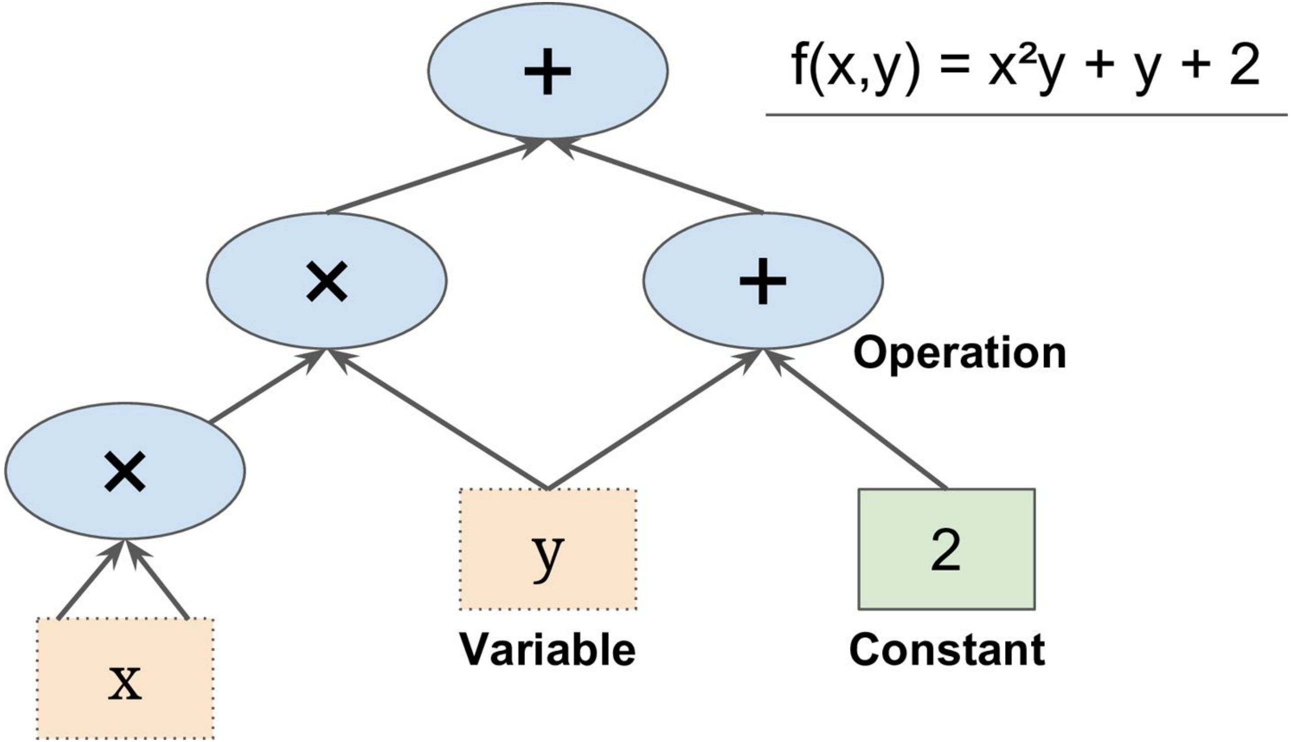
- New QGIS plugin
- QGIS 3.x required
- Available from GitHub
- Final release planned to uploaded into official QGIS repository
- Data preparation done using GRASS GIS provider (pygrass)
- GRASS GIS 7.8 required (Python3)

```
# search for GRASS GIS installation
grass7bin = fg()

# get input parameters from dialog
self._get_input_params()

try:
    runner = QGISRunner()
except ProviderError as e:
    raise ProviderError(e)

# import input data into temporary GRASS location
runner.import_data(self._input_params)

# perform data preparation and run the model in temporary
# GRASS location using GRASS GIS provider
runner.run()

# export result data from temporary GRASS location into
# output directory and display in QGIS map canvas
runner.show_results()
```

Ongoing development | Parallelization experiments

- Reasons
  - To reduce the computation time
  - For both CPU and GPU
- Approach
  - Loop-based computations rewritten to matrix-based ones
  - Mathematical operations rewritten to **TensorFlow** (2.0) or **NumPy**
    - Graph-based computations (independent math operations run in parallel)

Ongoing development | Parallelization experiments



Results of experimental parallelized branch

Results

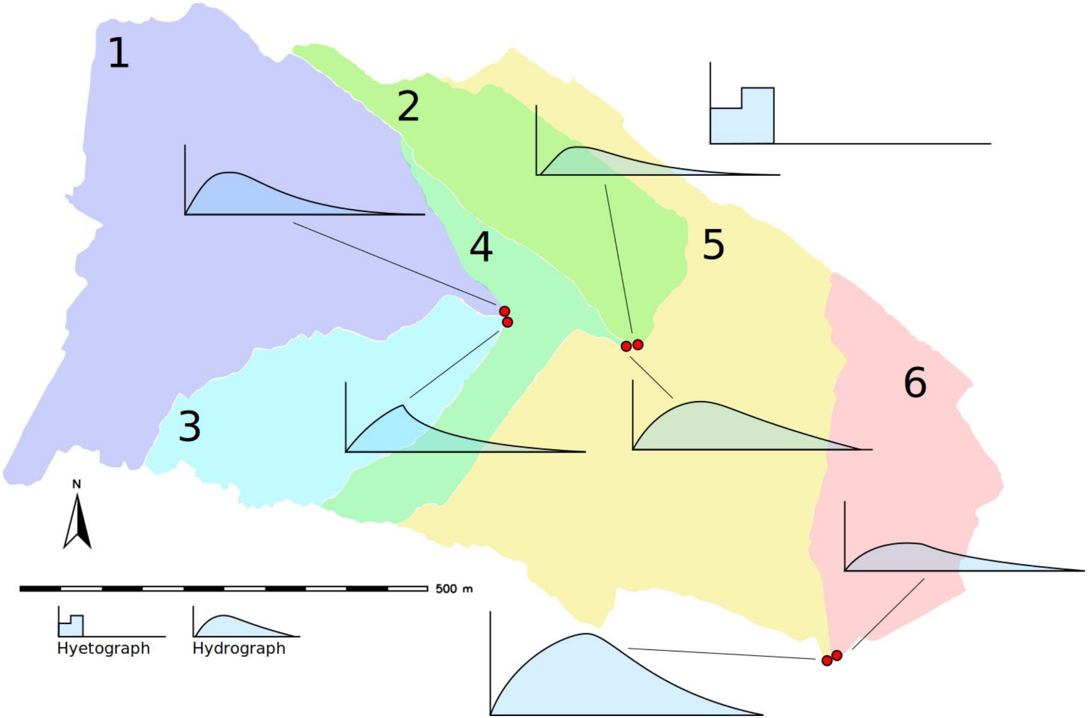
Table 1. Results of parallelization tests

RAM	Processing unit	Data 62 KB [s]	Data 197 MB [s]
15 GB	GPU1	4.0	7,560
	CPU1	0.2	12,809
	CPU2	2.1	7,249
251 GB	GPU2	2.5	6,611
	CPU3	0.2	10,637
	CPU4	1.5	8,631

Table 2. Used processing units

ID	Model	Clock speed	Memory
GPU1	GeForce GTX 1060 3GB	33 MHz	3,016 MiB
GPU2	4× GeForce GTX 1080 Ti	33 MHz	11,178 MiB
CPU1	AMD Ryzen 7 1700 Eight Core Processor	1.373 GHz	512 KB
CPU2	16× AMD Ryzen 7 1700 Eight Core Processor	1.373 GHz	512 KB
CPU3	Intel Xeon CPU E5-2630 v4	2.4 GHz	25,600 KB
CPU4	40× Intel Xeon CPU E5-2630 v4	2.4 GHz	25,600 KB

Ongoing development | Parallelization experiments



Sub-catchments based parallelization approach (planned)

Conclusion

- Ongoing development
  - Release candidates Q4/2019
  - Final release Q1/2020
- Join us on GitHub
  - Report issues

Thanks for attention!

The research has been supported by the research grants TJ01000270, QK1910029, and internal CTU grant SGS17/173/OHK1/T3/11.