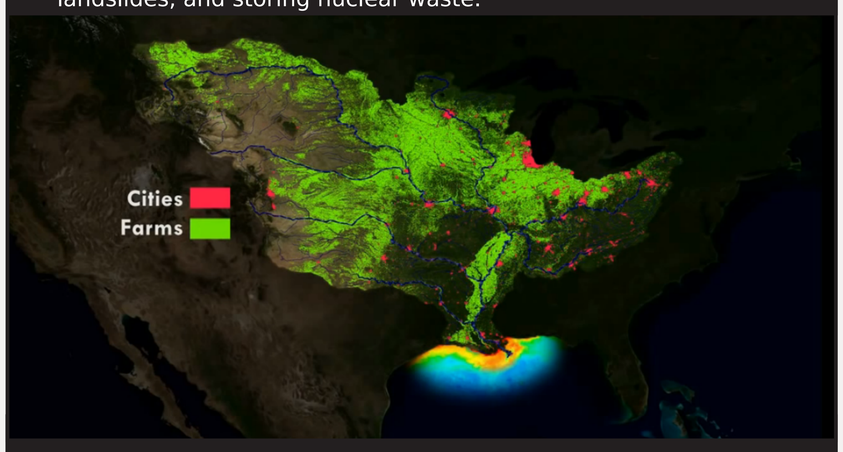
Communication-Avoiding Large Graph Algorithms for Flow Modeling

Richard Barnes
Energy & Resources Group
Electrical Engineering & Computer Science
UC Berkeley

1. Motivation

Many environmental problems require knowing **where and how water flows at large spatial and temporal scales.** These problems include remediating agricultural run-off, predicting landslides, and storing nuclear waste.

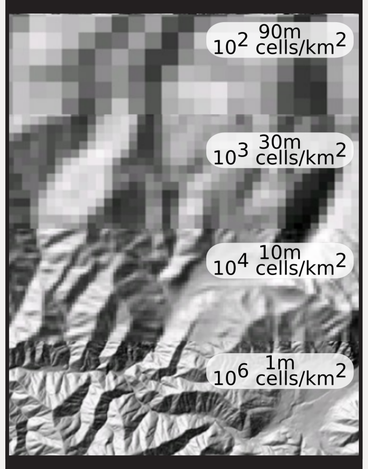


Run-off from 1.25M sq mi results in the Mississippi Dead Zone

Diverse users and applications...

Supercomputers & Laptops

...rapid increases in data resolution...



Trillions of Cells

....and extent make solving these problems challenging.

10s-100s of TBs

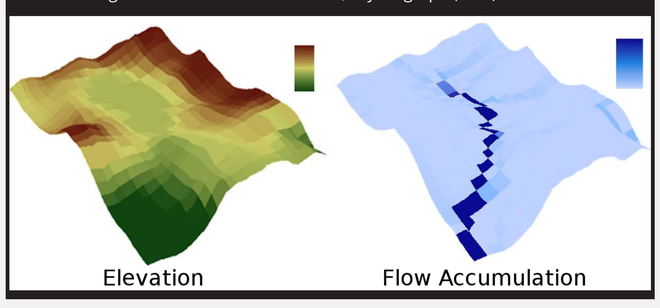
Research Question

Can we design
hydrological flow algorithms
suitable for both
supercomputers and laptops
which can process
continent-scale datasets at
high resolution (trillions of cells)
and rapidly solve
inverse problems
on smaller datasets
to answer previously
unsolvable questions?

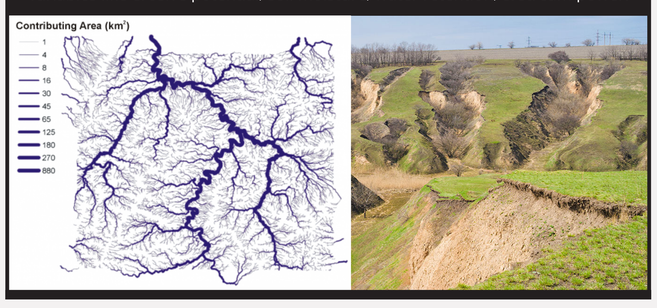
Yes!

2. Conceptual Model

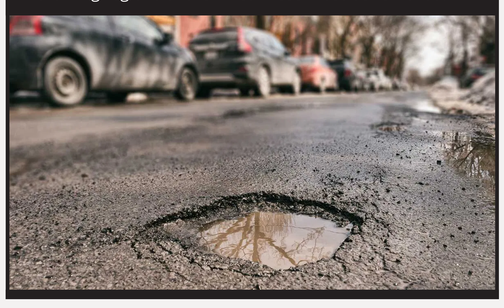
Digital elevation models (DEMs) are rectangular arrays of elevation data. In my application field, water is modeledwater as moving from higher cells to their lowest neighbours. This forms a directed, acyclic graph (DAG).



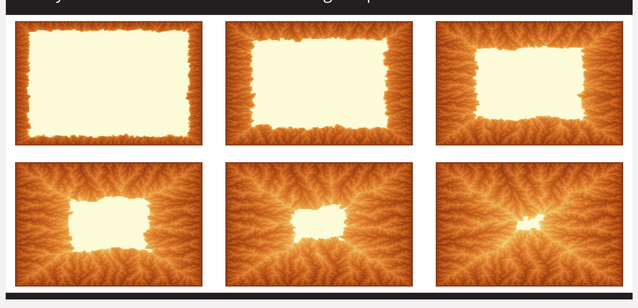
The DAG lets us calculate how much water passes through each cell: the Flow Accumulation. This allows us to understand key hydrologic & geomorphic variables like erosive potential, soil moisture, water retention, & stream power.



Both data noise and landscape features can create "pot-hole" depressions in the landscape, leading to inwardly-draining regions of the DAG. These must be removed.



On smaller datasets, coupling flow accumulation with an erosion equation lets us model the evolution of landscapes. Often we need to run such models many thousands of times to determine good parameter values.



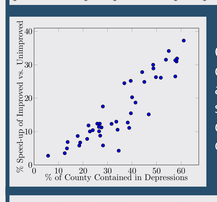
3. Methods

Source code: richdem.com

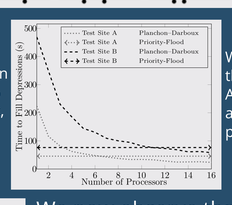
Depression-Filling

Barnes 2016

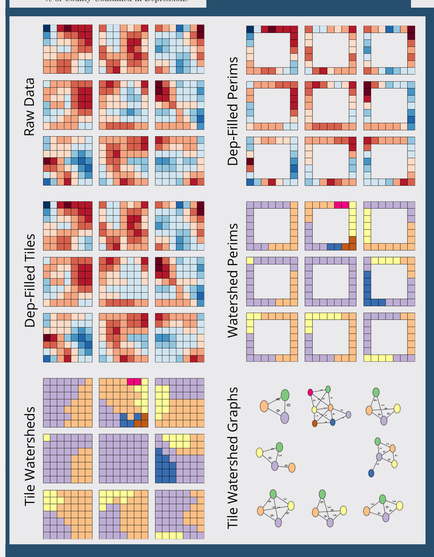
We start by developing a performant serial algorithm we call "Priority-Flood" (Barnes et al, 2014). The DEM's edge cells are placed in a priority-queue and the lowest cell is always popped. Its neighbours are then added. If a neighbour is lower than the popped cell, it is part of a depression. We eliminate the depression by raising the neighbour to the popped cell's level.



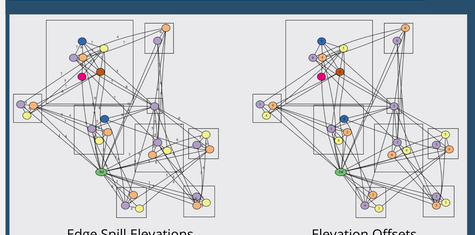
Cells that are in a depression can be placed in a plain queue for a 10-30% speed-up vs previous work, depending on the number of depressions.



When compared against the previous State of the Art in the field, our serial algorithm outperforms the parallel implementation.

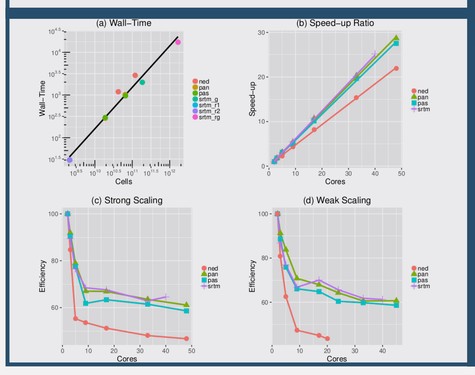


We now observe that each of the cells used to seed the priority-queue becomes the source of a "watershed": a set of cells which all flow downhill to the same point. If we were not filling depressions, they would form internally-draining watersheds. Therefore, the points where we beginning filling depressions always occur at the boundary of two watersheds. The watersheds within a tile form a graph whose edges are the elevations at which one watershed spills over into another. Therefore, retaining just the perimeter elevation and watershed information, plus these watershed graphs is sufficient to abstract the internal dynamics of a tile.



We gather the perimeter information into a single, large graph (21M nodes for our largest dataset) which is solved in serial. The result is a set of "offset values" representing the minimum elevation in each watershed. The tiles are then reprocessed and the minimum elevation applied to obtain the global solution. This means that we drive I/O to the theoretic minimum (1 read, 1 write) and require fixed communication events and size per tile regardless of the size of the input data.

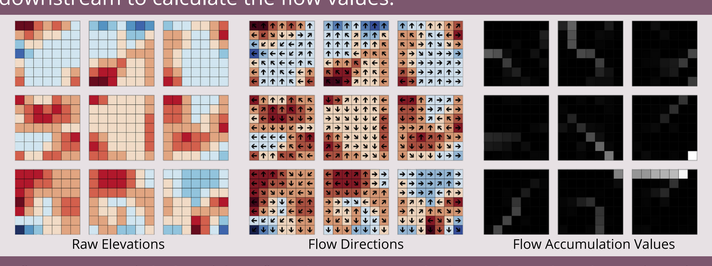
Our method's wall-time scales linearly across datasets from 1G-2Tcells requiring just 10s/Gcell and only 13GB and 84 seconds to solve the large graph problem.



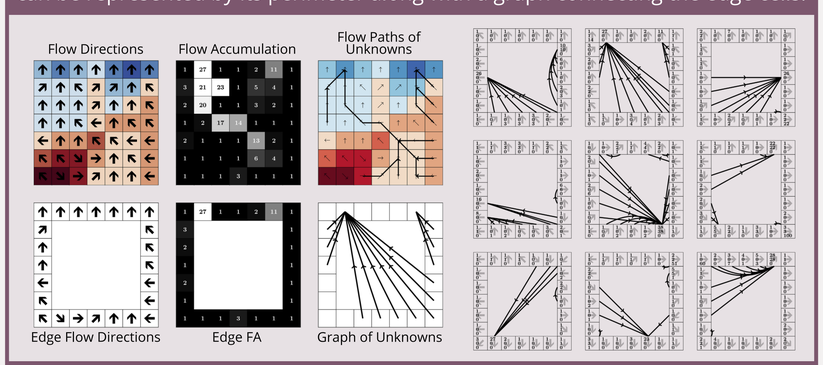
Flow Accumulation

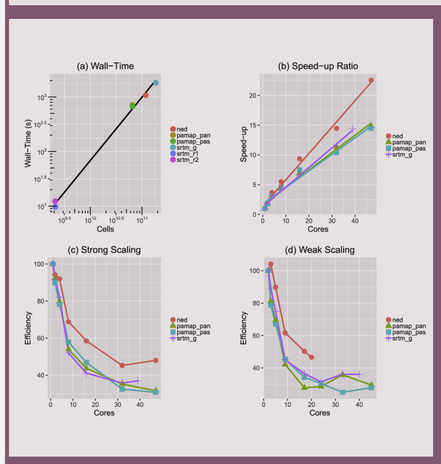
Barnes 2017

Calculating flow accumulation requires identifying which downstream neighbour(s) will receive a cell's flow and then accumulating flow from upstream to downstream to calculate the flow values.



A performant serial algorithm for Flow Accumulation places local maxima in a plain queue. As cells are popped from the queue, they gather-reduce their FA values from upstream neighbours, and place downstream neighbours with no remaining upstream dependencies into the queue. This a breadth-first traversal. In a tiled dataset, any local maxima contribute known values to downstream cells' FAs whereas cells on the border of the tile either emit information to neighbouring tiles or accept information in the form of unknowns. Thus, each tile can be represented by its perimeter along with a graph connecting the edge cells.





The large graph can, again, be gathered and solved in serial to determine the values of the unknown variables of each tile's perimeter. Each tile is then reprocessed (or pulled from cache) and the unknowns can be applied as offsets along the downstream flow paths of the unknown variables.

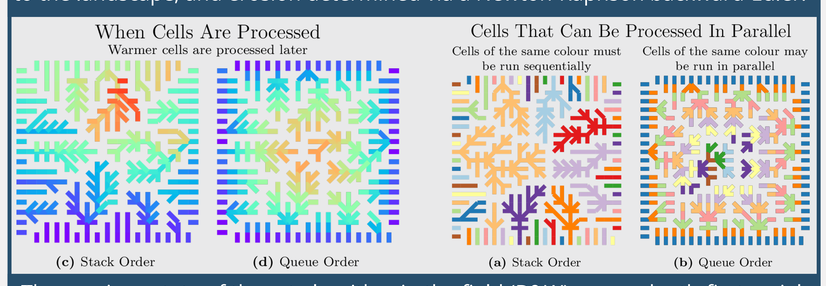
Our method's wall-time scales linearly across datasets from 1G-2T cells requiring 7s/Gcell and only 6GB and 19 seconds to solve the large graph problem.

For both flow accumulation and depression-filling the previous state of the art ceases to work at around 1Gcells. At this point my algorithms ran at least 6x faster, used 19x less bandwidth, and 70x less communication.

GPU Acceleration

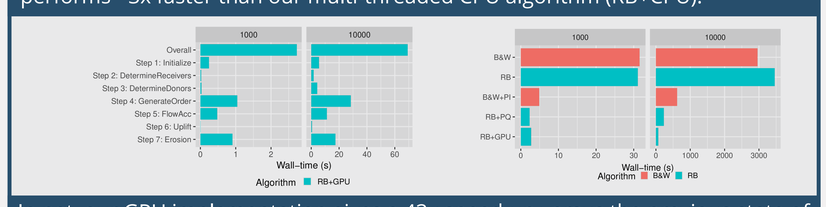
Barnes 2019

While the algorithms to the left solve large problems (up to 2Tcells) quickly, we sometimes need to solve smaller problems (up to 100M cells) even faster to simulate how flowing water affects a landscape, as depicted above. Doing so requires determining a cell processing order, which is facilitated by knowing what cells pass water downstream (donors) and receive water from upstream (receivers). Flow accumulation can be then be calculated, topographic uplift added to the landscape, and erosion determined via a Newton-Raphson backward-Euler.

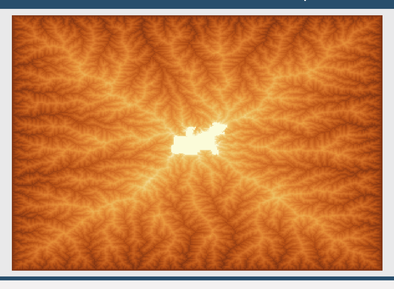


The previous state of the art algorithm in the field (B&W) uses a depth-first serial algorithm. Our serial breadth-first algorithm (RB) has similar performance, but facilitates parallelism. If we parallelize across depth first trees (B&W+PI) wall-time decreases, but long tails in some trees limit performance versus dividing the perimeter evenly between independently-running threads (RB+PQ).

On a GPU hardware restrictions and the sheer number of threads require load rebalancing, so we parallelize along the irregular wavefront that propagates from the edge (boundary condition) of the dataset. Our GPU algorithm (RB+GPU) performs ~3x faster than our multi-threaded CPU algorithm (RB+CPU).



In net, our GPU implementation gives a 43x speed-up versus the previous state of the art. The first attempt to use parameter-optimization to differentiate between competing theories of landscape evolution took place in 2018 and required over 1M compute hours, even though the grids used were only 500x500 cells. With our new algorithms the same calculations could be completed much faster.



Datasets 1000x larger. >6x faster. >19x less bandwidth. >70x less communication.

4. Conclusions

Previous algorithms for modeling hydrological flow at large spatial and temporal scales did not scale well. These algorithms required continuous communication and the ability to hold datasets of many terabytes in memory. In contrast, our new algorithms are able to abstract the computation to the perimeters of a tiled representation of the data. As a result, datasets of 2+TB can be processed using ~13GB of RAM - enough to fit on a single laptop.

Additionally, since only fixed communication and file I/O is required, our algorithms perform optimally on both laptops and supercomputers, making them accessible to users regardless of their computational resources. Such "algorithmic democracy" is important for ensuring that the benefits of algorithmic advances accrue to all users. Each paper is accompanied by complete, well-documented source code.

Finally, our algorithms can leverage emerging massive datasets and modern HPC to answer previously unaddressable questions involving global models of hydrology. We are now working to extend these techniques to groundwater and nonlinear flow regimes.

2014. Barnes, Lehman, Mulla. Priority-Flood: An Optimal Depression-Filling and Watershed-Labeling Algorithm for Digital Elevation Models. Computers & Geosciences 62:117-127. doi: 10.1016/j.cageo.2013.04.024

2016. Barnes. Parallel Priority-Flood Depression Filling For Trillion Cell Digital Elevation Models On Desktops Or Clusters. Computers & Geosciences 96: 56–68. doi: 10.1016/j.cageo.2016.07.001

2017. Barnes. Parallel Non-divergent Flow Accumulation For Trillion Cell Digital Elevation Models On Desktops Or Clusters. Environmental Modelling & Software: 92 202–212. doi: 10.1016/j.envsoft.2017.02.022

2019. Barnes. Accelerating a fluvial incision and landscape evolution model with parallelism. Geomorphology: 330. doi: 10.1016/j.geomorph.2019.01.002