Communication-Avoiding Large Graph Algorithms For Flow Modeling

Richard Barnes
richard.barnes@berkeley.edu
Electrical Engineering & Computer Science
Energy & Resources Group
Berkeley, CA

ABSTRACT

Large digital elevation models (DEMs) are used in geographic information systems (GIS) to model hydrology. Smaller DEMs, coupled with a governing equation, can be used to distinguish between competing theories of landscape evolution. However, existing algorithms scale poorly and don't use GPUs. To address this, I model hydrologic problems as graphs and develop new communication-avoiding algorithms which drive file I/O and communication to theoretic minimums. The result is parallel linearly-scaling algorithms for depression filling and flow accumulation—critical preprocessing steps in many GIS pipelines and landscape evolution models (LEMs). The algorithms exhibit 60% strong and weak scaling efficiencies up to 48 cores. The largest data set I test with has 2 trillion (10^{12}) cells: with 48 cores, processing required 4.8 hours of wall-time. This test is 1000 x larger than any previously performed in the literature. On datasets small enough for comparing against the old algorithms, the new algorithms use 19 x less bandwidth, 70 x less communication time, and 7 x less memory. Well-commented source code is available at (http://richdem.com).

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; • **Applied computing** → **Earth and atmospheric sciences**; • **Theory of computation** → *Design and analysis of algorithms*; *Graph algorithms analysis*; *Parallel algorithms*.

KEYWORDS

parallel computing; hydrology; geographic information system (GIS); terrain analysis

ACM Reference Format:
Richard Barnes. 2018. Communication-Avoiding Large Graph Algorithms For Flow Modeling. In *SC '20: November 9–19, 2020, Virtual Event*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SC '20, November 9–19, 2020, Virtual Event
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

DEMs represent terrain elevations as a rectangular array. They are used to estimate hydrologic and geomorphic properties, such as erosion [12]. Remote sensing can now produce meter-scale DEMs for the entire planet: the recently-released ArcticDEMv7 covers all land area north of 60° at 2 m resolution over several years giving a 20 PB DEM. How can we make sense of high-resolution data at this scale? Similarly, LEMs evolve smaller DEMs to compare geologic theories. This requires thousands of model realizations. A recent attempt to do this required over a million compute hours [6]. The key algorithms needed for DEM analysis, both large and small, are Depression-Filling and Flow Accumulation.

Existing algorithms [1, 8, 9, 11, 13, 15] for performing these operations on large DEMs either: (a) make extensive use of swap memory, causing cache and disk thrashing; or, (b) keep the DEM in RAM by using multiple compute nodes, requiring frequent communication [2]. Neither approach scales well. Here, I present new, scalable approaches to depression-filling, flow accumulation, and LEMs which outperform their predecessors (see Table 1 and Table 2).

2 ALGORITHMS

My algorithms use a producer-consumer architecture. Large DEMs are provided as rectangular tiles and the producer allocates these tiles. Consumers reduce tiles to a perimeter connected by a graph reducing each tile's information from $O(N^2)$ to $O(N)$. The producer uses perimeter information to calculate offsets. The consumers then adjust tiles' values using these offsets to obtain a global solution.

2.1 Depression Filling

Depression filling is accomplished by running the $O(N \log N)$ Priority-Flood algorithm [5] (Figure 1) on each tile (Figure 2a–c) in parallel. The tile's edge cells are added to a priority queue, each cell is the mouth of its own watershed (Figure 2c). The queue's lowest cell c is dequeued and its neighbours n are added to the queue. Each unvisited n inherits c's watershed label and is adjusted to be no lower than c. This has the effect of filling depressions. If n and c belong to different watersheds, the maximum of their elevations is noted. If this elevation is the lowest between the two watersheds, it is retained as

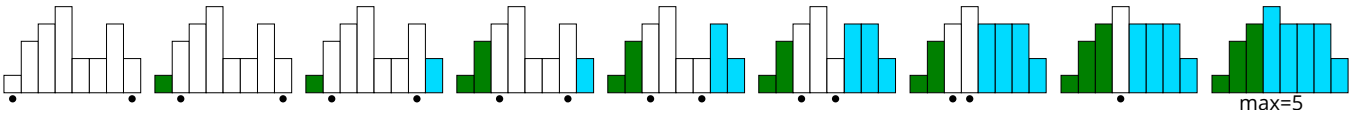


Figure 1: Running Priority-Flood on a single tile. Queued cells are represented by a black circle. Colours represent watersheds. See §2.1 for details. Figure drawn from [2].

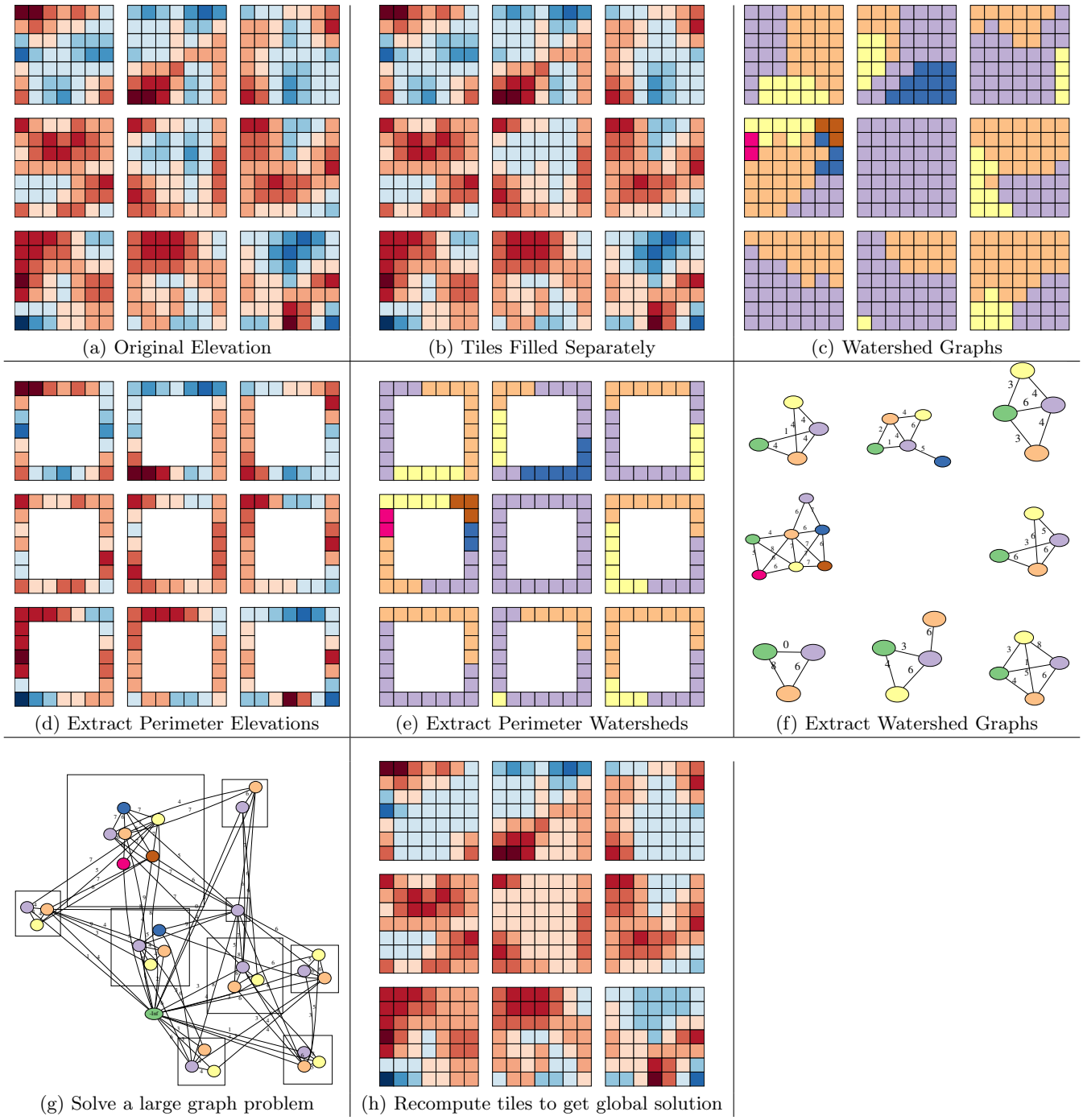


Figure 2: Parallel Priority-Flood.

Source	Year	Cells	Resolution	Dimensions	Adjective	Time (min)	Min/Cell
This work (RichDEM)	2016	$2 \cdot 10^{12}$	10 m	$\sim$1,291,715^2	*rather* large	287	$8 \cdot 10^{-9}$
Gomes et al. [8]	2012	$3 \cdot 10^{9}$	30 m	50,000 x 50,000	huge	58	$1 \cdot 10^{-8}$
Do et al. [7]	2011	$2 \cdot 10^{9}$	??	36,002 x 54,002	huge	??	
Yıldırım et al. [15] (TauDEM)	2015	$2 \cdot 10^{9}$	10 m	45,056 x 49,152	large	??	
Arge et al. [1] (GRASS)	2003	$1 \cdot 10^{9}$	10 m	33,454 x 31,866	massive	3720	$3 \cdot 10^{-6}$
Lindsay [9] (Whitebox GAT)	2015	$9 \cdot 10^{8}$	3 arc-sec	37,201 x 25,201	massive	8.6	$1 \cdot 10^{-8}$
Tesfa et al. [13]	2011	$6 \cdot 10^{8}$	??	24,856 x 24,000	large	20	$3 \cdot 10^{-8}$
Metz et al. [10, 11] (GRASS)	2010	$2 \cdot 10^{8}$	30 m	??	massive	32	$6 \cdot 10^{-7}$

Table 1: Statistics for depression filling from authors working with large DEMs. Times should not be directly compared due to hardware differences. The authors' description of the size of their data is also included. Some algorithms are part of larger terrain analysis suites, these are listed in parentheses.

the watersheds' spillover elevation. Full details are in Barnes et al. [5].

The result on a single tile is a graph of watersheds where each of the graph's edges is weighted by the watersheds' spillover elevations (Figure 2f). The producer gathers all the tiles' perimeter information and graphs (Figure 2d–f) and connects them using the elevations of the tiles' perimeters (Figure 2g). The resulting large graph can be solved in serial in <84 s and 13 GB RAM for a 2 Tcell DEM using Priority-Flood. Once solved, the minimum elevation for each watershed is known and can be applied in $O(N)$ time by the consumers. Afterward (Figure 2h), each tile is recomputed as in Figure 2a–c, but the offsets are applied.

See Barnes [2] and Figure 2 for details.

2.2 Flow Accumulation

After determining flow directions (Figure 3a), consumers calculate tiles' flow accumulation values (Figure 3b) by placing local maxima in a queue. Each cell popped from the queue passes its flow to one downstream cell. If this downstream cell has no upstream cells from which it is waiting to receive flow it is added to the queue. This terminates in $O(N)$ time when the queue is empty (Figure 3b).

Flow coming into a tile from the outside would add an offset along its flow path until the flow exits the tile (Figure 3c). Each cell touched by a flow path is therefore offset from its true value. Each tile sends its edge elevations, known flow accumulations for the edge cells, and connections between edge cells to the producer (Figure 3d–f). These form a large graph (Figure 3h) which can be solved in serial in 19 s with 6 GB RAM on a 2 Tcell DEM to produce offsets that are applied to each tile to obtain the global solution (Figure 3i).

See Barnes [3] and Figure 3 for further details.

2.3 GPU Acceleration

The LEMs in [6] used an $O(N^2)$ Python-based depression-filling algorithm. The serial algorithm in §2.1 runs 25,000 x faster. In previous work, LEMs modeled erosion (Figure 4a) using Newton-Raphson to solve a backward-Euler equation working their way upstream using a serial depth-first traversal (Figure 4b). We use a breadth-first traversal (Figure 4c) to gain additional parallelism (Figure 4d) when processing the irregular wavefront formed by the equation's moving boundary condition and implement this on a GPU (Figure 4e, RB+GPU). This gives a 43 x speed-up versus the previous state of the art (Figure 4e, B&W) and a 3 x speed-up versus a CPU-parallel method we develop as a baseline (Figure 4e, RB+PQ).

See Barnes [4] and Figure 4 for further details.

	This work	EMFlow	TauDEM
Depression filling			
Wall-time (s)	23	494	144
RAM used (GB)	5	2	37
Comm. Time (s)	82	-	5729
Comm. Size (MB)	46	-	887
Flow accumulation			
Wall-time (s)	24	658	42
RAM used (GB)	7	2	21
Comm. Time (s)	163	-	1256
Comm. Size (MB)	30	-	556

Table 2: Comparisons of the algorithm versus the previous state of the art. Tests were performed using XSEDE's Comet [14] (NSF Grant No. ACI-1053575).

REFERENCES

[1] L. Arge, J.S. Chase, P. Halpin, L. Toma, J.S. Vitter, D. Urban, and R. Wickremesinghe. 2003. Efficient flow computation on massive grid terrain datasets. *GeoInformatica* 7, 4 (2003), 283–313. https://doi.org/10.1023/A:1025526421410

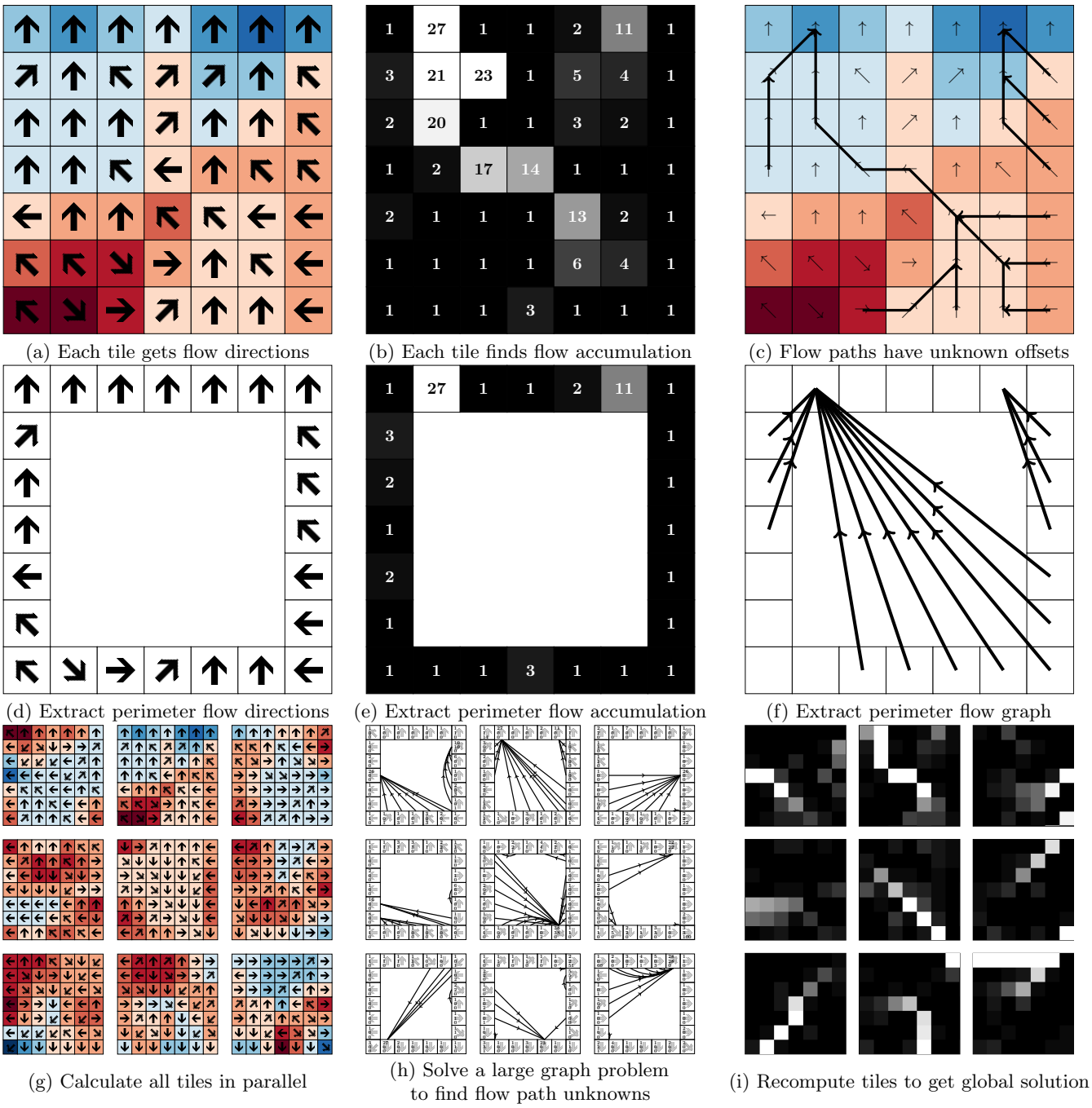
[2] Richard Barnes. 2016. Parallel priority-flood depression filling for trillion cell digital elevation models on desktops or clusters. *Computers & Geosciences* 96 (Nov. 2016), 56–68. https://doi.org/10.1016/j.cageo.2016.07.001

[3] Richard Barnes. 2017. Parallel non-divergent flow accumulation for trillion cell digital elevation models on desktops or clusters. *Environmental Modelling & Software* 92 (June 2017), 202–212. https://doi.org/10.1016/j.envsoft.2017.02.022

[4] Richard Barnes. 2019. Accelerating a fluvial incision and landscape evolution model with parallelism. *Geomorphology* 330 (2019), 28–39.

[5] Richard Barnes, Clarence Lehman, and David Mulla. 2014. Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models. *Computers & Geosciences* 62 (2014), 117 – 127. https://doi.org/10.1016/j.cageo.2013.04.024

[6] Katherine R Barnhart, Gregory E Tucker, Sandra Doty, Charles M Shobe, Rachel C Glade, Matthew W Rossi, and Mary C Hill. 2020. Inverting topography for landscape evolution model process representation: Part 1, conceptualization and sensitivity analysis. *Journal of Geophysical Research: Earth Surface* (2020),



(a) Each tile gets flow directions

(b) Each tile finds flow accumulation

(c) Flow paths have unknown offsets

(d) Extract perimeter flow directions

(e) Extract perimeter flow accumulation

(f) Extract perimeter flow graph

(g) Calculate all tiles in parallel

(h) Solve a large graph problem to find flow path unknowns

(i) Recompute tiles to get global solution

Figure 3: Parallel Flow Accumulation.

e2018JF004961.

[7] Hiep-Thuan Do, Sébastien Limet, and Emmanuel Melin. 2011. Parallel Computing Flow Accumulation in Large Digital Elevation Models. *Procedia Computer Science* 4 (2011), 2277–2286. https://doi.org/10.1016/j.procs.2011.04.248

[8] Thiago L. Gomes, Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Guilherme C. Pena. 2012. Computing the drainage network on huge grid terrains. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data* (Redondo Beach, California) *(BigSpatial '12)*. ACM, New York, NY, USA, 53–60. https://doi.org/10.1145/2447481.2447488

[9] John B. Lindsay. 2015. Efficient hybrid breaching-filling sink removal methods for flow path enforcement in digital elevation models: Efficient Hybrid Sink Removal Methods for Flow Path Enforcement. *Hydrological Processes* (2015). https://doi.org/10.1002/hyp.10648

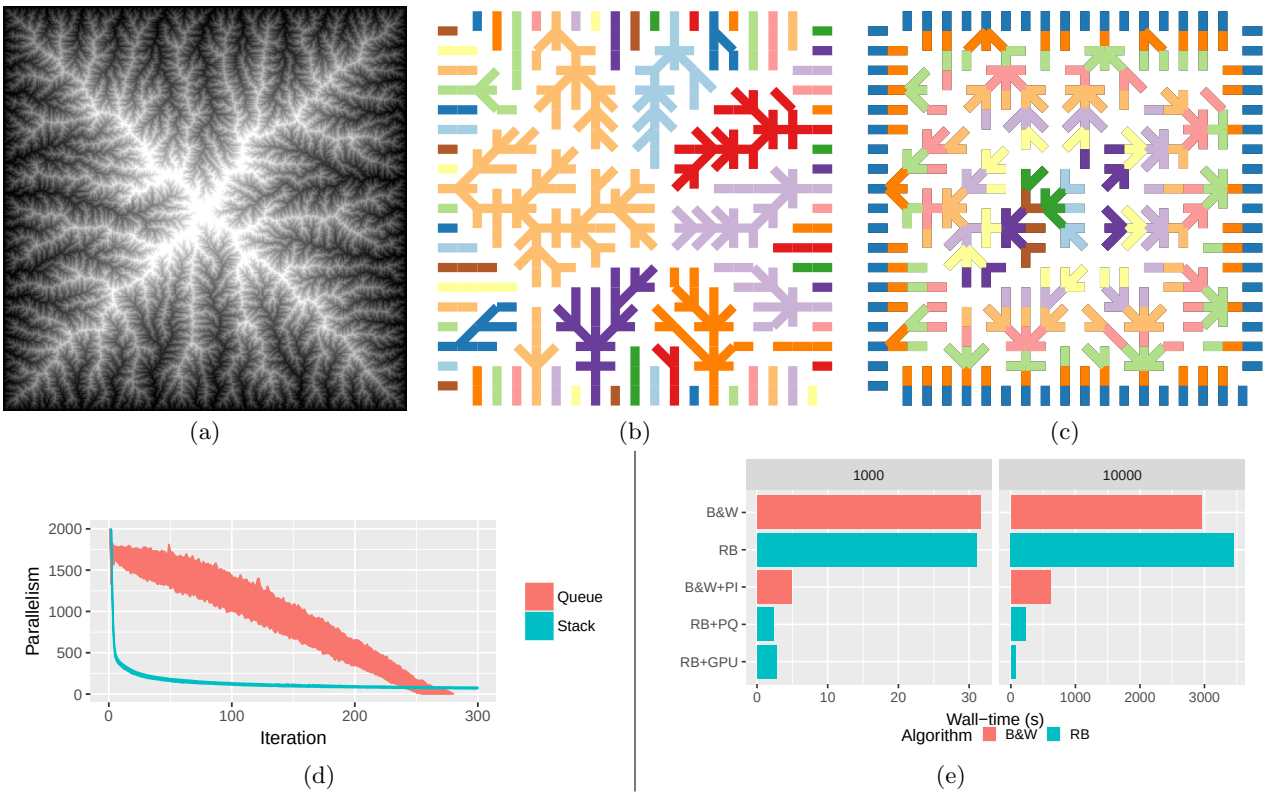


Figure 4: Landscape evolution models.

[10] M. Metz, H. Mitasova, and RS Harmon. 2010. Accurate stream extraction from large, radar-based elevation models. *Hydrology and Earth System Sciences Discussions* 7 (2010), 3213–3235. https://doi.org/10.5194/hessd-7-3213-2010

[11] M. Metz, H. Mitasova, and RS Harmon. 2011. Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search. *Hydrology and Earth System Sciences* 15, 2 (2011), 667. https://doi.org/10.5194/hess-15-667-2011

[12] F. Nardi, S. Grimaldi, M. Santini, A. Petroselli, and L. Ubertini. 2008. Hydrogeomorphic properties of simulated drainage patterns using digital elevation models: the flat area issue/Propriétés hydro-géomorphologiques de réseaux de drainage simulés à partir de modèles numériques de terrain: la question des zones planes. *Hydrological Sciences Journal* 53, 6 (2008), 1176–1193. https://doi.org/10.1623/hysj.53.6.1176

[13] Teklu K. Tesfa, David G. Tarboton, Daniel W. Watson, Kimberly A.T. Schreuders, Matthew E. Baker, and Robert M. Wallace. 2011. Extraction of hydrological proximity measures from DEMs using parallel processing. *Environmental Modelling & Software* 26, 12 (Dec. 2011), 1696–1709. https://doi.org/10.1016/j.envsoft.2011.07.018

[14] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, et al. 2014. XSEDE: accelerating scientific discovery. *Computing in Science & Engineering* 16, 5 (2014), 62–74.

[15] Ahmet Artu Yıldırım, Dan Watson, David Tarboton, and Robert M. Wallace. 2015. A virtual tile approach to raster-based calculations of large digital elevation models in a shared-memory system. *Computers & Geosciences* 82 (2015), 78 – 88. https://doi.org/10.1016/j.cageo.2015.05.014