Computers & Geosciences 96 (2016) 56–68

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

Parallel Priority-Flood depression filling for trillion cell digital elevation models on desktops or clusters

Richard Barnes

Energy & Resources Group, Berkeley, USA

A R T I C L E I N F O

Article history:
Received 28 January 2016
Received in revised form
17 June 2016
Accepted 1 July 2016
Available online 4 July 2016

Keywords:
Parallel computing
Hydrology
Geographic information system (GIS)
Pit filling
Sink removal

A B S T R A C T

Algorithms for extracting hydrologic features and properties from digital elevation models (DEMs) are challenged by large datasets, which often cannot fit within a computer's RAM. Depression filling is an important preconditioning step to many of these algorithms. Here, I present a new, linearly scaling algorithm which parallelizes the Priority-Flood depression-filling algorithm by subdividing a DEM into tiles. Using a single-producer, multi-consumer design, the new algorithm works equally well on one core, multiple cores, or multiple machines and can take advantage of large memories or cope with small ones. Unlike previous algorithms, the new algorithm guarantees a fixed number of memory access and communication events per subdivision of the DEM. In comparison testing, this results in the new algorithm running generally faster while using fewer resources than previous algorithms. For moderately sized tiles, the algorithm exhibits ~60% strong and weak scaling efficiencies up to 48 cores, and linear time scaling across datasets ranging over three orders of magnitude. The largest dataset on which I run the algorithm has 2 trillion (2×10^{12}) cells. With 48 cores, processing required 4.8 h wall-time (9.3 compute-days). This test is three orders of magnitude larger than any previously performed in the literature. Complete, well-commented source code and correctness tests are available for download from a repository.

1. Introduction

Digital elevation models (DEMs) are representations of terrain elevations above or below a chosen zero elevation. Raster DEMs, in which the data are stored as a rectangular array of floating-point or integer values, are widely used in geospatial analysis for estimating a region's hydrologic and geomorphic properties, including soil moisture, terrain stability, erosive potential, rainfall retention, and stream power. Many algorithms for extracting these properties require that, by following flow directions downhill from one cell to another, it is always possible to reach the edge of the DEM.

Depressions (see Lindsay, 2016 for a typology) are inwardly draining regions of a DEM which have no outlet and, therefore, confound such algorithms. Although depressions may be representative of natural terrain, such as in the Prairie Pothole Region of the United States, they may also result from technical issues in the DEM's collection and processing, such as from biased terrain reflectance or conversions from floating-point to integer precision Nardi et al. (2008). Note that depressions are distinct from pits, which are single DEM cells whose neighbors all have a higher elevation.

Depressions may be dealt with by filling them into the level of their lowest outlet, as will be done here. Several authors have argued that this approach produces inferior results compared to approaches which either solely breach depression walls or combine breaching and filling (Lindsay, 2016; Martz and Garbrecht, 1998; Grimaldi et al., 2007; Lindsay and Creed, 2005; Danner et al., 2007). As a particularly egregious example of a situation in which breaching would be better, Metz et al. (2010) show one river along which 92% of cells were adjusted by depression-filling. However, a DEM may be modified extensively without compromising results, depending on the nature of the analysis being done. Additionally, breaching and hybrid approaches continue to lag behind recent developments in depression-filling, including the one described here, both in terms of execution times and the size of the DEM it is possible to process.

For a given DEM Z, depression-filling, such as described by this paper, produces a new DEM W defined by the following criteria (Planchon and Darboux, 2002):

1. The elevation of each cell of W is greater than or equal to its corresponding cell in Z.
2. For each cell c of W, there is a path that leads from c to the boundary by moving downwards by an amount of at least ϵ between any two cells on the path, where ϵ may be zero. Such a path is referred to as an ϵ-descending path.

E-mail address: richard.barnes@berkeley.edu

http://dx.doi.org/10.1016/j.cageo.2016.07.001
0098-3004/© 2016 Elsevier Ltd. All rights reserved.

Table 1
DEM sizes, dimensions, and processing times for authors working with large DEMs. The table should be used only to develop a sense of the maximum sizes and the range of times it can take to process large DEMs. Times between algorithms should not be directly compared as different hardware has been used in all cases and different operations have been performed in many cases. For instance, Yildirim et al. (2015) perform depression-filling while Lindsay (2016) performs depression breaching. The authors' description of the size of their data is also included; all authors used "large". Some algorithms are part of larger terrain analysis suites, these are listed in parentheses.

Source	Year	Cells	Resolution	Dimensions	Adjective	Time (min)	Min/cell
This paper (RichDEM)	2016	2×10^{12}	10 m	$\sim$1,291,715^2	*Rather* large	287	8×10^{-9}
Gomes et al. (2012)	2012	3×10^{9}	30 m	$50{,}000\times50{,}000$	Huge	58	1×10^{-8}
Do et al. (2010)	2010	2×10^{9}	??	$36{,}002\times54{,}002$	Huge	21	1×10^{-8}
Do et al. (2011)	2011	2×10^{9}	??	$36{,}002\times54{,}002$	Huge	??	
Yildirim et al. (2015) (TauDEM)	2015	2×10^{9}	10 m	$45{,}056\times49{,}152$	Large	??	
Arge et al. (2003) (GRASS)	2003	1×10^{9}	10 m	$33{,}454\times31{,}866$	Massive	3720	3×10^{-6}
Lindsay (2016) (Whitebox GAT)	2015	9×10^{8}	3 arc-sec	$37{,}201\times25{,}201$	Massive	8.6	1×10^{-8}
Tesfa et al. (2011)	2011	6×10^{8}	??	$24{,}856\times24{,}000$	Large	20	3×10^{-8}
Wallis et al. (2009) (TauDEM)	2009	4×10^{8}	??	$14{,}949\times27{,}174$	Large	8	2×10^{-8}
Danner et al. (2007)	2007	3×10^{8}	3 m	??	Massive	445	1×10^{-6}
Metz et al. (2010, 2011) (GRASS)	2010	2×10^{8}	30 m	??	Massive	32	6×10^{-7}

3. W is the lowest surface allowed by properties (1) and (2).

This paper considers only the most common case wherein $\epsilon=0$. Setting $\epsilon>0$ requires more complex methods than those described here.

DEMs have increased in resolution from 30 to 90 m in the recent past to the sub-meter resolutions becoming available today. Increasing resolution has led to increased data sizes: current DEMs are on the order of gigabytes and increasing, with billions of cells. Even in situations where only comparatively low-resolution data is available, a DEM may cover large areas: 30 m Shuttle Radar Topography Mission (SRTM) elevation data has been released for 80% of Earth's landmass (Farr et al., 2007). While computer processing and memory performance have increased appreciably, development of algorithms suited to efficiently manipulating large DEMs is on-going.

If a DEM can fit into the RAM of a single computer, several algorithms exist which can efficiently perform depression-filling operations (see Barnes et al., 2014b for a review and Zhou et al., 2016 for the latest work in this area). If a DEM cannot fit into the RAM of a single computer, other approaches are needed.

In this paper, I will argue that existing approaches are inefficient and do not scale well. I will then present a new algorithm which overcomes the problems identified. The new algorithm is able to efficiently fill depressions in DEMs with more than a trillion cells and will work on both single-core machines and supercomputers. The algorithm achieves this by subdividing not just the data, but the problem itself: it is able to limit communication to a fixed number of events per subdivision and I/O to a fixed number of events per DEM cell. The algorithm may also offer efficiency advantages even if a DEM can fit entirely into RAM.

2. Background

Existing algorithms have taken one of the two approaches to DEMs that cannot fit entirely into RAM. They either (a) keep only a subset of the DEM in RAM at any time by using virtual tiles stored to a computer's hard disk or (b) keep the entire DEM in RAM by distributing it over multiple compute nodes which communicate with each other. I argue here that existing algorithms pay high costs in terms of disk access and/or communication which prevent them from scaling well; the new algorithm pays much lower costs.

Table 1 lists several authors mentioned here who have developed algorithms specifically for large DEMs. The sizes of the largest DEMs they test are listed, along with their choice of adjective to describe this size. Gigacell (10^9 cells) DEMs represent the upper limit of these tests. Here, I will go further than "massive" and bigger than "huge" by testing a trillion cell, or teracell (10^{12} cells), DEM. After ruling out "ginormous", I refer to this new size class as being *rather* large.

2.1. Virtual tiles

The virtual tile approach subdivides a DEM into tiles, a limited number of which can fit into RAM at a given time. When the RAM is full, tiles which are not being used are written to the hard disk. Virtual tiles are advantageous because they can be easily incorporated into any existing algorithm by modifying the algorithm so that it accesses data through a tile manager. The tile manager maps cells to tiles and, if the tile is not in memory, retrieves it, possibly writing an old tile to disk first. Since hard disk access is slow, existing algorithms reduce I/O by favoring access to nearby rather than distant cells. This helps increase the locality of access, which is favourable for caching. Unfortunately, virtual tile algorithms are unable to make strong locality guarantees and therefore, are ultimately unable to limit how often a particular tile will be loaded into memory.

Arge et al. (2003), whose work is encapsulated in the TerraFlow[1] package and included with GRASS (GRASS Development Team, 2016), were one of the first to examine I/O efficient algorithms for depression-filling (among other operations). As discussed in their paper, since disk access is costly, blocks of data are read from memory in an attempt to amortize this cost. Arge et al. describe a depression-filling algorithm which is bounded by $O(N\log N)$ I/Os and $O(N\log N)$ operations (see their paper and Aggarwal and Vitter, 1988 for a more exact description of the access complexity). Details of the algorithm's memory management are not described. They compared the speed of their algorithm against ArcInfo 7.1.2 (an industry-standard for the time) and achieved run-times twice as fast and completed larger problems. Danner et al. (2007) describe an algorithm similar to Arge et al. (2003), but theirs performed a breaching operation on depressions.

Metz et al. (2011) present a Priority-Flood (Barnes et al., 2014b) depression-breaching algorithm (now included with GRASS). The algorithm uses the GRASS segment library as a tile manager and, in comparison testing, achieves run-times almost twice as fast as Arge et al. (2003), though the authors note that they expect that the algorithm by Arge et al. (2003) would be faster on larger datasets.

Gomes et al. (2012) present a virtual tile approach using an $O(N)$ integer variant Priority-Flood in their EMFlow package.[2] The DEM is subdivided into tiles accessed via a tile manager. Tiles are

[1] http://www.cs.duke.edu/geo*/terraflow/
[2] https://github.com/guipenaufv/EMFlow