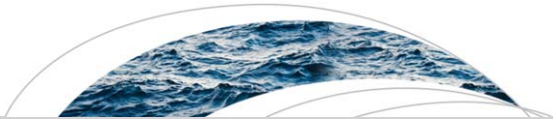
AGU PUBLICATIONS

Water Resources Research



RESEARCH ARTICLE

10.1002/2013WR014326

Key Points:
- Iterative methods can be used to scalably calculate drainage area in parallel
- Parallel computing is necessary to exploit large scale high-resolution DEMs
- The proposed method scales well to large numbers of processors

Correspondence to:
A. Richardson,
alan_r@mit.edu

Citation:
Richardson, A., C. N. Hill, and J. T. Perron (2014), IDA: An implicit, parallelizable method for calculating drainage area, *Water Resour. Res.*, *50*, 4110–4130, doi:10.1002/2013WR014326.

Received 25 JUNE 2013
Accepted 25 APR 2014
Accepted article online 2 MAY 2014
Published online 21 MAY 2014

IDA: An implicit, parallelizable method for calculating drainage area

Alan Richardson[1], Christopher N. Hill[1], and J. Taylor Perron[1]

[1]Department of Earth, Atmospheric and Planetary Sciences, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

Abstract Models of landscape evolution or hydrological processes typically depend on the accurate determination of upslope drainage area from digital elevation data, but such calculations can be very computationally demanding when applied to high-resolution topographic data. To overcome this limitation, we propose calculating drainage area in an implicit, iterative manner using linear solvers. The basis of this method is a recasting of the flow routing problem as a sparse system of linear equations, which can be solved using established computational techniques. This approach is highly parallelizable, enabling data to be spread over multiple computer processors. Good scalability is exhibited, rendering it suitable for contemporary high-performance computing architectures with many processors, such as graphics processing units (GPUs). In addition, the iterative nature of the computational algorithms we use to solve the linear system creates the possibility of accelerating the solution by providing an initial guess, making the method well suited to iterative calculations such as numerical landscape evolution models. We compare this method with a previously proposed parallel drainage area algorithm and present several examples illustrating its advantages, including a continent-scale flow routing calculation at 3 arc sec resolution, improvements to models of fluvial sediment yield, and acceleration of drainage area calculations in a landscape evolution model. We additionally describe a modification that allows the method to be used for parallel basin delineation.

1. Introduction

Hydrological flow routing models predict the path of surface runoff, and thus permit the delineation of drainage networks. Flow routing calculations are required in applications ranging from investigations of erosion and landscape evolution, where they are used to estimate flow discharge and fluvial sediment loads [e.g., *Syvitski et al*., 2005], to precision agriculture, where they are a means of inferring soil wetness [*Schmidt and Persson*, 2003]. The multiscale nature of drainage networks, in which convergent flows from hillslopes join to form tributary networks, which eventually merge to form continent-scale river networks, implies that accurate flow routing requires high-resolution data over large areas.

1.1. Drainage Area

One quantity that occurs frequently in hydrological calculations is the drainage area, *A*. The drainage area of a patch of terrain refers to the horizontal (map-view) area of the landscape that drains to that patch. Equivalently, it is the area of the landscape that is upslope from the patch. Drainage area can be estimated using only information about flow directions that can be inferred from land surface elevations, and it is therefore often used to estimate other quantities that are difficult to measure directly. It has been shown, for example, that *A* is one of the main variables that influences sediment loads in large rivers [*Milliman and Syvitski*, 1992; *Mulder and Syvitski*, 1996; *Syvitski and Milliman*, 2007]. Another use is in catchment analysis, where its correlation with channel slope is interpreted as a signature of erosional processes [*Willgoose et al*., 1991] and the transition from hillslope to valley morphology [*Tarboton et al*., 1991]. Drainage area is also commonly used as a surrogate for quantities such as water discharge and channel width in landscape evolution models, a set of approximations with strong empirical support that allow erosion and sediment transport rates to be expressed in terms of the topography [e.g., *Howard*, 1994; *Willgoose*, 2005; *Tucker and Hancock*, 2010].

1.2. Need for Higher Resolution and Current Limitations

Flows accumulate downstream, making drainage area a fundamentally nonlocal quantity that can depend on the topography in distant parts of the landscape. At the same time, flow directions depend sensitively

on local topography, making accurate determination of drainage area dependent on the resolution and accuracy of the topographic data. In addition, calculations that involve drainage area also frequently use local topographic quantities, such as slope or curvature, that depend even more sensitively on the fidelity of the topographic data. Finer resolution topographic data generally allow more accurate calculation of relevant topographic quantities, avoid the omission of small drainage basins and other fine-scale features when performing large-scale calculations, and reduce the likelihood of significant flow routing errors.

Remotely sensed, high-resolution elevation data are already available at continent scale. Advances in the acquisition of digital elevation model (DEM) data, the basis of many hydrological calculations, have led to dramatic increases in the available resolution. Regional data sets, such as the USGS National Elevation Dataset (NED) [*Gesch et al*., 2009], are available at horizontal resolutions down to 3 m in some locations, and the widespread acquisition of airborne laser altimetry is expanding coverage at even finer resolutions to entire states and countries [*Glennie et al*., 2013]. The SRTM [*Farr et al*., 2007] and ASTER [*Hirano et al*., 2003] projects have created global DEMs with resolutions of less than 100 m, and even finer resolution is expected from the TanDEM-X mission [*Zink et al*., 2008]. This rapid rise in data availability provides the opportunity to perform continent-scale flow routing calculations with unprecedented accuracy. As we explain in section 2, a limiting factor in such calculations is the algorithm used to compute drainage area. Conventional algorithms are restricted to execution on a single computer processor. Performance will therefore be significantly degraded if it becomes necessary to consider terrains larger than the memory one processor can accommodate, as frequent disk access will be required. Use of a single processor also implies that the run time of the conventional algorithms increases proportionally with terrain size.

As in the calculation of other hydrological quantities that rely on flow routing, such as hydrological proximity measures [*Tesfa et al*., 2011], attempts have been made to resolve these issues by proposing parallel drainage area algorithms [e.g., *Wallis et al*., 2009; *Bellugi et al*., 2011; *Braun and Willett*, 2013]. As we show in section 2.4, different parallel algorithms have different strengths that make them best suited to particular problems or computational architectures.

1.3. New Contribution and Outline

In this paper, we propose performing flow routing calculations in an implicit, iterative manner using linear solvers. This endows the computation with novel characteristics including good scalability to large numbers of processors, and suitability for execution on modern computational platforms such as graphics processing units (GPUs). In addition, our method's iterative nature renders it particularly suitable for repeated calculations such as those performed by numerical landscape evolution models. We also demonstrate that this algorithm can be adapted to delineate drainage basins with parallel computation.

Methods previously proposed for determining drainage area, and their limitations when applied to large data sets, are discussed in section 2. The implicit, iterative approach that we propose is described in section 3, with implementation considerations in section 4. The performance of the method is analyzed in section 5. Example calculations that demonstrate the advantages of this iterative approach and the potential scientific advances it makes possible are presented in section 6.

2. Calculation of Drainage Area

2.1. Preprocessing of Digital Elevation Data

There are two steps that must be performed before the drainage area calculation can begin. Early implementations of these, and the calculation of A itself, are described by *O'Callaghan and Mark* [1984] and *Jenson and Domingue* [1988]. The first is the hydrological conditioning of the input digital elevation model (DEM). This involves several processes to render the data more suitable for hydrological applications, such as removing spurious sinks from the data, which could cause flow paths in the model to artificially terminate early or be routed incorrectly. The second step is the creation of the flow direction vector field. This specifies the direction in which runoff will flow in each cell (Figure 1). A simple algorithm for assigning flow directions is D8, or "steepest descent" [*O'Callaghan and Mark*, 1984], which routes all flow to the neighboring point to which there is the steepest downward slope. More sophisticated algorithms split the flow between two or more neighbors. An example is the $D\infty$ method [*Tarboton*, 1997], which divides flow among two adjacent neighbors in proportion to each neighbor's proximity to a calculated flow angle. The flow directions, determined by

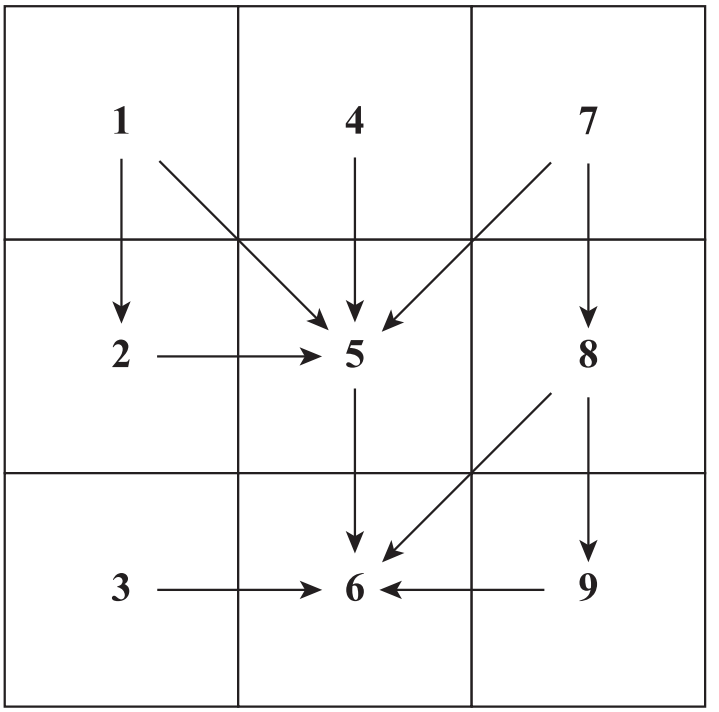


Figure 1. Schematic diagram illustrating hypothetical flow paths between nine cells in a digital elevation data set. Numbers are cell labels, and correspond to the subscripts in equations (1) and (4). Arrows indicate flow paths.

whichever method was chosen, are the input for the drainage area calculation. For a discussion of a modern preprocessing implementation, see *Metz et al.* [2011].

2.2. Conventional Serial Algorithm

The conventional drainage area algorithm [*Mark*, 1988] uses recursion to follow the drainage path upslope from drainage outlets to drainage divides (or other features that define the origins of flow paths). Once the divide points have been reached, they set their drainage area equal to their cell area, and execution travels downstream again, with the drainage area of each cell being equal to the sum of the drainage area of its immediate neighbors that drain directly to it, multiplied by the fraction of their area that they contribute to the cell, plus its own cell area. This algorithm is elegant in its simplicity and efficient for problems of limited size, but its fundamentally serial nature means that it must be run on a single computer processor. In practice, this limits the size of the data set that can be considered, as both processing time and memory requirements will increase linearly with the size of the problem domain.

2.3. Need for a New Algorithm

The rate of increasing data resolution, and therefore memory required, is faster than that of available resources for single processors. At 0.5° resolution, as used in the STN-30p data set [*Vörösmarty et al.*, 2000], less than 0.01 GB is needed to store drainage area data for North America from the Panama Canal to 60°N. This rises to more than 1000 GB for the 12 m expected resolution of TanDEM-X data [*Krieger et al.*, 2007]. This is at least an order of magnitude greater than is typically available to a single processor in current systems. The solution is to use multiple processors running in parallel. Doing so provides access to multiple times the memory and computational power of a single processor.

2.4. Previously Proposed Parallel Drainage Area Algorithms

Algorithms that were used in serial computations may no longer work for parallel execution, or may not be efficient. It therefore becomes necessary to design new algorithms to exploit such computing resources. A modification of the conventional recursive drainage area algorithm to permit parallel execution, and therefore enable the calculation on large landscapes at high resolution, was proposed by *Wallis et al.* [2009]. The idea is to only calculate the drainage area for a cell after all of the cells that drain into it have already been processed. This ensures that the required information, even if it is calculated by another processor, is available when it is needed. To do this, the method extends the conventional drainage area algorithm with a dependency grid. For each cell, the dependency grid stores the number of immediate neighbors that drain to that cell. When a cell is processed, the cells that it immediately drains to have their dependency number decreased by one. Cells are added to the queue to be processed when their dependency grid value reaches zero (none of the upstream cells remain to be processed). When the queue of a processor is empty (it has no more cells that can be processed with the available information), it communicates with the processors working on neighboring patches of the landscape. This allows information about flow across processor

domain boundaries to be transferred between processors. The information received from neighboring processors may allow additional cells to be processed. Alternating stages of computation and communication continue until all of the cells in the landscape have been processed. This method, named ParallelArea, is part of the TauDEM software package [*Tarboton*, 2013]. While ParallelArea eases the memory constraint on problem size, the number of communication stages required will grow if more processors are used on a landscape (as the domain size of each processor will decrease). Reported measurements indicate that on a large cluster the runtime of the algorithm decreases as (number of processors)$^{-0.93}$ up to 48 processors, but shows less consistent decreases in runtime for higher processor counts [*Wallace et al*., 2010]. This suggests that the ParallelArea method scales very well up to moderate processor counts for a landscape of the size tested (24,856 $\times$ 24,000 cells), but that it may not be possible to further reduce runtime by increasing processor counts beyond this.

Parallel drainage area algorithms have also been proposed by *Bellugi et al*. [2011] and *Braun and Willett* [2013]. Like ParallelArea, both of these methods arrange the order in which cells are processed so that computation begins at flow sources and progresses downslope, ensuring that whenever a cell is processed, the information it relies upon (the upstream drainage area) is already available. *Braun and Willett* [2013] report good scaling behavior on increasing numbers of cores of a single processor, obtaining a five times speedup over single core performance when running on eight cores. Such algorithms therefore appear to perform well when the terrain data set is small enough to fit into the memory of a single multiprocessor. Unlike *Wallis et al*. [2009], the authors do not describe implementation on distributed memory systems. The use of such systems, where processors have local memory and must communicate with other processors to access the local memory of those processors, is essential to efficiently process large terrains that cannot fit into the memory of a single multiprocessor. It is likely that the algorithm described by *Braun and Willett* [2013] could be implemented in a way that is suitable for distributed memory systems, but given its similarity to ParallelArea, it is not clear whether it provides scalability that improves on the results reported by *Wallace et al*. [2010]. In an unpublished experiment, the algorithm of *Bellugi et al*. [2011] scaled as (number of processors)$^{-0.8}$ between 2 and 32 processors, decreasing to (number of processors)$^{-0.3}$ between 32 and 128 processors, on a 128 $\times$ 10^6 pixel landscape (D. Bellugi, personal communication, 2013).

While these methods scale well for modest numbers of processors, scaling deteriorates as communication overhead increases. The high-performance computing systems of the future are expected to consist of very large numbers of processors [*Shalf et al*., 2011], so the current parallel algorithms may no longer be efficient. We therefore undertook to devise an alternative approach which may be more suitable for many-core architectures.

3. Implicit Drainage Area (IDA)

3.1. Implicit Formulation

The Implicit Drainage Area (IDA) algorithm reformulates the flow routing problem as a system of linear equations. Consider the nine cell system shown in Figure 1. If a_n is the drainage area of cell n, b_n is the area of cell n, and w_{mn} is a weight equal to the fraction of the area of cell m that drains to cell n, then the flow paths shown in Figure 1 lead to a set of linear equations:

$$a_i = b_i \quad i = 1, 3, 4, 7 \tag{1a}$$

$$a_2 = b_2 + a_1 w_{12} \tag{1b}$$

$$a_5 = b_5 + a_1 w_{15} + a_2 w_{25} + a_4 w_{45} + a_7 w_{75} \tag{1c}$$

$$a_6 = b_6 + a_3 w_{36} + a_5 w_{56} + a_8 w_{86} + a_9 w_{96} \tag{1d}$$

$$a_8 = b_8 + a_7 w_{78} \tag{1e}$$

$$a_9 = b_9 + a_8 w_{89} \tag{1f}$$

where the b_i in each equation indicates that each cell receives drainage from itself in addition to any drainage from neighboring cells. For example, the drainage area of cell 9 in Figure 1 is calculated by adding the portion of the area of cell 8 that drains to cell 9 ($a_8 w_{89}$) to the area of cell 9 itself (b_9). Most widely used flow

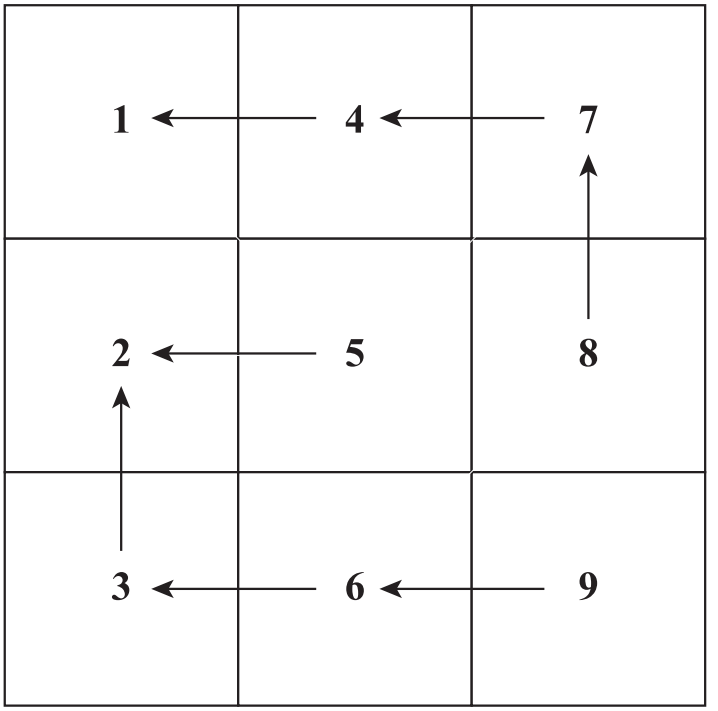


Figure 2. Hypothetical flow paths between nine cells in a digital elevation model, using the same notation as Figure 1. In this example, D4/D8 flow routing is used, resulting in each cell draining entirely to only one of its neighbors. Arrows indicate flow paths. The corresponding system of equations for drainage area is shown in equation (5).

routing schemes can be written in this way with an appropriate choice of weights, w_{mn}. In the simple "steepest descent" or D8 algorithm [*O'Callaghan and Mark*, 1984], for example, each cell contributes flow to only one downslope neighbor, and therefore all the nonzero weights are equal to one. Multiple flow direction (MFD) algorithms, in which each cell may contribute flow to multiple downslope neighbors [e.g., *Quinn et al.*, 1991; *Tarboton*, 1997], as in the example in Figure 1, are accommodated by specifying weights between zero and one that sum to one for each cell:

$$\sum_n w_{mn} = 1 \tag{2}$$

The value of w_{mn} is determined by the flow direction algorithm. It is the same as the value that occurs in the conventional drainage area algorithm. For example, in a flow routing scheme in which a cell's flow passes to all downslope neighbors in proportion to slope, the weights for cell 8 in Figure 1 are:

$$w_{86} = \frac{S_{86}}{S_{86} + S_{89}} \tag{3a}$$

$$w_{89} = \frac{S_{89}}{S_{86} + S_{89}} \tag{3b}$$

$$w_{84} = w_{85} = w_{87} = 0 \tag{3c}$$

where S_{mn} is the slope from cell m to cell n.

The linear system in equation (1) can be cast as a matrix equation:

$$\begin{bmatrix} 1 & & & & & & & & \\ -w_{12} & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & 1 & & & & & \\ -w_{15} & -w_{25} & & -w_{45} & 1 & & -w_{75} & & \\ & & -w_{36} & & -w_{56} & 1 & & -w_{86} & -w_{96} \\ & & & & & & 1 & & \\ & & & & & & -w_{78} & 1 & \\ & & & & & & & -w_{89} & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{bmatrix} \tag{4}$$

where the elements left blank are zeros. This system has the form $\mathbf{Wa} = \mathbf{b}$, in which $\mathbf{a}$ is the vector of unknown drainage areas, and can be solved with a suitable algorithm for calculating $\mathbf{W}^{-1}\mathbf{b}$. A similar

formulation has been proposed independently by S. Eddins of Mathworks, Inc. [*Eddins*, 2007], and also by *Schwanghart and Kuhn* [2010], although these authors do not emphasize the ease of parallelization when solving such a system in an iterative manner. This example describes a situation in which a regular grid is used, but the method could equally be applied to any other form of mesh. As in most drainage area calculations, all areas discussed in this paper are areas of the horizontal projection of features, not their surface areas. However, there is nothing in this method that precludes its use for surface areas.

The system in equation (4) is implicit because the drainage area of each cell is expressed in terms of the unknown drainage areas of its neighbors. It is sparse because the matrix $\mathbf{W}$ contains only the diagonal terms corresponding to the areas of individual cells and off-diagonal terms corresponding to at most eight (and typically fewer) immediate neighbors of each cell.

The situation is simplified when cells drain entirely to only one neighbor, such as when D8 flow routing is used. For the D8 system shown in Figure 2, assuming that the area of each cell is 1, the system corresponding to equation (4) above would be:

$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{5}$$

3.2. Solution Method

Posing the problem in the form of a sparse linear system renders the computation of drainage area similar to many other scientific and engineering calculations. A significant body of work is dedicated to the solution of such problems [e.g., *Saad*, 2003]. This method therefore permits the application of such research and future advances to the determination of drainage area. The system in equation (4) could be solved with a direct method, such as Gaussian Elimination followed by back substitution. Such a method would be robust, but not amenable to parallelization. Instead, we suggest a solution approach that uses a preconditioned iterative method.

The iterative approach to solving a system of equations of the form $\mathbf{Wa}=\mathbf{b}$ (where $\mathbf{W}$ is an $M \times N$ matrix, $\mathbf{a}$ is an $N \times 1$ vector, and $\mathbf{b}$ is an $M \times 1$ vector) is to first use an initial guess for $\mathbf{a}$, which we will call $\mathbf{a_0}$. The residual is then $\mathbf{r}=\mathbf{b}-\mathbf{Wa_0}$. This can be used to improve the guess for $\mathbf{a}$ (with the exact procedure depending on the solver being used), giving rise to a new residual. Many different solvers are available when the matrix $\mathbf{W}$ is positive definite ($\mathbf{a}^T\mathbf{Wa} > 0 \,\forall \mathbf{a} \in \mathbb{R}^N, \mathbf{a} > 0$) and symmetric ($\mathbf{W}^T=\mathbf{W}$). When these criteria are not met, as is the case in the implicit formulation of flow routing problems, the number of choices is reduced.

Iterative methods allow the desired accuracy of the solution to be specified. The method will iterate until the convergence condition is met. We use the condition that $\frac{||\mathbf{r}||}{||\mathbf{b}||} < O(1\times10^{-16})$, which ensures that the residual is small. If less accuracy is required, then a higher bound on the relative residual norm could be used, resulting in the method terminating after fewer iterations.

Iterative methods are typically combined with a preconditioner [*Benzi*, 2002]. These attempt to instead solve the system $\mathbf{P}^{-1}\mathbf{Wa}=\mathbf{P}^{-1}\mathbf{b}$, where $\mathbf{P}^{-1}\mathbf{W}$ is a matrix with a lower condition number (a measure which indicates the difficulty of solving the system) than $\mathbf{W}$, resulting in faster convergence. A wide variety of solver-preconditioner pairs are therefore available [*Saad*, 2003]. In sections 4 and 5, we discuss solvers that are well suited to the systems typically encountered in drainage area computations.

3.3. Hybrid Method

To exploit the speed of the conventional recursive method, it can be used to determine an approximation of drainage area on the portion of a landscape stored on each processor, neglecting any flow received from outside that portion of the landscape. This estimate can then be used as an initial guess for the IDA method.

Performance can be further improved by recognizing that the serial algorithm calculates the correct drainage area for cells that do not receive flow from another processor. These cells can therefore be omitted from the system of equations passed to the linear solver, greatly reducing memory requirements and runtime.

The performance improvement does, however, come at the expense of additional effort required to implement the hybrid algorithm. Cells that are not correctly solved by the serial algorithm must be identified, inserted into the reduced system, and then have their updated values copied back to the correct location from the output of the linear solver. Rather than solving for $a_i = b_i + \sum_j \{w_{ji} a_j : j \in$ immediate neighbors that drain to $i\}$, as described in section 3.1, the hybrid method solves the system composed of $a_i = a_i^l + \sum_j \{w_{ji}\left(a_j - a_j^l\right) : j \in$ immediate neighbors in the same processor domain that drain to i and are downstream from a process boundary crossing$\} + \sum_k \{w_{ki} a_k : k \in$ immediate neighbors that drain to i, are on the other side of a processor boundary$\}$. Here the superscript l (for "local") refers to the drainage area computed for a point using the serial algorithm (and so neglecting flow across processor boundaries). This change to the system solved is necessary to properly account for flow from cells that were correctly calculated by the serial algorithm to those that were not. The algorithm accomplishes this by adding the change caused by the inclusion of flow across processor boundaries $\left(\sum_j \{w_{ji}\left(a_j - a_j^l\right)\} + \sum_k \{w_{ki} a_k\}\right)$ to the value calculated by the serial algorithm $\left(a_i^l\right)$. The edge contamination concept described by *Tesfa et al.* [2011] could aid the identification of cells that do need to be passed to the solver.

As an example, we consider the flow paths shown in Figure 2 for the case when each column of the landscape is stored on a different processor. Processor 1 stores cells 1–3, 4–6 are on processor 2, and 7–9 are on processor 3. After running the serial algorithm on each processor, cells 5, 7, 8, and 9 have the correct value. Assuming each cell has unit area, the initial guesses for each cell at this stage (which correspond to the a_i^l values above), are:

$$\begin{bmatrix} a_1^l : 1 \\ a_2^l : 2 \\ a_3^l : 1 \end{bmatrix} \begin{bmatrix} a_4^l : 1 \\ a_5^l : 1 \\ a_6^l : 1 \end{bmatrix} \begin{bmatrix} a_7^l : 2 \\ a_8^l : 1 \\ a_9^l : 1 \end{bmatrix} \tag{6}$$

In this example, only cell 8 can be excluded from the reduced system, because cell 8 is the only cell with a correct serial solution that does not drain across a processor boundary. The system corresponding to equation (5) of the regular IDA method is then:

$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_9 \end{bmatrix} = \begin{bmatrix} a_1^l \\ a_2^l - a_3^l \\ a_3^l \\ a_4^l \\ a_5^l \\ a_6^l \\ a_7^l \\ a_9^l \end{bmatrix} \tag{7}$$

For example, the drainage area of cell 2 is calculated by adding the difference between the correct value for cell 3 and the value computed for that cell by the serial algorithm $\left(a_3 - a_3^l\right)$, and the value of cell 5, which was computed correctly by the serial algorithm $\left(a_5 = a_5^l\right)$, to the value calculated for cell 2 by the serial algorithm $\left(a_2^l\right)$. Note that in the steepest descent example in Figure 2, the weights are all equal to 1.

3.4. Drainage Basin Delineation

The iterative method for computing drainage area can be adapted to delineate drainage basins without computing the drainage areas of all points within the basins. The objective of this operation is to begin with a set of outlet points, such as points along a coast, and identify the collection of points that drains to each outlet. This yields a mask that allows properties of basins, such as size, shape, and average slope, to be determined for each basin.

The canonical serial basin separation algorithm [*Mark*, 1988] starts from the outlets, assigning each a unique number. It then recursively identifies all upstream points of each outlet, similarly to the conventional serial drainage area algorithm, and fills these points with the outlet's unique value.

To solve the basin separation problem in parallel, it can be reformulated as a system of equations $\mathbf{Wu}=\mathbf{b}$ and then solved in a similar way to the IDA algorithm. As with the canonical case, the outlets must first be identified. The entries in the $\mathbf{b}$ vector corresponding to these points are set to unique values, while the rest are set to zero. The $\mathbf{W}$ matrix has ones on the diagonal and $-\frac{1}{n}$ in $W_{i,j}$, if cell i directly drains to cell j, where n is the number of immediate neighbors that cell i drains to. All other elements of $\mathbf{W}$ are zero. As the system of equations is solved, the unique values will propagate upstream from the outlets. When the solver completes, the $\mathbf{u}$ vector will contain an entry for each cell equal to the unique value of the outlet to which that cell ultimately drains, or zero if it does not drain to any of the requested outlets. As with the canonical method for basin separation, flow direction schemes that permit a cell to drain to more than one of its neighbors, such as D∞ [*Tarboton*, 1997], complicate basin separation as they permit a single cell to ultimately drain to more than one outlet. With the proposed method, the basin number of cells upstream from the cell where a divergent flow occurs would be set equal to the sum of the basin numbers of the cells immediately downstream from the divide, potentially making it difficult to identify the basins that such cells are part of.

As an example, we use the landscape and associated flow paths shown in Figure 2. All of the cells in this landscape drain through the two outlets: cells 1 and 2. We associate unique numbers with these two outlets, o_1 and o_2, respectively. The application of basin delineation will set the u value for each cell to the unique number corresponding to the outlet to which it drains. The linear system of equations for this example is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} o_1 \\ o_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

4. Implementation

4.1. Advantages of the Implicit, Iterative Formulation

The advantage of the formulation we present in section 3 is not only the availability of established solvers for sparse, linear systems, but also the ease of parallelization. Optimized, parallel iterative solvers and preconditioners are widely available in libraries, such as PETSc [*Balay et al*., 2012] and HYPRE [*Falgout et al*., 2012]. Once the matrices have been constructed, they need only be passed to such a library to be solved in parallel.

The implementation of the central component of the IDA method, solving the system of equations, therefore requires relatively little programming effort. The construction of the input matrices and an initial guess,

if desired, are the only steps that may require some effort, but as long as the flow directions of each cell's neighbors are available, this is likely to be simpler than the parallel implementation of some other drainage area algorithms.

PETSc, HYPRE, and hand-coded central processing unit (CPU) and graphics processing unit (GPU) versions of the code were created. Of these options, HYPRE provided the easiest implementation but offered fewer choices of solver and preconditioner than PETSc. We found that the slightly more difficult development experience when using PETSc was justified by the ease of testing many different solver and preconditioner combinations, together with its ability to run on GPUs in addition to CPUs. The results we show in the remainder of this paper were therefore generated from the PETSc implementation, unless otherwise noted. This code may be downloaded from the online repository of the Community Surface Dynamics Modeling System (CSDMS) [*CSDMS*, 2013]. Optimizations are possible when the algorithm is hand-coded, as described below, and so this may be preferable for production codes.

4.2. Construction of the W Matrix

As mentioned above, when a linear solver and preconditioner library, such as PETSc, is used, the only step that remains to be implemented in the regular IDA algorithm is the construction of the *W* matrix. This can be accomplished with the following algorithm, where N_y is the number of cells in the *y* direction (often North-South) of the local process domain, N_x is the number of cells in the *x* direction (often East-West), and $w^{j,j}_{i+p,j+q}$ is the fraction of flow contributed from the cell at $(i+p, j+q)$ to the cell at (i, j), which may be zero.

```
for i = 0 to N_y do
    for j = 0 to N_x do
        /* row index                                              */
        W_row = i * N_x + j;
        for p = −1 to 1 do
            for q = −1 to 1 do
                /* column index                                   */
                W_col = (i + p) * N_x + (j + q);
                if p == 0 and q == 0 then
                    /* 1 along main diagonal                      */
                    W(W_row, W_col) = 1;
                else
                    W(W_row, W_col) = −w^{i,j}_{i+p,j+q};
```

Cells may contribute flow across process domain boundaries, which will be manifested as trying to set the value of an element outside of the local $N_xN_y \times N_xN_y$ matrix. Libraries such as PETSc provide functions for constructing the local portions of sparse matrices, including the ability to set these nonlocal elements.

Once the construction of the *W* matrix is complete, the area of each cell is stored in the *b* vector for regular IDA, or set as described in section 3.3 for the hybrid method. The *a* vector is set to the initial guess of each cell's drainage area, if available. These inputs are then provided to the preconditioned solver, which will store the final drainage area solution in *a*, completing the IDA algorithm.

4.3. Choice of Solver

The performance of the IDA algorithm is strongly dependent on the choice of solver and preconditioner. Many iterative solvers and preconditioners are only suitable for specific forms of coefficient matrices. As we note in section 3, the matrix in this case is not symmetric or positive definite, which reduces the number of choices.

The conjugate gradient method [*Hestenes and Stiefel*, 1952] is a popular linear solver, but cannot be used for this system for the aforementioned reasons. Other Krylov subspace methods can be used, however,

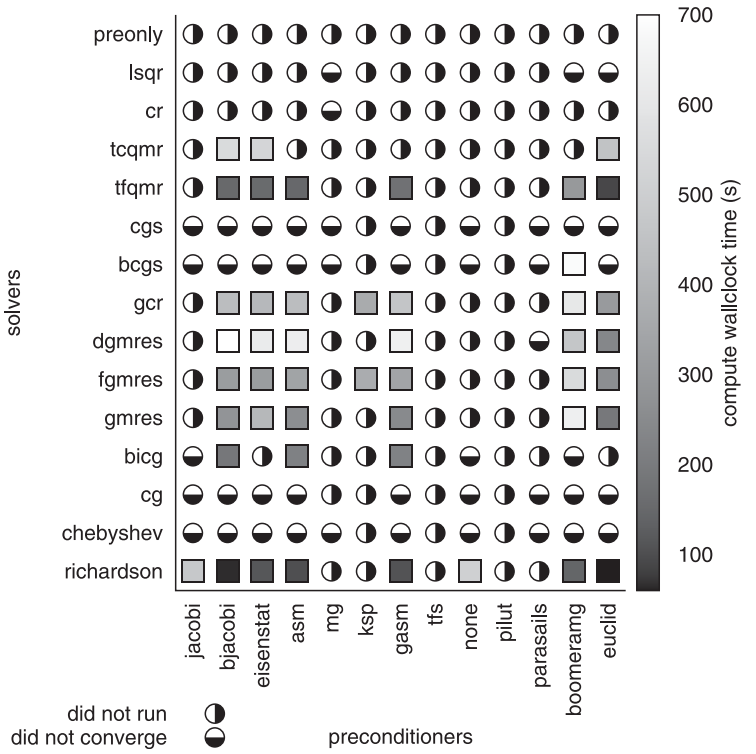


Figure 3. Runtime of IDA on a 4000 × 4000 landscape extracted from the 15 arc sec HydroSHEDS North America data set, using eight processing cores and a variety of solver-preconditioner combinations. We use PETSc's naming conventions for these solvers and preconditioners. Certain combinations failed to run, probably due to assumptions of the methods, such as matrix symmetry, being violated. Other combinations ran but did not converge within the 15 min limit imposed. A large variation in runtime is observed for those combinations that did run and converge. A single Intel Xeon 5570 with 24 GB of memory was used, together with PETSc 3.4.

such as GMRES (generalized minimal residual method) [*Saad and Schultz*, 1986] and BiCGSTAB (biconjugate gradient stabilized method) [*van der Vorst*, 1992]. Non-Krylov solvers are also available, including the Richardson method [*Richardson*, 1910].

The choice of preconditioner similarly presents several options. The Jacobi preconditioner, one of the simplest methods, approximates the coefficient matrix by its diagonal. It is therefore most suitable for diagonally dominant matrices. That is not true of the drainage area system, rendering this a poor choice. The successive over-relaxation (SOR), incomplete LU factorization (ILU), additive Schwarz method (ASM), and algebraic multigrid (AMG) preconditioning methods are all potential candidates. For a broad discussion of linear solvers and preconditioning methods, see *Saad* [2003].

We tested various solver-preconditioner combinations available in PETSc by processing a 4000 × 4000 cell area of the United States (approximately covering the region from 35°N, 117°W to 52°N, 100°W) on eight processor cores using the 15 arc sec flow direction data set from the HydroSHEDS project [*Lehner et al*., 2008]. For each combination, we recorded the wall clock time required for the calculation to converge. Solver-preconditioner combinations that did not run due to incompatibility with the flow routing matrix equation were noted, and calculations that did not converge within 15 min were terminated. The results, including runtimes for combinations that successfully converged, are presented in Figure 3. For the solver-preconditioner combinations that were successful, we performed the same calculation on two processor cores to examine scaling behavior. The results of this informal scaling analysis are shown in Figure 4.

Several of the solvers converged for a variety of preconditioners. The Richardson solver [*Richardson*, 1910] converged with the greatest number of preconditioners (8 out of 13), followed by the FGMRES [flexible generalized minimal residual method) [*Saad*, 1993] and GCR (generalized conjugate residual) [*Widlund*, 1978]. In general, the Richardson and TFQMR (transpose-free quasi-minimal residual method) [*Freund*, 1993] methods provided the fastest runtimes. Richardson also exhibited superior scaling, approaching linear scaling for

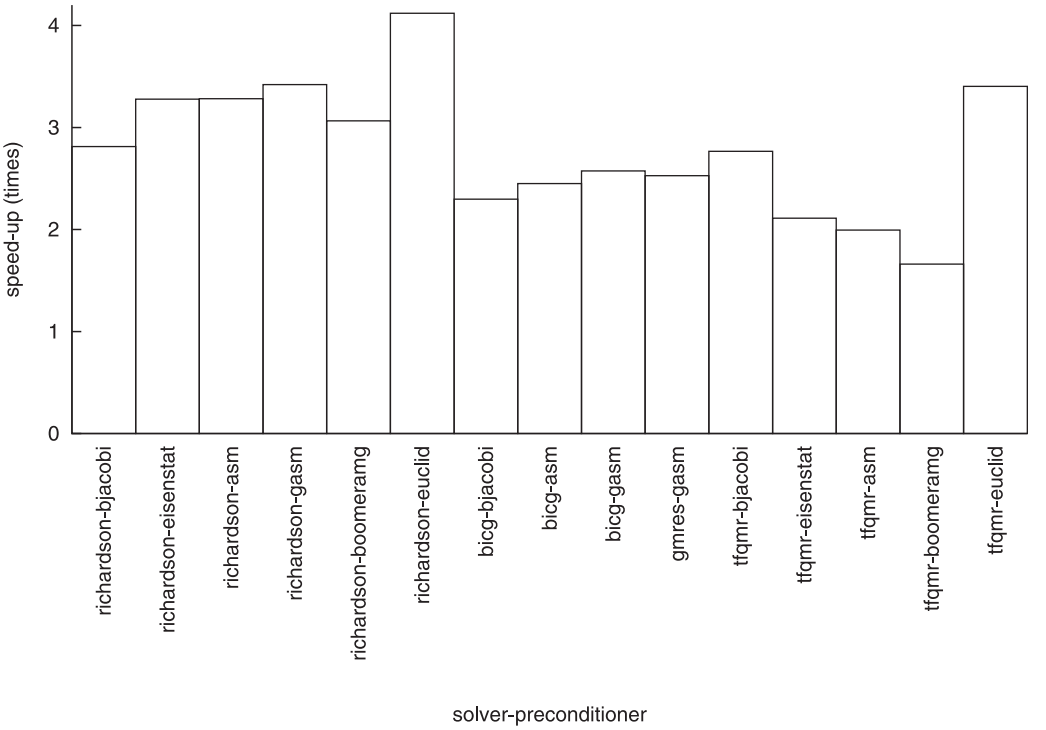


Figure 4. Speed-up of the IDA runtime on the same 4000 × 4000 landscape as in Figure 3, using a variety of solver-preconditioner combinations, when increasing the number of processing cores from 2 to 8. Many of the combinations are observed to obtain close to the linear four times speed-up, but the Richardson-Euclid combination slightly exceeds this, achieving super linear speed-up. A single Intel Xeon 5570 with 24 GB of memory was used.

several preconditioners. We found that the combination of the Richardson solver and the Euclid preconditioner, part of the external HYPRE package [*Falgout et al*., 2012], exhibited the best scaling behavior. The Richardson solver is a simple method that updates the guess by using the equation $\mathbf{a}_{j+1}=\mathbf{a}_j+\omega(\mathbf{b}-\mathbf{W}\mathbf{a}_j)$, for the $j+1$ iteration, where ω is a scalar that we left at PETSc's default of 1. Euclid is an incomplete LU factorization preconditioning method. Such methods factor the *W* matrix into upper triangular and a lower triangular components, while attempting to preserve sparseness [*Hysom and Pothen*, 2001]. We therefore use Richardson-Euclid in all subsequent performance results presented in this paper, unless otherwise noted. It is possible that combinations not tested may yield better scalability, and that scalability may be affected by the computer hardware in use. It is also possible that the optimum combination may depend on the characteristics of the terrain. To test this, we ran a selection of solver-preconditioner pairs with five terrains of the same size centered at randomly selected coordinates in North America, with the condition that no more than 20% of the terrain be covered by ocean. Although runtime differed by up to 20% between the different landscapes, there was almost no variation in the relative times of the different solver-preconditioner combinations.

4.4. Memory Requirements

To accommodate the working space required for the linear solver, a larger amount of memory may be required for IDA compared to other parallel drainage area calculation methods. Nevertheless, as with other parallel drainage area methods, the memory requirements of IDA are divided among the number of processors participating, and so larger problem sizes can be considered by increasing the number of processors. The hybrid IDA method has significantly reduced memory requirements compared to the regular version, as only the cells that could not be calculated by the serial algorithm are considered by the linear solver.

One of the largest memory requirements of IDA is due to the storage of the *W* matrix. One method of storing this matrix is to notice that it has nine nonzero diagonals, of which one (the main diagonal) is always 1 and so can be incorporated into the equations rather than stored. The *W* matrix could therefore be stored

as an $8 \times N$ matrix, where N is the number of elements in the landscape. Even in this compressed matrix, the vast majority of elements will still be zero (as flow is in general only contributed by a small number of a cell's neighbors), and so further compression is possible by storing in other formats such as compressed row storage (CRS) [see *Tewarson*, 1973, for a description]. CRS is the storage format used by default by PETSc. While the W matrix contains $N \times N$ elements, exploiting its sparse nature therefore results in storage requirements only growing linearly with N.

As prepackaged solutions, libraries such as PETSc may implement general optimizations, but typically do not take advantage of optimization opportunities specific to the system they are being used to solve. In the case of the drainage area calculation, such optimizations include noticing that when D8 flow directions are being used, the elements of the **W** matrix can be stored as single bit Boolean values, rather than in floating-point format. Such optimizations can significantly reduce memory requirements and potentially improve performance. We have implemented such optimizations in a hand-written GPU code and found that they do improve performance compared to general library implementations. For a 2000×2000 landscape and running on an Nvidia GeForce GTX 480 GPU using the BiCGSTAB solver [*van der Vorst*, 1992], our hand-coded implementation is 5% faster than the PETSc implementation. Nevertheless, the results we present in subsequent sections were obtained using PETSc, unless otherwise noted.

The memory requirements of IDA will depend on the solver and preconditioner used. The BiCGSTAB solver, for example, requires approximately $7 \times N$ elements, in addition to the memory to store the W matrix (which will depend on the storage format used, as discussed above, but should be smaller than $8 \times N$ elements). The Richardson solver [*Richardson*, 1910], on the other hand, can be implemented with only $3 \times N$ elements plus the storage for W.

5. Performance Analysis

One of the advantages of IDA is that the number of communication stages, when processors communicate with each other to transfer data from the boundaries of their domains to neighbors, is almost independent of the number of processors. This compares favorably to methods such as ParallelArea, where increasing the number of processors will lead to a larger number of communication stages. For such methods, using a large number of processors may therefore result in communication overhead dominating the reduced runtime. We therefore expect that IDA will scale well, even to many processors. We examine the scaling behavior by measuring the time to calculate the drainage area of the contiguous USA at 15 arc sec resolution ($20{,}640 \times 8880$ pixels) using different numbers of processors. The flow directions data set of the hydrologically preconditioned USGS HydroSHEDS project [*Lehner et al.*, 2008], which we discuss in greater detail below, is used. The timing measurements we show therefore only correspond to the application of the IDA algorithm: calculating the drainage area. We performed the computation on a distributed memory cluster with one 12 core Intel Xeon X5650 and 48 GB of memory per node, connected by an InfiniBand interconnect. The runtime variation with number of processors is shown in Figure 5. Even at 192 processing cores, the method is still able to improve performance by increasing processor count. The runtime could be reduced by using the conventional serial algorithm on each processor's local domain (neglecting flow between processor domains), and providing this to the solver as an initial guess. On a sample data set of 5000×5000 cells per processor, with between two and five processing cores, we observed a speed-up of about 20% through the use of this modification, independent of the number of cores. Omitting the cells that are correctly solved by the serial algorithm, yielding the hybrid method described in section 3.3, further improves performance. The scaling plot for the hybrid IDA method and TauDEM is presented in Figure 6. It is apparent that the hybrid is significantly faster than regular IDA. This speed-up, more than 100 times in this experiment, suggests that the more complicated implementation is likely to be justified for most applications. On a sample 4000×4000 cell data set, running on 8 cores, employing the hybrid method reduces the linear solver system from 16×10^6 to about 55×10^3 elements (a 290 times reduction), so the dramatic performance improvement is not surprising. As with the regular case, the scaling of the hybrid method computation time does not diverge dramatically from ideal (linear) scaling. Likewise, continued performance improvements are obtained with increasing processing core count, even at 192 cores. TauDEM initially improves performance as the number of processing cores is increased, but after 96 cores the runtime steadily increases. It is possible that for this landscape and computer cluster, this is the point at which the

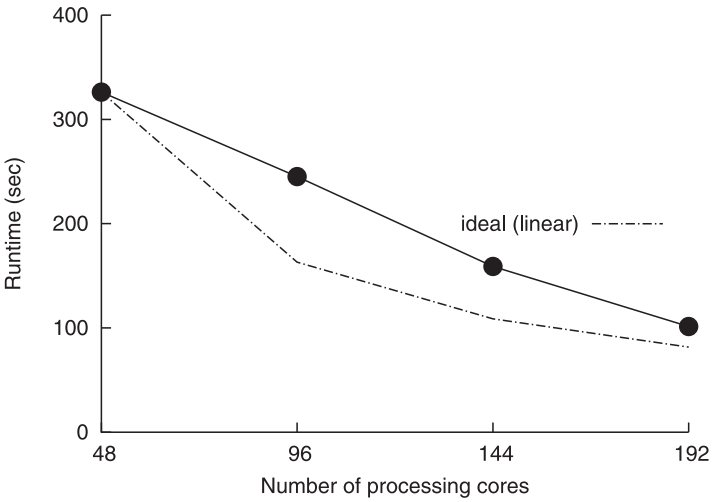


Figure 5. The computation runtime of the IDA algorithm using the Richardson-Euclid solver-preconditioner combination on the 20,640 × 8880 cell HydroSHEDS North America data set. Intel Xeon X5650 CPUs with 48 GB of memory were used, connected by an InfiniBand network. Perfect, linear scaling is shown for reference.

additional communication stages required by reducing each core's domain size outweigh the benefits of increased total processing power.

An additional form of performance scaling measurement, known as weak scaling, examines the variation in computation runtime as the total problem size increases, while keeping the problem size per processor fixed. We add a 450 × 450 cell tile to the landscape for each additional processing core used. The results of this experiment for the hybrid method and TauDEM are shown in Figure 7. The computation runtime of hybrid IDA is observed to only increase by two times when calculating the drainage area on a landscape eight times as large (0.18 s for a 1800 × 2700 landscape, compared to 0.32 s for a 10,800 × 3600 landscape). In comparison, the computation time of TauDEM increases by 22 times over the same range. For a smaller number of processing cores and larger tiles per processing core (1000 × 1000 cells), shown in Figure 8, TauDEM provides almost ideal scaling of computation time, while hybrid IDA's computation time increased by 2.4 times when calculating the drainage area on a landscape four times as large.

The major strength of IDA, both the regular and hybrid versions, is the close to ideal strong scaling that continues to large numbers of processors. This suggests that the method is well suited to many-core computing resources, and that it enables problems to be solved faster by simply increasing the number of processors. If its scaling behavior continues to even larger numbers of processors, a point may be reached at which the regular IDA method provides runtimes that are competitive with other methods. The hybrid IDA algorithm significantly reduces the threshold at which the method becomes attractive. Indeed, for all of our experiments, the computation time of the hybrid IDA method was shorter than that of the ParallelArea algorithm implemented in TauDEM. A weakness of IDA compared to ParallelArea is its poorer weak scaling for small numbers of processing cores. Considering this result in combination with the strong scaling performance suggests that IDA is most

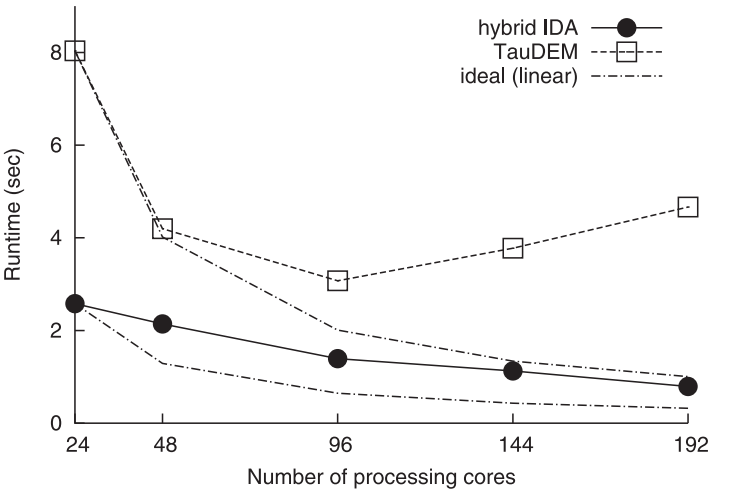


Figure 6. The computation runtime of the hybrid IDA algorithm and ParallelArea algorithm, as implemented in TauDEM 5.2, on the same landscape as in Figure 5. Intel Xeon X5650 CPUs with 48 GB of memory, connected by an InfiniBand network, were used. The TauDEM times are those reported as "Compute time" by that code, while the IDA measurements are the sum of the local serial calculation and the linear solver runtime.

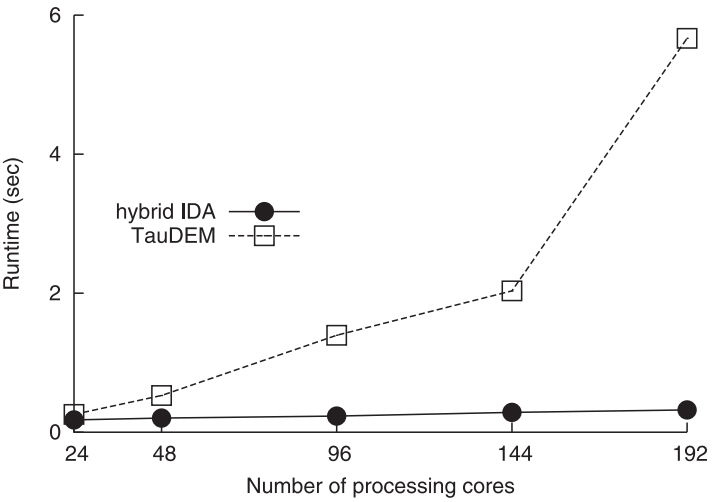


Figure 7. The variation in computation runtime of hybrid IDA using Richardson-Euclid, and TauDEM 5.2, as the total problem size increases while the problem size per processor (450 × 450) remains fixed. The landscape tested was extracted from the 15 arc sec HydroSHEDS data set covering North America. A distributed memory cluster consisting of Intel Xeon X5650 CPUs with 48 GB of memory, connected by an InfiniBand network, was used.

suitable for situations when a large number of processing cores are available. Given their similarity to ParallelArea, the methods described by *Bellugi et al*. [2011] and *Braun and Willett* [2013] may be expected to perform similarly to that algorithm.

We anticipate that the performance of drainage area algorithms for very large numbers of processors will become increasingly relevant as computer hardware architectures evolve. The highly parallelizable nature of iterative linear solvers renders them appropriate for execution on modern many-core architectures, such as graphics processing units (GPUs). Furthermore, such methods typically involve the repeated application of the same operations on every element of the domain, another characteristic necessary for good GPU performance. It has been shown by *Naumov* [2012] that speed-ups of several times over the runtime obtained with CPUs are possible. Indeed, our hand-written GPU implementation of IDA performed up to 30 times faster than our CPU version running on one core, as shown in Figure 9. Significant speed-ups (greater than ten times for a 2000 × 2000 cell landscape) were also observed with our PETSc-based implementation of IDA, when switching from CPU to GPU execution. This suggests that the IDA method may be particularly suited to such computational platforms. The computations performed by the linear solver are similar in nature in the regular and hybrid IDA algorithms, and so the hybrid algorithm should exhibit comparable GPU performance.

Thus, the advantage of the present implementation of regular and hybrid IDA over the conventional recursive algorithm is the ability to overcome memory limitations and to exploit the additional computational power of multiple processors. The advantage over previously proposed parallel algorithms is the potentially greater scalability to large numbers of processors and its suitability for execution on GPU-like architectures.

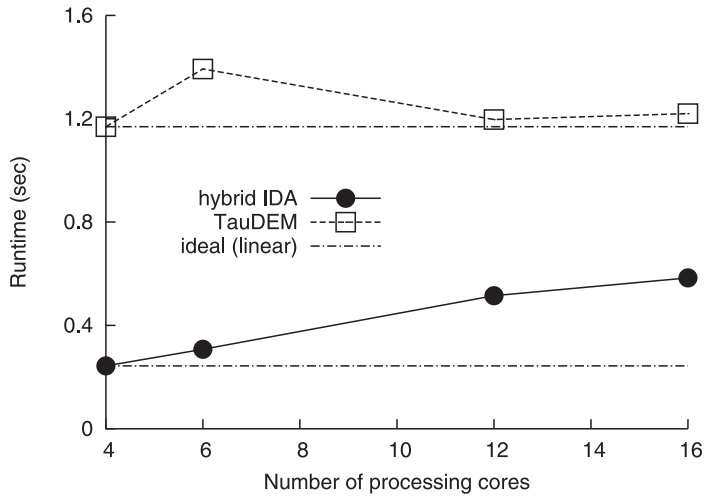


Figure 8. The weak scaling of hybrid IDA using Richardson-Euclid, and TauDEM 5.2, using a 1000 × 1000 cell landscape tile per processing core. The landscape the tiles were extracted from, and the computing resources used, were the same as those of Figure 7. Only two processing cores on each node were used, to ensure that the effects of internode communication were included in the results.

There are many more linear solvers and preconditioner methods than those that were tested, and so there are potentially more appropriate combinations than Richardson-Euclid. It is also likely that the solution time could be reduced through the imposition of constraints based on the physical nature of the flow routing problem. One obvious constraint is that of positivity, as drainage area cannot be negative.

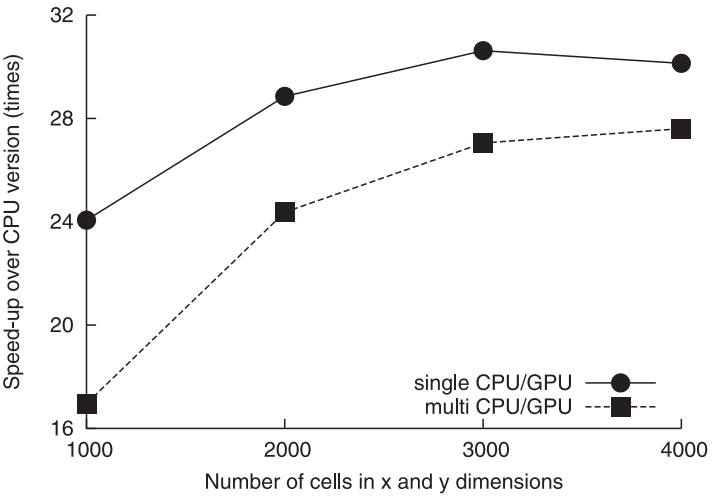


Figure 9. Speed-up of IDA when running on GPUs compared to CPUs. Landscapes ranging in size from 1000 × 1000 cells to 4000 × 4000 were extracted from the 15 arc sec HydroSHEDS North America data set and used as input, with a hand-written BiCGSTAB code used as the solver. As this data set uses D8 flow directions, the **W** matrix was stored using single-bit Boolean values, as described in section 4.4, for both the CPU and GPU implementations. Two cases are shown: one GPU compared to one CPU core, and two GPUs compared to two CPU cores. Nvidia GeForce GTX 480 GPUs, and an Intel Xeon E5520 CPU, were used.

6. Example Applications

The ability to process large landscapes at high resolution creates the opportunity to perform investigations that are challenging using the conventional drainage area algorithm on widely available computer resources. Examples include continent-scale flow routing, which we demonstrate in section 6.1, and the ability to perform terrain analysis calculations that use flow routing over large spatial extents while simultaneously resolving local topographic characteristics such as slope gradients, as examined in section 6.2. Another application where the IDA algorithm is particularly promising due to the iterative nature of the computational methods is in landscape evolution modeling, as illustrated in section 6.3.

6.1. Continent-Scale Flow Routing

To improve understanding of global freshwater supplies, the World Wildlife Fund initiated the HydroSHEDS project [*Lehner et al*., 2008], which produced a hydrologically corrected version of SRTM data [*Farr et al*., 2007] at resolutions of 30, 15, and 3 arc sec, including elevations and flow directions. The drainage areas calculated from this data set are also available, but only at 30 and 15 arc sec resolution. We are not aware of any continent-scale drainage area computations that have been performed with the 3 arc sec version of this data set. The probable reason for this is the difficulty of processing this volume of data with the conventional serial algorithm. Such a calculation is possible with the IDA method, however, and so the HydroSHEDS flow directions were used to produce a 3 arc sec (approximately 90 m) drainage area map of North America from the Panama Canal to 60°N, a grid with dimensions of 104,400 × 63,600. The calculation was performed by dividing the landscape into three pieces which were solved for local flow using the conventional serial algorithm to construct an initial guess, as described above. The flow between these three sections was then accounted for by running IDA on 130 Nvidia M2070 GPUs at the Keeneland Computing Facility at the Georgia Institute of Technology. We used our own GPU implementation, rather than PETSc, due to its reduced memory requirement, as described in section 4.4. The drainage area map, which is available from the CSDMS repository [*CSDMS*, 2013], is used to perform the analyses in the next section.

Most basins were entirely contained within one of the pieces solved with the conventional serial algorithm. The initial guess passed to IDA for these basins was therefore correct and so the application of IDA left their drainage area unchanged. To verify that the basins split between two pieces during the serial solve were correctly joined by IDA, we separately solved one such basin, that of the Nelson River, with the serial solver. The resulting drainage area was within machine precision of that obtained with IDA.

6.2. Slope Versus Relief for Estimating Erosion Rates and Sediment Yields

Previous work has shown a strong correlation between A and average long-term suspended sediment load Q_s [*Milliman and Syvitski*, 1992], which is expected: for a given erosion rate, larger basins should produce more sediment. A link has also been established between the maximum topographic relief of a basin, R, and Q_s [*Mulder and Syvitski*, 1996]. This may be because maximum relief acts as an

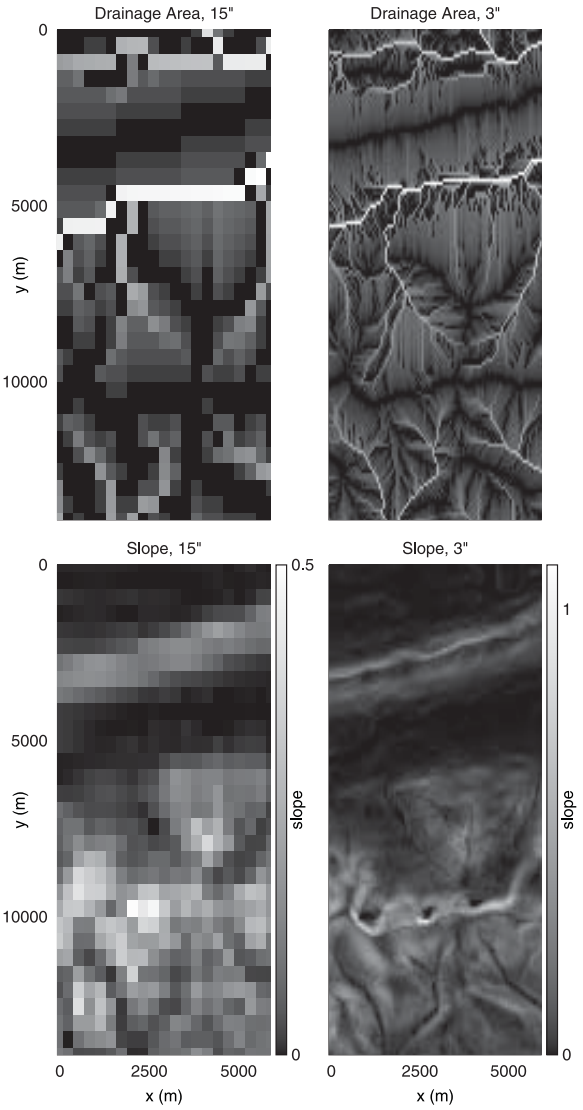


Figure 10. (top) Differences in calculated drainage area and (bottom) slope for a portion of the (left) 15 arc sec and (right) 3 arc sec HydroSHEDS data set covering Ireland's Galtee Mountains and Glen of Aherlow at 52.4°N, 8.19°W. The difference in drainage area is apparent from the absence of several rivers from the 15 arc sec result. The difference in slope is particularly noticeable in the magnitudes (see color scales), with the 15 arc sec result significantly gentler than the 3 arc sec result, and the inability of the 15 arc sec data to resolve important features such as the steep slope above the corrie lakes, clearly resolved at $y = 10{,}000$ on the 3 arc sec image.

approximate surrogate for average slope, which should influence hillslope erosion rates as well as fluvial transport capacity. Such an approximation might not be valid, however, in river basins with nonuniform topography where the maximum relief is not representative of the average basin slope, or in comparisons between basins with different relationships between local slopes and basin relief. Using slope rather than relief may therefore yield better estimates of sediment yield, but calculating the slope between cells at resolutions typically used for continent-scale drainage area computations, often several kilometers, may not produce meaningful values, because this point spacing is longer than typical hillslope lengths. On the other hand, it is typically not feasible to calculate continent-scale A at a resolution that is also suitable for slope calculations due to the computational limitations that motivated this paper. Many studies have therefore employed the approximation of only using relief, which does explain a significant fraction of the variance in sediment yields from large rivers [e.g., *Syvitski et al.*, 2005].

We tested whether improved estimates of continent-scale fluvial sediment yields can be obtained by calculating both slope and drainage area at a finer resolution than previous efforts that relied on relief. The sub-90 m resolution of the HydroSHEDS data set is sufficiently fine that meaningful slopes can be measured for some landscape features, as shown in Figure 10.

By employing the basin separation method described in section 3, with outlets specified to be all points along the coast, we identified which basin each point on the landscape belongs to, and used this mask, together with the drainage network, to calculate the average slope S_{av} in each basin.

Using 63 rivers that were coregistered with the MF11 database [*Milliman and Farnsworth*, 2011], incorporating predam data where available (11 of the rivers), multiple linear regression of log-transformed data was used to determine a relationship between Q_s and (A, S_{av}), and between Q_s and (A, R). The resulting equations were $Q_s = 10^{-0.31} A^{0.77} S_{av}^{1.22}$ and $Q_s = 10^{-4.0} A^{0.44} R^{1.21}$, with Q_s in kg/s, A in km^2, and R in meters. Because $S \sim RA^{-\frac{1}{2}}$, the larger power dependence on A is expected when the model uses average slope, rather than

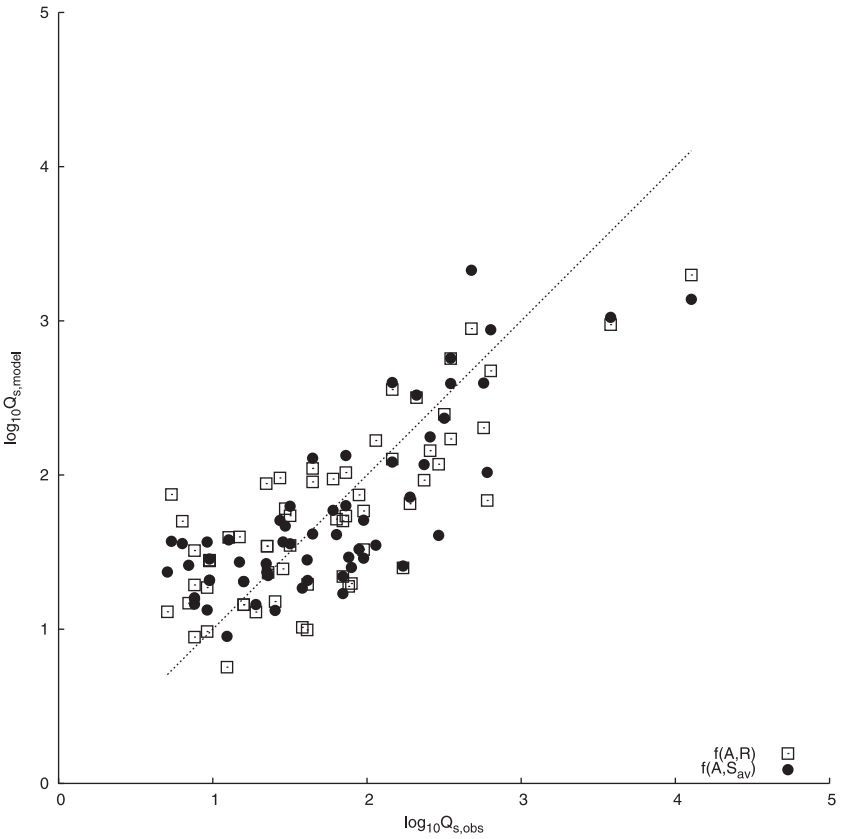


Figure 11. A comparison of observed and predicted sediment load (Q_s) for a model in which Q_s is a power function of drainage area (A) and maximum relief (R), and another model in which Q_s is a power function of drainage area (A) and average basin slope (S_{av}).

maximum relief. However, that does not entirely explain the difference in the exponents on A. A comparison between observed Q_s and the value predicted by these equations is shown in Figure 11. Using S_{av} rather than R improves the correspondence between predicted and observed values of Q_s. The (A, R) model was found to explain 61% of the variance in the data, whereas the (A, S_{av}) model explained 65%. While these are both lower than the 67% that *Mulder and Syvitski* [1996] observed with an (A, R) model, this difference can be explained by the use of different data sets. When only the 11 predam measurements were used in the regression, the (A,S_{av}) and (A, R) models had R^2 values of 94% and 92%, respectively.

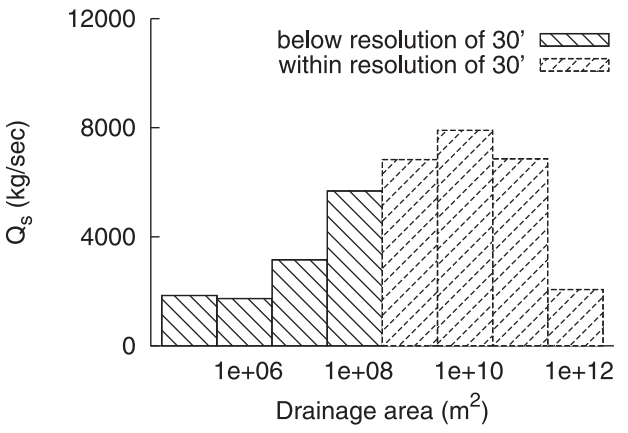


Figure 12. The total coastal sediment load in the region from the Panama Canal to 60°N predicted by the (A, S_{av}) model in Figure 11, binned by basin area. Approximately one third of the sediment flux is produced by basins that are below the resolution of 30′ data sets.

For the particular application of predicting the average coastal sediment flux of large basins, the reduction in accuracy caused by using a coarser grid to determine A is likely to be small, as it will probably not change the calculated total drainage area of each basin significantly. For smaller basins, where a single cell of a coarse grid may be a substantial

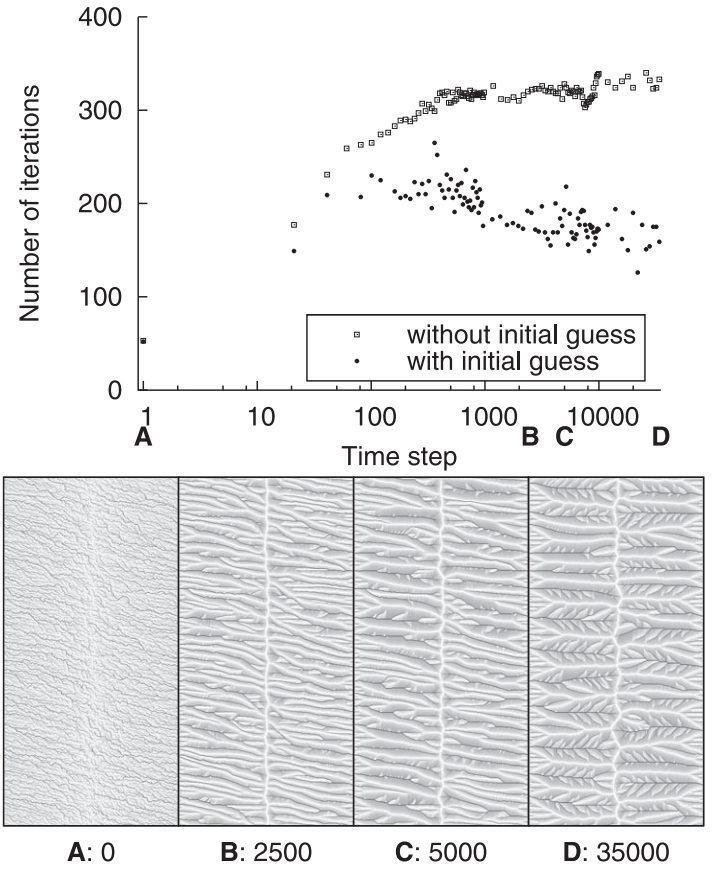


Figure 13. (top) The number of iterations required for convergence of the IDA algorithm in each time step of a landscape evolution model simulation, with and without using the solution from the previous time step as an initial guess for the current time step. The results for every 20th time step are shown for early times, with progressively larger intervals for later time steps to avoid crowding. (bottom) The drainage area of the landscape after different numbers of time steps (using a $\log_{10}$ color scale, with darker grays indicating larger drainage areas).

fraction of the basin size, using a high-resolution drainage area grid becomes more important. Calculating upstream drainage area at a similar resolution to S_{av} is also necessary when predicting average sediment flux at a point inside a basin, such as during landscape evolution simulations. In such situations, interpolating a coarse A to the resolution of the landscape elevation and S_{av} grid can introduce dramatic errors, such as assigning the large A of a river channel to the cells of neighboring valley slopes with large S_{av}. We focus on the average coastal sediment flux of small basins in what follows.

We use the (A, S_{av}) model to predict Q_s for all of the basins in the 3 arc sec North America data set. The result is shown in Figure 12. Applied to these data, the model predicts that the contribution of small basins to total coastal sediment yield is non-negligible. A similar conclusion was reached by *Milliman and Syvitski* [1992]. This example demonstrates how parallel flow routing calculations can be used to explore the influence of drainage basins that would not be resolved by coarser global data sets. The accuracy of this result may be reduced by the lack of small rivers located in North and Central America in the MF11 database with suspended sediment measurements. As these data were not used in the regression that created the model, the model may not fit observations for such rivers well.

The (A, R) model is quite simplistic compared to others proposed more recently, such as BQART [*Syvitski and Milliman*, 2007] and WBMsed [*Cohen et al.*, 2011]. However, even these more recent models use maximum relief rather than slope. With the advent of data at sufficiently fine resolution to calculate slope on many landscape features, and methods such as IDA that permit drainage area to be determined at the same resolution, it might be possible to further improve sediment flux models by replacing relief with slope.

6.3. Landscape Evolution

Iterative methods permit an initial guess to be provided. If the initial guess is close to the solution, the number of iterations required for convergence can be significantly reduced. This suggests that the IDA method may be particularly suitable for landscape evolution simulations, in which it is generally expected that flow directions, and thus drainage area, will not exhibit large changes between time steps. The drainage area solution from the previous time step could therefore be used as an initial guess.

To demonstrate this idea, we use the IDA method to calculate drainage areas for the elevation field at each successive time step of a landscape evolution simulation. Starting from a smooth ridge with a small amount of superimposed random noise, we use a finite difference approximation of a nonlinear advection-diffusion equation describing river incision and soil creep [*Perron et al*., 2008, 2009, 2012] to model the evolution of the landscape due to erosion and soil transport with a time step duration of 100 years. The solution vector a in the iterative linear solver of the IDA method is set equal to the solution from the previous time step. The solver therefore begins its search for the new solution from this point in the solution space. When the landscape does not change greatly between time steps, the drainage area is likely to also remain almost unchanged. The number of iterations required to converge on the new solution should therefore be lower than when the landscape undergoes large changes between time steps. If the drainage network does not change between iterations, the initial guess will be correct and the solver will return immediately.

The results are examined in Figure 13. Initially, the landscape lacks complex flow paths, resulting in IDA converging on the solution within a few iterations of the solver. During the early stages of the simulation following this, as drainage networks invade the domain and change rapidly (between points A and B in the figure), the drainage area calculation using the initial guess value always requires fewer iterations to converge on the solution, but the difference is modest. This is probably because the new drainage area calculated in each iteration is substantially different from the previous solution. Once the drainage networks are established and change more gradually as they approach a steady state (the interval between points C and D, which constitutes most of the simulation), the number of iterations required when using a guess value is approximately half that required when no guess value is used, resulting in a faster computation of the drainage area. The number of iterations required will depend on the number of cells that need to be updated. If the landscape is dominated by a single river basin, and a change in the drainage network occurs near the source of that river, then a large percentage of the cells in the landscape will need to be updated. In this situation, the number of iterations will not be significantly reduced by providing an initial guess. If there is no change in the drainage network from the previous time step, then the solver will calculate a zero initial residual and so will not run any iterations.

The attractiveness of this approach will depend on the problem size and the number of processors being employed. Despite the advantages of using a parallel, iterative method, for small problems the conventional serial algorithm is still likely to be faster. As the problem size increases, the greater computing power of parallel methods will outweigh the higher overhead of such approaches. For problems large enough to require a many-core, distributed-memory computing platform, the scalability of IDA and suitability for execution on GPU-like architectures may give it the advantage. The threshold at which IDA becomes competitive will be lower for landscape evolution problems, due to IDA's ability to exploit information from the previous time step through an initial guess.

7. Conclusions

In this paper, we propose using an implicit, iterative algorithm for calculating drainage area (IDA) that matches current and expected future computer architecture. The algorithm formulates the calculation as a system of linear equations. This system can then be solved with a preconditioned iterative solver. By avoiding the recursive nature of other drainage area algorithms and taking advantage of widely available, parallelizable linear solvers, IDA is well adapted for parallel processing. We demonstrate that the IDA algorithm exhibits good scaling to large numbers of processors. This suggests that it is well suited to large, high-resolution data sets. The conventional serial algorithm may be fastest for small data sets, and queue-based parallel algorithms are potentially best suited to modest numbers of processors, but IDA's continued performance improvements even at large numbers of processors suggest that it might be an appropriate method for many-core systems such as large GPU clusters. A hybrid approach, which combines the

efficiency of the serial calculation with IDA's parallel scalability, reduces the problem size at which IDA becomes competitive. The option to provide an initial guess to the iterative solver creates the possibility of reducing the solution time when performing repeated flow routing calculations on similar terrain, such as in numerical landscape evolution models, by using the solution from the previous step as an initial guess. We anticipate that further performance improvements could be obtained through a judicious choice of solver and preconditioner. With a small modification, this method can be used to delineate drainage basins with the same suitability for parallel computation.

Applying the IDA algorithm to the HydroSHEDS 3 arc sec data set, we demonstrate the suitability of this method for processing very large data sets. At such a resolution it is possible to obtain more meaningful values of topographic slope than with coarser grids, which in turn permits more accurate estimates of quantities such as large-scale erosion rates and sediment yields. With drainage area data at a scale that permits meaningful slope calculations, it becomes possible to compare predictions of riverine suspended sediment flux based on slope with predictions based on topographic relief, which is frequently used as a surrogate for slope at coarse spatial resolutions. We show that an empirical model based on drainage area and slope accounts for a larger fraction of the variance in a data set of continental sediment fluxes than a model relying only on drainage area and drainage basin relief. Calculating drainage area at high resolution allows us to apply this relationship to predict average coastal sediment flux for small basins.

Acknowledgments

This work was partly supported by the NSF Geomorphology and Land-Use Dynamics Program under award EAR-0951672 to J.T.P. and C.N.H. It additionally used resources of the Keeneland Computing Facility at the Georgia Institute of Technology, which is supported by the National Science Foundation under contract OCI-0910735. We are grateful to Dave Yuen for fostering this effort through his enthusiasm for GPU computing, Matt Knepley for helpful discussions of parallel algorithms, and to David Tarboton and two anonymous reviewers for critically reading the manuscript and suggesting improvements. Data from the HydroSHEDS database [*Lehner et al.*, 2008] were used. The IDA code and high-resolution drainage area map of North America produced with it are available from the CSDMS repository [*CSDMS*, 2013].

References

Balay, S., J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang (2012), Argonne National Laboratory, PETSc Web page. [Available at http://www.mcs.anl.gov/petsc.]

Bellugi, D., W. E. Dietrich, J. Stock, J. McKean, B. Kazian, and P. Hargrove (2011), Spatially explicit shallow landslide susceptibility mapping over large areas, in *Fifth International Conference on Debris-flow Hazards Mitigation, Mechanics, Prediction and Assessment*, edited by R. Genevois, D. L. Hamilton, and A. Prestininzi, Casa Editrice Universita La Sapienza, Rome, 309–407 p., doi:10.4408/IJEGE.2011-03.B-045.

Benzi, M. (2002), Preconditioning techniques for large linear systems: A survey, *J. Comput. Phys.*, *182*(2), 418–477.

Braun, J., and S. D. Willett (2013), A very efficient O(n), implicit and parallel method to solve the stream power equation governing fluvial incision and landscape evolution, *Geomorphology*, *180–181*, 170–179, doi:10.1016/j.geomorph.2012.10.008.

Cohen, S., A. J. Kettner, J. P. Syvitski, and B. M. Fekete (2011), WBMsed, a distributed global-scale riverine sediment flux model: Model description and validation, *Comput. Geosci.*, *53*, 80–93.

CSDMS (2013), Univ. of Colorado, Boulder, CSDMS Web page. [Available at http://csdms.colorado.edu.]

Eddins, S. (2007), Upslope area—Forming and solving the flow matrix, MathWorks. [Available at http://blogs.mathworks.com/steve/2007/08/07/upslope-area-flow-matrix/.]

Falgout, R., T. Kolev, J. Schroder, P. Vassilevski, and U. M. Yang (2012), Lawrence Livermore National Laboratory, HYPRE Web page. [Available at https://computation.llnl.gov/casc/hypre/software.html.]

Farr, T. G., et al. (2007), The shuttle radar topography mission, *Rev. Geophys.*, *45*, RG2004, doi:10.1029/2005RG000183.

Freund, R. (1993), A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems, *SIAM J. Sci. Comput.*, *14*(2), 470–482, doi:10.1137/0914029.

Gesch, D., G. Evans, J. Mauck, J. Hutchinson, and W. Carswell Jr. (2009), The national map: Elevation, *U.S. Geol. Surv. Fact Sheet* 3053.

Glennie, C., W. Carter, R. Shrestha, and W. Dietrich (2013), Geodetic imaging with airborne LiDAR: The Earth's surface revealed, *Rep. Prog. Phys.*, *76*(8), 086801.

Hestenes, M., and E. Stiefel (1952), Methods of conjugate gradients for solving linear systems, *J. Res. Natl. Bur. Stand. U.S.*, *49*(6), 409–436.

Hirano, A., R. Welch, and H. Lang (2003), Mapping from ASTER stereo image data: DEM validation and accuracy assessment, *ISPRS J. Photogramm. Remote Sens.*, *57*(5–6), 356–370, doi:10.1016/S0924-2716(02)00164-8.

Howard, A. D. (1994), A detachment-limited model of drainage basin evolution, *Water Resour. Res.*, *30*(7), 2261–2285.

Hysom, D., and A. Pothen (2001), A scalable parallel algorithm for incomplete factor preconditioning, *SIAM J. Sci. Comput.*, *22*(6), 2194–2215.

Jenson, S., and J. Domingue (1988), Extracting topographic structure from digital elevation data for geographic information system analysis, *Photogramm. Eng. Remote Sens.*, *54*(11), 1593–1600.

Krieger, G., A. Moreira, H. Fiedler, I. Hajnsek, M. Werner, M. Younis, and M. Zink (2007), TanDEM-X: A satellite formation for high-resolution SAR interferometry, *IEEE Trans. Geosci. Remote Sens.*, *45*(11), 3317–3341.

Lehner, B., K. Verdin, and A. Jarvis (2008), New global hydrography derived from spaceborne elevation data, *Eos Trans. AGU*, *89*(10), 93, doi:200810.1029/2008EO100001.

Mark, D. M. (1988), Network models in geomorphology, in *Modelling Geomorphological Systems*, edited by M. G. Anderson, pp. 73–97, John Wiley, N. Y.

Metz, M., H. Mitasova, and R. S. Harmon (2011), Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search, *Hydrol. Earth Syst. Sci.*, *15*(2), 667–678, doi:10.5194/hess-15-667-2011.

Milliman, J. D., and K. L. Farnsworth (2011), *River Discharge to the Coastal Ocean: A Global Synthesis*, Cambridge Univ. Press, Cambridge, U. K.

Milliman, J. D., and J. P. Syvitski (1992), Geomorphic/Tectonic control of sediment discharge to the ocean: The importance of small mountainous rivers, *J. Geol.*, *100*(5), 525–544.

Mulder, T., and J. P. Syvitski (1996), Climatic and morphologic relationships of rivers: Implications of sea-level fluctuations on river loads, *J. Geol.*, *104*(5), 509–523.

Naumov, M. (2012), Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS, *Tech. Rep. NVR-2012-003*, Nvidia. Santa Clara, Calif.

O'Callaghan, J. F., and D. M. Mark (1984), The extraction of drainage networks from digital elevation data, *Comput. Vis. Graph. Image Process.*, *28*(3), 323–344, doi:10.1016/S0734-189X(84)80011-0.
Perron, J., W. Dietrich, and J. Kirchner (2008), Controls on the spacing of first-order valleys, *J. Geophys. Res.*, *113*, F04016, doi:10.1029/2007JF000977.
Perron, J., J. Kirchner, and W. Dietrich (2009), Formation of evenly spaced ridges and valleys, *Nature*, *460*(7254), 502–505.
Perron, J. T., P. W. Richardson, K. L. Ferrier, and M. Lapôtre (2012), The root of branching river networks, *Nature*, *492*(7427), 100–103.
Quinn, P., K. Beven, P. Chevallier, and O. Planchon (1991), The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models, *Hydrol. Processes*, *5*(1), 59–79, doi:10.1002/hyp.3360050106.
Richardson, L. (1910), On the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam, *Proc. R. Soc. London. Ser. A*, *83*(563), 335–336.
Saad, Y. (1993), A flexible inner-outer preconditioned GMRES algorithm, *SIAM J. Sci. Comput.*, *14*(2), 461–469.
Saad, Y. (2003), *Iterative Methods for Sparse Linear Systems*, Soc. for Ind. and Appl. Math., Philadelphia, Penn.
Saad, Y., and M. Schultz (1986), GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, *7*(3), 856–869.
Schmidt, F., and A. Persson (2003), Comparison of DEM data capture and topographic wetness indices, *Precis. Agric.*, *4*, 179–192, doi:10.1023/A:1024509322709.
Schwanghart, W., and N. J. Kuhn (2010), Topotoolbox: A set of Matlab functions for topographic analysis, *Environ. Modell. Software*, *25*(6), 770–781, doi:10.1016/j.envsoft.2009.12.002.
Shalf, J., S. Dosanjh, and J. Morrison (2011), Exascale computing technology challenges, in *High Performance Computing for Computational Science—VECPAR 2010, Lecture Notes in Comput. Sci.*, vol. 6449, pp. 1–25, Springer, Berlin, doi:10.1007/978-3-642-19328-6_1.
Syvitski, J. P., and J. D. Milliman (2007), Geology, geography, and humans battle for dominance over the delivery of fluvial sediment to the coastal ocean, *J. Geol.*, *115*(1), 1–19.
Syvitski, J. P., C. J. Vörösmarty, A. J. Kettner, and P. Green (2005), Impact of humans on the flux of terrestrial sediment to the global coastal ocean, *Science*, *308*(5720), 376–380, doi:10.1126/science.1109454.
Tarboton, D. G. (1997), A new method for the determination of flow directions and upslope areas in grid digital elevation models, *Water Resour. Res.*, *33*(2), 309–319, doi:10.1029/96WR03137.
Tarboton, D. G. (2013), Utah State University, TauDEM Web page. [Available at http://hydrology.usu.edu/taudem/taudem5.]
Tarboton, D. G., R. L. Bras, and I. Rodriguez-Iturbe (1991), On the extraction of channel networks from digital elevation data, *Hydrol. Processes*, *5*(1), 81–100.
Tesfa, T. K., D. G. Tarboton, D. W. Watson, K. A. Schreuders, M. E. Baker, and R. M. Wallace (2011), Extraction of hydrological proximity measures from DEMs using parallel processing, *Environ. Modell. Software*, *26*, 1696–1709.
Tewarson, R. P. (1973), *Sparse Matrices, Math. Sci. and Eng.*, Elsevier Sci., Amsterdam.
Tucker, G. E., and G. R. Hancock (2010), Modelling landscape evolution, *Earth Surf. Processes Landforms*, *35*(1), 28–50.
van der Vorst, H. (1992), Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, *13*, 631.
Vörösmarty, C., B. Fekete, M. Meybeck, and R. Lammers (2000), Global system of rivers: Its role in organizing continental land mass and defining land-to-ocean linkages, *Global Biogeochem. Cycles*, *14*(2), 599–621.
Wallace, R. M., D. G. Tarboton, D. W. Watson, K. A. T. Schreuders, and T. K. Tesfa (2010), Parallel algorithms for processing hydrologic properties from digital terrain, in *Sixth International Conference on Geographic Information Science*, edited by R. Purves, and R. Weibel, Zurich, Switzerland.
Wallis, C., R. M. Wallace, D. G. Tarboton, D. W. Watson, K. A. T. Schreuders, and T. K. Tesfa (2009), Hydrologic terrain processing using parallel computing, in *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation*, edited by R. S. Anderssen, R. D. Braddock and L. T. H. Newham, pp. 2540–2545, Modell. and Simul. Soc. of Aust. and N. Z. and Intl. Assoc. for Math. and Comput. in Simul., Cairns, Australia.
Widlund, O. (1978), A Lanczos method for a class of nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.*, *15*(4), 801–812.
Willgoose, G. (2005), Mathematical modeling of whole landscape evolution, *Annu. Rev. Earth Planet. Sci.*, *33*, 443–459.
Willgoose, G., R. L. Bras, and I. Rodriguez-Iturbe (1991), A physical explanation of an observed link area-slope relationship, *Water Resour. Res.*, *27*(7), 1697–1702.
Zink, M., G. Krieger, H. Fiedler, I. Hajnsek, and A. Moreira (2008), The TanDEM-X mission concept, in *7th European Conference on Synthetic Aperture Radar (EUSAR)*, pp. 1–4, Berlin, Germany.