Advances in Water Resources 53 (2013) 23–32

Contents lists available at SciVerse ScienceDirect

Advances in Water Resources

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

An asynchronous solver for systems of ODEs linked by a directed tree structure

Scott J. Small [a,*], Laurent O. Jay [b], Ricardo Mantilla [a], Rodica Curtu [b], Luciana K. Cunha [a], Morgan Fonley [b], Witold F. Krajewski [a]

[a] IIHR-Hydroscience & Engineering, The University of Iowa, Iowa City, IA 52242, USA
[b] Department of Mathematics, The University of Iowa, Iowa City, IA 52242, USA

ARTICLE INFO

Article history:
Received 10 February 2012
Received in revised form 14 September 2012
Accepted 22 October 2012
Available online 31 October 2012

Keywords:
Asynchronous integrator
Tree structure
Hillslope-link river basin model
Distributed memory
Asynchronous communication

ABSTRACT

This paper documents our development and evaluation of a numerical solver for systems of sparsely linked ordinary differential equations in which the connectivity between equations is determined by a directed tree. These types of systems arise in distributed hydrological models. The numerical solver is based on dense output Runge–Kutta methods that allow for asynchronous integration. A partition of the system is used to distribute the workload among different processes, enabling a parallel implementation that capitalizes on a distributed memory system. Communication between processes is performed asynchronously. We illustrate the solver capabilities by integrating flow transport equations for a $\sim$17,000 km^2 river basin subdivided into 305,000 sub-watersheds that are interconnected by the river network. Numerical experiments for a few models are performed and the runtimes and scalability on our parallel computer are presented. Efficient numerical integrators such as the one demonstrated here bring closer to reality the goal of implementing fully distributed real-time flood forecasting systems supported by physics based hydrological models and high-quality/high-resolution rainfall products.

1. Introduction

Consider an initial-value problem for a general system of ordinary differential equations (ODEs) of the form

$$\frac{dy}{dt} = f(t,y), \tag{1a}$$

$$y(t_0) = y_0, \tag{1b}$$

where $y \in \mathbb{R}^d$ is a vector and f is a vector valued function. Numerical integrators for systems of ODEs are plentiful, and numerical analysis textbooks typically contain a chapter on them. A common example is offered by the family of explicit Runge–Kutta methods, with various orders of approximations, which can be applied directly to (1) in order to obtain a numerical solution. However, truly efficient solvers oftentimes utilize particular properties of the specific system under consideration.

For instance, if the system of ODEs exhibits stiffness, an explicit method can be a poor choice of integrator, whereas an implicit Runge–Kutta method would allow for larger step sizes and improved efficiency. If discontinuities are present in the derivatives of the right-hand side function f then low order integrators could result in fewer computations without compromising the order of the solution's approximation. If high order methods are desired, then spending CPU time locating these discontinuities may be warranted. When the system (1a) can be decomposed into the form

$$\frac{dy_1}{dt} = f_1(t,y_1), \tag{2a}$$

$$\frac{dy_2}{dt} = f_2(t,y_1,y_2), \tag{2b}$$

the first equation (2a) can be solved independently of (2b), which requires solving two smaller systems of ODEs, with the numerical solution of (2a) used as an input to the other system (2b). This can be quite beneficial to implicit solvers, as the size of the corresponding nonlinear systems to be solved is reduced.

In this paper, we focus on systems of ODEs in which the linkage between equations is determined by a sparse directed tree structure. This kind of linkage structure arises in distributed hydrological models in which a river network of interconnected streams provides a natural tree-like connecting structure (see Fig. 1). Under the assumption that the model for the flow of water through a single stream can be written as a system of ODEs, the equations from every stream in the entire network can be viewed as one large system of ODEs (e.g., [1,2]). In general, hydrological models that use a semi-discrete approximation of flow transport of the Saint–Venant equations or the Muskingum method fall into this category [3]. Although this application is of principal importance to the authors (see [1,4–6]), the results in this paper are more general and extend to other types of physical problems where the connectivity

* Corresponding author. Tel.: +1 319 335 5956; fax: +1 319 335 5238.
E-mail address: scott-small@uiowa.edu (S.J. Small).

0309-1708/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.advwatres.2012.10.011

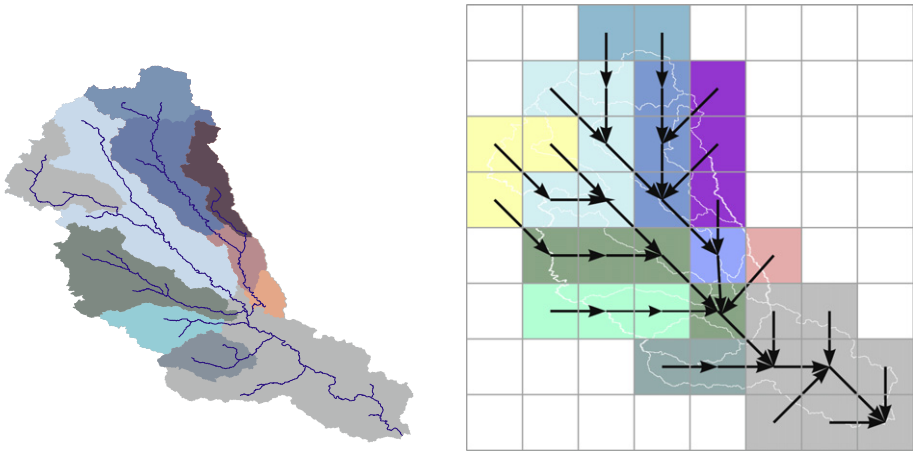


Fig. 1. Partitioning schemes in distributed hydrological models that lead to a global system of ODEs. Sub-watershed partitioning on the left and square pixel partitioning on the right.

between equations is determined by a directed tree structure. Thus, the problem and algorithms will make no assumptions about the underlying application, except for the presence of discontinuities.

We offer a description of an efficient solver for systems of ODEs linked by a directed tree structure. Section 2 briefly introduces the system of ODEs that arise in solving flow transport on a hillslope-link based river basin model. Section 3 covers the problem statement and some basic definitions on directed tree structures. Section 4 considers the actual implementation of the solver, while the approach taken for parallelizing the solver is discussed in Section 5. Numerical examples from the river basin model are presented in Section 6. This paper ends with some conclusions in Section 7.

2. Flow transport on a hillslope-link based river basin model

We consider the link-hillslope decomposition of a river basin described by Mantilla and Gupta [1]. Equations for each link follow the general form described by Mantilla [7] that has been applied in other contexts of self-similar trees by Menabde [8]. The topology of a river network can be described as a directed tree structure under the assumption that no loops are present in the network and no channel splits (e.g., braided streams) are considered. For each link (i.e., river segment between two junctions), a differential equation can be written for water discharge by describing the link as a nonlinear reservoir. Index each link in the basin, and let $\mathcal{U}_i$ be the collection of the indices of each link with channel flowing directly into the link with index i. Then, for link i, the discharge $q(i,t)$ of water (in m^3/s) from the channel and the depth of water (in m) ponded on the channel's surrounding hillslope $s_p(i,t)$ at time t (measured in minutes) can be modeled by the system of ODEs

Table 1
Description of the constants used in the river basin model. Typical values are given under the last column.

Constant	Description	Range
$A_h(i)$	Area of the surrounding hillslope at link i	$(0,5]$ km^2
$A(i)$	Sum of the A_h's for all links upstream from i	$[0.01,10^6]$ km^2
$L(i)$	Length of link i	$[10,1000]$ m
$S(i)$	Average slope of hillslope surrounding i	$[0.0,0.5]$
$\eta(i)$	Manning coefficient at link i	$[0.1,0.8]$
v_r	Reference flow velocity	$[0.2,1.0]$ m/s
RC	Runoff coefficient	$(0.0,1.0]$
λ_1	Exponent for flow velocity discharge	$[0.0,0.7]$
λ_2	Exponent for flow velocity upstream area	$[-0.4,-0.05]$

$$\frac{dq}{dt}(i,t) = \frac{1}{\tau} q(i,t)^{\lambda_1} \left(\sum_{j \in \mathcal{U}_i} q(j,t) - q(i,t) + c_1 s_p(i,t)^{5/3} \right), \tag{3a}$$

$$\frac{ds_p}{dt}(i,t) = c_2 p(i,t) - c_3 s_p(i,t)^{5/3}, \tag{3b}$$

where the values τ, c_1, c_2, and c_3 are constants at each link and are given by

$$\tau = \frac{(1-\lambda_1)L(i)}{60 v_r A(i)^{\lambda_2}}, \tag{4a}$$

$$c_1 = \frac{2L(i)}{0.6} \cdot \frac{S(i)^{1/2}}{\eta(i)}, \tag{4b}$$

$$c_2 = \frac{10^{-3}}{60} RC, \tag{4c}$$

$$c_3 = \frac{2L(i)}{0.6 A_h(i)} \cdot \frac{S(i)^{1/2}}{\eta(i)} (60 \cdot 10^{-6}). \tag{4d}$$

These constants also handle unit conversion. The parameters that appear in (4) are described in Table 1. The constants v_r, RC, λ_1, and λ_2 are universal amongst the entire system. The term $c_1 s_p^{5/3}$ in Eq. (3a) can be interpreted as the flow of water from the surrounding hillslope into the channel, and $p(i,t)$ is a known function for the precipitation at link i measured in mm/s.

3. Problem description and definitions

We consider initial-value problems for systems of ODEs

$$\frac{dy}{dt} = f(t,y), \tag{5a}$$

$$y(t_0) = y_0. \tag{5b}$$

The function $y(t) \in \mathbb{R}^d$ is the unknown and the variable $t \in [t_0, t_f]$ is the independent variable, referred to as time. For this paper, we consider $f(t,y)$ in (5a) as additionally having a tree structure, i.e., the system satisfies the following definition.

Definition 3.1. The initial-value problem (5) is said to have a *tree structure* if

1. The unknown $y(t)$ and function $f(t,y)$ are of the form

$$\begin{aligned} y(t) &= [y_1(t), y_2(t), \ldots, y_N(t)], \\ f(t,y) &= [f_1(t,y), f_2(t,y), \ldots, f_N(t,y)], \end{aligned} \tag{6}$$

with each vector $y_i(t), f_i(t,y) \in \mathbb{R}^{n_i}$, for $i = 1, \ldots, N$, and $d = \sum_{i=1}^{N} n_i$.

2. There exists a directed rooted tree T with labeled edges $1, 2, \ldots, N$ such that every f_i is a function restricted to y_i and to all y_j with edge j being a parent of edge i in T.

The root of the tree corresponds to the unique y_i that is not an argument of any f_j, except when $j = i$. Every f_i is a function of y_i, and every y_i can be an argument of at most one function f_j, for $j \neq i$. A system of ODEs satisfying the above definition can be represented graphically as a directed tree T, as demonstrated in the next example.

3.1. A simple example

As a simple example, consider the linear system of ODEs of the form $\frac{dy}{dt} = Ay$, where the matrix A and unknown vector y are given by Fig. 2. This system is of the form (5) with $d = 10$ and the right-hand side function $f(t, y) = f(y) = Ay$. This system also satisfies condition 1 from Definition 3.1, with $N = 5$, $n_i = 2$ for all i, and

$$\begin{aligned} &y_i(t) = [x_{2i-1}(t)\ x_{2i}(t)], \quad i = 1, \ldots, 5, \\ &y(t) = [y_1(t)\ y_2(t)\ \ldots\ y_5(t)], \\ &f(y) = [f_1(y)\ f_2(y)\ \ldots\ f_5(y)]. \end{aligned} \tag{7}$$

Further, condition 2 is satisfied by the tree given in Fig. 3, as the functions f_i satisfy

$$\begin{aligned} &f_1(y) = f_1(y_1, y_2, y_3), \quad f_2(y) = f_2(y_2), \\ &f_3(y) = f_3(y_3, y_4, y_5), \quad f_4(y) = f_4(y_4), \\ &f_5(y) = f_5(y_5). \end{aligned} \tag{8}$$

In addition, the variable y_1 is an argument only to the function f_1, while each y_i with $i \neq 1$ is an argument of some f_j with $j \neq i$. Each edge in Fig. 3 represents one of the systems with right-hand side function f_i.

3.2. Conceptualization and definitions

In light of the tree representation of problem (5) with Definition 3.1, the system of ODEs can be viewed as N systems of n_i-dimensional ODEs. The system $\frac{dy_i}{dt} = f_i(t, y)$ will be referred to as *link i* (or *edge i*), for $i = 1, \ldots, N$. In accordance with the language of graph theory (see for example [13]), each link i that depends only upon y_i is referred to as a *leaf*, and a link j is called a *root* if it is the only link with y_j as an argument of the right-hand side function f_j. For instance, in the example of Section 3.1, link 1 is the root, and links 2, 3, and 5 are leaves.

Link j is a *parent* of link $i \neq j$ if f_i is a function of y_j. In this situation, link i is called the *child* of link j. Every root has no child, and every non-root link has exactly one child. A link has no parents if and only if it is a leaf. This terminology is clear when the system is viewed graphically.

$$A := \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad y := \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix}$$

Fig. 2. The matrix A and unknown y for the explicit example.

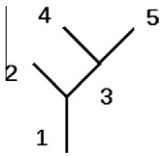


Fig. 3. Tree representation for the explicit example. The edge labeled 1 corresponds to the system $\frac{dy_1}{dt} = f_1(t, y_1, y_2, y_3)$, the edge labeled 2 corresponds to the system $\frac{dy_2}{dt} = f_2(t, y_2)$, etc. The vertices of the tree have been suppressed. The edges are directed down.

Although we focus on problems with a tree structure, the techniques in this paper are easily generalized to systems of ODEs with a forest structure; i.e., systems with multiple, independent trees. More precisely, the system (5) has a forest structure if it contains more than one root. Clearly, for a forest structure, multiple trees must be allowed in Definition 3.1.

The system of ODEs that results from considering the river basin model in Section 2 is a system of ODEs with a tree structure, with $n_i = 2$ for all i and N denoting the number of channels in the river basin.

4. Solving the system

An intuitive solution for systems of ODEs with a tree structure involves finding the solution for external leaves and propagating the solution inwards into the internal tree links. However, care must be taken when selecting step sizes and with memory management.

Suppose we wish to solve the system (5) having a tree structure from an initial time t_0 to a final time t_f. Our procedure is to first integrate the leaves a few steps. Now, all links with only leaves as parents may be integrated up to the time in which their parents have been integrated. This procedure is repeated for any link with every parent that is integrated a few steps until a solution at the root is computed. The leaves are now integrated a few more steps, and the procedure is repeated. This whole process can be carried out concurrently on various subtrees. Using this incremental procedure, we can reduce memory usage (see Section 4.4).

4.1. Synchronous vs. asynchronous integration

The solution of the leaves does not depend upon the solutions of any other link. Therefore, these equations are free to be solved anytime using a numerical integration technique (for example, a Runge–Kutta method) from an initial time t_0 to some intermediate time $t_i \leqslant t_f$. Once the equations for the leaves have been solved, some links in the system have only parents that have been integrated to at least t_i (all their parents are leaves). We can thus integrate each of these links, using the solutions computed for the leaves, through time t_i. The procedure can then be repeated, integrating any links whose parents have been integrated to at least time t_i, until a numerical solution is calculated for every link to at least time t_i. The root of the tree will be the last link to be integrated. An example of this procedure is given in Fig. 4.

Care must be taken when each equation is to be solved in this manner. To demonstrate, consider a link i with two parents, p and q (see Fig. 5). The solution of link i is known at time 0; the solution of link p has been approximated at times 0 and 0.1; and the solution of link q has been approximated at times 0 and 0.2. In this

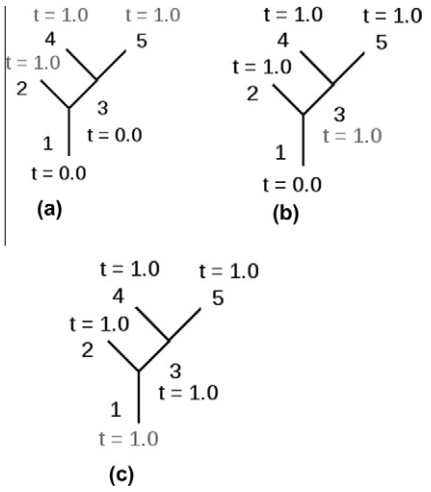


Fig. 4. An example of how to integrate a system. All links start at time $t_0 = 0.0$. (a) All leaves can be integrated, for example, to time $s = 1.0$. (b) Each parent of link 3 has been integrated, so link 3 can now be integrated to time $s = 1.0$. (c) Link 1 can now be integrated out to time $s = 1.0$.

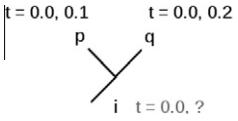


Fig. 5. If links p and q have been integrated at two different times, then their numerical solutions must be interpolated to generate data for link i.

situation, link i can only be solved to, at most, time 0.1. But to integrate link i at time 0.1, the solution of q at time 0.1 is needed. There are two methods to deal with such situations.

1. Avoid this situation entirely by synchronizing the time steps at each link so that whenever link i needs information about the solution of a link j at a time t, that information will be available. In other words, the numerical solver at every link will take the same time steps and all equations are solved concurrently. This has the advantage that no solution history is required to be stored. Such a scheme is called a *synchronous integration scheme*.
2. Allow the numerical solvers for each link to have different step sizes. Whenever link i needs information about the solution of a link j at a time t, that information can be interpolated from other computed values of the solution. The equations are not solved concurrently and the history of the solution must be partially stored. Such a scheme is called an *asynchronous integration scheme*.

We opt for the second approach. Although a synchronous approach is simpler to implement, there is a disadvantage when using error control techniques. Note that when a uniform step size is taken, the smallest step size that guarantees a desired error tolerance amongst all the links must be used to integrate at *every* link. This can lead to some links taking much smaller time steps than necessary, which creates inefficiencies. However, with an asynchronous integration approach, each link can be integrated independently with an appropriate step size.

4.2. Numerical methods

In our implementation, we use the so-called *dense output Runge–Kutta methods*. These methods have a built-in mechanism to obtain a continuous solution between two time steps, allowing the aforementioned asynchronous integration approach.

Consider a general system (5). A dense output Runge–Kutta method applied to this problem is defined as follows.

Definition 4.1. One step of an s-stage dense output Runge–Kutta method with step size h applied to the general system of ODEs (5) is a formula of the form

$$k_i = f\left(t_0 + c_i h, y_0 + h\sum_{j=1}^{s} a_{ij}k_j\right), \quad i = 1,\ldots,s, \tag{9a}$$

$$u(\theta) = y_0 + h\sum_{i=1}^{s} b_i(\theta)k_i \tag{9b}$$

with each $a_{ij}, c_i \in \mathbb{R}$, and $0 \leqslant \theta \leqslant 1$. The coefficients $b_i(\theta)$ are polynomials so that $u(\theta)$ is an approximation to the exact solution $y(t)$ of (5) at time $t_0 + h\theta$.

For $\theta = 1$, we obtain $u(1) = y_1 \approx y(t_0 + h)$. Many such methods exist in the literature. We give the dense output for the RK4 method in Fig. 6, with coefficients listed in a Butcher tableau. The dense output of this method gives an order 3 approximation between steps. Another example is the dense output of the Dormand & Prince method, see, for example ([9], Section II.6). This method is of order 5 and has a dense output approximation of order 4. The order of dense output methods is discussed, for example, in [10]. In general, the order of the dense output should be at least equal to the order of the integrator minus 1. Otherwise the numerical solution used by the downstream links will be reduced in order to that of the dense output. Certainly, if stiffness of the system is a consideration, implicit RK methods with dense output may be used. For our implementation, we apply dense output Runge–Kutta methods to the system of ODEs at each link.

Note that we need not solve all links with the same integration method. For instance, using higher order numerical methods on the leaves could lower error propagated through the entire system, or implicit solvers could be used at stiff links while explicit solvers could be used at nonstiff links. The order of convergence of the numerical solver at a link can be affected by the asynchronous integration scheme if integrators of lower order are used at upstream links. Although we do not discuss the exact nature of this here, the numerical integration order can be obtained through a variation of parameters type argument (see, for example, [9]).

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

$$b_1(\theta) = \theta - \tfrac{3}{2}\theta^2 + \tfrac{2}{3}\theta^3$$
$$b_2(\theta) = \theta^2 - \tfrac{2}{3}\theta^3$$
$$b_3(\theta) = \theta^2 - \tfrac{2}{3}\theta^3$$
$$b_4(\theta) = -\tfrac{1}{2}\theta^2 + \tfrac{2}{3}\theta^3$$

Fig. 6. Coefficients for the dense output RK4 method.

4.3. Discontinuities

Care should also be taken if the right-hand side function f is discontinuous or has discontinuous derivatives. Ignoring these discontinuities can result in loss of error control and increased runtimes from numerous step rejections. A discussion on discontinuities in ODEs can be found in ([9], Section II.6). For general functions f, discontinuities can be located using dense output methods (see [9,11]), for example (9). For example, a model problem that incorporates reservoirs will introduce derivative discontinuities (see, for example, [12]). The discontinuities introduced by the model considered in Section 2, however, are only time dependent and are known in advance.

More specifically, in the river basin model (3), the function $p(i,t)$ representing precipitation is specified by a piecewise constant function where the precipitation rate is constant over a set interval of time at each link. To integrate these systems efficiently, the integrators must take care to "step" on any times where a change in precipitation occurs.

Suppose link i has been integrated to a time t_m, and the step size is h_m. Suppose further that at time t^* with $t_m < t^* < t_m + h_m$, the function $p(i,t)$ has a jump discontinuity. We would therefore reduce the step size h_m to $t^* - t_m$. For the numerical solver, we take for the value of $p(i,t^*)$ its limit from the left when computing the numerical solution s_p^* of (3b) at time t^*. But we use the limit from the right when using s_p^* to compute numerical solutions at times greater than t^*. This will allow the solver to control the error regardless of the discontinuities.

Another consideration is the propagation of discontinuities through the system of differential equations. Discontinuities in the function $p(i,t)$ in the Eqs. (3) manifest themselves as discontinuities in the derivative of $s_p(i,t)$ and discontinuities in the second derivative of $q(i,t)$. If j is the child link of i, then a discontinuity in $p(i,t)$ becomes a discontinuity in the second derivative of the differential equations at link j. If k is the child link of j, then this discontinuity becomes a discontinuity in the third derivative at k, and so on down to the root link. An example of this is given in Fig. 7. To improve efficiency, these discontinuities should be tracked.

In our implementation, this tracking process is done by storing at every link a list of the times in which a discontinuity in the right-hand side function or one of its derivatives occurs. Note however that this propagation need not be tracked throughout the entire system. Once the order of the derivative in which a discontinuity occurs is greater than the order of the highest order integrator used in the system, tracking can stop.

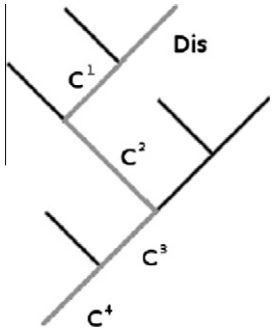


Fig. 7. A demonstration of how discontinuities propagate through the river basin model (3). The right-hand side function of the differential equation at the link labeled *Dis* is discontinuous in time, the right-hand side function of the differential equation at the link labeled C^1 has a continuous derivative, the right-hand side function at the link labeled C^2 has a continuous second derivative, etc. All unlabeled links have no discontinuities in the corresponding right-hand side function. For simplicity, a change in rainfall is assumed to occur only at the link labeled *Dis*.

4.4. Memory management

Solving a system of ODEs in a tree structure using an asynchronous integration scheme does not require that we store the entire history of every link. For a link i, we must store a time t_j and the corresponding k_μ for $\mu = 1,\ldots,s$ in Definition 4.1, so that other links can use appropriate approximate solutions of link i. For simplicity, we denote the collection of s vectors k_μ for link i at time t_l as V_l^i. For each link, we will have in memory a list of the form

$$L_i := \{t_l^i, V_l^i, t_{l+1}^i, V_{l+1}^i, \ldots, t_m^i, V_m^i\}. \tag{10}$$

Each pair t_j^i, V_j^i represents a computed step of the numerical method. Similarly, if link j is a parent of link i, then link j will have a list

$$L_j := \{t_q^j, V_q^j, t_{q+1}^j, V_{q+1}^j, \ldots, t_r^j, V_r^j\}. \tag{11}$$

We can assume initially that $t_m^i < t_q^j$. Link i will use the list L_j to further its calculations to a time $t_n^i > t_q^j$. This time will fall between times t_p^j and t_{p+1}^j in the list L_j. Once the calculations for link i have been performed for the numerical solution at time t_n^i, all data stored in L_j for times before t_p^j can be freed, as they will no longer be needed in any further calculations. This will leave the list L_j as

$$L_j = \{t_p^j, V_p^j, t_{p+1}^j, V_{p+1}^j, \ldots, t_r^j, V_r^j\}, \tag{12}$$

which will contain less data than the list (11). By using this approach, we limit the amount of stored data to that which may still be needed and remove any unneeded data. This approach will use much less memory than storing all computations.

5. Parallel implementation

To implement the above algorithm, we propose a parallel implementation to improve performance. We focus here on a distributed architecture. Our approach is to partition the edges of the tree corresponding to the system of ODEs amongst the processes, creating a partition of the entire system. Ideally, we want to partition the links into p clusters (one for each process) with an equal number of links. Further, if link i is the parent of link j, and link i is assigned to a cluster A while link j is assigned to a different cluster B, then communication must occur. We would further like to minimize the number of links with children in other clusters and reduce the time in which the processes remain idle.

To summarize, the most ideal partitioning algorithm partitions the system into p clusters with a process assigned to each cluster so that

1. each cluster has the same number of links,
2. the number of links assigned to cluster A with a child link in a different cluster B is minimized,
3. the amount of time a process spends waiting for data from other processes is minimized.

The first two conditions give a special case of the graph-partitioning problem. The general graph partitioning problem is well known to be NP-complete [13]. However, heuristic approaches for solving the problem exist (see for example [14,15]). For graphs that are trees, the partitioning problem is still NP-complete [16]. Polynomial time algorithms have been proposed for the graph partitioning problem for trees using slightly different restrictions for the clusters [16]. These algorithms are designed to distribute workload on shared memory systems. [17] proposes a linear time algorithm with both conditions 1 and 2 above but requires the number of clusters to be variable.

We propose a heuristic approach for partitioning the system. Assuming there are p processes indexed from $1,\ldots,p$, the

algorithm is given in Algorithm 1. Recall that N is the number of links and M is the number of leaves.

Algorithm 1. Partitioning the system amongst p processes

1 Perform a depth-first search on the tree formed by the system of N ODEs, starting from the root.
Maintain a list of the M leaves in the order in which they are visited.
Call the list $l_1, l_2, \ldots, l_M$.
2 Calculate $L \leftarrow \left\lfloor \frac{N}{p} \right\rfloor$ (i.e., the greatest integer less than or equal to $\frac{N}{p}$).
Assign $l_1, \ldots, l_L$ to process 1, $l_{L+1}, \ldots, l_{2L}$ to process 2, etc.
Any remaining unassigned leaves are assigned to process p.
3 **for** $i \leftarrow 1$ to M **do**
  $P \leftarrow l_i$;
  while *Parent link P has a child link C that is not assigned to a process* **do**
    Assign C to the process to which P is assigned.
    $P \leftarrow C$:
  end
end

An example of this algorithm with three processes is given in Fig. 8. The idea of step 1 is that if a link has parents that are leaves each parent will appear consecutively in this list. Graphically, this can be viewed as listing the leaves "from left to right." For example, in Fig. 3, the leaves would be listed as 2, 4, and 5. In step 2, the first process receives the first L leaves in the list; the second receives the next L leaves; etc. Any unassigned leaves (because p may not divide M) are assigned to the last process p.

While solving the system, if a link i is assigned to a process a and one of its parents j is assigned to process b ($a \neq b$), then communication will occur between processes a and b, as link i needs information about the solution of link j to be integrated. The main idea of the assignment scheme given in the algorithm above is that communication is *expensive* and should be minimized. The depth-first search of step 1 is designed to try to keep the parents of each edge with the same process assignment as much as possible.

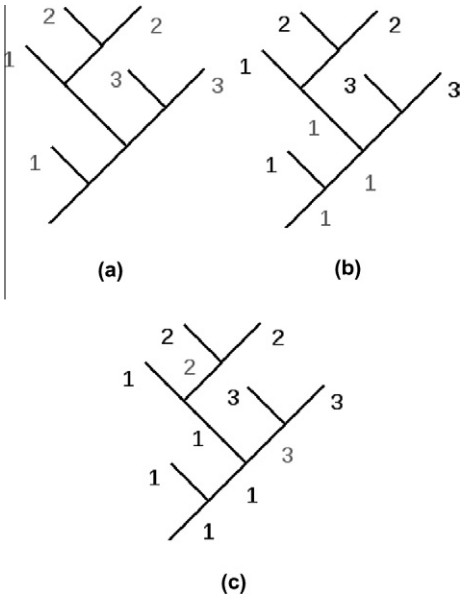


Fig. 8. An example of how to partition the system amongst three processes. Labels on the links represent process assignment: (a) first assign the leaves from left to right (the results of a depth-first search), (b) assign the links to process 1 starting from the leaves assigned to process 1, (c) repeat for the leaves assigned to processes 2 and 3.

The runtime of a depth-first search of a tree is linear with respect to the number of links. So the runtime of step 1 in Algorithm 1 is $\mathcal{O}(N)$. In step 2, each leaf will be assigned to a process, which results in a runtime of $\mathcal{O}(M)$. In step 3, each link is visited once for its assignment. However, a total of $M-1$ links will also be visited again from the terminating condition of the while loop. Therefore, the runtime of step 3 is $\mathcal{O}(N+M)$. The overall runtime for Algorithm 1 is the sum of the runtimes of the three steps, which gives $\mathcal{O}(N+M+N+M) = \mathcal{O}(N)$. So the partitioning algorithm is linear with respect to the number of links in the system.

For a given problem, partitioning the system needs to occur only once and is dependent only upon the arguments of each f_i and not upon any parameters, integration times, or numerical methods. For long integration times, the partitioning algorithm requires a small amount of time compared to the integration time.

We also note that partitioning the basin evenly will not in general lead to a perfectly balanced workload amongst the different processes. Some links may require more computational time as a result of the differential equation used at that link. For instance, more work must be performed at links with varying rainfall rates.

6. Numerical experiments

To demonstrate our algorithms, we have created an implementation in the C programming language. Communication between processes is done using the Message Passing Interface (MPI) (see [18]). All experiments are run on a cluster with 200 nodes, and each node has Dual Quad Core Intel (R) Xeon (R) CPU X5550 @ 2.67 GHz processors with 24 Gb DDR3 1333 MHz memory and 1 Tb of storage.

6.1. Verification of numerical results

To verify our implementation, we apply the algorithms presented here to a particular case of the parameters for which a closed form of the solution of (3) is known. We consider water flows on a Peano network using only the transport equation (3a). Assuming constant velocity (i.e., $\lambda_1 = \lambda_2 = 0.0$), $v_r = 1.0$ m/s, and $L(i) = 500$ m for every link i. In physical terms, this corresponds to a case in which runoff from an instantaneous and uniform storm has made it into the channels of the river network and flows freely downstream. The constant velocity assumption treats each channel link as a linear reservoir. The exact solution at the root node has been recently given by [19] as

$$q_\Omega(t) = q_0 e^{-\frac{t}{\tau}} \sum_{k=1}^{2^\Omega - 1} \frac{3^{b_k} \left(\frac{t}{\tau}\right)^{k-1}}{(k-1)!}, \tag{13}$$

where Ω is the Horton order of the root link in the Peano tree and the sequence b_k takes the integer k into the number of 1's appearing in the binary representation of k (so $b(1) = 1$, $b(3) = 2$, $b(11) = 3$, etc). We have taken the discharge at time 0 in each link to have the same value, q_0. Again, this corresponds to runoff from an instantaneous and uniform pulse of rainfall that instantaneously moves into the channel.

For the experiment, we use the Peano network with a root link having Horton order $\Omega = 10$. This system has a total of $N = 4^9 = 262,144$ links. The initial discharge is taken as $q_0 = 1.0\ \mathrm{m}^3/\mathrm{s}$. For the numerical methods, we apply the 7-stage dense output Dormand and Prince method of order 5 to the ODE at each leaf and the dense output Runge–Kutta method of order

4 from Fig. 6 to all other links. The step sizes are chosen to be the same at each link, and the simulation is run using multiple fixed step sizes. The true solution and numerical errors at the outlet at the final time of 5000 min for the various step sizes are given in Fig. 9. The order of the numerical solution at the root is seen to be 4. This is expected as 4 is the lowest integration order used.

6.2. Effectiveness of implementation

To test the effectiveness of our implementation, we compare its runtime with that of a standard numerical implementation. For the test problem, we use the full model (3), with $v_r = 0.64$ m/s, $RC = 0.5$, $\lambda_1 = 0.24$ and $\lambda_2 = -0.12$. The constants specific to each link are again taken from measurements for the 16,878 km^2 basin formed by the Cedar River basin located in Eastern Iowa. The 30-meter resolution digital elevation model (DEM) was obtained from the National Elevation Dataset [20,21]. A 30-meter resolution digital elevation model (DEM) was used to determine geometrical and topological parameters for the river network and adjacent hillslopes. The corresponding system of ODEs has a directed tree structure with $N = 305,352$ links. Initial inputs of $q(i,0) = 1$ m^3/s and $s_p(i,0) = 0.0$ m are used for every link i in the system.

We apply our implementation to this problem as well as the DOPRI5 routine provided in [9]. The latter solves the problem as one large system of N ODEs using the Dormand and Prince method of order 5. The DOPRI5 solver is written in the Fortran programming language. For our implementation, we also use the same Runge–Kutta method at each link. The solution is computed for 10 days. The runtimes for varying relative error tolerances are given in Table 2, and the error tolerances are for all components. An absolute tolerance of 10^{-20} is also used at each link for instances when a hillslope or channel is nearly empty. The runtimes presented include only the time to calculate the numerical solutions at each time step as well as the time to read the rainfall data from disk. All other operations, including partitioning the basin and initializing the system, are independent of implementation and tolerance and took an average of 7 s. The error is computed every five minutes of integration time with respect to a numerical solution with a small relative error tolerance (10^{-8}). The maximum of these errors is given in Table 2. The solution at the outlet of the basin is given in Fig. 10. The results show that our implementation requires significantly less time than the standard DOPRI5 solver. However, the maximum errors of each implementation are comparable in size.

Table 2
Runtimes for our implementation versus the DOPRI5 solver.

Relative tol	Our implementation		DOPRI5	
	Runtime (min)	Max error	Runtime (min)	Max error
10^{-2}	47.45	1.594598	1020.38	0.838145
10^{-3}	54.00	0.199336	1028.52	0.103745
10^{-4}	68.02	0.020997	993.48	0.282850
10^{-5}	98.05	0.016246	1007.52	0.020642
10^{-6}	148.07	0.013581	965.63	0.010159

6.3. Effects of parallelization

We implemented the algorithms described in this paper using the model presented for a river basin in Section 2. We ran experiments assuming only transport in the channels (i.e., (3a) only), the

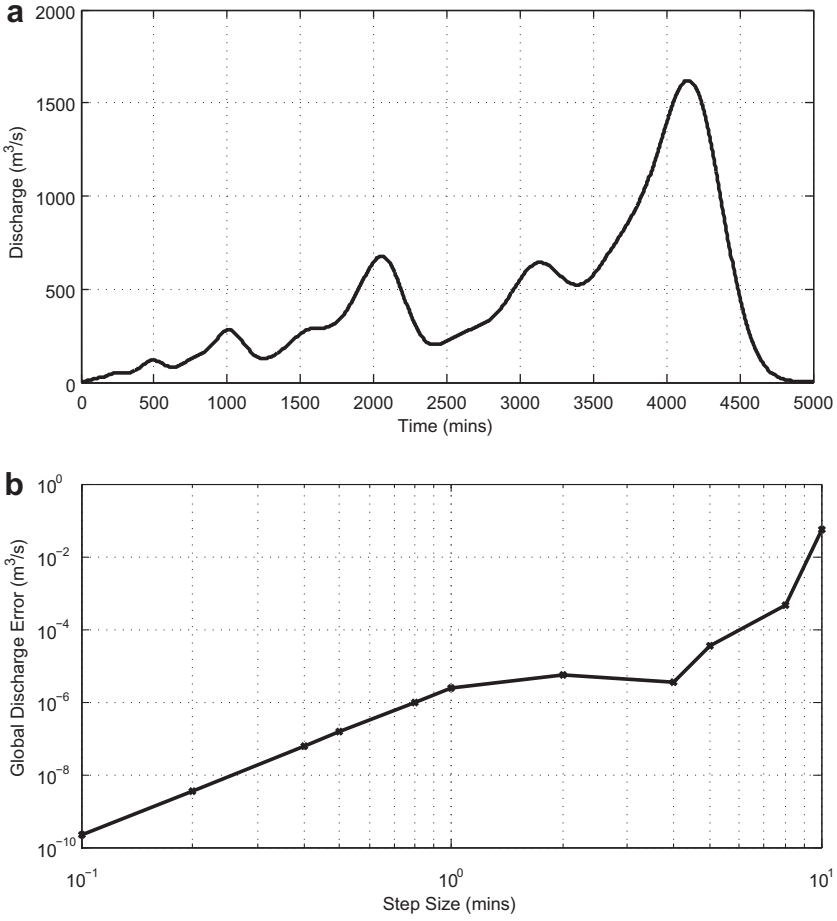


Fig. 9. Exact solution and numerical error for the simulated discharge of the Peano network of Horton order 10.

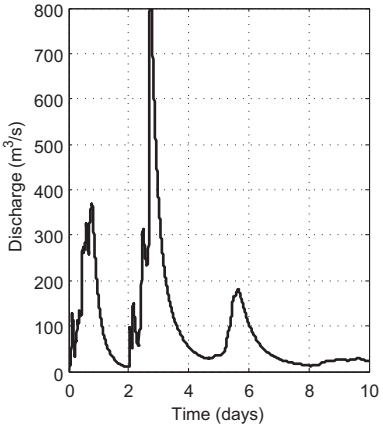


Fig. 10. Simulated discharge at the outlet of the Cedar River basin over 10 days for the comparison with DOPRI5. An initial discharge of 1.0 m^3/s and no initial hillslope ponding applied at each link.

model (3) without precipitation, and finally the full model with precipitation. For these experiments, we used the values $v_r = 0.64$ m/s, $RC = 0.5$, $\lambda_1 = 0.24$ and $\lambda_2 = -0.12$. The constants specific to each link are again taken from measurements for the basin formed by the Cedar River basin located in Eastern Iowa. The precipitation product used to force the hydrological model is a 4-km resolution NEXRAD data provided by the NCDC. The error tolerance for the dense output Runge–Kutta methods is set to a relative tolerance of 10^{-4} for all components of all links. The Dormand and Prince method of order 5 is applied to each link, and the initial time step size at each link is set to 0.1 min. A more comprehensive description of the basin is given by Cunha et al. [22].

Once again, the runtimes presented in this section include only the time to calculate the numerical solutions at each time step as well as time to read the rainfall data from disk. All other operations are roughly independent of the number of processes used and again required an average of 7 s.

6.3.1. River basin model without precipitation

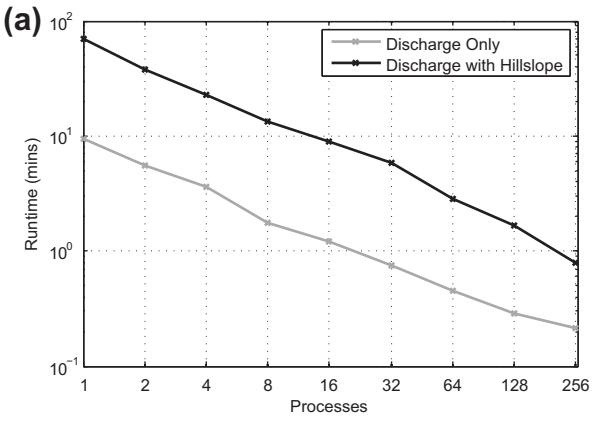
The runtimes and speedups for the river basin model using only the discharge equation (3a) are given in Fig. 11. The numerical solution is given in Fig. 12. The speedup of p processes is defined to be the ratio of the runtime for one process to the runtime for p processes. This gives a measure of the scalability of the implementation. The ponding depth s_p is assumed to be equal to zero for all times, and the integration time is taken to be 20 days. At each link, we assume an initial discharge of 30 m^3/s. The runtimes decrease by a factor of about 1.5 as the number of processes is doubled.

The runtimes and speedups for the river basin model using (3) with hillslope is also given in Fig. 11, assuming no precipitation ($p(i,t) = 0$ for all t and links i). The numerical solution is given in Fig. 13. The integration time is again taken to be 20 days. Each link is assumed to have an initial hillslope ponding depth of $s_p(i,0) = 0.1$ m, and the initial discharge is again $q(i,0) = 30$ m^3/s. The runtimes drop by almost a factor of around 1.5 as the number of processes is doubled.

Ideally, the speedups should be linear with p processes (i.e., equal to p) for very good scalability. The speedups from both experiments in Fig. 11 are much less than p, especially when p is large. This demonstrates a need for improved load balancing.

6.3.2. River basin model forced by a radar-derived precipitation product

The full model (3) is tested assuming continuous rainfall. The rainfall product for the year 2008 was obtained from



(b)

Processes	Discharge	Speedup	Discharge & Hillslope	Speedup
1	9.42	–	69.60	–
2	5.60	1.68	38.27	1.82
4	3.60	2.62	22.68	3.07
8	1.77	5.33	13.43	5.18
16	1.22	7.74	8.83	7.88
32	0.75	12.56	5.86	11.86
64	0.45	20.93	2.85	24.42
128	0.28	33.24	1.68	41.35
256	0.21	43.46	0.80	87.00

Fig. 11. Runtimes for the discharge only and discharge with hillslope models. All times are measured in minutes.

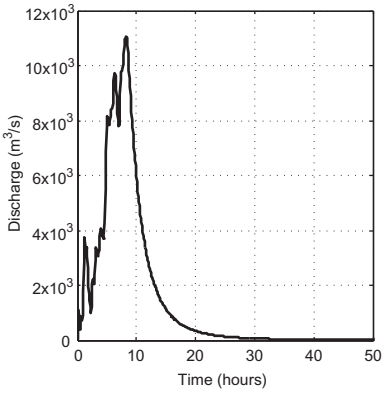


Fig. 12. Simulated discharge at the outlet of the Cedar River basin over two days using only channel discharge in the model. Initial channel discharge is taken as 30.0 m^3/s at each link.

Hydro-NEXRAD [23]. The data is provided as hourly rates in mm/h with a spatial resolution of 4 km. Therefore, the function $p(i,t)$ from (3b) can be viewed as a step function. For the measured data, for most links, the value of $p(i,t)$ tends to be 0 for most times t. Because step functions are discontinuous, we must not integrate across any jumps, as this will destroy all error controls.

We assume that there is initially no water present on the hillslope of each link ($s_p(i,0) = 0.0$ m). We also assume a small initial discharge of 1 m^3/s at each link ($q(i,0) = 1$ m^3/s). Rainfall measurements taken over the state of Iowa during the spring-summer-fall of 2008 are used for the function $p(i,t)$. The total integration time is 264 days. The hydrograph is given in Fig. 14 and the runtimes are given in Fig. 15. The speedups for low numbers of processes are very good, but deviate substantially from the ideal case when the

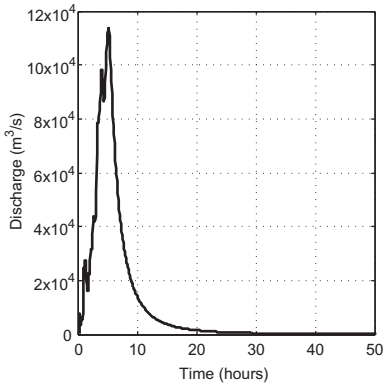


Fig. 13. Simulated discharge at the outlet of the Cedar River basin over two days with no rainfall forcing using the channel discharge with hillslope model. Initial channel discharge is taken as 30.0 m^3/s and an initial ponding depth is taken as 0.1 m at each link.

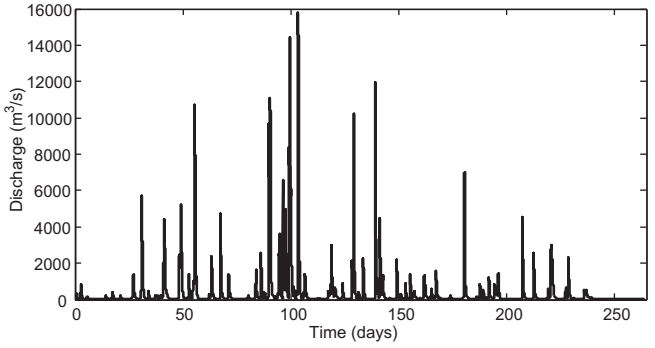
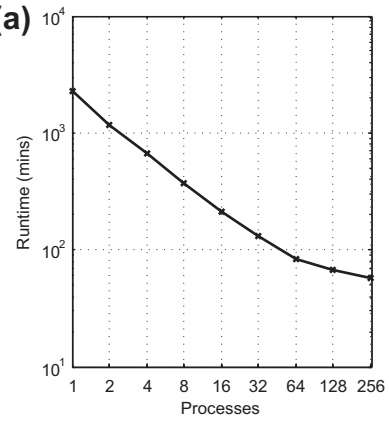


Fig. 14. Simulated discharge at the outlet of the Cedar River basin with radar-derived rainfall forcing using the channel discharge with hillslope model.



(b)

Processes	Runtime	Speedup
1	2,256	–
2	1,179	1.9
4	665	3.4
8	374	6.0
16	213	10.6
32	132	17.2
64	83	27.2
128	67	33.5
256	57	39.7

Fig. 15. Runtimes for the river basin model with precipitation using the channel discharge with hillslope model. Runtimes are given in minutes.

number of processes increases. Although this does suggest a need for improved load balancing, I/O considerations also affect the scalability. In this implementation, the rainfall data is read from disk by each process at roughly the same time. Thus, using more processes creates more reads from the rainfall data.

7. Conclusion

We have described a numerical solver for systems of ODEs with a directed tree structure. An important application of these systems appears in distributed hydrologic models of river basins. We have discussed the use of dense output Runge–Kutta methods to allow for an asynchronous integrator. A partition of the system is used to distribute the links amongst different processes, allowing for a parallel implementation. The communication between processes is performed asynchronously.

In addition to the experiments presented in Section 6, we have performed tests on the full river basin model presented in [22]. This model includes soil dynamics, resulting in a system of 4 ODEs ($n_i = 4$) at each link, and is analogous to the problem of Section 6.3.2. With 128 processes, our implementation can solve this system in less than 17 min on our parallel computing system.

Other authors have investigated similar problems for river basins. The work of Apostolopoulos and Georgakakos [24] solves a system of ODEs having a directed tree structure which also includes soil dynamics and evapotranspiration. Using their computer architecture, a maximum speedup of around 2.25 was achieved using 9 processors. Their work performed simulations on a 21 km^2 basin.

A second point of comparison is the results by Vivoni et al. [25] for a triangular element-based hydrological river basin model. Their problem differs from ours mathematically due to the consideration of subsurface flow among neighboring elements, which provides connectivity between elements not described by a directed tree structure. Further, a grid model is used to describe the flow of water through a hillslope. However, their problem shares some similarities because the connectivity between control volumes is still sparse. Their results for an 808 km^2 basin gave a best speedup of around 35 for 64 processes for their architecture. As in our work, a static partition of the basin is used. The partitioning algorithm in Vivoni et al. [25] groups geographical sub-watersheds, followed by a standard solution to the graph-partitioning problem on the smaller number of sub-watersheds. Their speedup results, along with the results presented in this paper, suggests that the problem of load balancing is not trivial for these problems and is worthy of further investigation.

Efficient numerical integrators such as the one demonstrated here bring closer to reality the goal of implementing fully distributed flood real-time forecasting systems supported by physics based hydrological models and high-quality/high-resolution rainfall products. In addition, it opens the door to more comprehensive Monte-Carlo based studies of parameter sensitivity in complex hydrological models, ensemble simulations for data assimilation studies and estimation of long term flood frequencies under realistic physical conditions.

Acknowledgements

We thank the reviewers for their helpful remarks. Research for this paper was supported by the National Science Foundation grant DMS-1025483.

References

[1] Mantilla R, Gupta VK. A GIS Framework to Investigate the Process Basis for Scaling Statistics on River Networks. Geosci Remote Sens Lett, IEEE 2005;2(4):404–8.
[2] Qu Y, Duffy CJ. A semidiscrete finite volume formulation for multiprocess watershed simulation. Water Resour Res 2007;43(8).
[3] Kampf SK, Burges SJ. A framework for classifying and comparing distributed hillslope and catchment hydrologic models. Water Resour Res 2007; 43:W05423.
[4] Mandapaka PV, Krajewski WF, Mantilla R, Gupta VK. Dissecting the effect of rainfall variability on the statistical structure of peak flows. Adv Water Resour 2009;32(10):1508–25. ISSN 0309-1708.
[5] Mantilla R, Gupta VK, Troutman BM. Scaling of peak flows with constant flow velocity in random self-similar networks. Nonlinear Process Geophys 2011;18(4):489–502.

[6] Cunha L, Krajewski W, Mantilla R. A framework for flood risk assessment under nonstationary conditions or in the absence of historical data. J Flood Risk Manage 2011;4(1):3–22. ISSN 1753-318X.
[7] Mantilla R. Physical basis of statistical scaling in peak flows and stream flow hydrographs for topologic and spatially embedded random self-similar channel networks. PhD thesis, University of Colorado, 2007.
[8] Menabde M, Sivapalan M. Linking space-time variability of river runoff and rainfall fields: a dynamic approach. Adv Water Resour 2001;24:1001–14.
[9] Hairer E, Nørsett S, Wanner G. Solving Ordinary Differential Equations I, Nonstiff Problems. New York: Springer; 2000.
[10] Shampine LF. Interpolation for Runge-Kutta Methods. SIAM J Numer Anal 1985;22:5.
[11] Enight WH, Jackson KR, Nørsett SP, Thomsen PG. Effective Solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants. Appl Math Comput 1988;27:313–35.
[12] Jeppson R. Open channel flow. CRC Press; 2011.
[13] Kleinberg J, Tardos E. Algorithm Design. Pearson Education Inc.; 2006.
[14] Kernighan B, Lin S. An Efficient Heuristic Procedure for Partitioning Graphs. Bell Syst Tech J 1970;49(2):291–307.
[15] Talbi E-G, Bessière P. A parallel genetic algorithm for the graph partitioning problem. In: Proceedings of the 5th international conference on supercomputing, ICS '91. ACM, New York, NY, USA, ISBN 0-89791-434-1, 1991. p. 312–20.
[16] Ray S, Jiang H. Improved algorithms for partitioning tree and linear task graphs on shared memory architecture. In: Distributed computing systems, 1994, Proceedings of the 14th International Conference on, 1994. p. 363–70.
[17] Lukes JA. Efficient algorithm for the partitioning of trees. IBM J Res Devel 1974;18(3):217–24. ISSN 0018-8646.
[18] Pacheco P. Parallel programming with MPI. Morgan Kaufmann Publishers, Inc.; 1997.
[19] Mantilla R, Navarro W, Ramirez JM. Analytic solution of the IUH for a river basin partitioned into hillslopes and channel links. Unpublished results.
[20] Gesch D, Oimoen M, Greenlee S, Nelson C, Steuck M, Tyler D. The national elevation dataset. Photogr Eng Remote Sens 2002;68(1):5–11.
[21] Gesch D. The national elevation dataset. In: Maune D, editor. Digital elevation model technologies and applications: the DEM users manual, 2nd ed. American Society for Photogrammetry and Remote Sensing, 2007. p. 99–118.
[22] Cunha LK, Mandapaka PV, Krajewski WF, Mantilla R. Impact of rainfall error structure on estimated flood magnitude across scales: an investigation based on a parsimonious distributed hydrological model. Water Resour Res, vol. 48, 2012.
[23] Krajewski WF, Kruger A, Smith JA, Lawrence R, Gunyon C, Goska R, Seo B-C, Domaszczynski P, Baeck ML, Ramamurthy MK, Weber J, Bradley AA, DelGreco SA, Steiner M. Towards better utilization of NEXRAD data in hydrology: an overview of Hydro-NEXRAD. J Hydroinform 2011;13(2):255–66.
[24] Apostolopoulos TK, Georgakakos KP. Parallel computation for streamflow prediction with distributed hydrologic models. J Hydrol 1997;197:1–24.
[25] Vivoni ER, Mascaro G, Mniszewski S, Fasel P, Springer EP, Ivanov VY, Bras RL. Real-world hydrologic assessment of a fully-distributed hydrological model in a parallel computing environment. J Hydrol 2011;409:483–96.