Applying microprocessor analysis methods to river network modelling

Frank Liu\textsuperscript{a}, Ben R. Hodges\textsuperscript{b,*}

\textsuperscript{a}IBM Research Austin
\textsuperscript{b}Department of Civil, Architectural, and Environmental Engineering
The University of Texas at Austin

Abstract

This paper describes the Simulation Program for River Networks (SPRINT) that is proposed as a tool for studying Continental River Dynamics (CRD), the solution of physics-based equations for large-scale river networks. Existing coupled hydrologic/hydraulic models have been unable to solve the full Saint-Venant equations for river networks larger than $O(10^3)$ elements, but continental scales require $10^6$ to $10^7$ elements. The new model solves the full nonlinear Saint-Venant equations for one-dimensional (1D) unsteady flow and stage height in river channel networks with non-uniform bathymetry, and is demonstrated to compute networks of $O(10^5)$ elements more than 330 times faster than real time on a desktop computer. The model incorporates ideas that were originally developed to address Very Large System Integration (VLSI) problems in microprocessor design, where solving large nonlinear computational problems is a common challenge. Computational speed is increased by applying Jacobian bypass techniques in a Newton-Raphson solution and smoothing the geometric depth-area and friction-area relationships where discontinuities otherwise slow convergence. Pre-processing of junction relationships is used to remove temporal nonlinearities where river tributaries meet. Model input/output are simplified and made readily accessible to other software through use of Application Programming Interface (API) standards and a “netlist” idea that was previously used to describe electric circuit topology. The model is tested on both simple and complex geometry through comparisons with the HEC-RAS model. A example simulation is conducted for $1.5 \times 10^4$ river km of the Guadalupe and San Antonio river network during a 14 day rain event.

NOTICE: This is the authors’ version of a work that has been accepted for publication in Environmental Modelling & Software. This Accepted Author Manuscript is being published online at the authors’ website in accordance with the journal’s policies at [http://www.elsevier.com/about/open-access/open-access-policies/article-posting-policy#accepted-author-manuscript](http://www.elsevier.com/about/open-access/open-access-policies/article-posting-policy#accepted-author-manuscript). Note that changes resulting from the publishing process, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document.

*Corresponding author.

Email addresses: frankliu@us.ibm.com (Frank Liu), hodges@utexas.edu (Ben R. Hodges)
1. Introduction

River networks are an integral part of hydrology, providing flow paths that move water through the landscape more rapidly than either overland or groundwater flows. In advancing hydrology, a number of challenges must be met to establish Continental River Dynamics (CRD) as a viable part of continental- to global-scale modelling (Hodges [2013]). Two associated technical challenges are (1) execution speed for tens of millions of river segments and (2) robust, portable standards for describing the river geometry and network connectivity. Rapid execution is necessary because river models demand smaller time steps (seconds to minutes) than hydrological models in order to capture both downstream movement of flood waves and backwater effects of flow constrictions (§2.1). Furthermore, improving computational speed allows sensitivity and uncertainty analysis using multiple model runs (e.g. Pappenberger et al., 2005; Hutton et al., 2012). New geometry standards are necessary because the complexities describing geometry in a high-order river network can be addressed in a variety of mutually incompatible schemes. Model representation of network connections and river reach geometry have been heretofore defined by modelers to meet the needs of individual hydraulic and hydrologic models. The resulting proliferation of methods creates incompatibilities between different models, which slows innovation (§2.5). The key motivation for this paper was the realization that these two challenges are similar to those experienced in the rapid development of the semiconductor industry in the 1970s.

This paper explores some ideas previously used in microprocessor design to rapidly simulate complex semiconductor circuit networks with hundreds of millions of elements. These methods are adapted and applied in a new code using Application Programming Interface (API) protocols to allow the model to be directly called by other software without requiring writing/reading files. The underlying numerical discretizations follow traditional approaches solving the one-dimensional (1D) Saint-Venant equations for river flow, but the numerical solution methods are adapted from Very Large System Integration (VLSI) design. This paper is an initial exploration of ideas available in microprocessor design, showing the relationships between disciplines that might inspire further advances. As this paper may be of interest to electrical or computer engineers beginning a collaboration in hydrologic/hydraulic modelling, the background presented (§2) is broader than would normally appear in this journal.

The Simulation PRogram for RIiver NeTworks (SPRINT) presented herein is an example of advancing hyperresolution hydrology through state-of-the-art computational methods – a need discussed in Wood et al. (2011). By using computational approaches and ideas developed in microprocessor design, we can create efficient, large-scale river network models that apply the full nonlinear Saint-Venant equations rather than volume-based routing or reduced-physics equations. The present version of the code provides three key technical advances to improve computational speed and make large-scale river network modelling practical: 1) implementation of
Jacobian bypass techniques, §3.4 for rapid solution without ad hoc linearization, 2) smoothing of cross-section auxiliary functions, §3.5 to ensure $C^1$ smoothness and prevent slow solution convergence, and 3) pre-processed junction relationships, §3.6 to ensure nonlinearity in time is modeled with linear relationships in space. A further advance is implementation of the VLSI “netlist” idea, §4.1 to standardize topographical representation and make changing between different river network models a simpler process.

This paper provides background and a short literature review in §2, explanation of the numerical methods in §3, details of the software implementation in §4, model tests in §5 and discussion in §6.

2. Background

2.1. The state of dynamic river modeling

Accurately capturing all the flow scales in a river channel requires solution of the 3D Navier-Stokes equations, which is readily accomplished for short river reaches of 1-10 km (e.g. Sandbach et al., 2012) but cannot, at present, be used to solve an entire continental network (Hodges, 2013). Cross-section integration of the 3D equations produces the 1D Saint-Venant equations, which model how the flow rate and surface elevation evolve along the length of a river channel (de Saint-Venant, 1871; Decoene et al., 2009). Because the along-channel velocity scale is typically an order of magnitude larger than both cross-channel and vertical velocities, the 1D model provides a reasonable representation of reach-scale mass and momentum balances (Cunge et al., 1980). There is a long history of river modeling using the 1D Saint-Venant equations, but they have never been used in a continental-scale river network model, i.e. $\geq O(10^6)$ km of river. Saint-Venant models in the past several decades have generally focused on improved numerical methods for reach-scale river dynamics, i.e. over $10^1$-$10^2$ km, rather than the $10^4$-$10^6$ km required for a high-order regional or continental river network. Numerical methods have included finite difference (e.g. Stepien, 1984; Chau and Lee, 1991), finite volume (e.g. Sanders, 2001; Finaud-Guyot et al., 2011), and finite-element (e.g. Sen and Garg, 1998) approaches. A thorough review of prior river models and their numerical methods is beyond the focus of this paper, but it can be said that to date, no method or modelling approach has been demonstrated to be superior for either short reach-scales or large river-network solutions. A more detailed discussion of the state-of-the-art and challenges is found in Hodges (2013).

2.2. Discretization methods

A numerical model can be categorized by its time and space discretization methods. The time discretization typically controls the model time step (for stability) whereas the space discretization typically affects the characteristics of model error. A wide variety of spatial discretization methods have been developed for numerical modeling, which would require an extensive literature review to discuss their comparative advantages and disadvantages. Temporal discretizations are somewhat simpler,
falling into three main categories: explicit, semi-implicit, and fully-implicit – depending on the treatment of advection and free-surface motion. Explicit and fully-implicit methods use the same temporal discretization for both advection and the free surface, whereas semi-implicit methods use explicit discretization for advection with implicit discretization for the free surface. Explicit methods (Sanders, 2001; Kesserwani et al., 2009) typically require small time steps for stability, whereas semi-implicit methods (Rosatti et al., 2011; Arico and Tucciarelli, 2007; Crnkovic et al., 2009) and fully-implicit methods (Sart et al., 2010; Islam et al., 2008) allow larger time steps. The difference between semi-implicit and fully-implicit models is that the former is only unconditionally stable for subcritical flows; thus, when advective speed exceeds the free-surface gravity-wave celerity (supercritical conditions) the stable time step for semi-implicit schemes is limited by the explicit advective discretization. Although fully-implicit schemes can be unconditionally stable for both subcritical and super-critical conditions, it has been recognized that transcritical conditions (the transition from sub- to super-critical) can cause stability issues, depending on the spatial discretization (§3.2).

Explicit and semi-implicit numerical methods are generally preferred when numerical stiffness or the matrix size causes difficulty in solving a nonlinear system. However, implicit techniques are preferred for numerical stability, and were deemed a requirement for practical river models in the seminal work of Cunge et al. (1980). In the past, fully-nonlinear implicit methods were often considered too expensive because iterative techniques (such as Newton’s method) are required to solve the discrete nonlinear system. In the traditional view, each Newton iteration in a single time step requires construction and inversion of a Jacobian matrix. Naïve matrix factorization has the complexity of $O(M^3)$ where $M$ is the matrix size, so implementation of a simple Newton method is clearly impractical for large systems. Thus, it is not surprising that implicit discretization of the full Saint-Venant equations has been viewed as “troublesome” (Brunner, 2010). Slow computation and poor convergence of early nonlinear implicit schemes led to widespread use of linearization techniques for Saint-Venant models. Zhu et al. (2011) and Islam et al. (2008) have recently solved the nonlinear Saint-Venant equations in fully implicit schemes for short river reaches, but thus far no one has taken advantage of recent computational efficiency increases for solution of large nonlinear matrices that allow larger problems to be solved (§2.3). The fully-implicit approach used herein allows us to apply the numerical techniques pioneered in VLSI for accelerated nonlinear solutions (e.g. Acar et al., 2002), which make it possible to consider river network modeling posed as a large nonlinear matrix problem.

2.3. Efficiency of modern linear solvers

Numerical linear algebra packages have become an integral part of scientific computing and are generally more efficient than custom-coded solvers. Such packages typically solve a linear system in the form of:

$$A \cdot x = b$$

(1)
or equivalently:

\[ x = A^{-1}b \]  

(2)

where \( A \in \mathbb{R}^{n \times n} \) is a nonsingular constant matrix (either real or complex), \( b \in \mathbb{R}^n \) is a given real (or complex) vector (often called right-hand-side or RHS) and \( x \in \mathbb{R}^n \) is the vector of unknowns to be compute.

Computationally, the problem is usually addressed by performing an LU decomposition by factorizing the \( A \) matrix into the product of two matrices of the same size:

\[ A = L \cdot U \]  

(3)

where \( L \) is strictly lower triangular (all entries above main diagonal are zero) and matrix \( U \) is strictly upper triangular (all entries below main diagonal are zero). By doing so, the original linear algebra problem can be converted into two sub-problems:

\[ L \cdot y = b \]  

(4)

and

\[ U \cdot x = y \]  

(5)

Since matrix \( L \) is strictly lower triangular, one can quickly compute the intermediate unknown vector \( y \) by processing from the first row and substituting the previous solutions into the subsequent rows. Once \( y \) is known, repeating the process in eq. (5) using reverse row order will generate the solution to the original problem \( x \).

This approach is often referred to as LU-factorization followed by forward-backward substitution. Gaussian elimination is the classic method to solve LU factorization problems, where \( L \) and \( U \) are equivalent to the intermediate results of Gaussian elimination. Traditional Gaussian elimination of dense matrices of size \( M \) is computationally expensive, scaling as \( O(M^3) \). However, when matrix \( A \) is sparse it is generally accepted that the runtime for LU factorization scales as \( O(M^{1.1\sim 1.3}) \), which is only slightly super-linear. Unfortunately, implementing an efficient sparse LU factorization is not a trivial undertaking as runtime may be substantially affected by the matrix topology, such as row ordering.

Over the past two decades, sparse LU factorization has been an active research area in numerical analysis (Amestoy et al., 2001; Davis, 2004; Demmel et al., 1999; Duff, 2004). Sparse LU factorization routines are highly sophisticated (e.g. UMFPACK, which has more than 34,000 lines of C code), resulting in rapid factorization of matrices from a few hundred thousand to more than a million elements. Indeed, the present solvers are so fast that LU factorization for sparse matrix size of \( 10^6 \) is considered a solved problem (Davis, 2004). Table I provides the runtime using UMFPACK for various sparse matrices vs. the CPU time to perform LU factorization and the “re-solve” time. The latter refers to the ability to rapidly solve multiple \( Ax = b \) problems for different values of \( b \) once the LU factorization has been completed. This capability occurs when LU factorization occurs “in-core”, e.g. the values of both \( L \) and \( U \) matrices are stored in the same memory of the original \( A \) matrix. When a
new RHS vector \( \mathbf{b} \) is provided, the corresponding solution \( \mathbf{x} \) can be computed in a fraction of the cost since the only calculations involved are forward-backward substitutions rather than complete LU factorization. The re-solve time is more than an order of magnitude faster than the LU factorization time, which is a useful property used in SPRINT for accelerated nonlinear solutions of the Saint-Venant equations (§3.4).

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Factorization Time (sec)</th>
<th>Re-Solve Time (sec)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>0.022</td>
<td>0.000</td>
<td>-</td>
</tr>
<tr>
<td>20,000</td>
<td>0.038</td>
<td>0.001</td>
<td>38:1</td>
</tr>
<tr>
<td>50,000</td>
<td>0.090</td>
<td>0.001</td>
<td>90:1</td>
</tr>
<tr>
<td>100,000</td>
<td>0.180</td>
<td>0.003</td>
<td>60:1</td>
</tr>
<tr>
<td>200,000</td>
<td>0.355</td>
<td>0.006</td>
<td>59:1</td>
</tr>
<tr>
<td>500,000</td>
<td>0.890</td>
<td>0.015</td>
<td>59:1</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1.766</td>
<td>0.031</td>
<td>57:1</td>
</tr>
</tbody>
</table>

Table 1: Measured runtime of various sparse matrix sizes and the corresponding factorization and solve time. All matrices are asymmetric with stencil of 5. Hardware: Intel Core i7-870; Memory 8G; OS: Linux kernel 2.6.32 x86_64; Solver: UMFPACK 5.5. The re-solve time for the first entry is too small to be meaningful.

2.4. Network modeling for microprocessors

In the semiconductor industry, exponential growth in microprocessor transistor counts created a design problem known as Very Large Scale Integration (VLSI); chip designers needed to rapidly verify complex designs before committing to lengthy and expensive manufacturing processes. Circuit network models were developed to fill this need for rapid virtual prototyping. These models are collectively known as Electronic Design Automation (EDA) tools, and include the open-source Simulation Program with Integrated Circuit Emphasis (SPICE) described in Nagel and Rohrer [1971] and Nagel and Pederson [1973]. We presented an extensive background on SPICE in Liu and Hodges [2012], which is summarized herein. SPICE and similar EDA tools provide simulations of circuit behavior in complex networks by solving nonlinear differential algebraic equations (e.g. Hachtel et al. 1971; Ho et al. 1975). The SPICE equations fall into three categories: Kirchoff’s Current Law (KCL), Kirchoff’s Voltage Law (KVL) and Branch Constituent Relations (BCR). Both KCL and KVL can be derived from Maxwell’s equations, the general mathematical description of electrodynamics. KCL requires continuity of current at any circuit node, with an effect similar to conservation of mass for fluid flow. KVL is a statement of energy conservation, similar to momentum conservation used for the Saint-Venant and Navier-Stokes equations for fluids. The third circuit category (BCR) provides the complex relationships between currents and voltage in semiconductor components, which range from linear Ohm’s law (relating voltage, current and resistance in a simple conductor) to complex nonlinear equations for transistors. A BCR is functionally equivalent to the empirical fluid flow resistance models for friction and
turbulence as well as for bridge abutments, culverts and weirs. These ideas provide a
strong analogy between hydrodynamic and electrodynamic modeling; both systems
have conservation laws and empirical relationships that are applied to reduce com-
plex physics to computationally-friendly equations.

The challenge for VLSI software is in translating the complicated circuit net-
work of a prototype semiconductor design into a mathematical description that can
be readily solved for either steady-state or transient analyses. Microprocessor cir-
cuits can be topologically complicated due to the large number of transistors and
interconnections, so SPICE uses a “netlist” applying rules and keywords to define
connectivity ([2.5]). In the early years, netlists were simply annotations of the design
schematics in a simple text format. As VLSI circuits evolved into more complex
systems, increasingly sophisticated EDA programs were written to parse designs and
generate models, but the text-based netlist was retained for data exchange. The
netlist format is the fundamental abstraction layer to describe the design topology
and relationships within a microprocessor, with the key benefit that different software
applications for simulation and analysis (often from competing software vendors) can
access the circuit topology using a common syntax. Such exchanges have been sim-
plified through an Open Library Architecture (OLA) covering broad aspects of VLSI
design ([IEEE Computer Society Design Automation Standards Committee 2010]).

Besides separating the model data from the simulation engine itself, a netlist also
makes it possible to realize hierarchical abstraction. In a netlist, a commonly used
portion of the circuit can be defined as a “subcircuit,” which can be instantiated
and parameterized multiple times without duplicating definitions. As an example,
the logical AND function can be implemented by defining a subcircuit with six tran-
sistors, so that further AND subcircuits do not require re-defining the transistors or
their connections.

Because interesting microprocessor behavior is unsteady, nonlinear, and covers
massive number of transistors, the circuit solutions typically use a modified Newton-
Raphson’s algorithm ([3.3]) applied in a numerical integration method (e.g. [Gear
1971]). The ideas behind SPICE have already been applied in other areas, notably
the micro-electromechanical systems simulator SUGAR ([Clark et al. 1998] and Bio-
SPICE for systems biology ([SRI International 2012]). In the SPRINT model de-
scribed herein, some of the ideas developed in SPICE are applied to the problem of
dynamic river simulation.

2.5. River and microprocessor network synthesis

A river network can be represented by its reaches, where a “reach” is defined as
a section of river between tributary confluences (junctions) or significant hydraulic
features (e.g. waterfall, dam, weir). Reaches are further subdivided into elements
(or segments) connecting nodes. A key task for any CRD model is translating the
topology and geometry of the river into a set of reaches, junctions, elements, and
nodes for numerical simulation – problems similar to those encountered in VLSI
Although a continental network of $10^6$ km (such as the Mississippi basin) seems massive, a state-of-the-art microprocessor network can have billions of elements. Furthermore, microprocessor circuits do not follow the intuitive branching schemes of river networks, so it is reasonable to state the size of river networks should not be a barrier to simulation.

Today, the practical problem for river network modeling is not defining the topology (as was the case at the start of VLSI), but instead the fact that every river model has a different approach to defining network topology, often using proprietary graphical user interfaces and undocumented binary files. Despite the development of OpenMI (e.g. Castronova et al., 2013) and CSDMS (Overeem et al., 2013) as API’s for linking models, coupling a hydrological model to different river network models is still a time-consuming data translation problem, which creates barriers to adopting new models and slows the rate at which hydrology can build on new ideas and methods. Adoption of standards for network descriptions similar to those used in VLSI could significantly simplify the problems associated with model coupling.

3. Methods

3.1. Governing equations

The Saint-Venant equations can be solved in a number of forms using different dependent variables. Common choices are $v-h$, $Q-h$, and $Q-A$ where $v$ is the velocity, $h$ is the water height above the channel bottom (equivalently, the water depth), $Q$ is the volumetric flow rate, and $A$ is the cross-sectional area. The present work uses $Q-A$ variables with incompressible continuity represented as

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} + q_l = 0 \quad (6)$$

where $q_l$ is the net lateral inflow rate per unit length of the river from the landscape, groundwater, and precipitation. Note that lateral outflows are simply negative lateral inflows. The momentum equation applied herein is the “dynamic” form used in many models (e.g. Shang et al., 2011, Paiva et al., 2011, Arico and Tucciarelli, 2007):

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} \left( \beta \frac{Q^2}{A} \right) + gA \frac{\partial h}{\partial x} = gA(S_0 - S_f) \quad (7)$$

where $\beta$ is the momentum coefficient (typically taken as unity in the absence of comprehensive field data), $g$ is gravity, $S_0$ is the bottom slope, $S_f$ is the friction slope, and $h = h(A)$ is an auxiliary function for depth as a function of cross-sectional area. Eq. (7) is sometimes known as the “non-conservative” form of momentum, in contrast to the “conservative” form used in many Saint-Venant models of short river reaches (e.g. Rosatti et al., 2011, Kesserwani et al., 2009). The different forms are derived and discussed in Cunge et al. (1980). SPRINT has been implemented with both equation forms; however for simplicity in exposition, only the non-conservative form is used in this research. This form is common in large-scale hydrological models.
(e.g. Paiva et al., 2013; Zhu et al., 2011) as well as hydraulic models (e.g. Trigg et al., 2009).

An issue not previously discussed in the Saint-Venant literature, but of importance in CRD modeling, is whether to use an auxiliary function for the cross-section averaged, free-surface elevation, \( \eta = \eta(A) \) or a depth-area relation, \( h = h(A) \). The former can be applied in eq. (7) based on \( \partial \eta / \partial x = \partial h / \partial x - S_0 \). The \( \eta \) approach is used several Saint-Venant models, (e.g. Brunner, 2010; Rosatti et al., 2011; Zhu et al., 2011). For SPRINT the \( \eta \) formulation was considered, but rejected. In any large river network, \( \eta \) increases monotonically upstream such that far upstream values may be several orders of magnitude greater than downstream values. Such magnitude variations can lead to undesirable numerical stiffness that slows solution convergence. Furthermore, the value of \( \eta \) in a river reach does not have an \textit{a priori} relationship to the relative importance of that reach in the network; i.e. a large \( \eta \) may occur in a small mountain tributary close to the river mouth with small \( \{Q, A\} \), or upstream in a segment of the main stem river with large \( \{Q, A\} \). Thus, the \( \eta \) form of the Saint-Venant equations is unlikely to be successful in CRD problems covering a wide range of topographical elevations. In contrast, values of \( h \) throughout a continental river network vary within a relatively small range so that numerical stiffness will only be an issue for very small values of \( h \), which are generally associated with smaller tributary reaches with small \( \{Q, A\} \). It follows that numerical errors associated with insufficient convergence of a stiff problem will be isolated to minor reaches when using an \( h \) formulation.

In eq. (7), the friction slope in SPRINT is modeled using the Chezy-Manning equation (also known under multiple pseudonyms using various combinations of Gauckler, Manning and Strickler), which can be represented as

\[
S_f = \tilde{n}^2 \frac{Q^2}{A^2 R^{4/3}}
\]  

(8)

in which the units are SI standards of \( \{m, s\} \), with Manning’s \( \tilde{n} \) used as a dimensional friction coefficient, and \( R \) as the traditional “hydraulic radius” defined as:

\[
R \equiv \frac{A}{P}
\]  

(9)

where \( P = P(A) \) is the perimeter of the wetted area as a function of \( A \). Computation herein is simplified by defining a dimensional “equivalent friction geometry,” denoted as \( F \):

\[
F(A) \equiv \frac{P^{4/3}}{A^{7/4}}
\]  

(10)

which is a more convenient parsing of \( P \) and \( A \) than conventionally used in \( R \), but does not change the Chezy-Manning relationship of eq. (8). The friction contribution in eq. (7) can be written as:

\[
g A S_f = g \tilde{n}^2 Q^2 F
\]  

(11)
Thus, friction’s contribution to momentum is reduced to a nonlinear function of
an empirical parameter (\(\tilde{n}\)), the flow rate \((Q)\), and a function of the wetted cross-
sectional shape \((F)\).

3.2. Discretization

As noted in §2.5, a river network is divided into reaches that are subdivided into
segments connected at nodes. SPRINT uses a numbering convention such that the
\(i^{th}\) reach is subdivided into \(N_i\) segments and \(N_i + 1\) nodes such that \(j^{th}\) segment of
reach \(i\) has upstream node \(j\) and downstream node \(j + 1\), where \(j = 1\) is always the
furthest upstream node of a reach and \(\text{max}(j)\) is always the furthest downstream
node. Clearly, the convention could easily have been chosen from downstream to
upstream, but there is neither a standard nor a compelling argument for a “best”
convention, which contributes to data exchange issues discussed in §2.5.

As our goal is examining how VLSI techniques can be used to improve river mod-
elling rather than developing and testing a new numerical scheme, SPRINT uses
the conventional non-staggered, finite-difference, implicit scheme from Preissmann
[1961] to discretize the governing equations at computational nodes. This method
has been used and studied extensively in the river modelling literature (e.g. Castel-
larin et al., 2009; Freitag and Morton, 2007; Sart et al., 2010; Zhu et al., 2011; Islam
et al., 2005). For the Preissmann scheme the function \(f\) is modeled as

\[
f(x,t) \approx \frac{1}{2}(f_{j+1}^{n+1} + f_j^{n+1})
\]

\[
\frac{\partial}{\partial x} f(x,t) \approx \frac{1}{\Delta x} (f_{j+1}^{n+1} - f_j^{n+1})
\]

\[
\frac{\partial}{\partial t} f(x,t) \approx \frac{1}{2\Delta t} (f_{j+1}^{n+1} - f_j^{n+1} + f_j^{n+1} - f_j^n)
\]

where subscripts represent the node location and superscripts represent the time
step. Using this 4-node Preissman stencil, continuity eq. (6) is discretized as

\[
f_1 = [A_{j+1}^{n+1} + A_j^{n+1}] + \frac{2\Delta t}{\Delta x} [Q_{j+1}^{n+1} - Q_j^{n+1}] - [A_j^{n+1} + A_j^n] - \frac{\Delta t}{\Delta x} (Q_l)_{j+1/2}^{n+1} = 0
\]

where \((Q_l)_{j+1/2}\) is the total lateral inflow rate from the landscape between the \(j\)
node and \(j + 1\) node, which is positive for an inflow and negative for an outflow.
To maintain simple and consistent boundary conditions for communication with
hydrological models, the present version of the SPRINT input uses lateral inflows
defined at nodes rather than elements so that

\[
(Q_l)_{j+1/2} = \frac{1}{2} [(Q_l)_j + (Q_l)_{j+1}]
\]
Momentum, eq. (7), is discretized as

\[
f_2 = \left[ Q_{j+1}^{n+1} + Q_j^{n+1} \right] - \left[ Q_j^n + Q_j^n \right] + \frac{2\beta \Delta t}{\Delta x} \left[ \frac{(Q_{j+1}^{n+1})^2}{A_{j+1}^{n+1}} - \frac{(Q_j^{n+1})^2}{A_j^{n+1}} \right] \\
+ g \frac{\Delta t}{\Delta x} \left( A_{j+1}^{n+1} + A_j^{n+1} \right) \left( h_{j+1}^{n+1} - h_j^{n+1} \right) - \Delta t g \left[ A_{j+1}^{n+1} + A_j^{n+1} \right] S_0(j+1/2) \\
+ \Delta t g \bar{n}_{j+1/2}^2 \left[ (Q_j^{n+1})^2 F_j^{n+1} + (Q_j^{n+1})^2 F_j^{n+1} \right] = 0 \tag{17}
\]

where \( S_0(j+1/2) \) and \( \bar{n}_{j+1/2} \) are the river bathymetric slope and Manning’s \( \bar{n} \) between nodes \( j \) and \( j+1 \), while \( h = h(A) \) is the auxiliary function relating water depth to cross-sectional area and \( F = F(A) \) is given by eq. (10).

Because the stability, error and boundary condition characteristics of the implicit Preissmann scheme have been documented throughout its 50-year history (more than 100 papers just from 1987 to 2012), a full analysis of method is not provided herein. However, a few characteristics should be noted with regard to boundary conditions, stability, and discontinuous solutions. Although the Preissmann discretization is unconditionally stable for both supercritical and subcritical flows, it is known to exhibit local instabilities where the flow transitions between subcritical and supercritical at boundaries (Venutelli, 2002; Meselhe and Holly, 1997). In a network model, obtaining consistent subcritical conditions in the far upstream and far downstream boundaries is not difficult; however, junctions between tributaries are internal boundaries to the 1D Saint-Venant solutions, so transcritical junctions can cause problems. The present implementation of SPRINT does not attempt to handle transcritical junctions; thus, the model fails to converge where both supercritical and subcritical flows are present at a single junction. Although such junctions are relatively rare, they can occur in the real world and also as a numerical artifact in an unsteady flow. Numerical artifacts can occur because CRD model node spacing \( \geq O(10^2) \) m is required to keep the total number of computational cells \( \leq O(10^7) \) in a large basin. This resolution is coarse compared to transcritical flow physics occurring over short distances, hence such features may appear as artifacts rather than models of the actual physics. It is planned that future versions of SPRINT will handle transcritical junctions using established techniques (e.g. Freitag and Morton, 2007; Sart et al., 2010).

Since the Saint-Venant equations admit discontinuous solutions, models are often tested using a hydraulic jump and/or dam break flow. Both tests require \( \leq O(1) \) m resolution to represent typical channel scales (e.g. Goutal and Sainte-Marie, 2011; Crnkovic et al., 2009; Gottardi and Venutelli, 2003), which cannot be captured in a large-scale CRD model. From a modeling perspective, the fine-scale accuracy of the model is less important than simply whether or not model remains stable through the transitions. Prior work (Freitag and Morton, 2007; Sart et al., 2010) and tests by the present authors conducted with SPRINT (not shown) have adequately demonstrated the implicit Preissmann scheme is stable through unsteady discontinuous solutions (with the exception noted above regarding transcritical propagation through a junction).
### 3.3. Newton-Raphson solution method

Discrete continuity and momentum, eqs. (15) and (17), provide four variables coupling two adjacent nodes: \( Q_j, Q_{j+1}, A_j \) and \( A_{j+1} \). Symbolically, a system of nonlinear equations for the nodes on a reach is:

\[
\mathbf{f}(\mathbf{X}) = \mathbf{0}
\]  \hspace{1cm} (18)

This system can be solved by a damped Newton-Raphson iteration, represented as:

\[
\mathbf{X}_{k+1} = \mathbf{X}_k - \gamma \left[ \frac{\partial \mathbf{f}}{\partial \mathbf{X}} \right]^{-1} \mathbf{f}(\mathbf{X}_k)
\]  \hspace{1cm} (19)

where \( \gamma \) is the damping factor and \( k \) represents the Newton iteration level. The difficult part of this system is the matrix inversion, which is equivalent to eq. (2), and can be solved by the standard approaches discussed in §2.3. The Jacobian, \( \partial \mathbf{f} / \partial \mathbf{X} \), requires the derivative of the functions with respect to all four terms in the discretization stencil. Fortunately, most terms can be calculated analytically to reduce computational effort. The derivatives of continuity eq. (15) are, with respect to \( A^{n+1} \)

\[
\frac{\partial f_1}{\partial A_j} = 1.0 \hspace{1cm} (20)
\]

\[
\frac{\partial f_1}{\partial A_{j+1}} = 1.0 \hspace{1cm} (21)
\]

and with respect to \( Q^{n+1} \)

\[
\frac{\partial f_1}{\partial Q_j} = -\frac{2\Delta t}{\Delta x} \hspace{1cm} (22)
\]

\[
\frac{\partial f_1}{\partial Q_{j+1}} = \frac{2\Delta t}{\Delta x} \hspace{1cm} (23)
\]

The derivatives of momentum eq. (17) are more complicated, with respect to \( A^{n+1} \)

\[
\frac{\partial f_2}{\partial A_j} = \frac{2\beta \Delta t}{\Delta x} \left[ \frac{Q_j}{A_j} \right]^2 - g \Delta t \left[ A_j \frac{\partial h_{j+1}}{\partial A_j} + h_{j+1} - A_j \frac{\partial h_j}{\partial A_j} - h_j \right] - \Delta t g S_0 (j+1/2) + \Delta t g n^2 [Q_j]^2 \frac{\partial F_j}{\partial A_j} \hspace{1cm} (24)
\]

\[
\frac{\partial f_2}{\partial A_{j+1}} = -\frac{2\beta \Delta t}{\Delta x} \left[ \frac{Q_{j+1}}{A_{j+1}} \right]^2 - g \Delta t \left[ A_{j+1} \frac{\partial h_{j+1}}{\partial A_{j+1}} + h_{j+1} - A_{j+1} \frac{\partial h_j}{\partial A_{j+1}} - h_j \right] - \Delta t g S_0 + \Delta t g n^2 [Q_{j+1}]^2 \frac{\partial F_{j+1}}{\partial A_{j+1}} \hspace{1cm} (25)
\]
and with respect to \( Q^{n+1} \)

\[
\frac{\partial f_2}{\partial Q_j} = 1 - 4\beta \frac{\Delta t}{\Delta x} Q_j A_j + 2\Delta t g n^2 Q_j F_j
\] (26)

\[
\frac{\partial f_2}{\partial Q_j} = 1 + 4\beta \frac{\Delta t}{\Delta x} Q_{j+1} A_{j+1} + 2\Delta t g n^2 Q_{j+1} F_{j+1}
\] (27)

With known derivatives, the Jacobian matrix can be constructed either by inspection or by adjacency matrix multiplication. If the unknowns are ordered sequentially, the sparsity structure of the Jacobian will be similar to Fig. 1. Note that there are only four nonzero entries in each row, therefore the Jacobian matrix may be large (encompassing a continental-scale network of \(10^6 - 10^7\) elements) but will remain sparse. If one follows the Newton-Raphson method of eq. (19), then the Jacobian matrix must be inverted within each Newton loop. However, naïve inversion is both undesirable and unnecessary (see §2.3).

![Sparsity structure of the Jacobian matrix when the unknowns are sequentially ordered. The pattern will be different if the sequential ordering is not observed, but sparsity will be similar.](image)

**Figure 1:** Sparsity structure of the Jacobian matrix when the unknowns are sequentially ordered. The pattern will be different if the sequential ordering is not observed, but sparsity will be similar.

### 3.4. Acceleration of Newton-Raphson method

To solve the nonlinear function in eq. (18), Newton-Raphson method shown in eq. (19) can be applied. The classic Newton-Raphson method can be considered an iterative search algorithm starting from an initial condition \( \mathbf{X}_0 \), with a generic flow chart as shown in Algorithm 1.
Algorithm 1 Classic Newton-Raphson Method

1: \textbf{procedure} NR(t, m) \hspace{1em} \triangleright t: tolerance, m: max number iteration, k: iteration
2: \quad k \leftarrow 1
3: \quad \textbf{repeat}
4: \quad \hspace{1em} \text{Evaluate } f(X_k)
5: \quad \hspace{1em} \textbf{if} \|f(X_k)\| < t \textbf{ then}
6: \quad \hspace{1em} \textbf{return} converged \hspace{1em} \triangleright \text{Success}
7: \quad \hspace{1em} \textbf{end if}
8: \quad \hspace{1em} \text{Evaluate derivatives and populate Jacobian matrix } J_k
9: \quad \hspace{1em} \text{Factorize Jacobian matrix } J_k
10: \quad \hspace{1em} \text{Solve for corrections } \Delta X \leftarrow J_k^{-1} \cdot f(X_k)
11: \quad \hspace{1em} \text{Update } X_{k+1} \leftarrow X_k - \gamma \cdot \Delta X
12: \quad \hspace{1em} k \leftarrow k + 1
13: \quad \textbf{until } k > m
14: \quad \textbf{return} nonconvergent \hspace{1em} \triangleright \text{Nonconvergent within max iteration number}
15: \textbf{end procedure}

A different interpretation of the above algorithm is that each Newton iteration, \(k\), makes a small correction in the direction provided by the inverse of the Jacobian matrix \(\frac{\partial f}{\partial X}|_k\), with the correction magnitude determined by the value of \(f(X_k)\). When \(\|f(X_k)\| < t\), where \(t\) is a pre-defined tolerance (10\(^{-7}\) by default in SPRINT), no correction will be added and convergence is reached. From this point of view, the evaluation of the nonlinear \(f(X_k)\) at each iteration ensures the “correctness” of the final solution, while the Jacobian only provides the direction of the search. Theoretically, the Newton-Raphson method for \(f\) with \(C^2\) regularity has quadratic convergence if the Jacobian is updated at each iteration [Deuflhard, 2005].

However, the Newton-Raphson method does not require the exact Jacobian in order to converge to the correct solution. If an \textit{inexact} Jacobian is used, the convergence rate reduces from quadratic to linear, but the solution will still converge for \(C^1\) smooth problems. Using an inexact Jacobian appears to make the solution method more costly (by increasing the number of Newton iterations), but can be used to reduce the overall solution time if each iteration is significantly less expensive; that is, computing the inexact Jacobian and converging with more iterations may be faster than computing the exact Jacobian and converging with fewer iterations. This idea is exploited in a “Jacobian bypass” method by evaluating and factorizing (§2.3) the Jacobian matrix only once per time step. The factorized \(L\) and \(U\) matrices can therefore be re-used so that the corrections are made in the re-solve time rather than the factorization time (e.g. Table 1). A simplified version of this method is shown in Algorithm 2. Note that the Jacobian is evaluated and factorized only at the very first iteration, or when convergence issues are encountered. The latter idea is represented as a condition \(k \geq c\) in Algorithm 2, although more sophisticated heuristics are used in SPRINT to determine when Jacobian factorization is needed.
Algorithm 2 Accelerated Newton-Raphson Method

1: procedure NR_acc($t, m, c$) \hspace{1em} \triangleright t$: tolerance, $m$: max iter., $c$: conditional factorize
2: \hspace{1em} $k \leftarrow 1$
3: \hspace{1em} $flag \leftarrow 1$
4: repeat
5: \hspace{2em} Evaluate $f(X)$
6: \hspace{2em} if $\|f(X_k)\| < t$ then
7: \hspace{3em} return converged \hspace{1em} \triangleright\ Success
8: \hspace{2em} end if
9: \hspace{2em} if $flag$ then \hspace{1em} \triangleright\ Only factorize Jacobian when necessary
10: \hspace{3em} Evaluate derivatives and populate Jacobian matrix $J$
11: \hspace{3em} Factorize Jacobian matrix $J$
12: \hspace{3em} $flag \leftarrow 0$
13: \hspace{2em} end if
14: \hspace{2em} Solve for corrections $\Delta X \leftarrow J^{-1} \cdot f(X_k)$
15: \hspace{2em} Update $X_{k+1} \leftarrow X_k - \gamma \cdot \Delta X$
16: \hspace{2em} $k \leftarrow k + 1$
17: \hspace{2em} if $k \geq c$ then \hspace{1em} \triangleright\ Trigger Jacobian factorization
18: \hspace{3em} $flag \leftarrow 1$
19: \hspace{2em} end if
20: until $k > m$
21: return nonconvergent \hspace{1em} \triangleright\ Nonconvergent within max iteration number
22: end procedure

In SPRINT, failure to achieve convergence within the user-specified maximum number of iterations ($m$ in Algorithm 2), causes reduction in the time step and re-computation of the Jacobian. When convergence cannot be achieved with a user-defined minimum time step using the Jacobian bypass, SPRINT will apply the full Newton’s method (Algorithm 1) with Jacobian factorization at each iteration. SPRINT will declare the convergence has failed and stop execution if the minimal time step and the full Newton’s method is unable to reduce the residual below the tolerance. Future versions of SPRINT will be configured with an option for continued execution of time marching despite convergence failure.

Although the inexact Jacobian is used to accelerate the solution algorithm, it is important to emphasize that the converged result will still be the exact nonlinear dynamic solution of the full Saint-Venant equations. In contrast, methods that \textit{a priori} linearize the equations for faster solution (e.g.\cite{Brunner2010}) will converge to the linearized approximation of the Saint-Venant equations.

3.5. Smoothness of auxiliary functions

The Jacobians for the Newton-Raphson method in eqs. (24)–(27) require evaluation of derivatives in the form of $\partial h/\partial A$ and $\partial F/\partial A$. For simple rectangular
and trapezoidal channel cross-sections, analytical solutions can be easily derived and provide rapid Jacobian evaluation. However, analytical Jacobians do not exist for general river cross-sections defined by surveyed sets of $y:z$ data pairs. Therefore, discrete implementations of the auxiliary functions $F(A)$ and $h(A)$ and their derivatives have to be calculated from $y:z$ data pairs at each computational node. The approach used to compute these auxiliary functions is critical because the Newton-Raphson method requires the Jacobians to be computed from $C^1$ functions – e.g. both the function and its derivative must be continuous. The $C^1$ requirement is not generally discussed in river modeling literature for implicit models, despite this being a key factor determining whether a model is convergent. If the $C^1$ requirement is not satisfied, the Newton-Raphson method may not converge. The solution behavior in absence of $C^1$ smoothness depends upon the level of discontinuity and is not easily predicted, therefore ensuring the discrete equations are $C^1$ is a necessary step for a robust and dependable model.

As illustrated in a typical cross-section in Fig. 2(a), surveyed $y:z$ data are rarely $C^1$. Fortunately, the extreme discontinuities in cross-section shape are moderated in building a discrete $h:A$ function because $A$ monotonically increases with $h$. However in the example shown in Fig. 2(b), what appears to be a relatively smooth $h:A$ leads to $\partial h/\partial A$ with sharp discontinuities and strong local non-monotonic behavior. Note that this behavior occurs below bankfull depth as well as in the broader floodplain. Non-monotonic behaviors at larger scales for $\partial h/\partial A$ do not cause Jacobian problems, e.g. the peak at 4 m$^2$ in Fig. 2(b) bracketed by lower values at 1 m$^2$ and 10 m$^2$; it is the sharp discontinuities and development of local minima or maxima for small changes in $A$ that can lead to solution divergence or oscillation. The $F:A$ data in the Fig. 2(c) example is smoother than the $h:A$ data in 2(b), with $\partial F/\partial A$ showing a generally monotonic increase, but similar to Fig. 2(b) the small local perturbations of $\partial F/\partial A$ that are evident in the data, e.g. at $A \sim 5$m$^2$, can still cause convergence problems.

This relatively simple issue profoundly affects the practical usability of a river model. If the $C^1$ requirement is not met, the solution may fail to converge so that accurate time integration is impossible. Jacobians with lesser discontinuities may lead to converged solutions with spatial and/or temporal oscillations. Theoretically, a user could manually adjust cross-section data where solution problems occur, but such failure-based fixes are impractical for large-scale river network modeling. $C^1$ smoothness in $h:A$ and $F:A$ could be obtained by smoothing the $y:z$ river cross-section, however a simple fix is readily obtained by ensuring smoothness in the $h:A$ and $F:A$ relationships, which are already smoother than the cross-section geometry.

For general cross-sections given in $y:z$ pairs, tabulations for the related auxiliary functions for $F:A$ and $h:A$ can be computed using trapezoidal-rule approximations, as used for the tabulated data points (symbols) in Fig. 2(b,c). A smooth representation can be created by fitting cubic spline function to each tabulation (de Boor, 2001; Wahba, 1990). By definition, cubic splines have smooth continuous first-order
Figure 2: Surveyed data of typical cross-section from Waller Creek, Texas (a) raw $y$: $z$ survey data; (b) depth-area and derivative relationships (symbols), lines represent spline; (c) friction-depth and derivative relationships (symbols), lines represent spline. Data courtesy of City of Austin.
derivatives, therefore $\partial h/\partial A$ and $\partial F/\partial A$ in the Jacobian will also be smooth. When the “knots” of the spline functions are chosen as all the tabulated data points, the spline function will return the same values as the original data, preserving exactly the tabulated $F:A$ and $h:A$. However, exact spline fitting will maintain localized non-monotonic behavior in the underlying data, resulting in the derivatives that, although smooth, may have multiple minima and maxima over relatively small changes in $A$. This cycling of the derivative sign can cause slower convergence or temporal oscillation of the solution. The localized derivative maxima and minima can be considered cross-section features well below the resolution of a 1D model that are therefore unimportant to solution accuracy. That is, a 1D model is similar to applying a filter of length scale $\sqrt{A}$ to the flow physics, such that flow features below that scale are smoothed in the partial differential equations. Thus, local maxima and minima imply small perturbations in the flow below resolvable length scales of the 1D model. To obtain smoother derivatives than achievable with exact spline fitting, the cross-section data for SPRINT are sub-sampled before splining. The dashed lines in Fig. 2(b,c), show sub-sampled splines that are close fits to the $h:A$ and $F:A$ data, while providing smooth derivatives with relatively few local minima and maxima. Complete analysis of the effects of this splining on model results remains a subject for future inquiry.

3.6. Reach junctions

In SPRINT, reach junctions are defined (at a minimum) as locations where two or more upstream river branches connect with a downstream branch. Additionally, reach junctions are defined wherever river features, whether manmade or natural, develop flow conditions that are not reasonably approximated by the Saint-Venant equations; e.g. dams, weirs, waterfalls, culverts, bridges. Cunge et al. (1980) discusses these “interior boundary conditions” at length. Between reach junctions, multiple solution nodes may be defined with segment spacings that are either uniform or non-uniform. The SPRINT discretization is a non-staggered, node-based approach solving for $\{Q,A\}$ at each node, resulting in $2N + 2$ unknowns over a reach with $2N$ equations, therefore requiring two boundary conditions. The first boundary condition is conservation of mass. For $R$ upstream reaches connecting to a single downstream reach:

$$Q_D = \sum_{k=1}^{R} Q_{U(k)}$$

where $Q_{U(k)}$ and $Q_D$ are the upstream and downstream flow rates for each reach at the junction. Future development of SPRINT will include the ability to handle multiple downstream reaches. A simple example is shown in Fig. 3 for the junction of two upstream reaches with a single downstream reach.

The second boundary condition at a junction is continuity of water surface levels, such that a single elevation describes the water surface elevation ($\eta$) at the junction node. However, this conceptually simple idea presents non-trivial problems for any model using $A$ as a solution variable and either $h$ or $\eta$ as an auxiliary variable. A
key problem is that the relationship between $A$ and $\eta$ is usually poorly defined for tributaries at a junction (e.g. Szupiany et al., 2012). That is, for the purposes of the cross-section integrated Saint-Venant equations, a junction is a single point in space which must have some $h:A$ characteristics, but as yet no one has described a convincing theoretical approach to abstract the complex multi-dimensional geometry of river junctions into unique $h:A$ relationships. Furthermore, the utility of such a universal theory of junction geometry would be somewhat limited: unless multi-beam sonar surveys are conducted at every river junction on a regular basis, the geometry of these geomorphically-dynamic regions will never be precisely known. Thus, we seek a reasonable approximation in keeping with resolved scales associated with the cross-section integrated 1D equations. In SPRINT, the downstream node (i.e. uppermost node of the downstream reach) is the “junction node,” which is part of the Saint-Venant discretization of the downstream reach. The furthest downstream nodes of the $R$ upstream reaches should be close enough to junction node such that it is reasonable to require

$$\eta_D = \eta_{U(k)} \quad : \quad k = \{1...R\}$$

(29)

where $\eta_D$, $\eta_{U(k)}$ are the surface elevations of the downstream and upstream reaches at the junction. Note that if the measured upstream cross-sections are a significant distance from the junction node, an artificial cross-section should be introduced in each upstream reach sufficiently close to the junction node such that eq. (29) applies. Along a river’s main stem, an artificial cross-section may be created by interpolation between the junction and the nearest upstream section. Where a smaller tributary joins a larger river, the downstream cross-section could be created by downstream projection of the nearest upstream cross-section using the bottom slope and distance downstream. Automating this type of geometric processing for large-scale river networks remains a challenge. Given appropriate geometry, satisfying eq. (29) requires
using \( \eta = h(A) + z_b \), where \( z_b \) is local elevation of the river bottom. It follows that

\[
\eta_D + z_D = h_{U(k)} + z_{U(k)} : \quad k = \{1...R\}
\]  

(30)

where \( z_D \) is the channel bottom at the downstream node and our auxiliary function is \( h = h(A) \) derived from cross-section geometry as previously described. It is certainly possible to directly discretize eq. (30); however, for general cross-sections such discretization creates nonlinearity at the junction that slows global solution convergence. We question the necessity of exactly enforcing eq. (30) since the discrete \( h:A \) relationships are approximations and the upstream bottom elevations near the junction \( (z_{U(k)}) \) are obtained by interpolation or projection to be sufficiently close to the junction node. As a simpler approach, in pre-processing the river geometry, SPRINT uses relationships of the form

\[
A_{U(k)}(\eta) = r_{(k)} A_D(\eta) : \quad k = \{1...R\}
\]

(31)

The resulting \( r_{(k)} \) are functions of \( A_D(\eta) \), which are computed using cubic splines from geometric data such that eq. (30) is approximately satisfied. SPRINT uses the time \( n \) value of \( r_{(k)} \) for the junction condition during the time step from \( n \) to \( n + 1 \). This approach ensures the junction boundary condition at any time step is computationally linear in space, but nonlinearly evolves through time. Using these linear junction conditions accelerates solution convergence in a system with numerous branches.

3.7. Small depths

A challenge for any flow model is the behavior as \( h \to 0 \) and \( A \to 0 \). This area is documented extensively for 2D shallow-water models where “wetting and drying” is a common problem (e.g., Li and Duffy, 2011; Yu and Lane, 2006), but the issue is mentioned for only a few 1D models (Chen et al., 2012; Tucciarelli, 2003). The principal issue is that the governing equations are not well posed for \( A \leq 0 \), but matrix solution techniques cannot guarantee positive values of \( A \) during Newton-Raphson iteration. From the perspective of CRD, wetting/drying occurs at small depths and flows, which will have little impact on the overall river network solution. Ideally, the discrete equations should asymptotically provide \( Q \to 0 \) as \( A \to 0 \). Unfortunately, at small \( h \) the relationships in the friction slope, eq. (3), are not well behaved and can cause solution instability when \( A \to 0 \) faster than \( Q \to 0 \), which translates into unphysically high velocities. Since the assumptions for the Saint-Venant equations as well as the friction model are invalid as the depth approaches the topographical roughness scale, it is reasonable to model 1D wetting/drying at small scales by an \textit{ad hoc} approach. To address this issue, SPRINT uses a dynamically-modified threshold friction coefficient \( \tilde{n}_{TH} \) in place of \( \tilde{n} \) in eq. (11) so that for \( A \) below threshold area \( A_{TH} \)

\[
\tilde{n}_{TH} = k \tilde{n}
\]

(32)

where the scaling factor \( k \) is

\[
k = 1 + \frac{1}{a} \log \left\{ 1 + \exp \left( a \left[ \frac{A_{TH}}{A} - 1.0 \right] \right) \right\}
\]

(33)
where $a$ (default value of 12) controls the slope of the transition across $A_{TH}$. For $A < A_{TH}$ as $A \to 0$ the value of $k$ increases exponentially. The resulting exponential increase in $\hat{n}_{TH}$ serves to drive $Q \to 0$ faster than $A \to 0$. Although this approach reduces the likelihood of infinitesimally small or negative $A$, it cannot eliminate the possibility (chiefly because of Gibbs oscillations at a sharp front). Thus, it is necessary to further provide small minimum value of $A_{min} < A_{TH}$ used as a lower bound to reset $A$ if $A < A_{min}$ during Newton-Raphson iteration.

4. SPRINT implementation details

4.1. Netlist

A netlist syntax for river network topology has been developed using ideas from VLSI design ([2.5]). Our river netlist is organized using a set of `def` blocks. Typically defined within a block is either a segment (river reach), a computational node (connection between two reaches), or a junction (connection between multiple reaches). Data within a block includes the river cross-section shape, length of the computational element, and flow resistance coefficient (Manning’s $\hat{n}$). Additional blocks are used to define the upstream boundary conditions (flow rates), downstream boundary conditions (wetted area or stage), and lateral inflows. There are also optional blocks to control the simulation and printing behavior of the SPRINT simulator. A simple example netlist is provided in the appendix. Due to space limitations, the details of SPRINT netlist syntax are not explained herein; complete documentation is provided in [Liu (2012)].

The SPRINT netlist specification can be provided “out-of-order;” that is, the connectivity between reaches is defined explicitly by nodal connections listed within the `def` blocks rather than implicitly by a required ordering of the reaches. This enables networks to be readily built from existing network databases. A topographical checker is implemented to analyze network continuity using a network graph and a depth-first-search (DFS) algorithm. The topographical checker also ensures boundary conditions are defined for extreme upstream and downstream nodes. If the netlist has broken connectivity (isolated reaches) or lacks boundary conditions the network fails topographical checking and the SPRINT returns an error state without trying to run a simulation.

4.2. Software design

SPRINT is implemented in C++ in object-oriented design style. It has been tested on various x86_64 platforms from both Intel and AMD, as well as IBM PowerPC platforms. The code implements a clean interface between the sparse linear solver and other parts of the software so that the solution can use any state-of-the-art sparse linear algebra package, such as Berkeley Sparse or UMFPACK ([Davis, 2004]); the latter is the default linear matrix solver.

An important design aspect of SPRINT is that all routines are Application Programming Interface (API) compliant. This approach is integrated through the code,
e.g. the construction of geometry for a computational node in the river network is simply an API call. In overall operation, the top level driver parses the user’s netlist and makes successive API calls to the kernel engine. A higher-level application could make successive API calls to construct river models and run simulations without actually requiring on-disk file exchange. This design philosophy will allow SPRINT to be embedded in other hydrosience packages. For example, a 2D land surface model could call SPRINT model through the API to handle the channel routing functions.

The SPRINT front-end has been written and tested to be http compatible, implemented as a special Apache service. The system has been tested wherein a remote user submits a model netlist through the http front-end to a remote server on which SPRINT is installed. Results are returned using http tools.

5. Model performance

5.1. Overview of tests

SPRINT has been applied to several test cases providing model-model comparisons and illustrating model capabilities and execution speed. Ideally, the model would be tested against a large-scale benchmark case of complex cross-sections, boundary conditions, observed flows and water surface elevations in a multi-branch river, but public data for such test cases do not presently exist. In §5.2 test simulations demonstrate the effect of the Jacobian bypass and smoothing auxiliary functions on model solutions. In §5.3, §5.4, and §5.5 SPRINT is compared against model results from HEC-RAS, the U.S. Army Corps of Engineers Hydrologic Engineering Center River Analysis System, which is an accepted reference engineering model for solution of the Saint-Venant equations and has been evaluated in numerous field settings. In §5.6 details of model configurations are provided and the computational times for SPRINT and HEC-RAS are compared. In §5.7 the capability of SPRINT for modeling a large river network is demonstrated on 15 x 10^3 km of river in the Guadalupe and San Antonio River basin in Central Texas.

5.2. Jacobian bypass and smoothed auxiliary functions

Performance effects of the Jacobian bypass method and smoothing the auxiliary functions were tested with unsteady flow simulations over two different reaches of Waller Creek in Central Texas (USA) described in Table 2. These reaches have complex varying cross-sectional bathymetry (similar to Fig. 2). The simulations were based on inflows generated by the HEC-HMS modeling system operated by the City of Austin. The goal of this test was not to produce a calibrated model, but to compare the operation of the SPRINT model both with and without the Jacobian bypass and smoothed auxiliary functions.
Table 2: Model configuration for unsteady flow test simulations of Waller Creek reaches for storm event of Sept. 7, 2010. “Simulated time” refers to the number of hours in the event simulated by the model.

<table>
<thead>
<tr>
<th>Case</th>
<th>total river length (km)</th>
<th>simulated time (h)</th>
<th>time step (s)</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC1</td>
<td>11.6</td>
<td>20</td>
<td>60</td>
<td>288</td>
</tr>
<tr>
<td>WC2</td>
<td>6.4</td>
<td>35</td>
<td>60</td>
<td>149</td>
</tr>
</tbody>
</table>

Fig. 4(a) shows results for a single node in the simulation of Case WC2 both with (Algorithm 1) and without (Algorithm 2) the Jacobian bypass. As expected from the discussion in §3.4, the surface elevation and flow results are identical throughout the river reach over the entire simulation. Results for other nodes in the flow simulation are similar (not shown).

Simulations results in Fig. 4(b) compare effects of using the smoothed auxiliary function to using raw $y:z$ data. The raw data is used to create discrete auxiliary functions $h: A$ and $F: A$ with the trapezoidal rule, which captures all the cross-section variability illustrated in Figs. 2b and 2c. Unsurprisingly, the small changes associated with smoothing cause small changes in the modeled surface elevations (and flows, not shown); these changes are consistent with the underlying uncertainty in cross-section survey data, such that it is difficult to label one of the simulations more correct than the other. Indeed, there is a logical difficulty in presuming that a numerical solution of partial differential equations invoked on a system characterized by a non-$C^1$ auxiliary functions (i.e. symbols in Fig. 2b and 2c) is a better physical representation than a numerical solution to the same equations using approximations that are $C^1$ (i.e. dashed lines in Fig. 2b and 2c).

Tables 3 and 4 provide the performance characteristics of the Waller Creek simulations of Table 2. As expected from the §3.4 discussion, the Jacobian bypass method increases the number of Newton iterations by slowing the convergence rate, but decreases the overall computational (CPU) time by reducing the number of factorizations. Use of the smoothed $h(A)$ and $F(A)$ decreases the number of Newton iterations by reducing convergence problems associated with non-$C^1$ auxiliary functions, and therefore further decreases the CPU time. These acceleration techniques can have more dramatic impacts in larger systems where the Jacobians provide an ill-conditioned system and raw $h: A$ and $F: A$ data have more extreme discontinuities. Note that these simulations use the UMFPACK solver for all simulations, which already has extremely efficient LU decomposition and solution.
Table 3: Average Newton iterations per time step

<table>
<thead>
<tr>
<th>Case</th>
<th>Raw $h(A)$ and $F(A)$ w/o bypass</th>
<th>Raw $h(A)$ and $F(A)$ with bypass</th>
<th>Smooth $h(A)$ and $F(A)$ with bypass</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC1</td>
<td>12.1</td>
<td>17.0</td>
<td>13.1</td>
</tr>
<tr>
<td>WC2</td>
<td>11.3</td>
<td>15.0</td>
<td>11.9</td>
</tr>
</tbody>
</table>

Table 4: Average CPU time (milliseconds) per time step. CPU time decrease is calculated from the raw data $h : F$ simulations without Jacobian bypass to the smoothed $h : F$ simulations with Jacobian bypass.

<table>
<thead>
<tr>
<th>Case</th>
<th>Raw $h(A)$ and $F(A)$ w/o bypass</th>
<th>Raw $h(A)$ and $F(A)$ with bypass</th>
<th>Smooth $h(A)$ and $F(A)$ with bypass</th>
<th>CPU time decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC1</td>
<td>8.4</td>
<td>4.5</td>
<td>3.4</td>
<td>60%</td>
</tr>
<tr>
<td>WC2</td>
<td>4.1</td>
<td>2.3</td>
<td>1.6</td>
<td>61%</td>
</tr>
</tbody>
</table>
Figure 4: Water surface elevation (stage height) at a single node of a 6.4 km river reach modeled in Waller Creek (Texas, USA). (a) solutions with and without Jacobian bypass acceleration of the Newton-Raphson method (§3.4). (b) solutions using raw $y:z$ and smoothed auxiliary functions (§3.5).
5.3. Case 1: uniform reach simulations

To demonstrate that SPRINT is identical to HEC-RAS for simple simulations, Fig. 5 shows the steady-state results for simulations using rectangular (Case 1a) and trapezoidal (Case 1b) cross-section channels. Both models used unsteady flow algorithms starting from uniform initial conditions. A reach length of 5,600 m modeled by 70 segments of 80 m each. The channels have uniform bottom slopes of \(2.5 \times 10^{-3}\) and uniform Manning’s \(\tilde{n}\) of 0.0035. The rectangular channel width is 20 m. The bottom width for the trapezoidal channel is 3.048 m (10 ft) with a sidewall rise/run of 2.0. The upstream boundary condition is a constant flow rate of 500 m\(^3\)/s and the downstream boundary condition is a constant depth of 5 m. A selected subset of the results are tabulated in Table 5. With uniform cross-sections and constant flow conditions the nonlinear terms are negligible, so the difference between the exact solution of SPRINT and the linearized solution of HEC-RAS are less than the convergence criterion of HEC-RAS.

<table>
<thead>
<tr>
<th>Node</th>
<th>RAS result</th>
<th>SPRINT result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>76.864</td>
<td>76.8638</td>
</tr>
<tr>
<td>8</td>
<td>75.463</td>
<td>75.4630</td>
</tr>
<tr>
<td>16</td>
<td>73.861</td>
<td>73.8611</td>
</tr>
<tr>
<td>24</td>
<td>72.257</td>
<td>72.2567</td>
</tr>
<tr>
<td>32</td>
<td>70.647</td>
<td>70.6469</td>
</tr>
<tr>
<td>40</td>
<td>69.024</td>
<td>69.0245</td>
</tr>
<tr>
<td>48</td>
<td>67.373</td>
<td>67.3725</td>
</tr>
<tr>
<td>56</td>
<td>65.647</td>
<td>65.6471</td>
</tr>
<tr>
<td>64</td>
<td>63.706</td>
<td>63.7060</td>
</tr>
</tbody>
</table>

Table 5: Tabulated surface elevation (in feet using raw data baseline, as provided by HEC-RAS) at selected nodes for Case 1a. Differences are below the convergence criterion of 0.001 ft of HEC-RAS.
Figure 5: Model comparison of surface elevations above lowest downstream point for (a) Case 1a, uniform rectangular cross section and (b) Case 1b, trapezoidal cross-section.
5.4. Case 2: two-section reach with unsteady BC

An abrupt transition in channel slope and roughness provides a challenge due to the nonlinearities associated with sharp transitions. SPRINT and HEC-RAS are compared for an open channel using a 378 m upstream section with a steeper slope ($S_{0u} = 5 \times 10^{-3}$) and a 573 m downstream section with a milder slope ($S_{0d} = 2.5 \times 10^{-3}$). Both sections used rectangular cross-sections with uniform widths of 6.096 m (20 ft). All computational segments were 12.192 m (40 ft) in length. Different Manning’s $\bar{n}$ were used in the two sections: $2.5 \times 10^{-3}$ in the upstream reach and $1.5 \times 10^{-3}$ in the downstream reach. To evaluate unsteady performance, the upstream and downstream boundary conditions were time varying, as shown in Fig. 6.

A comparison of the modeled unsteady flow rates and depths at a single computational node are illustrated in Fig. 7. Although these anecdotal results indicate excellent agreement between the two models is locally achieved, a global analysis shows discrepancies. The steady-state water surface elevation beyond 100 h is shown in Fig. 8. There are two notable differences. Firstly, at the transition between sections HEC-RAS shows a slight increase in the upstream water surface gradient before the adjusting to the lower gradient of the downstream section, resulting in a 4 cm discrepancy between the models at the junction. SPRINT shows a smoother transition of the water surface through this area. Secondly, HEC-RAS shows a small dip in the water surface elevation (approximately 7 cm) centered about 600 m downstream. We have not been able to determine the cause of either discrepancy, but we have ruled out errors in the model setup. Without access to the HEC-RAS source code we have not been able to further investigate; however, the second dip appears to be a numerical artifact rather than any expected physical phenomena. Given the smoothness of the boundary other than at the transition, a reasonable hypothesis is that HEC-RAS is having convergence problems near the transition, which develops an error that is contaminating the downstream solution. The temporal evolution of the water surface profiles for SPRINT and HEC-RAS is shown in Fig. 9. The overall appearance of the two model results is similar; however, Fig. 9(c) shows that the relative difference (normalized by the maximum and minimum surface elevations of the two reaches) is on the order of 2% to 3%, which appears to be a function of the discrepancies at the transition and the downstream dip in HEC-RAS. The difference data normalized by depth (not shown) is qualitatively similar.
Figure 6: Time-varying boundary condition for Case 2 two-section unsteady flow

Figure 7: Water depth and flow rate at a single node in the low-gradient section of the channel for Case 2.
Figure 8: Steady-state surface elevations from unsteady solutions of both SPRINT and HEC-RAS for Case 2.
Figure 9: Evolution of unsteady surface elevation ($\eta$) for Case 2 in (a) SPRINT, (b) HEC-RAS, and (C) relative difference at three hour time intervals.
5.5. Case 3: Non-uniform cross-sections with unsteady flow

A more complex model-model comparison can be made with the nonuniform reach provided in the HEC-RAS installation package. A subset of the cross-section data is used for this test. The total reach length is $3.74 \times 10^4$ m, modeled by 31 computational nodes with non-uniform separations. In Fig. 10, the cross-sections are superimposed over an interpolated surface to visualize the variability of the system. Manning’s $\bar{n}$ varies from 0.030 to 0.036 throughout the system. The upstream boundary condition is a time-varying flow rate and the downstream boundary condition is a time-varying surface elevation. The simulations model 100 hours of flow using a 5 minute time step. Fig. 11 shows the steady-state surface elevation for the two models. The largest deviations are in a region of sharp transitions approximately 18 km downstream. Fig. 12 shows that HEC-RAS and SPRINT are in reasonable agreement, with differences being less than 3% of depth. The errors in Fig. 12 are depth normalized, showing more substantial error at the start and the end of the simulations when the depths are smaller. Note that the error lines at 12 and 96 hours echo the bottom profile in Fig. 11. These results indicate that SPRINT does not produce identical solutions to HEC-RAS, but the results are close enough that differences can be reasonably attributed to the difference in the choice of governing equations and numerical behavior over sharply-changing topography.
Figure 11: Steady state solutions at 100 h for Case 3.

Figure 12: Evolution of unsteady surface elevation ($\eta$) for Case 3, (a) — HEC-RAS; $\circ$ SPRINT and (b) simulation difference normalized by the local depth.
5.6. Computational time

Computational (CPU) time comparisons between SPRINT and HEC-RAS were made for the three test cases presented in §5.3 to §5.5. Note that because of the difficulty in preparing a large HEC-RAS file from the Guadalupe river network data (§5.7), it was not possible to make a comparison of the CPU time on a larger $O(10^5)$ node simulation. Because SPRINT has been developed under Linux OS whereas HEC-RAS is only available as a Windows OS executable, a single x86 server was configured as a dual boot computer (Windows XP SP3 for HEC-RAS and Linux with kernel version 3.2.0 for SPRINT). SPRINT code was compiled using gcc version 4.6.3. The x86 server had two Intel dual-core Xeon 5160 CPUs operating at 3.00 GHz with 16GB of physical memory.

In the HEC-RAS unsteady simulations, only the “geometry preprocessor” and “unsteady flow simulation” options were selected. The computationally expensive “post processor” step was deactivated. The same time step size was used in both SPRINT and HEC-RAS simulations, which varies for the cases as presented in Table 6. The data output intervals were set to 1 hour. For iterative convergence of each time step, the tolerances of surface calculation and flow calculation in HEC-RAS were set to $10^{-5}$ while SPRINT used the default value of $10^{-7}$. The maximum number of iterations in HEC-RAS was set to 40 (maximum for HEC-RAS), and 25 for SPRINT. Note that under HEC-RAS convergence is not guaranteed, but the code will simply stop the matrix solution after 40 iterations no matter what error remains; the remaining unconverged error will propagate forward into the next time step. However in SPRINT, initial failure to converge causes changes in the solution algorithm and reduction of the time step for another 25 iterations – a process that repeats until convergence is achieved (§3.4). The computational time results are provided in Table 6 which clearly demonstrates the speed advantage of SPRINT.

<table>
<thead>
<tr>
<th></th>
<th>Number of elements</th>
<th>Number of time steps</th>
<th>Time step size (s)</th>
<th>RAS CPU (s)</th>
<th>SPRINT CPU (s)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>71</td>
<td>720</td>
<td>10</td>
<td>4.97</td>
<td>0.073</td>
<td>68:1</td>
</tr>
<tr>
<td>Case 2</td>
<td>78</td>
<td>2970</td>
<td>120</td>
<td>21.33</td>
<td>1.56</td>
<td>14:1</td>
</tr>
<tr>
<td>Case 3</td>
<td>31</td>
<td>5760</td>
<td>60</td>
<td>36.94</td>
<td>2.35</td>
<td>16:1</td>
</tr>
</tbody>
</table>

Table 6: Computational time comparison for HEC-RAS and SPRINT.
5.7. Guadalupe River Basin

The Guadalupe and San Antonio River basins in Central Texas are used to illustrate SPRINT’s ability to solve a large river-network problem without requiring substantial computational time. These combined river basins were chosen as they were recently the subject of a Muskingum reduced-physics modelling project (David et al., 2011) whose data were made available to the present authors. The full NHD-Plus data provides 5175 reaches for these river basins. Approximately 1500 first-order streams were removed from the data set before modelling; these streams contributed little to the river dynamics but made visualization of the results and integration with the land surface model considerably more difficult. The SPRINT model includes 3679 NHDPlus reaches with more than 1800 junctions and 15513 km of river represented by more than $1.3 \times 10^5$ computational nodes. The typical separation between nodes is $O(10^2 m)$. Cross-sectional surveys for the network were not available, so this test case used trapezoidal cross-sections synthesized using mean annual flow rate data from NHDPlus. The trapezoid widths ($W$) were calculated based on the empirical equation in Western et al. (1997) of $W = \alpha Q^{0.5}$, where we set $\alpha = 1.5$. Precipitation data for a 14-day storm event beginning 2 April 2002 was used with the Noah-D land-surface model previously applied by David et al. (2011) to generate river inflow boundary conditions for the model. The default model time step was set to 6 minutes, although the SPRINT code makes adjustments based on convergence rate (see §3.4). Flow and depth results at several typical time intervals are shown in Fig. 13.

The literature does not contain any demonstrations of a Saint-Venant model solving a river network of similar scales; indeed, this computational problem is two orders of magnitude larger than the recent Saint-Venant solution in the Amazon basin hydrological model of Paiva et al. (2013). The Guadalupe river network SPRINT simulation was run on the same desktop computer described in §5.6, but without using a virtual machine. Computation of the 14-day event required slightly more than 62 minutes of computer (CPU) time, which means the model computations were completed roughly $330 \times$ faster than the real event time, implying SPRINT is capable of computing a year of flow through $O(10^5)$ elements using only a day of desktop computer simulation – without using a supercomputer or parallel algorithms for multiple processors. Because the solution methods are built on standard numerical libraries that have already been optimized for parallel solution (§4.2), SPRINT should be substantially faster when ported to a supercomputer. Indeed, David et al. (2011) reported a $6 \times$ speed up in matrix solution time for a Muskingum model when the number of computational cores was increased from 1 to 16 for a similar $O(10^5)$ matrix.
Figure 13: Guadalupe and San Antonio Rivers. Color scale represents normalized channel cross-sectional area. Line width represents scaled flow rate.
6. Discussion and Summary

The SPRINT model demonstration shows that unsteady Saint-Venant solutions in large river networks are not necessarily computationally expensive. By applying techniques developed in VLSI microprocessor design, efficient and stable solutions to unsteady nonlinear problems can be achieved at reasonable computational cost. Results in §5.7 show that SPRINT implemented on a common desktop computer can simulate more than $10^5$ computational elements at a speed more than 330 times faster than real time. If the computational scaling of David et al. (2011) holds, it is reasonable to expect 5 simulated years in a single day of computer time for $10^5$ elements solved on a 16-core workstation. For the near future, scaling up to $10^6$–$10^7$ elements will likely require supercomputers. Because SPRINT uses standard numerical solution software packages that are already in use on parallel supercomputers, scaling up the model should be easier than adapting hydraulic models that depend on the Windows computing environment. Based on these results, future coupled hydrological/hydraulic models at regional to continental scales should use the Saint-Venant equations except in limited reaches where it can be conclusively demonstrated that all possible flow conditions can be modelled with reduced-physics approaches. The critical future limitation for practical CRD models is that dynamic Saint-Venant solutions require adequate channel cross-sectional and roughness data, along with extensive observations for calibration and validation (Hodges, 2013).

The Jacobian bypass method (§3.4) and smoothing the river geometry auxiliary functions (§3.5) are shown herein to accelerate the river network solution (§5.2). These techniques can be adapted to other models, making it possible to use a fully-implicit numerical schemes without linearizing either the governing equations or the Jacobian matrix. SPRINT is shown to produce identical results to the accepted hydraulic model HEC-RAS in a simple flow (§5.3), and results that are very close in more complicated flows (§5.4, 5.5). Identical results in more complex flows are not expected because the unsteady solution algorithm in HEC-RAS is linearized (Brunner, 2010) and does not provide for smoothing of depth-area and friction-area relationships developed for SPRINT.

Of the advances in this paper, the most far-reaching effects could result from widespread adoption of something similar to the VLSI netlist (§4.1) and development of community standards and conventions for defining river geometry (§2.5). Using a netlist as an open-source geometry abstraction has several tangible benefits. The first benefit is encouraging greater competition and innovation in simulation and analysis models. Being able to run different software packages without requiring topology translation between proprietary standards lowers the barriers to evaluating new models. The experience in VLSI is that model developers are more innovative when new models are easily tested and adopted, resulting in rapid model improvement. The second benefit is in user flexibility. Instead of being locked into a closed environment, a user can freely apply different models for different problems or compare results of a single problem simulated with different models. A third benefit is
encouraging collaboration between academic researchers and industrial practitioners by providing easier data exchange between the communities. Finally, the open-source topological abstraction allows creation of public benchmarks and open contests for simulation engines \cite{Li2011}. Clearly these benefits would also be advantageous to the hydrological modeling community, where a few established river models dominate both commercial and academic work but interchange between models is difficult and time consuming. Rapid advancement of hydrology at continental scales will depend, in a large part, on how efficiently scientists can move between models and make use of new ideas. For large-scale parallelization, we believe the most effective approach will include standard data formats that can be invoked through text files, netCDF binary files, or data structures in an API call. Adoption of standard conventions for describing river geometry – the equivalent latitude/longitude conventions applied in an along/cross-stream framework – would be a major step forward in enabling easier linking of land-surface and river routing models \cite{Hodges2013}.

A more speculative potential benefit of the netlist approach would be development of hierarchical modeling for river networks. The tree structure of river networks is a natural fit for the hierarchical approaches enabled by the netlist that are used in VLSI (§2.4). Furthermore, river survey data are dispersed across a wide spectrum of private companies and government agencies, making it difficult for a single organization to obtain and validate geometry for a complete continental-scale river network model. A model hierarchy might be created such that local agencies could work on developing and maintaining models for local river networks, while regional agencies stitch these together for river basins, and national agencies maintain higher-level models that amalgamate the basin-level models for continental-scale results.

The VLSI ideas used in SPRINT are a starting point for exploiting technology created in microprocessor design. Developing and testing some of the more complex VLSI ideas requires a CRD testbed application that directly links a distributed land-surface model with a river network model over a continental scale. Such an effort will require further investment beyond the collaboration that produced the SPRINT model. Furthermore, a major impediment to advancing CRD is the lack of any public data set for model validation. Such a data set should include cross-section geometry for all stream orders, modelled landscape runoff into each river reach, bottom composition or estimated hydraulic roughness, and observed water surface elevations and flow rates for a set of significant low, high, and moderate flow events.

7. Acknowledgments

The first author was supported under U.S. National Science Foundation Grant No. 0710901 during the early part of this project. The authors would like to acknowledge Professor David Maidment and Dr. Cedric David for helpful discussions, as well as for providing the Guadalupe-San Antonio river basin data. The authors are grateful to
Fernando R. Salas and Prabhas R. Gupta for preparing the Guadalupe-San Antonio River simulation as well as data processing for Waller Creek.

8. Appendix

The following is a small sample of a SPRINT netlist. Although the netlist may seem difficult to read, it can be parsed readily by any computer language that reads ASCII text. The netlist itself requires relatively few rules to implement syntactically-correct read/write routines. We considered development of a more general XML format, but the rules for XML made the syntax more complicated than necessary and, we believe, would be a barrier to expanding netlist usage over the wider hydrological community.

In the following netlist, multiple nodes are defined, each distinguished by its unique node name (‘id’=’). The blocks are defined using “def” and “end” keywords at the start and end of a block. Inside each block, values are defined by multiple “key=value” pairs. Keys are similar to standard hydraulic terminology, e.g. s0 is the bottom slope, n is Manning’s $n$. The cross section geometry is defined using $x$ as a horizontal cross-channel measurement and $y$ as a vertical measurement. Node descriptions are independent of their placement in the network. Their connectivity into computational segments is created in the “def segment” lines. Upstream and downstream boundary conditions are defined by “Qsource” and “BoundaryCondition” blocks. The “options” blocks are used to control the behavior of the simulation and output (printing) requirements. Thus, the netlist controls the entire behavior of a simulation, from geometry to forcing, in a single file. For more details on the SPRINT netlist, please refer to the SPRINT users’ manual [Liu, 2012].

```plaintext
##
## one segment river bed
## start from base flow 1 and spin down

def options metric=1 end
def options TimeStep=60 TimeStepUnit=second end
def options PrtInterval=2 PrtIntervalUnit=minute end
def options PrtQ=1 PrtA=1 PrtDepth=1 PrtSurfElev=1 PrtCoord = 1 end

def node id=node_1 s0=0.0083 n=0.04 z0=707.23 h0=0.0
        xcoord=123.45 ycoord=567.89
    def xy
        x=0.0 y=6.0
        x=2.5 y=1.0
        x=3.5 y=1.0
        x=6.0 y=6
    end
end

def node id=2 s0=0.0083 n=0.04 z0=707.23 h0=0.0
    def trapezoidal
```
BottomWidth=1 slope=0.5
end
end

def node id=3 s0=0.0083 n=0.04 z0=707.23 h0=0.0
def xy
  x=0.0 y=6.0
  x=2.5 y=1.0
  x=3.5 y=1.0
  x=6.0 y=6
end
end

def node id=4 s0=0.0083 n=0.04 z0=707.23 h0=0.0
def trapezoidal
  BottomWidth=1 slope=0.5
end
end

def node id=5 s0=0.0083 n=0.04 z0=707.23 h0=0.0
def xy
  x=0.0 y=6.0
  x=2.5 y=1.0
  x=3.5 y=1.0
  x=6.0 y=6
end
end

def node id=6 s0=0.0083 n=0.04 z0=707.23 h0=0.0
def xy
  x=0.0 y=6.0
  x=2.5 y=1.0
  x=3.5 y=1.0
  x=6.0 y=6
end
end

def segment up=node_1 down=2 Length=40 end
def segment up=2 down=3 Length=40 end
def segment up=3 down=4 length=40 end
def segment up=4 down=5 length=40 end
def segment up=5 down=6 length=40 end

def Qsource
  location=node_1
  def TimeSeries
    TimeUnit=minute
    t=0 v=1.0
    t=1 v=1.0
    t=2 v=0.5
    t=4 v=0.1
    t=6 v=0.5
    t=80 v=0.1
\[ t = 100 \quad v = 0.1 \\
 t = 1000 \quad v = 0.1 \\
 t = 1200 \quad v = 0.5 \\
 t = 1400 \quad v = 1.0 \\
 t = 2500 \quad v = 1.0 \\
 t = 5000 \quad v = 1.0 \]

```python
end
end

def BoundaryCondition
    location = 6
    type = area
    def timeseries
        TimeUnit = minute
        t = 0 \quad v = 1
        t = 2800 \quad v = 1
    end
end

def options
    StopTime = 8
    StopTimeUnit = minute
end

```

## end


