Scalable Riemann Solvers with the Discontinuous Galerkin Method for Hyperbolic Network Simulation

Aidan Hamilton
aidan@udel.edu
Department of Mathematics
University of Delaware
Newark, DE, USA

Jing-Mei Qiu
jingqiu@udel.edu
Department of Mathematics
University of Delaware
Newark, DE, USA

Hong Zhang
hzhang@mcs.anl.gov
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA

ABSTRACT

We develop a set of highly efficient and effective computational algorithms and simulation tools for fluid simulations on a network. The mathematical models are a set of hyperbolic conservation laws on edges of a network, as well as coupling conditions on junctions of a network. For example, the shallow water system, together with flux balance and continuity conditions at river intersections, model water flows on a river network. The computationally accurate and robust discontinuous Galerkin methods, coupled with explicit strong stability preserving Runge-Kutta methods, are implemented for simulations on network edges. Meanwhile, linear and nonlinear scalable Riemann solvers are being developed and implemented at network vertices. These network simulations result in tools that are added to the existing PETSc and DMNetwork software libraries for the scientific community in general. Simulation results of a shallow water system on a Mississippi river network with over one billion network variables are performed on an extreme-scale computer using up to 8,192 processor with an optimal parallel efficiency. Further potential applications include traffic flow simulations on a highway network and blood flow simulations on a arterial network, among many others.

CCS CONCEPTS

• **Mathematics of computing** → Discretization; *Solvers*; • **Applied computing** → *Physics*; • **Computing methodologies** → **Massively parallel and high-performance simulations**.

KEYWORDS

Riemann solver, discontinuous Galerkin, hyperbolic networks, PETSc, scalable simulation

ACM Reference Format:
Aidan Hamilton, Jing-Mei Qiu, and Hong Zhang. 2023. Scalable Riemann Solvers with the Discontinuous Galerkin Method for Hyperbolic Network Simulation . In *Platform for Advanced Scientific Computing Conference (PASC '23), June 26–28, 2023, Davos, Switzerland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3592979.3593421

PASC '23, June 26–28, 2023, Davos, Switzerland

ACM ISBN 979-8-4007-0190-0/23/06…$15.00
https://doi.org/10.1145/3592979.3593421

1 INTRODUCTION

Hyperbolic network models and simulations, that effectively predict nonlinear wave interactions in global complex networks among their local components, are of great importance in many fields of science and engineering. Much research effort has been made to the modeling, analysis and computational algorithm development [9]. Yet the algorithm design for general nonlinear network junction coupling conditions, together with a parallel-efficient implementation with support from scalable software libraries, are still lacking.

In this paper, we present our two new contributions in addressing such a gap. On one hand, we propose a new linearized coupling condition, as an approximation to Riemann solutions at network vertices. Compared with the nonlinear implementation of coupling condition, the new algorithm is more computationally efficient and user friendly, while providing comparable accuracy in numerical solutions. Meanwhile we implement the highly efficient, accurate and robust discontinuous Galerkin (DG)- Runge-Kutta (RK) method with the traditional nonlinear Riemann solver, as well as the newly proposed linearized coupling condition, for general hyperbolic network simulations by adding two new classes, namely *NetRiemannProblem* and *NetRiemannSolver*, into an experimental branch of the open-source library PETSc [5–7]. We applied these classes to a Mississippi river network simulation with over one billion network variables (as modeled with 892,740 edges and 872,300 vertices) on the shallow water system over an extreme-scale computer using up to 8,192 processors, and demonstrate the scalability and potential scale of our proposed Riemann solvers with DG method for hyperbolic network simulations.

These implementations are built upon the existing PETSc [5–7] and DMNetwork [2] software libraries, which provide access to scalable solvers and distributed network. There is a wide range of application domains where our proposed Riemann solvers could be useful in predictive modeling and simulations. Examples include traffic flow modeling and simulation on a network of highways, shallow water simulations on a network of rivers, blood flow simulations in arterial networks, etc.

This paper is organized as follows. In Section 2, we introduce hyperbolic network problems. We review nonlinear Riemann problems at network junctions and propose a new user friendly and computationally efficient linearization procedure for general hyperbolic systems. In Section 3, we introduce the RK DG method as a robust, efficient and effective solver for network hyperbolic systems. In Section 4, the implementation in PETSc is discussed. Section 5 presents the numerical experimental results. Section 6 gives a summary of the paper and our ongoing research.

2 HYPERBOLIC CONSERVATION LAW ON A NETWORK

In this section we introduce the notion of a hyperbolic network, along with relevant background material. Throughout we use the shallow water system as our example of hyperbolic conservation laws, however the material presented generalizes to other hyperbolic conservation laws as well.

2.1 Hyperbolic conservation laws on a network

We start with introducing a system of m conservation laws in a 1D domain,

$$\partial_t u + \partial_x f(u) = 0, \tag{1}$$

where $u \in \mathbb{R}^m$ is vector-valued function called the conserved variables and $f : \mathbb{R}^m \to \mathbb{R}^m$ is the flux function. The system is called hyperbolic, if the Jacobian matrix of the flux function is diagonalizable. The hyperbolic nature of the system leads to nonlinear wave propagation on the 1D domain. Now a **network** is a topological graph, i.e. a couple (V, E), where E is a collection of intervals, and V is a collection of vertices connecting these intervals. A system of m conservation laws then naturally extends to a network edge-wise; However, coupling conditions, such as the continuity of certain physics quantities, balance of fluxes, etc, will need to be imposed on the vertices of the network, as boundary conditions for the edge conservation laws. Nonlinear wave propagation will then be further compounded by these coupling conditions.

We give a specific example of a hyperbolic network, modeled by the shallow water system. Assuming a uniform cross section and zero slope, the shallow water system on each edge reads

$$\begin{aligned} \partial_t h + \partial_x (hv) &= 0, \\ \partial_t hv + \partial_x (hv^2 + \frac{g}{2}h^2) &= 0, \end{aligned} \tag{2}$$

where $h(x,t)$ is the water height and $v(x,t)$ the water velocity, g the gravitational constant. Letting $q = hv$, such a system is in the form of (1) with

$$u = \begin{pmatrix} h \\ q \end{pmatrix}, \qquad f(u) = \begin{pmatrix} q \\ q^2/h + \frac{1}{2}gh^2 \end{pmatrix} \tag{3}$$

Then the Jacobian matrix for this system is given by

$$Df(u) = \begin{pmatrix} 0 & 1 \\ -v^2 + gh & 2v \end{pmatrix}, \tag{4}$$

with eigenvalues

$$\lambda_1(u) = v - \sqrt{gh}, \qquad \lambda_2(u) = v + \sqrt{gh} \tag{5}$$

We assume that the system is *fluvial* or *subcritical*, meaning that the fluid velocity v is smaller than the speed $\sqrt{gh}$ of the gravity waves, i.e $|v| < \sqrt{gh}$. Under such a condition, $\lambda_1 < 0, \lambda_2 > 0$, i.e. the fluid has waves propagating in opposite directions. Note that the fluvial condition is essential to define the coupling conditions at network junctions [11, 14]. Extending coupling conditions to *super-critical* systems is still an open problem (see [10] for a simple case) and beyond the scope of this paper.

To complete the shallow water system on a network, we impose algebraic coupling conditions on conservation of mass, i.e. flux balance, and equal height or energy condition conditions. [11] Consider a vertex $v \in V$, then let $E(v)$ be the set of edges connected to v,

$$\sum_{\substack{e \in E(v), \\ incoming}} q_e(v) = \sum_{\substack{e \in E(v), \\ outgoing}} q_e(v), \qquad \forall t > 0, \tag{6}$$

is the flux balance condition. Note here incoming and outgoing is defined by the direction in parameterizing the edge. This condition is further coupled with the equal height for the well-posedness of hyperbolic network [11],

$$h_e = h_{e'} \qquad \forall e,\ e' \in E(v). \tag{7}$$

Note that there exist other coupling conditions such as equal energy [11], but for simplicity of presentation we consider only equal height.

2.2 Riemann Problems at Network Vertices

In this subsection, we introduce the Riemann problem, which provides a fundamental building block to modern numerical methods to hyperbolic systems arise in computational fluid dynamics. In the hyperbolic network setting, understanding the vertex Riemann problem is essential in understanding the theoretical well-posedness of the problem [14]; meanwhile it provides the main guiding principle in our algorithm design of treating coupling conditions at network junctions. For simplicity of the presentation, we will only discuss the Riemann problem for the shallow water system as an example.

2.2.1 A Riemann problem for 1D equation [24]. A Riemann problem for a 1D hyperbolic conservation law (1) has the jump initial condition

$$u_0(x) = \begin{cases} u_L, & x < 0, \\ u_R, & x > 0, \end{cases} \tag{8}$$

for constants u_L, u_R.

Linear system. The Riemann problem for a linear system can be solved by finding scalar Riemann solution along each eigenvector direction. The discontinuity will propagate with the speed determined by the corresponding eigenvalue of the Jacobian matrix. As these Riemann solutions are projected back to the original conservative variable, an intermediate state is u_* is generated. This intermediate state can be found by finding the intersection points of two lines connected to the left and right states (u_L and u_R) with the slope determined from the first and second eigenvectors respectively, in the phase space.

Nonlinear system. For a nonlinear shallow water system (2), the solution to the Riemann problem can be found by locating the intersection point of two nonlinear curves in the phase space plot as in Figure 1. These two curves correspond to the 1-wave connected to u_L and 2-wave connected to u_R. They can be described by the following formula:

$$\phi_L(h; u_L) = \begin{cases} v_L - 2(\sqrt{gh} - \sqrt{gh_L}) & \text{if } h < h_L \quad \text{(rarefaction)} \\ v_L - (h - h_L)\sqrt{g\frac{h+h_L}{2hh_L}} & \text{if } h > h_L \quad \text{(shock)} \end{cases} \tag{9}$$

and

$$\phi_R(h; u_R) = \begin{cases} v_R + 2(\sqrt{gh} - \sqrt{gh_R}) & \text{if } h < h_R \quad \text{(rarefaction)} \\ v_R + (h - h_R)\sqrt{g\frac{h+h_R}{2hh_R}} & \text{if } h > h_R \quad \text{(shock).} \end{cases} \tag{10}$$

Mathematically, solving the Riemann problem is reduced to find an intermediate states with h_* such that $u_* = (h_*, h_* v_*)$ is connected to u_L by a 1-wave and can be connected to u_R by an 2-wave. Computationally, one can numerically find such an intersection by applying a nonlinear root-finder to find h_* such that $\phi_L(h_*; u_L) = \phi_R(h_*; u_R)$. Meanwhile, $\phi_L(h_*; u_L) = \phi_R(h_*; u_R)$ returns the value of v_* such that $u_* = (h_*, h_* v_*)$ can be connected to u_L by a physically corrected 1-wave, and can be connected to u_R by a physically correct 2-wave. Please see Figure 1 for the nonlinear curves connected to u_L and u_R, and their intersection u_*. Notice that the dotted lines correspond to the eigenvector directions of the Jacobians from the linearization of system.

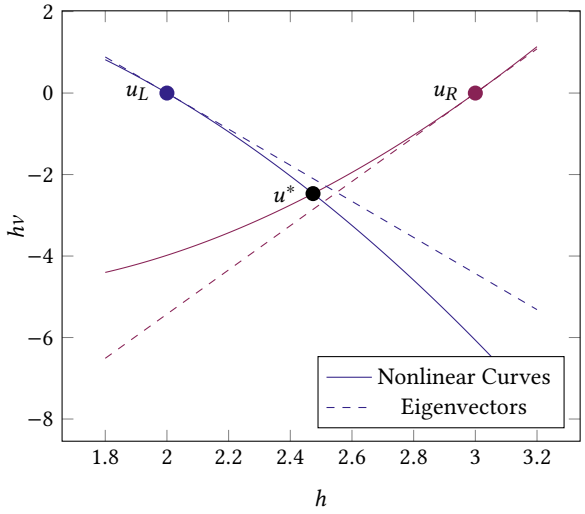

Figure 1: Phase space plot for the Riemann Problem of the shallow water system.

2.2.2 Riemann Problems at Vertices. We first introduce the 1D Riemann problem at a vertex v.

Definition 2.1 (Riemann Problem at a Vertex v [14]).

$$\begin{aligned} &\partial_t u_e + \partial_x f(u_e) = 0, && t \in \mathbb{R}^+ \quad e \in E(v) \\ &u_e(x, t=0) = \overline{u}_e, && x \in \begin{cases} R^+ & \text{if } e \text{ is outgoing} \\ R^- & \text{if } e \text{ is incoming} \end{cases} \end{aligned} \tag{11}$$

where $\overline{u}_e$ are constant states.

For well-posedness of this Riemann problem, we require additional $deg(v)$ algebraic equations known as the coupling conditions. Here $deg(v)$ is the number of edges connected to the vertex. The solution to the vertex Riemann Problem (11) is then given by intermediate states u_e^* such that u_e^* connects to $\overline{u}_e$ by a 1-wave if e is incoming and a 2-wave if e is outgoing, and $\{u_e^*\}_{e \in E(v)}$ satisfy the algebraic coupling conditions. In the shallow-water case, these

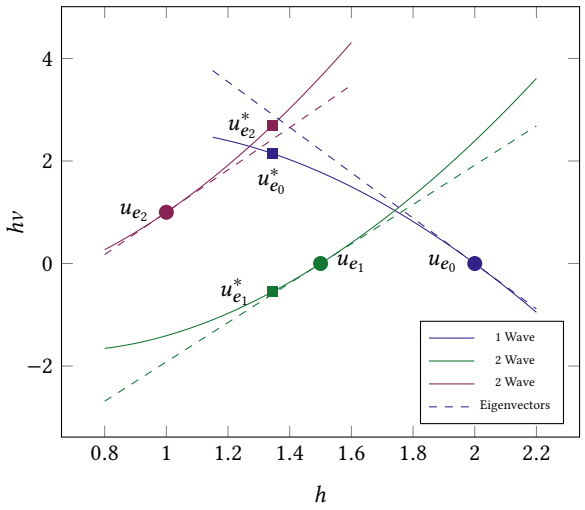

Figure 2: Phase space plot of the Riemann problem on a three-branch vertex. Edge 0 is for incoming edge and edges 1 & 2 are for outgoing. The $\{u_{e_l}^*\}_{l=0}^2$ are intermediate states satisfying the algebraic conditions (6) and (7), while lying along the wave curves.

coupling conditions are,

$$\begin{aligned} h_e^* &= h_{e'}^* \\ \sum_{\substack{e \in E(v), \\ incoming}} v_e^*(v) &= \sum_{\substack{e \in E(v), \\ outgoing}} v_e^*(v) \\ v_e^* &= \phi_R(h_e^*; u_e) \text{ if } e \text{ is outgoing} \\ v_e^* &= \phi_L(h_e^*; u_e) \text{ if } e \text{ is incoming} \end{aligned} \tag{12}$$

This nonlinear system can then be solved numerically by a nonlinear root-finder as in the 1D Riemann problem case. However, the formula for describing waves such as eq. (9) for a general nonlinear system are not easy to obtain analytically. Because of this, this procedure to construct an exact Riemann solver is not easily applicable to general systems.

Figure 2 presents the phase plot of Riemann solutions of a shallow water system on a three-branch vertex. Notice that $u_{e_l}^*$ is connected to $\bar{u}_{e_l}$ respectively by the corresponding incoming or outgoing (1 or 2) waves; and the set of $\{u_{e_l}^*\}_{l=0}^2$ satisfy the continuity condition (same water height h), as well as the flux balance condition.

2.3 Linearized Vertex Riemann Solver

Motivated by the difficulty in constructing exact vertex Riemann solvers and the simplicity of the Riemann problem for a linear system, we propose a generic linearized Riemann solver, that does not rely on formula such as (9). Let $r_1(u)$ denote the eigenvector corresponding to incoming wave and $r_2(u)$ for the outgoing wave. In Figure 2, dotted lines depict lines connected to $\bar{u}_e$ associated with localized incoming/outgoing eigenvector directions; they are tangent to the nonlinear wave curves.

To construct an approximate solver for the vertex Riemann problem, we look for intermediate states u_e^* connected to u_e along locally linearized eigenvector directions instead of the nonlinear wave

curves. That is, we have the following lines parameterized by τ_e connected to $\overline{u}_e$,

$$u_e^* = r_1(\overline{u}_e)\tau_e + \overline{u}_e \ (incoming), \quad u_e^* = r_2(\overline{u}_e)\tau_e + \overline{u}_e \ (outgoing)$$

In our shallow water example we have

$$r_1(u) = \begin{pmatrix} 1 \\ v - \sqrt{gh} \end{pmatrix}, \quad r_2(u) = \begin{pmatrix} 1 \\ v + \sqrt{gh} \end{pmatrix}.$$

The approximate Riemann solver at a vertex results in solving a system with

$$h_e^* \left(\overline{v}_e + \sqrt{g\overline{h}_e} \right) - q_e^* = \overline{h}_e \left(\overline{v}_e + \sqrt{g\overline{h}_e} \right) - \overline{q}_e. \tag{13}$$

for incoming e, with similar for outgoing, together with the coupling conditions (6) and (7). The proposed linearization procedure provides a second order approximation to the exact nonlinear wave curves for $\|\overline{u}_e - u_e^*\|$ sufficiently small. We refer to Chapter 1, Theorems 3.1 and 4.1 of [17] to related discussions.

This linearized procedure has not, to the authors knowledge, been previously presented as a vertex Riemann Solver. Its benefit is that it only requires knowledge of the eigenvectors r_1 and r_2 of the Jacobian of the linearized system, which should be easily available for any system under consideration. Thus it can be generically implemented. Furthermore, as we shall demonstrate in section 5.3 the linearized solver can be much faster than an exact solver, requiring only a single linear solver as opposed to a nonlinear solve in the shallow water system.

3 DISCONTINUOUS GALERKIN DISCRETIZATION OF HYPERBOLIC NETWORK

In this section, we introduce our numerical approach in simulating conservation laws on a 1D domain, followed by discussions on network vertices. We choose to use the explicit strong stability preserving RK DG method [13], due to its high order accuracy, compactness in handling boundary/coupling conditions, as well as its superior performance in a long term simulation.

We assume a discretization of the 1D spatial domain $\Omega = [a, b]$ with N elements

$$a = x_{\frac{1}{2}} < x_{\frac{3}{2}} < \cdots < x_{N+\frac{1}{2}} = b.$$

Each of the element is denoted as $I_j = [x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}]$ with mesh size $h_j = x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}$, $j = 1 \cdots N$. Let h denote the maximal mesh size. We let

$$P_h^k(\Omega) = \{w : w|_{I_j} \in P^k(I_j), m = n, \dots N\}, \tag{14}$$

where $P^k(I_j)$ denote the space of polynomials of degree at most k on the element I_j. The vector version is defined naturally, with a slight abuse of notation P_h^k will be assumed to be the vector version for a system of conservation laws. Note that a function $u_h \in P_h^k$ is piece-wise defined and in general is discontinuous at the element boundaries. The discontinuity at each of the element boundary is handled by an exact or approximate solver to a Riemann problem as in (8). The Riemann problem is essential in understanding the nonlinear shock phenomenon for nonlinear hyperbolic systems. In fact, exact or approximate solutions to Riemann problem is a fundamental building block for modern numerical methods for hyperbolic systems [24].

The DG method for (1) is formulated by a weak formulation, derived from multiplying an equation by some test functions $v \in W_h^k$ and performing integration by parts. Assume (1) is a scalar equation, the DG method looks for $u_h \in P_h^k(\Omega)$, s.t.

$$\begin{aligned} \int_{I_j} v\partial_t u_h dx = \int_{I_j} f(u_h)\partial_x v dx \\ - \left(\hat{f}_{j+\frac{1}{2}} v^-|_{x_{j+\frac{1}{2}}} - \hat{f}_{j-\frac{1}{2}} v^+|_{x_{j-\frac{1}{2}}} \right), \end{aligned} \tag{15}$$

for all test functions $v \in P^k(I_j)$. Here we let $v^\pm$ denote the left and right limits of the function values. The numerical fluxes $\hat{f}_{j+\frac{1}{2}} = f(u^-_{j+\frac{1}{2}}, u^+_{j+\frac{1}{2}})$ with $u^\pm_{j+\frac{1}{2}} = \lim_{x \to x^\pm_{j+1/2}} u_h(x)$, are approximate Riemann solvers with monotone properties at element interfaces. For example, the Lax-Friedrich flux $f(u^-, u^+) = \frac{1}{2}(f(u^-) + f(u^+)) + \frac{\alpha}{2}(u^- - u^+)$ is a monotone flux. The monotonity of the flux gaurantees the entropy stability of the DG method [13]. Further discussion on approximate Riemann solvers can be found in [24]. In the system case, we apply the DG method in a component-wise manner to each equation in the system.

For implementation, we consider a set of basis given by $\{\psi_l(\xi)\}_{l=0}^k$ with $\xi = \frac{x - x_j}{h_j} \in [-\frac{1}{2}, \frac{1}{2}]$ for on each computational element I_j. Then the DG solution u_h can be represented as

$$u_{h,j}(x,t) = \sum_{l=0}^{k} c_{j,l}(t)\psi_l(\xi(x)), \quad x \in I_j \tag{16}$$

where $c_{j,l}$ are the coefficients of the basis. Plug (16) into the DG formulation (15), and let the test function run through the set of basis $\{\psi_l(\xi)\}_{l=0}^k$, we end up with a system of ODEs for $c_{j,l}$, for $j = 1, \cdots N$, $l = 0, \cdots k$. An explicit third order strong stability preserving (SSP) RK method can be applied as an efficient and accurate time integrator. For robustness and nonlinear stability of the algorithm, we apply a characteristic-wise TVB slope limiter slope at each inner stage of the RK integrator [12].

Extending the DG discretization from a 1D domain to the network case requires proper passing of information between DG polynomials at boundary elements on each edge and the corresponding vertex Riemann solver. On one hand, evaluating DG polynomials from boundary elements of all branches connected to a vertex, provides the $\{\bar{u}_e\}_{e \in E(v)}$ as in Riemann initial data in (11); on the other hand, solving the vertex Riemann problem yields intermediate states u_e^*, evaluating which gives $f(u_e^*)$ that is used as flux at the boundary element of DG solutions on each edge connected to the vertex, i.e. $\hat{f}_{\frac{1}{2}}$ and $\hat{f}_{N+\frac{1}{2}}$ in (15).

4 IMPLEMENTATION IN PETSC

We implement our simulations using the Portable, Extensible Toolkit for Scientific Computation (PETSc) [5–7], an open-source library (BSD-style license) for the numerical solution of large-scale applications. PETSc consists of a set of libraries for creating parallel vectors (Vec), matrices (Mat), distributed networks (DMNetwork), scalable linear (KSP), nonlinear (SNES) and timestepping (TS) [3] solvers.

DMNetwork manages the construction and parallel distribution of the networks, and facilitates query of network elements (edges and vertices) and physical components [2]. We use DMNetwork to manage the network structure of our problem, which allows for our scalable performance.

We added two classes, NetRiemannProblem and NetRiemannSolver to the DMNetwork/PETSc to build scalable Riemann Solvers for the DG method for hyperbolic network simulation. Figure 3 illustrates the software structure of NetRiemannProblem and NetRiemannSolver in PETSc. We note that these are experimental classes that are currently in a PETSc branch. The code is available at request.

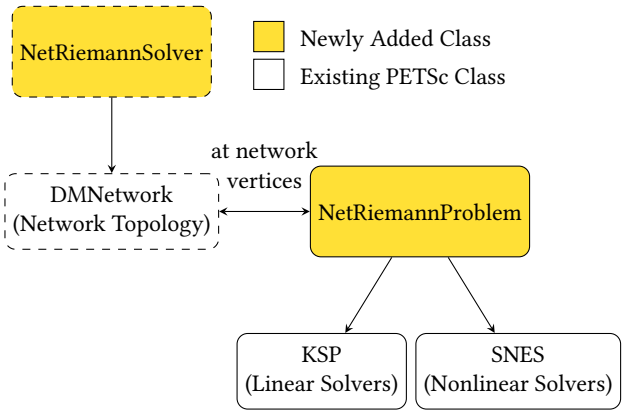

Figure 3: Software Structure of NetRiemannProblem and NetRiemannSolver within PETSc.

The process of solving the vertex Riemann problems (11) on every vertex of a network is logically split into the two classes.

(1) **NetRiemannProblem**: Specifies and solves each local vertex Riemann problem. Not parallel.
(2) **NetRiemannsolver**: Provides scalable parallelism for solving the vertex Riemann problem on all vertices of a distributed network.

The details of each class will follow.

4.1 NetRiemannProblem

NetRiemannProblem is essentially a function object, where a user is responsible for setting up the equations for the Riemann Problem they want to solve, and NetRiemannProblem will then, given initial data for the Riemann problem, actually perform the solve. This abstraction separates the details of particular Riemann problem from its usage in other algorithms such as our DG discretization.

A user is responsible for providing function callbacks for specifying the either nonlinear or linear system (e.g. eq. (12) or (13)) that comprises their Network Riemann problem. It is important to note that the user is not responsible for creating any of the Vec or Mat objects in these routines, nor is responsible for generating or directly using solver objects such as KSP or SNES.

In fact, one essential feature of NetRiemannProblem is its reuse of solver objects. A particular local Riemann Solver such as (12) will be called repeatedly, with different vertex degrees, which requires different size systems and different solver objects. NetRiemannProblem will cache all solver objects by vertex degree, so subsequent calls can reuse the existing solvers. This is hidden from a user, who only needs to provide simple function callbacks specifying systems without thinking about any performance optimizations.

This also allows reuse of symbolic solves in the linear solvers for both linear Riemann problems as well as the Jacobian solves inside of the nonlinear solvers. As the systems are small, a direct LU factorization solve is chosen as the default solver, the symbolic factorization of which is reused among all Riemann solves for a vertex degree. This greatly reduces users' development time and accelerates the solve speed for these systems.

An additional usage is that standard 1D boundary conditions, such as the outflow boundary condition for the shallow water system [22], can be implemented as special case of NetRiemannProblem for degree 1 vertices. This is exactly how we enforce boundary conditions in our DG discretization package.

4.2 NetRiemannSolver

NetRiemannSolver is responsible for assembling and solving vertex Riemann problems on all vertices of a network, stored as a DMNetwork object. A user adds all NetRiemannProblems to a NetRiemannSolver, and assigns each NetRiemannProblem to a subset of the vertices of a DMNetwork. DMNetwork is a class for managing distributed networks, with data associated with the components of the network, see [2] for details. Figure 4 shows how DMNetwork maintains a distributed representation of a network, and it is this distribution that provides the parallelism for our simulation. It is important that only some vertices are shared among processors in DMNetwork.

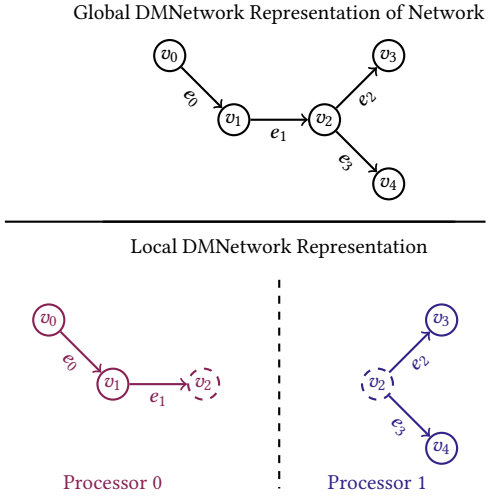

Figure 4: Example of distributed DMNetwork on two processors. The global representation is the logical representaiton of the network. The local representation is how it is actually stored and used. The dashed vertex v_2 is shared among processors, and any data associated with v_2 would be shared as well.

DMNetwork provides the functionality to generate these distributed representations. It uses packages such as ParMETIS [20] for generating a distributed network with load balance.

NetRiemannSolver is thus designed to manage the complexities of efficiently solving NetRiemannProblems on a collection of vertices on a distributed DMNetwork. This reduces to following three main tasks that NetRiemannSolver does.

(1) **Manage Parallelism:**
On the distributed network in DMNetwork, a processor owns a local subnetwork of the full global network as shown in Figure 4, and some vertices are shared among processors. The Riemann data $\overline{u}_e$ for the vertex v Riemann problem is assumed to be associated with the edges $e \in E(v)$. However, if v is shared among processors, like v_2 in Figure 4, each processor copy of v will not have direct access to all of the Riemann data. Thus NetRiemannSolver provides an interface to set the locally available Riemann data, and will take care of the communication and manipulation required to construct vertex Riemann problem for v. For v_2 vertex in Figure 4 processor 0 would set $\overline{u}_1$ and processor 1 would set $\overline{u}_2$ and $\overline{u}_3$. then NetRiemannSolver would perform the communication such that both processors would have access to the Riemann problem solutions u_e^*.

(2) **Batch Solve the NetRiemannProblems:**
NetRiemannSolver efficiently manages iteration through the distributed DMNetwork and performing the NetRiemannProblem solves. Communication and computation are interlaced by solving completely on processor Riemann problems while communicating the Riemann data $\overline{u}$ for the shared vertices. In the case shown in Figure 4 on processor 0, the NetRiemannProblems for v_0 and v_1 would solved while communicating the Riemann data for v_2. Additional user interlacing is also available via runtime options.

(3) **Multiple NetRiemannProblems:**
It is straightforward to add multiple NetRiemannProblems to a NetRiemannSolver, as long as assigned vertices do not overlap. This is essential for practical use as multiple Riemann problems on a single network is the default, when one considers the need for boundary conditions on degree 1 vertices. While not done in our numerical tests presented, this make it easy to assign some vertices to be a linearized approximate NetRiemannProblem, and some as an exact NetRiemannProblem where more accurate solutions are needed.

4.3 DG Implementation

Using NetRiemannProblem, NetRiemannSolver and DMNetwork we present how we implement the scalable DG discretization of hyperbolic networks described in Section 3. The scalability of the implementation primarily comes from DMNetwork distributing the network as shown in Figure 4. Only some vertices of a network are shared, so the DG algorithm described in Section 3 is an entirely local to a processor, except for the computation of the boundary fluxes $\hat{f}_{\frac{1}{2}}$, $\hat{f}_{N+\frac{1}{2}}$, which are handled by the NetRiemannSolver. Thus the only part of the algorithm that performs communication is a NetRiemannSolverSolve() routine from the NetRiemannSolver class.

The implementation requires a setup phase where a user provides the details of the problem they wish to solve. A user would begin by setting the physics of the NetRiemannProblems they wish to use, i.e. setting equations (12), along with algebraic coupling conditions. A DMNetwork would be created, either through a creation routine or loading the information from file and distributed. It is associated with a NetRiemannSolver and a user would choose the set of vertices each NetRiemannProblem should apply to. Other standard algorithmic decisions such as the number of cells per edge, choice of time-integrator, time step size etc, would also be chosen.

After this setup phase the main driving algorithm is the evaluation of the right-hand side of equation (15) which we denote the DG RHS evaluation. This algorithm is shown in Algorithm 1.

Algorithm 1 DG RHS Evaluation for a Hyperbolic Network

```
for vertex v in DMNetwork do
    for e ∈ E(v) do
        Evaluate u_e(v)
        Place u_e(v) in NetRiemannSolver
    end for
end for
NetRiemannSolverSolve()                ▷ Communication is here
▷ All boundary fluxes f̂_{1/2}, f̂_{N+1/2} are now available
for Edge e in DMNetwork do
    for Cell c in edge e do
        Evaluate integral (15)
    end for
end for
```

As it shows, the introduction of the NetRiemannSolver class greatly simplifies the implementation of a scalable DG method.

5 NUMERICAL EXPERIMENTS

We preform four numerical experiments in this section. In Section 5.1 we test the numerical accuracy of the scheme on a reference problem, which verifies the correct implementation of the DG method. In Section 5.2 we test our method on a dam break problem, which verifies the stability of our method on a shock problem. Section 5.3 displays the potential speedup of the linearized vertex Riemann solver versus an exact nonlinear vertex Riemann solver. Section 5.4 demonstrates the scalability of NetRiemannSolver and the overall DG discretization using a large Mississippi river network.

5.1 Convergence Test

We test for the order of convergence of the scheme to verify the implementation. For our domain Γ we use the directed graph G shown in Figure 5, the network Γ is formed by associating $[0, 10]$ for every edge $e \in G$.

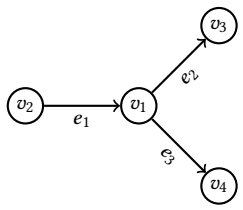

Figure 5: Graph for the convergence test

We consider the shallow water system (2) on Γ, along with the algebraic coupling conditions of height continuity (7) and the conservation of mass (6). We use the following initial condition, chosen such that the solution remains smooth on edges up to $t = 0.1$:

$$\begin{aligned} h_1(x,0) &= 1 + e^{-5(x-9)^2}, \\ h_1 v_1(x,0) &= h_1(x,0)/2, \\ h_2(x,0) &= h_3(x,0) = 1 + e^{-5(10-9)^2}, \\ h_2 v_2(x,0) &= h_3 v_3(x,0) = h_2(x,0)/4. \end{aligned} \tag{17}$$

This example constitutes a pulse of moving water on edge e_1 that passes through the vertex v_1 to interact with edges e_2 and e_3. This solution remains smooth to $t = 0.1$, but has nontrivial vertex Riemann problems. Since the true solution at $t = 0.1$ is not available, we use a refined DG solution with a P^3 basis and 10000 elements per edge, along with the exact vertex Riemann Solver (12), as a reference solution. The DG solutions are evolved in time by a third order SSP RK method [18] with time-step chosen to make the spatial error dominant. Figure 6 and Table 1 show the convergence results of the scheme using the Linearized vertex Riemann Solver (13). Expected order of $k+1$ for a kth order polynomial is observed for a smooth problem. The results from the scheme using the exact vertex Riemann Solver have a negligible difference, showcasing the effectivness of the linearized vertex Riemann Solver.

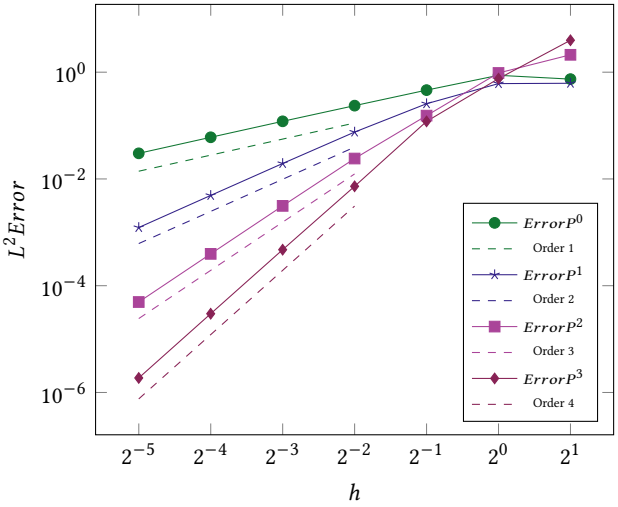

Figure 6: L^2 Errors and orders of convergence for the Section 5.1 test.

Table 1: L^2 Errors and orders of convergence for the Section 5.1 test.

h	P^0 Error	P^0 Order	P^1 Error	P^1 Order	P^2 Error	P^2 Order	P^3 Error	P^3 Order
2	0.74	NaN	0.62	NaN	2.12	NaN	3.96	NaN
1	0.87	−0.24	0.61	0.02	0.97	1.13	0.76	2.39
$\frac{1}{2}$	0.46	0.92	0.26	1.25	0.15	2.65	0.12	2.66
$\frac{1}{4}$	0.24	0.97	$7.55 \cdot 10^{-2}$	1.76	$2.41 \cdot 10^{-2}$	2.67	$7.28 \cdot 10^{-3}$	4.04
$\frac{1}{8}$	0.12	0.98	$1.95 \cdot 10^{-2}$	1.95	$3.13 \cdot 10^{-3}$	2.95	$4.73 \cdot 10^{-4}$	3.94
$\frac{1}{16}$	$6.03 \cdot 10^{-2}$	0.99	$4.91 \cdot 10^{-3}$	1.99	$3.95 \cdot 10^{-4}$	2.99	$2.98 \cdot 10^{-5}$	3.99
$\frac{1}{32}$	$3.02 \cdot 10^{-2}$	1.00	$1.23 \cdot 10^{-3}$	2.00	$4.95 \cdot 10^{-5}$	3.00	$1.87 \cdot 10^{-6}$	4.00

5.2 Dam Break Simulation

Here we describe a test demonstrating the capability to handle a shock problem, as well as providing a visualization of the numerical solution. We test using the graph shown in Figure 7. The network Γ is formed by associating edge e_0 with the interval $[0, 50]$ and all other edges with the interval $[0, 10]$. We again consider the shallow water system (2) on Γ, along with the algebraic coupling conditions of height continuity (7) and the conservation of mass (6).

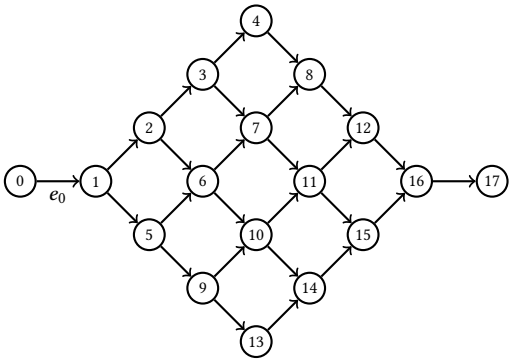

Figure 7: Grid graph variant used in the dam break test, Section 5.2.

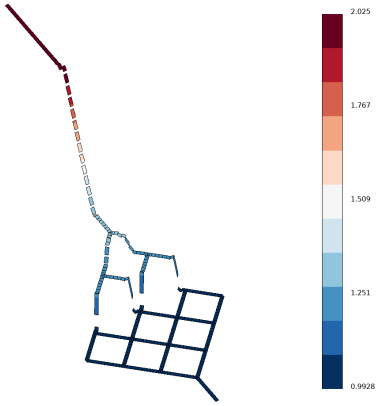

Figure 8: Water height for dam break problem, Section 5.2.

We use "dam break" initial conditions, with

$$\begin{aligned} h_0(x,0) &= 2, \quad h_0 v_0(x,0) = 0 \\ h_e(x,0) &= 1, \quad h_e v_e(x,0) = 0, \quad \forall e \in \Gamma, \quad e \neq 0, \end{aligned} \tag{18}$$

which would simulate a dam breaking at vertex 1 in the network Γ. This initial data is chosen such that the data and the solution are fluvial, using the method presented in [11]. This results in rarefaction wave passing backwards along e_0 and a shock wave propagating throughout the rest of the network, generating reflected waves at every encountered vertex.

We discretize by P^2 polynomials and cells of length 1, and use a Linearized NetRiemannProblem (13) for vertices $1, \ldots, 16$, and outflow boundary conditions for vertices 0 and 17. The numerical solution after the shock has propagated through a set of vertices, is visualized using GLVis [1] in Figure 8. Qualitatively the absence of oscillations in Figure 8 indicates the stability of the discretization,

as well as the Linearized NetRiemannProblem, in the presence of weak shocks.

5.3 Nonlinear vs. Linearized Vertex Riemann Solver

We compare the performance of the linearized vertex Riemann solver (13) versus the exact nonlinear vertex Riemann solver (12). We test the solvers on directed graphs G_n, where edge 1 is incoming on to vertex 1 and edges $2, \ldots, n$ are outgoing from vertex 1.

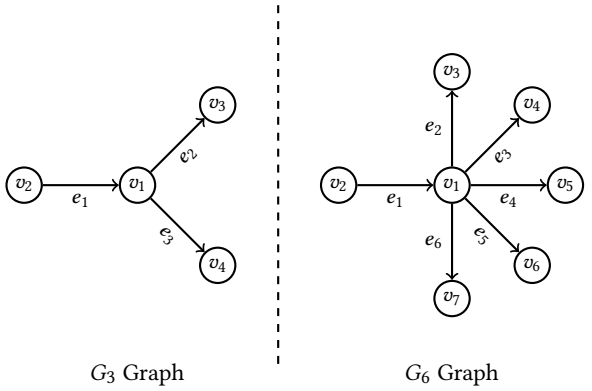

Figure 9: Example of the graphs used in the Section 5.3 tests.

For example, G_3 and G_6 are shown in Figure 9. The following set of Riemann problems is then tested on these directed graphs G_i for $i = 3, \ldots, 8$, with results shown in Table 2. We consider the shallow water system Riemann problem (11), subject to the algebraic coupling conditions (6) and (7). The Riemann data is chosen to be

$$\overline{u}_{e_i} = (1, 0), \quad i = 2, \ldots, n$$

with $\overline{u}_{e_1}$ taking a uniform 1000×1000 grid of values in the block $[1, 2] \times [-3, 3]$, for a total of 1e6 separate Riemann problems to solve. This range of values better informs the relative performance of the two solvers in a generic setting.

The two solvers were implemented as NetRiemannProblem objects, see Section 4.1, and the time to solve all 1e6 Riemann problems was logged, ignoring any setup time for the NetRiemannProblem. The default direct LU factorization solver was used for all linear system solves. The same NetRiemannProblem was reused for all 1e6 solves, so solver objects were reused throughout the solves; in particular the symbolic factorization in the LU factorization solver was reused for all linear system solves, being computed only once for all 1e6 NetRiemannProblem solves. For the exact vertex Riemann Solver, the nonlinear system (12) is solved using a PETSc/SNES nonlinear solver internal to the NetRiemannProblem. Default parameters for the SNES were used from PETSc version 3.18.2 , in particular the solver was a Newton method via linesearch with a relative tolerance of 1e-8. For this application it is a finer tolerance than is truly required, because we wished to show the relative performance one could expect with no tuning of any solver parameters.

Table 2 shows the timing results for solving all 1e6 Riemann problems with the two types of NetRiemannSolvers. The simulations were run on a Macbook Pro with a 2.9 GHz Dual-Core Intel Core i5 and 8 GB 2133 MHz LPDDR3 memory. The table demonstrates that, at least with the default SNES solver, the exact NetRiemannSolver takes roughly 10 times as long as the Linearized NetRiemannSolver for the this range of vertex Riemann problems. For most values of $\overline{u}_{e_1}$ the SNES solver took 4-5 iterations to converge, with the overhead for the line search accounts for the time difference.

Table 2: Execution time (seconds) for the Exact nonlinear vs Linearized NetRiemannProblem detailed in Section 5.3.

Number of Edges	Exact nonlinear NetRiemannSolver	Linearized NetRiemannSolver
3	38.90	3.25
4	40.20	3.44
5	39.60	3.75
6	40.60	3.94
7	40.70	4.24
8	44.00	4.57

This significant time difference can motivate the usage of the linearized method even when an exact method is available, as long as the $\overline{u}_{e_i}$ states are sufficiently close to afford the linearization error.

5.3.1 A Comment on Accuracy. While a complete accuracy analysis of the linearized method is beyond the scope of this paper and will be explored in subsequent work, we provide an intuitive sketch of the behavior of the linearized method for small jumps. Let $\overline{u}_{e_i}$ constitute Riemann data for a vertex v such that the resulting Riemann problem (12) has exact solution $u^*_{e_i}$ with a "small" jump, measured by $\sum_{i=1}^{3} \|\overline{u}_{e_i} - u^*_{e_i}\|_2$. Noting that the linearized wave-curves are a 2nd order approximation to the exact-wave curves [17] for sufficiently small jumps, we expect this to carry over to the Linearized Riemann solution, $u^{Lin}_{e_i}$.

We test this by again considering the Vertex Riemann Problem on the G_3 graph, see Figure 9, for the shallow water system (11), subject to the algebraic coupling conditions (6) and (7). We sample the Riemann data $\overline{u}_{e_i}$ independently for each edge $i = 1, 2, 3$ on a uniform 8×8 grid of values in the block $[1, 1.1] \times [-0.15, 0.15]$ for a total of 8^6 Riemann Problems. The exact Riemann solutions $u^*_{e_i}$ were computed and compared with the Linearized Vertex Riemann Solver solutions $u^{Lin}_{e_i}$ by plotting their difference $\sum_{i=1}^{3} \|u^*_{e_i} - u^{Lin}_{e_i}\|_2$ against the jump in solution $\sum_{i=1}^{3} \|\overline{u}_{e_i} - u^*_{e_i}\|_2$ in Figure 10. From Figure 10 we can see the roughly 2nd order approximation of the Linearized Vertex Riemann solver for "small" jumps. Whether a specific application can tolerate this error would need to be judged on a case by case basis, and for a fully robust usage require some form of a posteriori error estimation.

We would also like to note that most of the vertices in our DG discretization of a shallow system network problem will naturally have near continuous water heights for their Riemann problems, unless a shock wave is passing through that vertex, and thus the linearized method should be sufficient for most vertices of a network. This can be seen in the convergence test of Section 5.1, where the Linearized Vertex Riemann Solver did not impact the order of convergence.

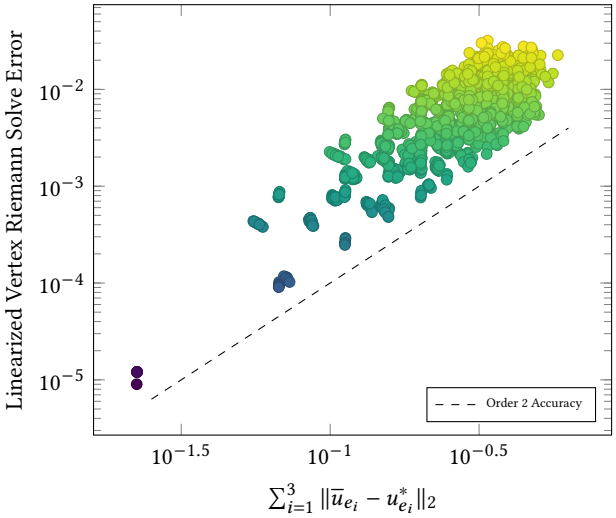

Figure 10: Error in the Linearized Vertex Riemann Solver, computed as $\sum_{i=1}^{3} \|u^*_{e_i} - u^{Lin}_{e_i}\|_2$, for the example presented in Section 5.3.1.

5.4 Scalable Simulation of Mississippi River Network

We test the strong scaling of our Riemann Solvers by simulating river flow in the Mississippi river basin [8], which is the largest river in North America. The Mississippi river system consists of one-eighth of the total river segments in the conterminous United States [19]. The data for the river network and the physical properties for each river segment (e.g., length, width, slope, flow depth and velocity) are obtained from the NHDPlus dataset [23]. This river network contains 892,740 edges and 872,300 vertices, and edges are discretized with cells at most length 10 meters and a P^2 basis for the DG method, resulting in 1,034,349,240 degrees of freedom to evolve in time by a 5-stage SSPRK2 method [21]. We again use the dam break initial conditions (18) for simplicity of the setup. We present the results using the exact NetRiemannSolver as the scaling results are similar for the linearized NetRiemann Solver. Numerical simulation is conducted on the Theta supercomputer at Argonne Leadership Computing Facility [4]. Theta has 4,392 nodes, each with 64 1.3GHz Intel Xeon Phi 7230 cores with 16 GiB of MCDRAM per node.

Table 3: Execution times (seconds) per function call in 5-stage SSPRK2 for the Mississippi River Network Simulation. See Figure 11 for an explanation of the terms.

Number of Cores	Edge DG	NetRiemann Solver	DG Limiter	Time Step	RHS Eval
256	5.87	2.03	4.15	49.00	6.63
512	2.94	1.02	2.08	24.40	3.34
1,024	1.45	0.51	1.03	12.10	1.66
2,048	0.73	0.26	0.52	6.08	0.83
4,096	0.37	0.15	0.27	3.14	0.43
8,192	0.18	$7.42 \cdot 10^{-2}$	0.13	1.55	0.22

In Table 3 a fine breakdown of the strong-scaling result is presented. A plot of these results is presented in Figure 11. This plot demonstrates a near perfect strong-scaling for the various pieces

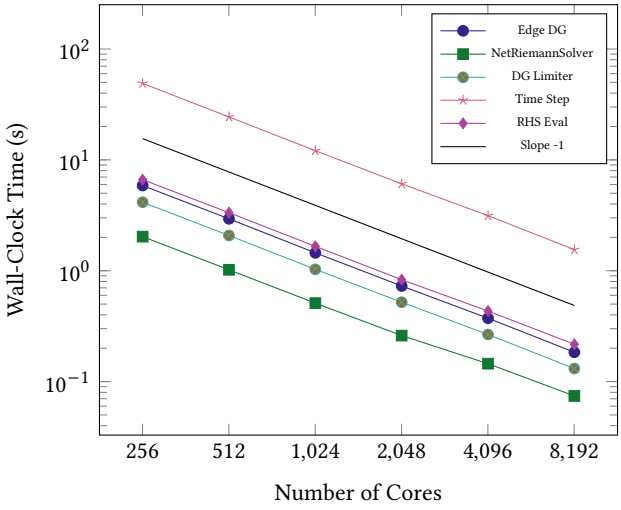

Figure 11: Strong scaling result per function call in 5- stage SSPRK2 for Mississippi River Network Simulation. Edge DG is the time to perform the integral on the RHS of (15) assuming boundary fluxes are available, NetRiemannSolver is the time to call NetRiemannSolverSolve() including communication time, DG Limiter is a TVB limiter called after each stage of the time integrator, RHS Eval is the time for the full DG RHS evaluation detailed in algorithm 1. Time Step is the total time for a time step, roughly 5*(RHS Eval+DG Limiter).

of the application, detailed in Table 3. In particular, the NetRiemannSolver class demonstrates strong-scaling for this problem even though it is the only subroutine of Time Step that performs communication.

Note that the scaling result from this study is satisfactory compared with the Routing Application for Parallel Computation of Discharge model [15] that scales up to 16 cores for the simulation of the upper Mississippi river using different physics. We achieved this scaling result through:

(1) Usage of DMNetwork to manage the network topology, and thus efficiently using ParMETIS [20] to distribute the network for load balancing.
(2) Using DG, a discretization method with a small stencil, requiring minimal communication.
(3) Interlacing communication and computation in NetRiemannSolver.
(4) Using a explicit time integrator, SSPRK2, which does not require any nonlinear or linear solves.

Furthermore, due to small stencil of DG in (2), communication occurs only at the shared vertices of the network subdomains, when computing the boundary fluxes with NetRiemannSolver. Thus communication costs only scale with the sizes of the Network Riemann Problems on the shared vertices of the network subdomain, which is proportional to the degrees of the shared vertices. This cost is hidden by interlacing with computation (3), and thus with the above strategies (1)-(4), we anticipate satisfying weak scaling results with the number of edges and vertices of networks.

The total execution time of the simulation includes (a) reading river network data from files; (b) setup and parallel distribution

of DMNetwork; and (c) repeated time steps to evolve the physics. The first two phases had a fixed cost, for which we did a sequential reading of data, setting up a sequential network, and then using ParMETIS to distribute it to a parallel network for the rest of the simulation. While this setup phase contributes to the execution of a full simulation, it is dominated by the actual simulation time (c), and is not the focus of this paper.

6 CONCLUSION AND ON-GOING WORK

This article introduced the progress of developing a scalable solver for generic hyperbolic network simulations, using the shallow water system as an example. We developed a simplified linearized vertex Riemann solver, to remove the onerous requirement to construct exact vertex Riemann solvers. Two classes, *NetRiemannProblem* and *NetRiemannSolver*, were built on the top of PETSc [5–7]. NetRiemannProblem abstracts the solution of individual vertex Riemann problems, and NetRiemannSolver solves a collection of NetRiemannProblems on a distributed network. Both classes were implemented using DMNetwork[2] to build scalable Riemann solvers with DG method for hyperbolic network applications. The scalablity of which was demonstrated on 892,740 edges and 872,300 vertex shallow water system network, which achieved perfect scalability up to 8,192 cores. This is, to the authors knowledge, the first instance of a scalable simulation of hyperbolic networks of this massive size.

The development of the package is an ongoing effort, and the interface and implementation details for NetRiemannSolver and NetRiemannProblem are continuously improving. As the development of these classes stabilizes it is hoped to be included in PETSc directly for general use. Currently we are experimenting with the parallel simulation of other hyperbolic networks such as network traffic flow [16]. Visualization of the dynamic solution of hyperbolic networks, such as seen in Figure 8 is an ongoing effort. It is the hope of the authors to provide a set of tools and software support that allow for scalable simulations of hyperbolic networks, that has a wide range of application in science and engineering.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. Aidan Hamilton and Hong Zhang were supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract DE-AC02-06CH11357 and by the Exascale Computing Project 17-SC-20-SC, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Jingmei Qiu was supported by the Air Force Office of Scientific Research under grant FA9550-22-1-0390, the National Science Foundation under grant NSF-DMS-2111253, and the Department of Energy under grant DE-SC0023164.

REFERENCES

[1] . GLVis: OpenGL Finite Element Visualization Tool. glvis.org. https://doi.org/10.11578/dc.20171025.1249

[2] S. Abhyankar, G. Betrie, D.A. Maldonado, L.C. McInnes, B. Smith, and H. Zhang. 2020. PETSc DMNetwork: A Library for Scalable Network PDE-Based Multiphysics Simulations. *ACM Trans. Math. Software* 46, 1 (2020). https://doi.org/10.1145/3344587

[3] Shrirang Abhyankar, Jed Brown, Emil M. Constantinescu, Debojyoti Ghosh, Barry F. Smith, and Hong Zhang. 2018. *PETSc/TS: A Modern Scalable ODE/DAE Solver Library.* Technical Report. arXiv:1806.01437

[4] ALCF. 2018. Theta supercomputer. https://www.alcf.anl.gov/theta.

[5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. 2022. *PETSc/TAO Users Manual.* Technical Report ANL-21/39 - Revision 3.18. Argonne National Laboratory.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. 2022. PETSc Web page. https://petsc.org/. https://petsc.org/

[7] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. 1997. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen (Eds.). Birkhäuser Press, 163–202.

[8] Getnet Betrie, Hong Zhang, Barry Simith, and Eugene Yan. 2018. A scalable river network simulator for extereme scale computers using the PETSc library. In *AGU Fall Meeting*.

[9] Alberto Bressan, Sunčica Čanić, Mauro Garavello, Michael Herty, and Benedetto Piccoli. 2014. Flows on networks: recent results and perspectives. *EMS Surveys in Mathematical Sciences* 1 (2014). Issue 1. https://doi.org/10.4171/EMSS/2

[10] Maya Briani and Benedetto Piccoli. 2018. Fluvial to torrential phase transition in open canals. *Networks and Heterogeneous Media* 13, 4 (2018), 663–690. https://doi.org/10.3934/nhm.2018030

[11] Maya Briani, Benedetto Piccoli, and Jing-Mei Qiu. 2016. Notes on RKDG Methods for Shallow-Water Equations in Canal Networks. *J. Sci. Comput.* 68, 3 (sep 2016), 1101–1123. https://doi.org/10.1007/s10915-016-0172-2

[12] Bernardo Cockburn and Chi-Wang Shu. 1989. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. *Computational Mathematics* (1989).

[13] Bernardo Cockburn and Chi-Wang Shu. 2001. Runge–Kutta Discontinuous Galerkin Methods for Convection-Dominated Problems. *Journal of Scientific Computing* 16, 3 (2001), 173–261. https://doi.org/10.1023/A:1012873910884

[14] R. M. Colombo, M. Herty, and V. Sachers. 2008. On 2×2 Conservation Laws at a Junction. *SIAM Journal on Mathematical Analysis* 40, 2 (2008), 605–622. https://doi.org/10.1137/070690298 arXiv:https://doi.org/10.1137/070690298

[15] Cédric H David, David R Maidment, Guo-Yue Niu, Zong-Liang Yang, Florence Habets, and Victor Eijkhout. 2011. River network routing on the NHDPlus dataset. *Journal of Hydrometeorology* 12, 5 (2011), 913–934.

[16] M. Garavello and B. Piccoli. 2006. *Traffic Flow on Networks: Conservation Laws Models.* American Institute of Mathematical Sciences. https://books.google.com/books?id=LVwYAwAACAAJ

[17] Edwige Godlewski and Pierre-Arnaud Raviart. 2014. *Numerical Approximation of Hyperbolic Systems of Conservation Laws.* Springer Publishing Company, Incorporated.

[18] Sigal Gottlieb, David I. Ketcheson, and Chi-Wang Shu. 2009. High Order Strong Stability Preserving Time Discretizations. *Journal of Scientific Computing* 38, 3 (2009), 251–289. https://doi.org/10.1007/s10915-008-9239-z

[19] John C Kammerer. 1987. *Largest rivers in the United States (water fact sheet).* Technical Report. U.S. Geological Survey,.

[20] G. Karypis and V. Kumar. 1995. *METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0.* Technical Report. University of Minnesota, Department of Computer Science, Minnesota.

[21] David I. Ketcheson. 2008. Highly Efficient Strong Stability-Preserving Runge–Kutta Methods with Low-Storage Implementations. *SIAM Journal on Scientific Computing* 30, 4 (2008), 2113–2136. https://doi.org/10.1137/07070485X arXiv:https://doi.org/10.1137/07070485X

[22] R.J. LeVeque and D.G. Crighton. 2002. *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press. https://books.google.com/books?id=QazcnD7GUoUC

[23] L McKay, T Bondelid, T Dewald, J Johnston, R Moore, and A Rea. 2012. NHDPlus Version 2: user guide. *National Operational Hydrologic Remote Sensing Center, Washington, DC* (2012).

[24] Eleuterio F Toro. 2013. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction.* Springer Science & Business Media.