Comput. Methods Appl. Mech. Engrg. 259 (2013) 154–165

Contents lists available at SciVerse ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

A parallel local timestepping Runge–Kutta discontinuous Galerkin method with applications to coastal ocean modeling

Clint Dawson[a,*], Corey Jason Trahan[b], Ethan J. Kubatko[c], Joannes J. Westerink[d]

[a] The Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX 78712, United States
[b] The Engineer Research and Development Center (ERDC), Coastal and Hydraulics Laboratory, Vicksburg, MS 39180, United States
[c] Department of Civil, Environmental and Geodetic Engineering, The Ohio State University, Columbus, OH 43210, United States
[d] Department of Civil and Environmental Engineering and Earth Sciences, The University of Notre Dame, Notre Dame, IN 46556, United States

ARTICLE INFO

Article history:
Received 18 September 2012
Received in revised form 22 March 2013
Accepted 24 March 2013
Available online 4 April 2013

Keywords:
Local timestepping
Multirate methods
Shallow water equations
Runge–Kutta discontinuous Galerkin methods
Tidal flows
Hurricane storm surge

ABSTRACT

Geophysical flows over complex domains often encompass both coarse and highly resolved regions. Approximating these flows using shock-capturing methods with explicit timestepping gives rise to a Courant–Friedrichs–Lewy (CFL) timestep constraint. This approach can result in small global timesteps often dictated by flows in small regions, vastly increasing computational effort over the whole domain. One approach for coping with this problem is to use locally varying timesteps. In previous work, we formulated a local timestepping (LTS) method within a Runge–Kutta discontinuous Galerkin framework and demonstrated the accuracy and efficiency of this method on serial machines for relatively small-scale shallow water applications. For more realistic models involving large domains and highly complex physics, the LTS method must be parallelized for multi-core parallel computers. Furthermore, additional physics such as strong wind forcing can effect the choice of local timesteps. In this paper, we describe a parallel LTS method, parallelized using domain decomposition and MPI. We demonstrate the method on tidal flows and hurricane storm surge applications in the coastal regions of the Western North Atlantic Ocean.

1. Introduction

It is well-known that for explicit time discretizations of conservation laws, the timestep must satisfy a CFL condition for numerical stability. From a global perspective, the timestep calculated from the CFL constraint is governed by the size of the smallest element and the eigenvalues of the system. In many situations, for example when the grid is unstructured or the physics is highly localized, the element sizes and eigenvalues may vary significantly over the domain, resulting in inefficiencies in regions where the local CFL timestep could be much larger than the global CFL timestep.

One way to deal with this problems is to use local timestepping (LTS), where the step size varies on each element and is dependent on a local CFL condition. Such methods have been previously derived and applied to general conservation laws by a number of authors [1–7] and to a variety of applications; see for example [8–14]. This procedure is also similar to multirate methods and adaptive mesh refinement (AMR) methods. The AMR method used in the GeoClaw software [15,16] uses forward Euler timestepping with timesteps dictated by local CFL constraints on each refinement patch. The fluxes at the interfaces between levels are conserved in the same way described here, and as described in [1]. The multirate methods described in [6] for conservation laws are shown to preserve second order accuracy and the TVD property.

In previous work [8], we developed and applied an LTS method within the framework of a second order Runge–Kutta discontinuous Galerkin (RKDG) method, and applied the method to the solution of the shallow water equations. The accuracy and stability of the method was examined and comparisons with RKDG solutions with no LTS were given for some relatively small-scale model problems, with all test cases executed on serial computers. The shallow water equations (SWE) are a set of hyperbolic partial differential equations (under the assumption of inviscid flow) which describe the circulation of an incompressible fluid where the water depth is much smaller than the horizontal wavelength. The SWE are used to study tides, storm surges and dam breaks, among other applications. In these problems, complex geometries such as irregular shorelines, channels, inlets, and regions with highly varying bathymetry, must be resolved to accurately capture the flow. Therefore, shock-capturing methods based on unstructured finite element discretizations, such as the discontinuous Galerkin (DG) method, are often applied to the SWE. DG methods are capable of incorporating special numerical fluxes and stability

* Corresponding author.
E-mail addresses: clint@ices.utexas.edu (C. Dawson), ctrahan@drc.com (C.J. Trahan), kubatko.3@osu.edu (E.J. Kubatko), jjw@nd.edu (J.J. Westerink).

0045-7825/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cma.2013.03.015

post-processing into the solution to model highly advective flows without excessive oscillations. Additionally, DG methods are highly parallel and allow for locally varying polynomial orders. Extensive previous work describing the development and application of DG methods to shallow water systems by the authors and collaborators can be found in [17–22].

In this paper, we study the LTS method described in [8] for some large-scale coastal flow applications. These applications require large domains, highly unstructured meshes with hundreds of thousands to millions of elements, and can involve simulation of complex phenomena over several days. Thus, efficient simulation requires the use of parallel computing. The extension of LTS methodologies to distributed memory parallel computers is nontrivial, and we describe one approach which has proven effective for two SWE applications with complex physics, namely modeling tidal flows in the Western North Atlantic ocean, and modeling coastal inundation due to hurricane storm surge in the Gulf of Mexico. This approach builds upon a fully parallel RKDG shallow water solver developed by the authors and several collaborators [18,19], where the parallelization is achieved through domain decomposition and MPI.

The rest of this paper is arranged as follows. In the next section, we outline the RKDG method and discuss the implementation of the LTS method in a parallel computing environment. In Section 3, we discuss the shallow water model, and study the LTS method in the context of the two applications mentioned above. In particular, we investigate the overall efficiency of LTS in parallel vs. standard global timestepping, and how well LTS performs within a complex coastal modeling system.

2. Numerical methods

2.1. The discontinuous Galerkin finite element method

In this section we briefly outline the DG method. Consider the hyperbolic equation,

$$\frac{\partial w}{\partial t} + \nabla \cdot \mathbf{F}(w) = s. \tag{1}$$

To formulate the semi-discrete DG method for (1), the physical domain, Ω, is first partitioned into non-overlapping finite elements, K_i for $i = 1, 2, \ldots, N$. If $P^k(K_i)$ is defined as the space of polynomials of degree $\leqslant k$ over element i, the DG method can be formulated as seeking a piecewise smooth function $w_h|_{K_i} \in P^k(K_i)$ which $\forall i$ satisfies:

$$\int_{K_i} \frac{\partial w_h}{\partial t}\, v_h\, dx - \int_{K_i} \mathbf{F}(w_h) \cdot \nabla v_h\, dx + \int_{\partial K_i} \widehat{\mathbf{F}} \cdot \mathbf{n}_i v_h\, ds = \int_{K_i} s v_h\, dx, \tag{2}$$

where $v_h|_{K_i} \in P^k(K_i)$ is the test function, and $\hat{\mathbf{F}}$ is an approximation to the normal flux at the element boundaries. Here $\mathbf{n}_i$ is the unit outward normal to ∂K_i.

Eq. (2) is obtained by multiplying the original equation by a test function, integrating over each element and integrating the divergence term by parts. The numerical flux is required since the discrete solutions allow for discontinuities between elements. For nonlinear equations, an approximate Riemann solver is used to define the numerical flux along element boundaries. Given a face between two elements, we label the elements on each side of this face K_- and K_+, and let $\mathbf{n}$ be the normal vector to the face which points from K_- to K_+, then

$$\hat{\mathbf{F}} \approx \mathbf{Fn}$$

and $\hat{\mathbf{F}}$ depends on the solution on either side of the face, that is,

$$\hat{\mathbf{F}} = \hat{\mathbf{F}}(w_{h,-}, w_{h,+}),$$

where $w_{h,-} = w_h|_{K_-}$ with similar definition for $w_{h,+}$. For elements on the boundary, the normal $\mathbf{n}$ is assumed to be the outward normal to the boundary of the domain, and $w_{h,+}$ is chosen to enforce external boundary conditions; see [17]. Many different numerical fluxes $\hat{\mathbf{F}}$ have been proposed in the literature. For the results presented below, the local Lax–Friedrichs flux is used.

The discrete solution and test functions are then expanded on element K_i with s degrees of freedom:

$$w_h|_{K_i} = \sum_{j=1}^{s} \tilde{w}_{j,i}(t)\, \phi_j(x,y), \quad v_h|_{K_i} = \sum_{k=1}^{s} \tilde{v}_{k,i}(t)\, \phi_k(x,y), \tag{3}$$

where $\{\tilde{w}, \tilde{v}\}$ are the basis function degrees of freedom, and $\{\phi_k\}$ are the basis functions. Using (3) and (2), Eq. (1) can be written as system of ODEs

$$\mathbf{M}\frac{d\tilde{\mathbf{w}}}{dt} = \mathbf{b}, \tag{4}$$

where $M_{j,k} = \int_{K_i} \phi_j \phi_k\, dx$ is the mass matrix and

$$\tilde{\mathbf{w}} = [\tilde{w}_{1,1}, \tilde{w}_{2,1}, \ldots, \tilde{w}_{s,1}, \tilde{w}_{1,2}, \ldots, \tilde{w}_{s,N}]^T, \tag{5}$$

$$\mathbf{b} = [R_1(\phi_1), R_1(\phi_2), \ldots, R_1(\phi_s), R_2(\phi_1), \ldots, R_N(\phi_s)]^T, \tag{6}$$

with,

$$R_j(\phi_i) = \int_{K_j} (\mathbf{F}(w_h) \cdot \nabla\phi_i + s\phi_i)\, dx - \int_{\partial K_j} \hat{\mathbf{F}} \cdot \mathbf{n}_j \phi_i\, ds. \tag{7}$$

2.2. Runge–Kutta time discretization

For time integration, the system of equations

$$\frac{d\tilde{\mathbf{w}}}{dt} = L_h(\tilde{\mathbf{w}}) \equiv \mathbf{M}^{-1}\, \mathbf{b} \tag{8}$$

is discretized in time using an explicit, strong stability preserving (SSP) Runge–Kutta scheme. These methods were originally referred to as total variation diminishing methods and were introduced by Shu and Osher (see Refs. [23,24]). For linear basis functions in space, generally a second order SSP Runge–Kutta scheme is used. Given a timestep Δt, and $t^n = n\Delta t,\ n = 0, 1, \ldots$, the method is defined as

$$\begin{aligned} \tilde{\mathbf{w}}^0 &= \tilde{\mathbf{w}}(t^n), \\ \tilde{\mathbf{w}}^i &= \tilde{\mathbf{w}}^{i-1} + \Delta t L_h(\tilde{\mathbf{w}}^{i-1}), \quad \text{for } i = 1, 2, \\ \tilde{\mathbf{w}}(t^{n+1}) &= \frac{1}{2}(\tilde{\mathbf{w}}^0 + \tilde{\mathbf{w}}^2). \end{aligned} \tag{9}$$

Thus, the method consists of taking two forward Euler steps, and averaging the final result with the solution at the previous timestep. This method is also known as Heun's method.

2.3. Slope limiting and wetting and drying

Other aspects of the DG implementation, such as slope limiting and wetting and drying, which are more specific to the shallow water application, are described in [8] and the references therein. Therefore we will not repeat them here except to say that in the numerical results below, we use a vertex-based slope limiter (the Bell–Dawson–Shubin limiter) as described in [25,26], and the wetting and drying algorithm described in [21] is used.

2.4. Local timestepping (LTS)

While the RKDG method described above allows for any polynomial order approximating space and higher order timestepping, we will restrict our attention in the remainder of this paper to piecewise linear approximations and second-order SSP

Runge–Kutta timestepping. The LTS method that we employ is described in [8], however we review it again here for completeness. It is based on a simple modification of the second order SSP Runge–Kutta method described above, to allow for different timesteps in different regions, and to conserve mass.

We also remark that the LTS scheme discussed here follows in spirit the method originally described and analyzed in [3] for the one-dimensional conservation law $u_t + f(u)_x = 0$. In that work, a local timestepping method was derived from an RKDG scheme, with piecewise linear approximations in space and Heun's method in time. It allowed for interfaces separating elements with timesteps differing by a factor M, with local CFL constraints imposed on each element. The method was shown to satisfy a strict maximum principle, and is hence stable, with a suitable slope-limiter applied to the linear component of the numerical solution. While the stability proof does not extend to the shallow water equations defined over very general domains and discretized on highly unstructured triangular meshes, the analysis in [3] does suggest that the approach described here could have similar stability properties. Our numerical tests to date suggest that this is in fact the case.

To describe the LTS scheme, we consider decomposing an example domain, Ω into two zones, Ω_1 and Ω_2, see Fig. 1. These zones are separated by a one-dimensional interface σ, indicated by the dashed line. Each zone consists of some number of elements. In Fig. 1 Ω_2 has four rectangular elements, and Ω_1 has eight rectangular elements which are half the size of the elements in Ω_2, which would lead to a more severe CFL timestep constraint in Ω_1 than in Ω_2. Each zone is assumed to have its own timestep, $\{\Delta T_1, \Delta T_2\}$, assigned so that (1) each satisfy its subdomain CFL constraint and (2) $\Delta T_2 = M\Delta T_1$, for some positive integer M.

Assume the solution is given over the whole domain at time t^n. In order to propagate the solution to time $t^{n+1} = t^n + \Delta T_2$, M subtimesteps within Ω_1 are taken. To evaluate numerical fluxes at the interface σ, the Ω_2 states are fixed at time t^n. Subsequently, Ω_2 is updated to t^{n+1} by taking one timestep. To calculate the fluxes at σ as seen from Ω_2, an average over the M Ω_1 fluxes is used.

More precisely, suppose K_1 and K_2 are two neighboring elements whose common boundary $\partial K_1 \cap \partial K_2$ intersects the local timestepping interface σ. Assume K_1 is in the subdomain Ω_1 and K_2 is in Ω_2. Suppose that the normal on $\partial K_1 \cap \partial K_2$ points from K_1 to K_2; i.e., K_1 is K_- and $K_2 = K_+$ in the notation above. Let $w_h^0 = w_h(t^n)$ and $w_h^{0,l} = w_h(t^n + l\Delta T_1)$, for $l = 0, \dots, M$.

Then combining (9) and (2), the RKDG method on K_1 at time $t^{n,l} \equiv t^n + l\Delta T_1$ is defined by

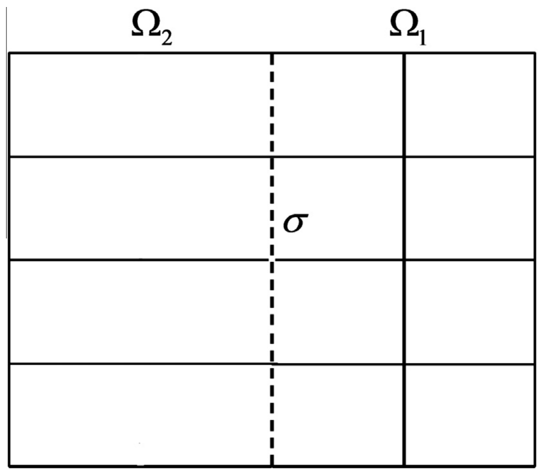


Fig. 1. Division of Ω into two zones with different timesteps.

$$\begin{aligned}\int_{K_1} w_h^{i,l}\, v_h\, dx &= \int_{K_1} w_h^{i-1,l}\, v_h\, dx + \Delta T_1 \int_{K_1} \mathbf{F}(w_h^{i-1,l}) \cdot \nabla v_h\, dx \\ &\quad - \Delta T_1 \int_{\partial K_1/\sigma} \widehat{\mathbf{F}}(w_{h,-}^{i-1,l}, w_{h,+}^{i-1,l}) \cdot \mathbf{n}_1 v_h\, ds - \Delta T_1 \\ &\quad \times \int_{\partial K_1 \cap \partial K_2} \widehat{\mathbf{F}}(w_{h,-}^{i-1,l}, w_{h,+}^{0}) \cdot \mathbf{n}_1 v_h\, ds + \Delta T_1 \\ &\quad \times \int_{K_1} s v_h\, dx \end{aligned} \tag{10}$$

for $i = 1, 2$. Here we have split the boundary ∂K_1 into the part which does not intersect σ, i.e., $\partial K_1/\sigma$, and the part which does intersect σ, i.e., $\partial K_1 \cap \partial K_2$. We are assuming that on $\partial K_1/\sigma$ no local timestepping is occurring. At each intermediate timestep $t^{n,l+1}$ we set

$$w_h^{0,l+1} = \frac{1}{2}(w_h^{0,l} + w_h^{2,l}) \tag{11}$$

for $l = 0, \dots, M-1$. Then on K_2, the RKDG method is defined by

$$\begin{aligned}\int_{K_2} w_h^{i}\, v_h\, dx &= \int_{K_2} w_h^{i-1}\, v_h\, dx + \Delta T_2 \int_{K_2} \mathbf{F}(w_h^{i-1}) \cdot \nabla v_h\, dx \\ &\quad - \Delta T_2 \int_{\partial K_2/\sigma} \widehat{\mathbf{F}}(w_{h,-}^{i-1}, w_{h,+}^{i-1}) \cdot \mathbf{n}_2 v_h\, ds - \sum_{l=0}^{M-1} \Delta T_1 \\ &\quad \times \int_{\partial K_1 \cap \partial K_2} \widehat{\mathbf{F}}(w_{h,-}^{i-1,l}, w_{h,+}^{0}) \cdot \mathbf{n}_2 v_h\, ds + \Delta T_2 \\ &\quad \times \int_{K_2} s v_h\, dx \end{aligned} \tag{12}$$

for $i = 1, 2$. Finally on K_2

$$w_h(t^{n+1}) = \frac{1}{2}(w_h^0 + w_h^2). \tag{13}$$

Combining (10)–(13), setting $v_h = 1$ on K_1 and K_2, and noting that $\mathbf{n}_1 = -\mathbf{n}_2$ on $\partial K_1 \cap \partial K_2$,

$$\int_{K_1 \cup K_2} w_h(t^{n+1}) dx = \int_{K_1 \cup K_2} [w_h(t^n) + S] dx$$

where S is an approximation to the time integral of the source/sink terms in the model over the time interval $[t^n, t^{n+1}]$. Thus, in this sense, mass and momentum are conserved in the region of the LTS interface.

2.5. Implementation and parallelization

The LTS method described above allows for a fairly general timestepping approach with one assumption, that neighboring elements are assumed to have timesteps which differ by some integer M. In order for the coding of the LTS method not to be overly cumbersome, we assume that each element K in the discretization of Ω is placed into a timestepping group or level. Level 1 will denote the elements with the smallest timestep, level 2 the next smallest, and so forth. We will denote the total number of levels by $\bar{N}$. For simplicity we will also assume that M is constant from one level to the next.

Elements are sorted into levels by first calculating a local CFL timestep. On each element K, we compute a local timestep

$$\Delta t_K = \alpha \frac{\bar{h}_K}{\lambda_K} \tag{14}$$

where α is a CFL parameter which is $\mathcal{O}(1)$, typically $\alpha = 1/\sqrt{2}$, $\bar{h}_K$ is the minimum distance between the centroid of the element and the midpoint of the edges of K, and λ_K is an estimate of the maximum eigenvalue of the Jacobian associated with the normal flux $\hat{\mathbf{F}}$. Let ΔT_l, $l = 1, \dots, \bar{N}$ denote timesteps associated with each timestepping level, where $M\Delta T_l = \Delta T_{l+1}$. We assume that

$$\Delta T_1 \leqslant \min_K \Delta t_K.$$

Then element K is placed into timestep group l if

$$\Delta T_l \leqslant \Delta t_K < \Delta T_{l+1}. \tag{15}$$

If $\Delta t_K \geqslant \Delta T_{\bar{N}}$ then element K is placed into level $\bar{N}$. The element timesteps are then reset, thus if element K is in group l, then $\Delta t_K \leftarrow \Delta T_l$.

In [8], this approach was investigated for several shallow water applications and observed to preserve second order accuracy for a problem with an analytical solution, and it was shown to give solutions comparable to those computed using a global CFL timestep (i.e., $\bar{N} = 1$). Furthermore, on serial machines, the method was shown to be nearly optimal in terms of computational efficiency.

For large-scale applications of interest, solutions cannot be computed in serial due to memory and CPU limitations, therefore parallel computing is necessary. We have investigated the implementation of the LTS method in parallel, again implementing the LTS method in a DG shallow water model. The parallel performance of this model without LTS has been investigated in other papers, most notably in [18]. The parallelization approach is based on domain decomposition, where the domain is first decomposed using the METIS software library [27,28]. METIS divides the domain into overlapping subdomains with "ghost" regions based on a graph-partition of the nodes that make up the finite element mesh. In our implementation, the ghost region consists of elements which are shared by neighboring processors. MPI is used to pass solution information defined on the ghost elements to the neighboring processor. METIS attempts to divide the domain to balance the work-load among processors, to preserve locality of the elements and nodes within the subdomain and to minimize the "surface-to-volume" ratio; that is, to keep the ratio of ghost nodes to resident nodes low in order to reduce the communications overhead. For improved load balancing, METIS allows the user to weight nodes in the finite element mesh using an estimate of the "work" related to the node; for example, by estimating the maximum amount of work performed in elements which are attached to the node.

For a fixed global timestep, the parallelization of the DG method is quite straightforward. Each element has a fixed amount of work, takes the same timestep, and parallelization is achieved by each subdomain communicating with neighboring subdomains at the end of each Runge–Kutta timestep. The communication remains constant throughout the simulation. For LTS, the situation is much more complicated.

First, there is the question of load balancing. The amount of work per element depends on the local timestep. We have attempted to address this in METIS by weighting each node by a factor which depends on the local timesteps associated with elements attached to the node. This factor is determined by the number of sub-cycling steps required for the element with the smallest timestep to go from time t^n to time t^{n+1}. The local timestep may also change during the course of the simulation, therefore in reality the load should be dynamically re-balanced during the simulation.

Second, there are communication issues associated with LTS. For example, consider a 1-D example as in Fig. 2. We picture four elements labeled $i-2,\ i-1,\ i$ and $i+1$. There are $\bar{N} = 3$ timestepping levels with $M = 2$. Elements $i-2$ and $i-1$ are on level 1, with the smallest timestep, element i is on level 2 and $i+1$ on level 3. Now assume elements $i-2,\ i-1$ and i are on processor 0 (PE0), and $i-1,\ i$ and $i+1$ are on the neighboring processor 1 (PE1), with elements $i-1$ and i in the ghost region. Both processors PE0 and PE1 compute the solution on these two elements, but element $i-1$ is "owned" by PE0 while element i is "owned" by PE1. For the solution to be computed correctly in the ghost region, information in element $i-1$ must be passed from PE0 to PE1 at each level 1 timestep, and the information in element i must be passed from PE1 to PE0 at all level 2 timesteps.

In general, each timestepping level must communicate information with neighboring processors which share elements on the same level, if these elements are within the ghost region. Therefore, we have implemented a message-passing construct which is level-dependent. This may reduce parallel efficiency in the sense of strong scalability, since not all subdomains may have the same number of elements on each level, in fact some subdomains may have no elements on a given timestepping level. Or, subdomains may have elements within a level but none in the ghost region, while other subdomains may have many elements within a certain level in the ghost region, and thus require message-passing. One could try to address this problem by attempting to evenly divide the elements on each level among the processors, however, this approach would most likely destroy locality, and result in a large number of isolated elements on each processor.

In summary, determining an optimal parallel strategy for LTS is complicated by several factors; however, as we will see in the results section below, LTS can still lead to an efficient and accurate approach in parallel as it does in serial.

3. Applications to the SWE

The SWE are based on the three-dimensional Reynold's averaged Navier–Stokes equations for a Newtonian fluid. Averaging these equations over the vertical depth of the water H and applying kinematic and no-flow boundary conditions at the top and the bottom, gives rise to the conservative form of the SWE:

$$\frac{\partial H}{\partial t} + S_p \frac{\partial (uH)}{\partial x} + \frac{\partial (vH)}{\partial y} = 0, \tag{16}$$

$$\frac{\partial (uH)}{\partial t} + S_p \frac{\partial \left(u^2 H + \frac{1}{2} g H^2\right)}{\partial x} + \frac{\partial (uvH)}{\partial y} = g S_p H \frac{\partial \eta}{\partial x} + (\tau_x^{\xi} - \tau_x^{\eta}) + F_x, \tag{17}$$

$$\frac{\partial (vH)}{\partial t} + \frac{\partial \left(v^2 H + \frac{1}{2} g H^2\right)}{\partial y} + S_p \frac{\partial (uvH)}{\partial x} = gH \frac{\partial \eta}{\partial y} + (\tau_y^{\xi} - \tau_y^{\eta}) + F_y, \tag{18}$$

where u and v are depth-average velocities, ξ is the water elevation relative to the geoid, $\eta = H - \xi$ is the bathymetry relative to the

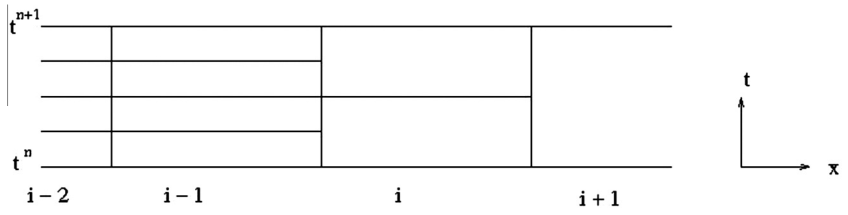


Fig. 2. Example of LTS in one space dimension with $\bar{N} = 3$ and $M = 2$.

geoid, g is gravitational acceleration, $\{\tau^{\xi}_{x,y}, \tau^{\eta}_{x,y}\}$ are the surface (wind) and bed (bottom friction) stresses, respectively, and $F_{x,y}$ accounts for other external forces,such as Coriolis force and tidal potential. The parameter S_p is a spherical correction factor which transforms the SWE in spherical coordinates $\phi,\ \lambda$ to Cartesian coordinates $x,\ y$ using an orthogonal cylindrical projection; see [19]. To arrive at these equations, a number of assumptions have been made; (1) the vertical acceleration of a fluid particle is small in comparison to the acceleration of gravity, (2) shear stresses due to the vertical velocity are small and (3) the horizontal shear terms, $\{\partial^2 u/\partial x^2, \partial^2 u/\partial y^2, \partial^2 v/\partial x^2, \partial^2 v/\partial y^2\}$ are small compared to vertical shears, $\{\partial^2 u/\partial z^2, \partial^2 v/\partial z^2\}$.

For closure, the bed stress terms must be parameterized via the depth-averaged velocities. The bed stress is often approximated by linear or quadratic functions of the velocities, however, we have used a hybrid form proposed by Westerink et al. [29] which varies the bottom-friction coefficient with the water column depth:

$$\tau^{\eta}_x = uH\left(C_f \frac{\sqrt{u^2+v^2}}{H}\right), \quad \tau^{\eta}_y = vH\left(C_f \frac{\sqrt{u^2+v^2}}{H}\right), \tag{19}$$

where,

$$C_f = C_{fmin}\left(1 + \left(\frac{H_{break}}{H}\right)^{f_\theta}\right)^{f_\gamma/f_\theta}. \tag{20}$$

This formulation applies a depth-dependent, Manning-type friction law below the break depth (H_{break}) and a standard Chezy friction law when the depth is greater than the break depth. For the applications below, C_{fmin} is allowed to vary, since the bed surfaces change.

The wind surface stress is computed by a standard quadratic drag law. Define

$$\frac{\hat{\tau}^{\xi}_x}{\rho_0} = C_d \frac{\rho_{\text{air}}}{\rho_0} |\mathbf{W}| W_x, \tag{21}$$

$$\frac{\hat{\tau}^{\xi}_y}{\rho_0} = C_d \frac{\rho_{\text{air}}}{\rho_0} |\mathbf{W}| W_y. \tag{22}$$

Here $\mathbf{W} = (W_x, W_y)$ is the wind speed sampled at a 10-m height over a 15 min time period and ρ_{air} is the air density. The drag coefficient is defined by Garratt's drag formula [30]:

$$C_d = (.75 + .06|\mathbf{W}|) * 10^{-3}. \tag{23}$$

We also remark that the wind surface stress is capped so that its magnitude is never greater than.002.

3.1. LTS in the SWE

The eigenvalues of the normal flux for the SWE are

$$\lambda_{1,2} = un_x + vn_y \pm \sqrt{gH}, \quad \lambda_3 = un_x + vn_y. \tag{24}$$

In shallow water simulations, one typically initializes the simulation by assuming a "cold-start;" i.e., water elevations are initially constant and water velocity is zero. Thus the largest eigenvalue initially is $\sqrt{gH}$, and the local timesteps are computed by

$$\Delta t_K = \alpha \frac{h_K}{\sqrt{gH_K}} \tag{25}$$

where H_K is the average water depth over the element. As the simulation progresses, the local timesteps may need to be adjusted based on the water velocity. In many cases $\sqrt{gH} \gg |un_x + vn_y|$ and the local timesteps can be fixed during the computation. For more challenging applications, for example, modeling hurricane storm surges, this is not the case. Therefore, at certain intervals during the computation, we may recompute the local timesteps by

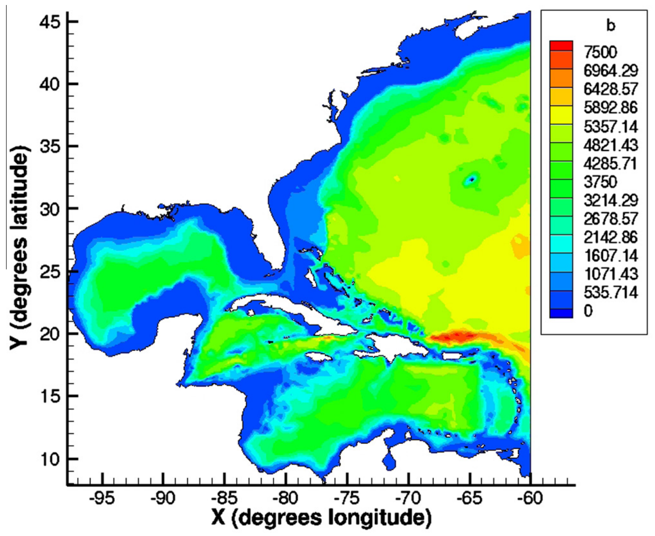


Fig. 3. Western North Atlantic Ocean domain. Tidal elevations are specified at the eastern boundary, all other boundaries are land boundaries. Contours represent bathymetry in meters relative to the National Geodetic Vertical Datum of 1988 (NAVD88).

$$\Delta t_K = \alpha \frac{h_K}{\lambda_K} \tag{26}$$

where $\lambda_K = |\mathbf{u}_K| + \sqrt{gH_K}$. Here $|\mathbf{u}_K|$ is the magnitude of the cell average of velocity over the element K. The elements are then redistributed among the levels on each processor. That is, the number of levels $\bar{N}$ and the ratio M is left fixed, but elements are allowed to move between levels, depending on Δt_K.

3.2. Tidal flows in the Western North Atlantic Ocean

The first problem we consider is that of tidal flow in the Western North Atlantic Ocean. The domain for this problem is pictured in Fig. 3 and consists of part of the Atlantic Ocean, the Caribbean Sea and the Gulf of Mexico. Tidal elevations are forced at the 60°W meridian open boundary. We utilize a standard tidal formula consisting of 7 tidal components, three diurnal (K_1, O_1, Q_1) and four semidirunal (M_2, S_2, N_2, K_2). The data can be found in [31]; see also Table 1 in [32]. We also impose tidal potential as a body force with the same 7 components. The simulation is cold-started and the tide is ramped-up using a smooth hyperbolic tangent ramp function over a 5 day time period. Other parameters in the model are:

- $C_{fmin} = .0025$
- $H_{break} = 1.0$ m
- $f_\theta = 10$
- $f_\gamma = .33333$

These parameters were obtained from [32].

The discretization of the domain into a mesh consisting of 98,635 elements and 52,774 nodes is plotted in Fig. 4.

Numerical studies comparing the RKDG SWE solution to tidal gauge data for this problem are given in [18]; there it was demonstrated that the DG method accurately reproduces measured tidal data. Here we consider various LTS scenarios and compare to RKDG solutions with no LTS. These scenarios are representative of many numerical experiments which have been performed in this study. The details of three particular LTS cases are outlined in Table 1. For example, LTS-Case 1 divides the domain into four timestepping groups or levels, with $M = 2$ between each level. The smallest timestep $\Delta t_{min} = 2$ s, thus the timesteps on each of the four levels

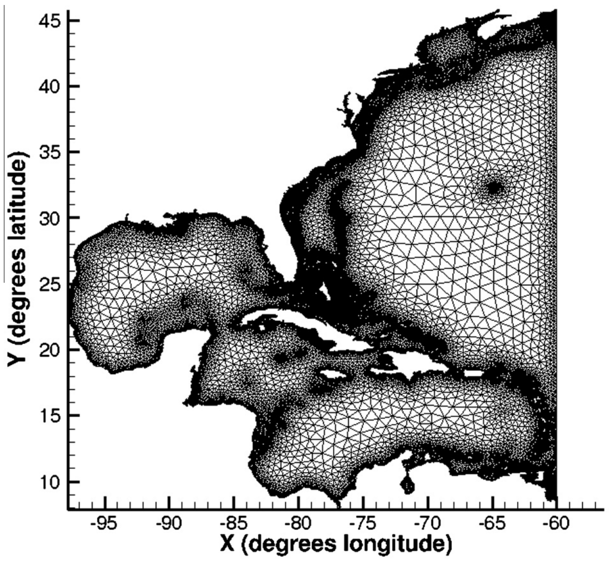


Fig. 4. Western North Atlantic mesh.

are 2, 4, 8 and 16 s, respectively. In the third column we see how many elements are initially in each timestepping group, based on the criteria discussed in Section 3.1. Note that for Case 1, the vast majority of elements take the largest timestep of 16 s. Therefore, for Case 2, we chose $\bar{N} = 7$ to allow elements to take even larger timesteps. In this case, elements can take a timestep as large as 128 s. Case 3 differs in that we take $M = 4$ and divide into 3 groups with timesteps of 2, 16 and 64 s. This distributes more elements among the latter two groups.

First, we examine whether LTS effects the accuracy of the solution. We compare the solution with no LTS to the LTS-Case 1 solution for a 10 day tidal simulation, to allow for the tide to fully ramp-up and to compute several tidal cycles. In Fig. 5, we compare the water elevation solution with no LTS with global timestep $\Delta t = 2.195$ s to the solution obtained with LTS-Case 1, at four measurement stations along the eastern coast of the US In this figure we are plotting the water elevation in meters vs. time in seconds over the 10 day simulation at specific points in the domain. These points are located near Boston, MA $(71.05^\circ W, 42.36^\circ N)$, Charleston, SC $(79.93^\circ W, 32.78^\circ N)$, Key West, FL $(81.81^\circ W, 24.55^\circ N)$, and Corpus Christi, TX $(97.22^\circ W, 27.58^\circ N)$. We note that the solutions are virtually indistinguishable, which indicates that LTS does not degrade the solution. We have examined solutions for other LTS parameters and obtained very similar results.

Next we examine the parallel efficiency of the code with and without LTS. In this problem, the local timesteps did not vary so dramatically during the course of simulation to warrant a re-partitioning of the mesh, therefore for the results presented here, the parallel partition is static. We will compare execution times for a 1 day simulation. All tests were performed on the Bevo2[1] cluster at the Institute for Computational Engineering and Sciences at The University of Texas at Austin. First, to set a benchmark, we test the parallel efficiency of the code with no LTS. In Table 2 we see that the code exhibits near perfect speed-up to 32 cores. Beyond that the parallel performance begins to degrade. Therefore, in comparing the various LTS strategies with no LTS, we will focus on runs with 8, 16 and 32 cores.

[1] Bevo2 is a 23 node compute cluster made up of Dell PowerEdge servers that house 2x quad core 2.66 GHz Intel Xeon processors for a total of 184 processors. Each node has 16 gigabytes of RAM, dual gigabit ethernet ports, and a single port Mellanox III Lx Infiniband adapter attached to a QLogic SilverStorm Infiniband 24 port switch capable of up to 20 Gb/s.

Table 1
LTS parameters for three different test cases.

Test case	$\bar{N}$	M	Δt_{min} (sec)	# Elements in each level
1	4	2	2	436, 4456, 13468, 80005
2	7	2	2	436, 4456, 13468, 19262, 21720, 26168, 12855
3	4	4	2	4892, 32730, 60743

For the LTS method, to see an example of how the elements and groups are split among processors, we show in Table 3 the distribution of elements among 8 processors (PEs) for LTS-Case 3. The number of elements per PE includes ghost elements. We also compute the total amount of "work" required on each PE to advance the solution over one time cycle from t^n to t^{n+1}. This is obtained as follows for Case 3: the number of elements on level 1 requires 16 timesteps to complete one time cycle, level 2 requires 4 timesteps, and level 3 requires 1 timestep. Therefore, on PE0, for example, the amount of work is

$$16 * 701 + 4 * 5983 + 983 = 36131$$

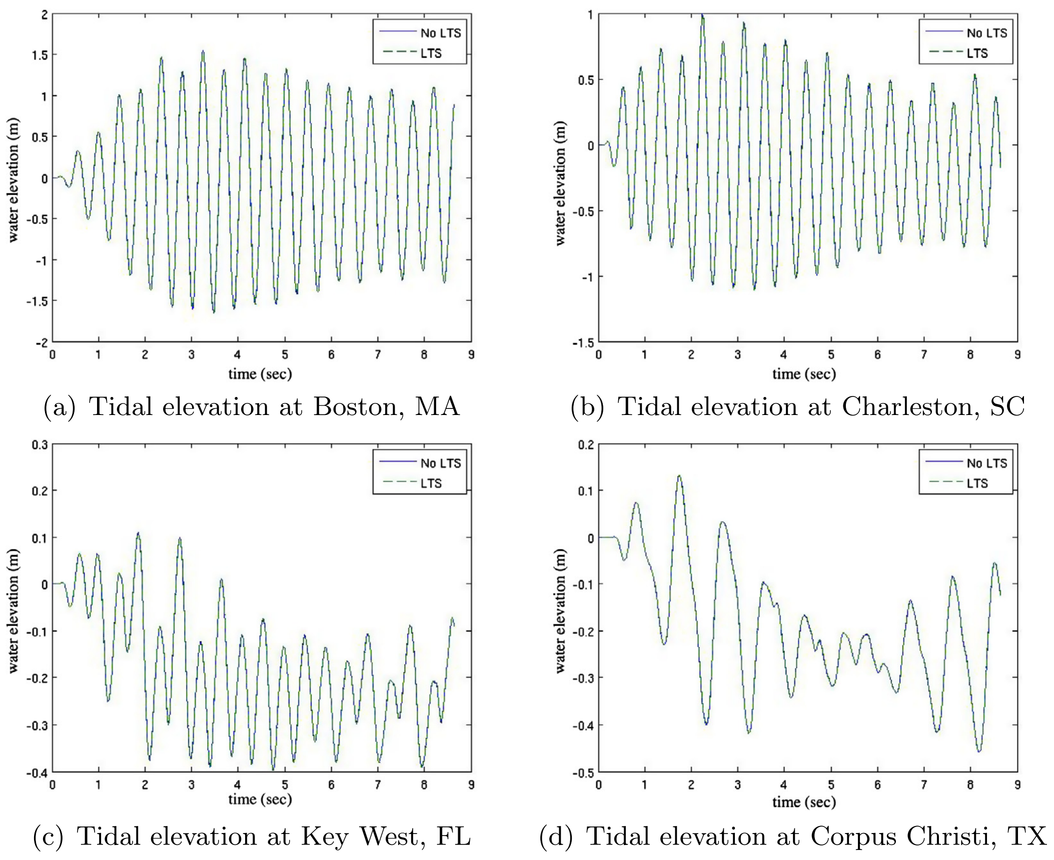
as seen in column 3 of the table. For the work to be distributed evenly among PEs this number should be roughly constant. We see in Table 3 that the work is fairly evenly distributed among the processors, with the maximum variation on the order of 10%. We also remark that during simulations the element timesteps are recomputed every 1000 timesteps, based on the formula given in Section 3.1. This allows elements to change timestepping levels during the simulation, and could effect the load balancing. However, for the three LTS cases considered, very few elements changed timestepping levels during the course of the simulations, thus the workload per PE remained essentially constant.

The parallel performance for the three LTS test cases is given in Table 4. We first note that the parallel scaling is not as good as without LTS, even though the run times are substantially reduced for every case in comparison to the results given in Table 2. The parallel efficiency for LTS drops off substantially between 16 and 32 PEs while no LTS still showed near optimal efficiency. The parallel scaling of LTS is limited by several factors. The number of elements on each level is not evenly distributed among PEs, due to locality constraints within METIS. Furthermore, even if the number of elements were evenly distributed, the computational efficiency is limited by the surface-to-volume ratio on each level on each PE. With LTS the surface-to-volume ratios could be worse on each timestepping level than without LTS, meaning there is less computation and more message passing at each level. We remark however, that even with these limitations LTS on 32 cores (in the best case) was a factor of 1.5 faster in compute time than no LTS on 64 cores, with no appreciable difference observed in the solutions.

3.3. Hurricane Ike storm surge forecast

One of the most challenging applications for coastal models is the simulation of storm surge due to hurricanes. In previous work, we have described the application of the DG method, with extensions to include wetting and drying and internal barriers such as levees, to the modeling of storm surge in the Gulf of Mexico [19]. In this section, we describe the application of LTS to a typical storm surge event. In particular, we consider Hurricane Ike, which struck the upper Texas coast in 2008.

The track of Ike is seen in Fig. 6. The storm progressed through the Western North Atlantic, through the Caribbean Sea making landfall in Cuba, and moved across the Gulf of Mexico, finally making a second landfall at Galveston, TX in the early morning of



(a) Tidal elevation at Boston, MA

(b) Tidal elevation at Charleston, SC

(c) Tidal elevation at Key West, FL

(d) Tidal elevation at Corpus Christi, TX

Fig. 5. Time history of water elevation comparing no LTS and LTS-Case 1. Units on the horizontal axis are seconds, and the vertical axis is in meters.

Table 2
Parallel performance and CPU times with no LTS for 1 day of simulation.

# Cores	CPU time (min:s)	Speedup
8	86:42	–
16	44:31	1.95
32	24:31	1.82
64	17:38	1.39

Table 3
Number of elements per PE on each timestepping level for 8 processors and the total work, computed as the total number of element-timesteps needed to advance the solution from time t^n to t^{n+1}.

PE	# Elements in each level			Total work
0	701	5983	983	36131
1	144	6124	9926	36726
2	288	3872	15652	35748
3	468	4773	8457	35037
4	617	5038	8943	38967
5	1725	1497	3589	37177
6	726	4404	8289	37521
7	1555	2865	2647	38987

Table 4
Parallel performance with LTS for 1 day of simulation.

LTS test case	Number of PEs	CPU time (min:s)	Speedup
1	8	30:43	
1	16	17:30	1.76
1	32	12:10	1.44
2	8	26:26	–
2	16	15:34	1.70
2	32	11:38	1.34
3	8	32:34	–
3	16	19:25	1.68
3	32	13:52	1.40

September 13, 2008. By this time, Ike had high category 2 winds but had an unusually large wind field and produced a category 4 storm surge in an area east of Houston, TX. In [19], we compared results computed using the RKDG method with no LTS to data taken from another model, namely the Advanced Circulation or ADCIRC code, which was used to study Hurricane Ike in [33]. In this section, we study a slightly different scenario, namely a "forecast" of Ike using approximate wind fields generated from data obtained from the National Hurricane Center, and the Holland hurricane wind/pressure model developed in [34]. In this wind model, the data given in the National Hurricane Center forecasts, namely the location of the eye of the storm, the central pressure, the radius-to-maximum winds, and the maximum sustained wind speed, are used to compute a vortex-shaped approximation of the hurricane wind and pressure field. This model is used in forecast simulations of hurricane storm surges as described in [35], for estimating surge as hurricanes approach land. Here we use the so-called "best" track data; i.e, the actual hurricane track as measured through the progression of the storm, as opposed to forecast tracks given during the event. The purpose of this exercise is to investigate the performance of the parallel DG code with LTS in this complex scenario, and compare to results generated using the no LTS, RKDG method described in [19].

The domain used in these simulations is similar to the domain used in the previous section, but with large sections of the Texas coast included; see Fig. 7. Here we include most sections of the coast which are less than 50 feet above sea level, since these

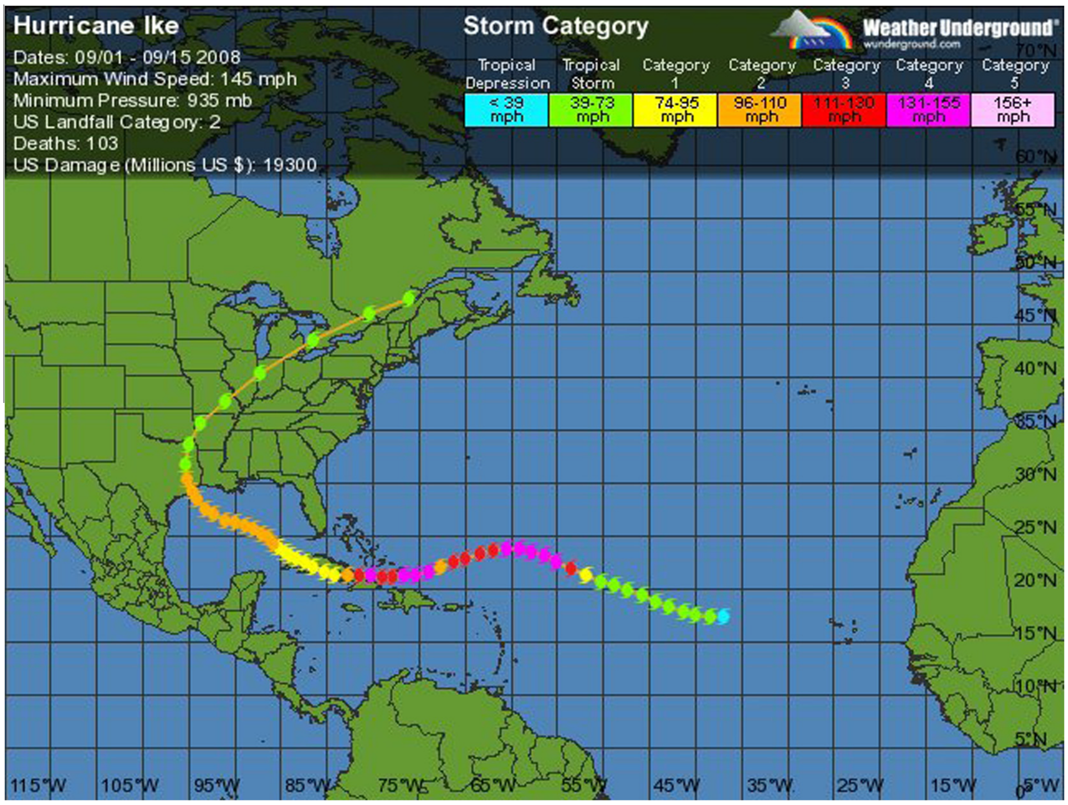

Hurricane Ike
Dates: 09/01 - 09/15 2008
Maximum Wind Speed: 145 mph
Minimum Pressure: 935 mb
US Landfall Category: 2
Deaths: 103
US Damage (Millions US $): 19300
Storm Category
Tropical Depression < 39 mph
Tropical Storm 39-73 mph
Category 1 74-95 mph
Category 2 96-110 mph
Category 3 111-130 mph
Category 4 131-155 mph
Category 5 156+ mph
Weather Underground wunderground.com
115°W 105°W 95°W 85°W 75°W 65°W 55°W 45°W 35°W 25°W 15°W 5°W
70°N 60°N 55°N 50°N 45°N 40°N 35°N 30°N 25°N 20°N 15°N 10°N 5°N 0°

Fig. 6. Track of Hurricane Ike, taken from http://www.wunderground.com.

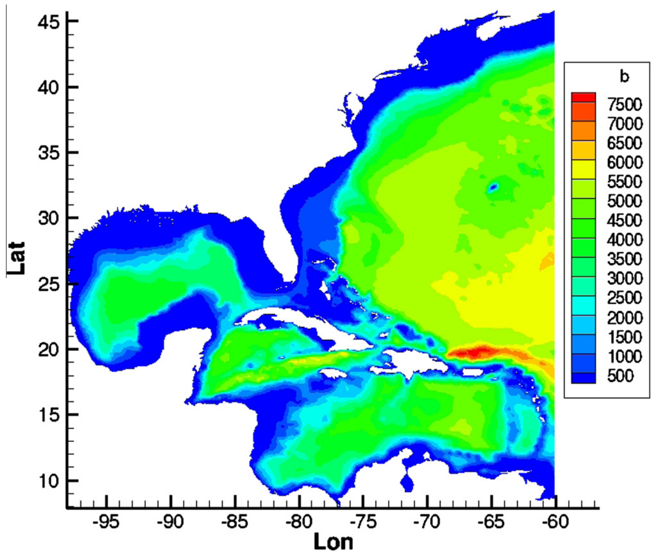


Fig. 7. Western North Atlantic/Texas domain with bathymetry (m).

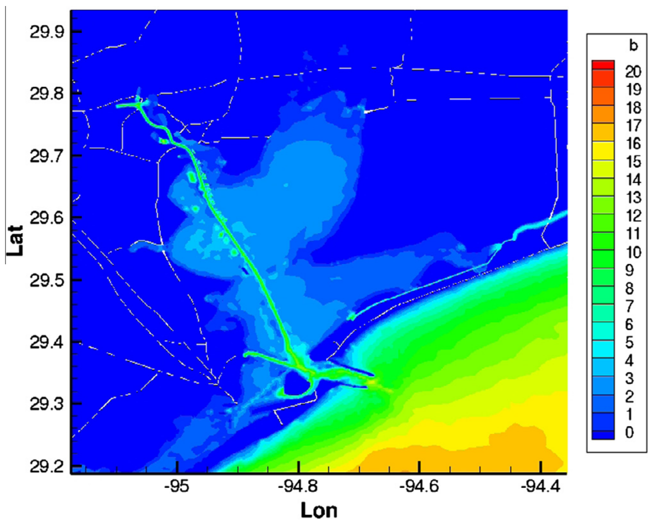


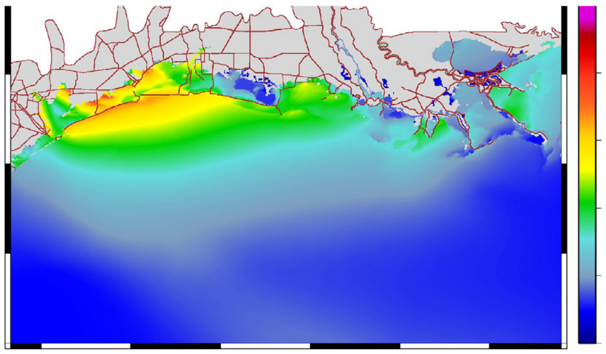
Fig. 8. Galveston Bay with bathymetry (m).

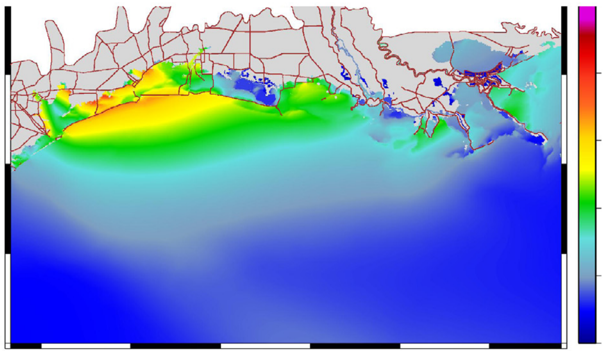
regions could be affected in a storm event. The contours in the figure represent bathymetry measured in meters. In Fig. 8, we zoom in on the Galveston Bay region, the narrow channel in the figure is the Houston Ship Channel, which connects the Port of Houston to the Gulf of Mexico. The land regions shown in the figure are also included in the computational domain.

We present results of simulations of a 5 day period during the storm, beginning at 12:00 p.m. on September 9, 2008 and progressing through 12:00 p.m. on September 14, 2008. The finite element mesh for these simulations consisted of 2,628,757 elements and 1,344,247 nodes, with most elements located in the Louisiana–Texas inland regions and continental shelf. The mesh is highly graded, with element areas on the order of several square kilometers in the deeper oceanic basins, transitioning to element areas on the order of 2000 square meters in the coastal regions of Texas and Louisiana. For simulations with no LTS, a global timestep of.5 s was used throughout the simulation. This was close to the minimum timestep computed using the CFL criteria (14) with velocity of zero.

Table 5
LTS parameters for scenarios 1 and 2 for Hurricane Ike.

LTS scenario	$\bar{N}$	$\bar{M}$	Δt's
1	2	2	.5, 1.0
2	4	2	.5, 1.0, 2.0, 4.0


(a) Ike maximum water elevation: No LTS


(b) Ike maximum water elevation: LTS Scenario 1

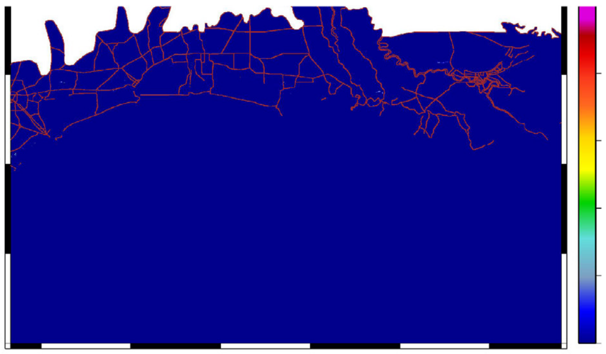

(c) Difference

Fig. 9. Comparison of maximum water surface elevation in meters for Hurricane Ike forecast. No LTS (top), LTS (middle) Scenario 1 and the difference (bottom).

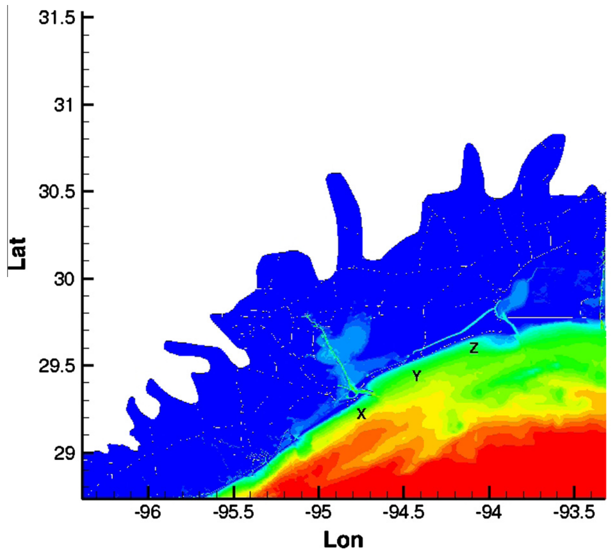


Fig. 10. Measurement locations X, Y and Z.

We considered several LTS scenarios and present the results for two such scenarios. The parameters used in these scenarios are summarized in Table 5.

Scenario 1: Upon an initial check of the local CFL constraints on each element, we determined that only a small fraction of elements required the minimum CFL timestep. The vast majority of elements had a local CFL timestep of 1 s or greater. Therefore, LTS scenario 1 is a simple LTS simulation with two timestepping levels ($\bar{N} = 2$) and with $\bar{M} = 2$. We ran the simulation on the Ranger parallel computer at the Texas Advanced Computing Center[2] with 800 processing cores. In this case, there were 65,727 elements in timestepping level 1 and 2,708,551 elements in timestepping level 2. These totals include elements in the overlap region between subdomains, therefore some elements are counted more than once. The elements remained fixed within their timestepping level throughout the 5 day simulation.

To compare the results of the LTS approach described above with no LTS, we look at two types of results, contours of maximum water elevation and hydrographs. The maximum water surface elevation is computed as

$$\eta_{\max}(x,y) = \max_{0 \leqslant t \leqslant T} \eta(x,y,t).$$

This quantity is of interest since it indicates where storm surge had the most impact over the course of the simulation. In Fig. 9, we compare the two solutions (LTS vs. no LTS) over the impact area (the upper Texas coast extending to southeastern Louisiana). We also computed the difference between the two solutions. Overall the agreement between the two solutions is quite close. There are a few small differences in the solutions in some isolated elements, primarily in regions which experience wetting and drying. These differences are most likely due to sensitivities in the wetting and drying algorithm used in the code.

We also compare hydrographs of solutions at three locations along the upper Texas coast, where actual instruments were deployed just before the storm, as described in [33]. These measurement locations are labeled as X, Y and Z in Fig. 10 and are in the region of maximum storm surge. The LTS and no LTS solutions are plotted together in Fig. 11, where we observe that the solutions are virtually identical.

Scenario 2: Upon further examination of the local CFL timesteps, we found that most elements are able to take an even larger timestep than 1 s, at least based on an initial estimate. Therefore, in the second scenario we divided the domain into 4 timestepping levels with $\bar{M} = 2$, with timesteps ranging from .50 to 4.00 s by factors of 2. The number of elements in each group at time $t = 0$ is gi-

[2] The Ranger system is comprised of 3936 16-way SMP compute nodes providing 15,744 AMD Opteron processors for a total of 62,976 compute cores, 123 TB of total memory and 1.7 PB of raw global disk space. It has a theoretical peak performance of 579 TFLOPS. All Ranger nodes are interconnected using InfiniBand technology in a full-CLOS topology providing a 1 GB/s point-to-point bandwidth.

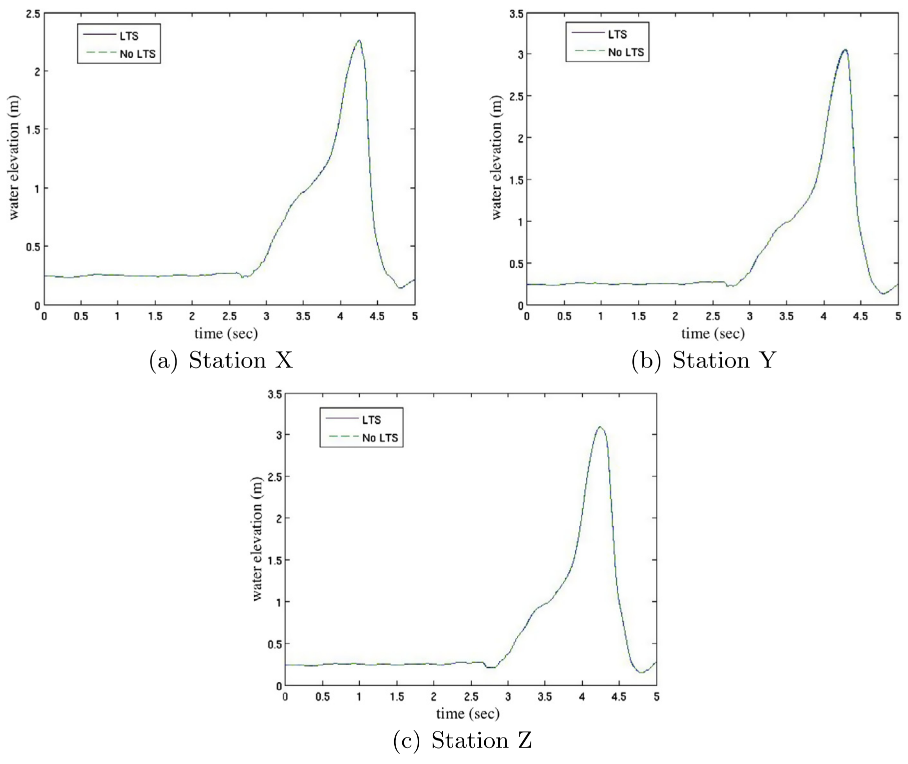


Fig. 11. Comparison of hydrographs for Hurricane Ike at measurement locations X, Y and Z.

Table 6
LTS scenario 1 data for Hurricane Ike simulation.

Timestep level	Δt	# Elements per level ($t = 0$)
1	.5	56795
2	1.0	60158
3	2.00	521461
4	4.0	2043731

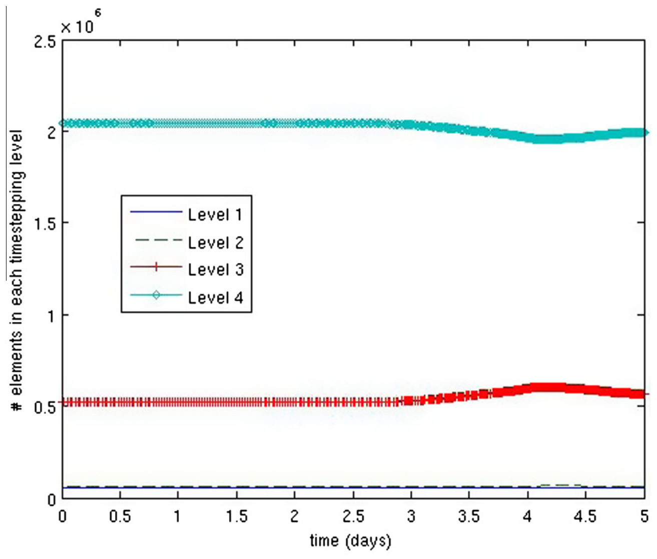


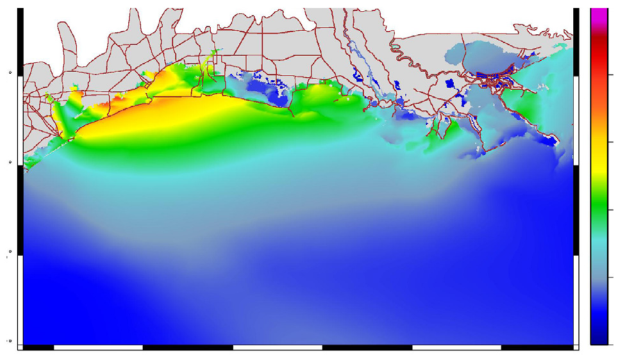
Fig. 12. Number of elements in each timestepping level vs. time for Hurricane Ike simulation.

ven in Table 6. Note that the vast majority of elements can take a 4 s timestep. In this case, we recomputed the local timestep by checking the CFL condition at periodic intervals throughout the computation. We chose to check and recompute local timesteps roughly every .1 days during the simulation. Thus, during the course of the hurricane, the elements can shift between timestep groups. In Fig. 12, we illustrate this further, where we show specifically how the number of elements within each timestepping level varies with time. We note that Hurricane Ike made landfall between 3.5 and 4.5 days of simulation, and this is the time frame during which most elements are shifted among timestepping groups.

The results produced by the LTS solution were nearly identical to those produced in LTS scenario 1 described above. In Fig. 13, we show the maximum elevation solution over the impact area, and the difference between the LTS and no LTS solutions for this case. Overall the agreement between the two solutions is quite close. As noted above, there are a few small differences in the solutions in isolated elements, primarily in regions which experience wetting and drying.

We remark that the water elevation hydrographs at Stations X, Y and Z are virtually identical to those observed in Fig. 11 and we do not reproduce them. In summary, the LTS method described herein captures the maximum surge comparable to the RKDG method with no LTS. We also remark that any attempts to run the simulation with no LTS and a timestep larger than the global CFL timestep of .527 s blew up early in the simulation. Therefore, experimentally at least this was a tight bound on the maximum allowable timestep for the standard RKDG method.

Finally, we discuss the parallel performance of the model with and without LTS. Using 800 cores on Ranger; i.e., dividing the domain into 800 subdomains, and using a global timestep of .5 s, the total compute time for the no LTS case was 930 min. The same run using LTS scenario 1 took 586 min for a 37% reduction in wall clock time. LTS scenario 2 took 432 min for a 53% reduction in wall clock time. We also performed simulations of the no LTS case and LTS scenario 2 on 1600 cores. The run times were 555 min and 240 min, respectively. Thus, no LTS exhibited a parallel speedup


(a) Ike maximum water elevation: LTS Scenario 2

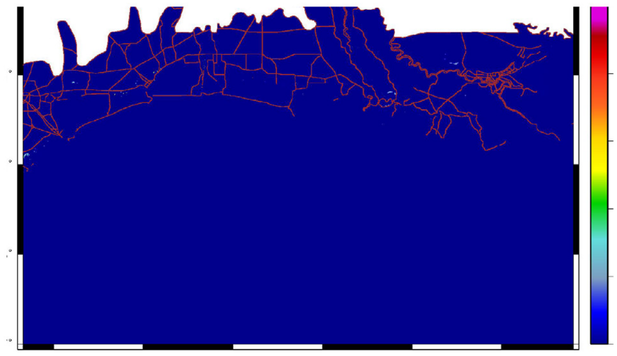

(b) Difference

Fig. 13. Maximum water surface elevation in meters for Hurricane Ike forecast. LTS scenario 2 and the difference between LTS and no LTS solutions.

factor of 1.67 and LTS scenario 2 a factor of 1.8 for this particular test case.

4. Conclusion

In this paper, we have investigated the LTS approach described in [8] for some large-scale applications in shallow water flows. These applications require the use of parallel computing, therefore we extended the serial LTS method described in [8] to utilize parallel, distributed memory computing platforms. We have examined the accuracy and efficiency of the method for standard tidal flow and for modeling hurricane storm surges. The method has proven to be robust even in extreme, wind-driven events.

The parallel performance of the LTS method for different choices of $\bar{N}$ and $\bar{M}$ is difficult to predict *a priori*. Since the element sizes in the meshes that we are given and the initial water depths both vary significantly over the domain, the local CFL constraints also vary significantly over the domain. Thus, it is difficult to determine in advance the $\bar{N}$ and $\bar{M}$ which would optimize the parallel performance. Our approach has been to test various values of $\bar{N}$ and $\bar{M}$ over fairly short time intervals, on the order of .1 days, before performing a full multi-day simulation.

Future work will focus on exploring further parallel efficiency of the method for shallow water flows and other applications where there are distinct separations in spatial discretization and temporal scales.

Acknowledgments

Author C. Dawson acknowledges the support of National Science Foundation Grants DMS-1217071 and DMS-0915223. J. Westerink acknowledges the support of National Science Foundation Grant OCI-0746232, and E. Kubatko acknowledges National Science Foundation Grants DMS-0915118 and DMS-1217218.

References

[1] S. Osher, R. Sanders, Numerical approximations to nonlinear conservation laws with locally varying time and space grids, Math. Comput. 41 (164) (1983) 321–336.
[2] C. Dawson, High resolution upwind-mixed finite element methods for advection–diffusion equations with variable time-stepping, Numer. Methods Partial Differ. Equ. 11 (5) (1995) 525–538.
[3] C. Dawson, R. Kirby, High resolution schemes for conservation laws with locally varying time steps, SIAM J. Sci. Comput. 22 (6) (2000) 2256–2281, http://dx.doi.org/10.1137/S1064827500367737.
[4] R. Kirby, On the convergence of high resolution methods with multiple time scales for hyperbolic conservation laws, Math. Comput. 72 (243) (2002) 1239–1250.
[5] B.F. Sanders, Integration of a shallow water model with a local time-step, J. Hydraul. Res. 46 (4) (2008) 466–475.
[6] E.M. Constantinescu, A. Sandu, Multirate timestepping methods for hyperbolic conservation laws, J. Sci. Comput. 33 (2007) 239–278.
[7] A. Sandu, E. Constantinescu, Multirate explicit Adams methods for time integration of conservation laws, J. Sci. Comput. 38 (2009) 229–249.
[8] C.J. Trahan, C. Dawson, Local time-stepping in Runge–Kutta discontinuous Galerkin finite element methods applied to the shallow water equations, Comput. Methods Appl. Mech. Engrg. 217–220 (2012) 139–152.
[9] E. Constantinescu, A. Sandu, Extrapolated multirate methods for differential equations with multiple time scales, J. Sci. Comput., http://dx.doi.org/10.1007/s10915-012-9662-z.
[10] L. Liu, X. Li, F. Hu, Nonuniform time-step Runge–Kutta discontinuous Galerkin method for computational aeroacoustics, J. Comput. Phys. 229 (19) (2010) 6874–6897.
[11] E. Montseny, S. Pernet, X. Ferriéres, G. Cohen, Dissipative terms and local time-stepping improvements in a spatial high order discontinuous Galerkin scheme for the time domain Maxwell's equations, J. Comput. Phys. 227 (14) (2008) 6795–6820.
[12] J. Remacle, J. Flaherty, M. Shephard, An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow, SIAM Rev. 45 (1) (2003).
[13] J. Diaz, M. Grote, Energy conserving explicit local time stepping for second-order wave equations, SIAM J. Sci. Comput. 31 (3) (2009) 1945–2014.
[14] N. Godel, S. Schomann, T. Warburton, M. Clemens, GPU accelerated Adams–Bashforth multirate discontinuous Galerkin FEM simulation of high-frequency electromagnetic fields, IEEE Trans. Magn. 46 (8) (2010) 2735–2738.
[15] M. Berger, D.L. George, R.J. LeVeque, K.T. Mandli, The GeoClaw software for depth-averaged flows with adaptive refinement, Adv. Water Resour. 34 (2011) 1195–1206.
[16] M. Berger, R.J. LeVeque, Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems, SIAM J. Numer. Anal. 35 (1998) 2298–2316.
[17] E.J. Kubatko, J.J. Westerink, C. Dawson, hp discontinuous Galerkin methods for advection dominated problems in shallow water flow, Comput. Methods Appl. Mech. Engrg. 196 (1–3) (2006) 437–451.
[18] E.J. Kubatko, S. Bunya, C. Dawson, J.J. Westerink, A performance comparison of continuous and discontinuous finite element shallow water models, J. Sci. Comput. 40 (2009) 315–339.
[19] C. Dawson, E. Kubatko, J. Westerink, C. Trahan, C. Mirabito, C. Michoski, N. Panda, Discontinuous Galerkin methods for modeling hurricane storm surge, Adv. Water Resour., http://dx.doi.org/10.1016/j.advwatres.2010.11.004.
[20] E. Kubatko, S. Bunya, C. Dawson, J. Westerink, Dynamic *p*-adaptive Runge–Kutta discontinuous Galerkin methods for the shallow water equations, Comput. Methods Appl. Mech. Engrg. 198 (2009) 1766–1774.
[21] S. Bunya, E. Kubatko, J. Westerink, C. Dawson, A wetting and drying treatment for the Runge–Kutta discontinuous Galerkin solution to the shallow water equations, Comput. Methods Appl. Mech. Engrg. 198 (17–20) (2009) 1548–1562, http://dx.doi.org/10.1016/j.cma.2009.01.008.
[22] D. Wirasaet, S. Tanaka, E.J. Kubatko, J.J. Westerink, C. Dawson, A performance comparison of nodal discontinuous Galerkin methods on triangles and quadrilaterals, Int. J. Numer. Methods Fluids 64 (2010) 1336–1362.
[23] C.-W. Shu, S. Osher, Efficient implementation of essentially non-oscillatory shock-capturing schemes: II, J. Comput. Phys. 83 (1) (1989) 32–78, http://dx.doi.org/10.1016/0021-999(89)90222-2.
[24] C.-W. Shu, TVB uniformly high-order schemes for conservation laws, Math. Comput. 49 (179) (1987) 105–121.
[25] C. Michoski, C. Mirabito, C. Dawson, D. Wirasaet, E.J. Kubatko, J.J. Westerink, Adaptive hierarchie transformations over dynamic *p*-enriched schemes applied to generalized DG systems, J. Comput. Phys. 230 (2011) 8028–8056.
[26] J.B. Bell, C.N. Dawson, G.R. Shubin, An unsplit, higher-order Godunov method for scalar conservation laws, J. Comput. Phys. 74 (1988) 1–24.
[27] G. Karypis, V. Kumar, METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN, 1998.
[28] G. Karypis, V. Kumar, A fast and high quality scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1999) 359–392.

[29] J. Westerink, R. Luettich, J. Feyen, J. Atkinson, C. Dawson, H. Roberts, M. Powell, J. Dunion, E. Kubatko, H. Pourtaheri, A basin to channel scale unstructured grid hurricane storm surge model applied to Sourthern Louisiana, Am. Meteorol. Soc. 136 (3) (2008) 833–864.

[30] J. Garratt, Review of drag coefficients over oceans and continents, Mon. Weather Rev. 105 (1977) 915–929.

[31] C.L. Provost, P. Vincent, Finite Elements for Modeling Ocean Tides, John Wiley and Sons, New York, 1991.

[32] A. Mukai, J. Westerink, R. Luettich, D. Mark, Eastcoast 2001, a tidal constituent database for Western North Atlantic, Gulf of Mexico, and Caribbean Sea, TR ERDC01-x, US Army Engineer Research and Development Center, Vicksburg, MS, 2002.

[33] A. Kennedy, U. Gravois, B. Zachry, J. Westerink, M. Hope, J. Dietrich, M. Powell, A. Cox, J.R.A. Luettich, R. Dean, Origin of the hurricane Ike forerunner surge, Geophys. Res. Lett. 38 (2011).

[34] G. Holland, An analytic model of the wind and pressure profiles in hurricanes, Mon. Weather Rev. 108 (1980) 1212–1218.

[35] J. Fleming, C. Fulcher, R. Luettich, B. Estrade, G. Alolen, H. Winer, A real time storm surge forecasting system using ADCIRC, in: M. Spaulding (Ed.), Estuarine and Coastal Modeling X, ASCE, pp. 893–912.