J Sci Comput (2009) 40: 315–339
DOI 10.1007/s10915-009-9268-2

A Performance Comparison of Continuous and Discontinuous Finite Element Shallow Water Models

Ethan J. Kubatko · Shintaro Bunya · Clint Dawson · Joannes J. Westerink · Chris Mirabito

Received: 31 May 2008 / Revised: 18 December 2008 / Accepted: 5 January 2009 /
Published online: 23 January 2009

Abstract We present a comparative study of two finite element shallow water equation (SWE) models: a generalized wave continuity equation based continuous Galerkin (CG) model—an approach used by several existing SWE models—and a recently developed discontinuous Galerkin (DG) model. While DG methods are known to possess a number of favorable properties, such as local mass conservation, one commonly cited disadvantage is the larger number of degrees of freedom associated with the methods, which naturally translates into a greater computational cost compared to CG methods. However, in a series of numerical tests, we demonstrate that the DG SWE model is generally more efficient than the CG model (i) in terms of achieving a specified error level for a given computational cost and (ii) on large-scale parallel machines because of the inherently local structure of

E.J. Kubatko (✉)
Department of Civil and Environmental Engineering and Geodetic Science, The Ohio State University, Columbus, OH 43210, USA
e-mail: kubatko.3@osu.edu

S. Bunya
Department of Systems Innovation, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, 113-8656, Japan
e-mail: bunya@sys.t.u-tokyo.ac.jp

C. Dawson · C. Mirabito
Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX 78712, USA

C. Dawson
e-mail: clint@ices.utexas.edu

C. Mirabito
e-mail: mirabito@ices.utexas.edu

J.J. Westerink
Department of Civil Engineering and Geological Sciences, University of Notre Dame, Notre Dame, IN 46556, USA
e-mail: jjw@photius.ce.nd.edu

the method. Both models are verified on a series of analytic test cases and validated on a field-scale application.

Keywords Shallow water equations · Finite elements · Discontinuous Galerkin · Generalized wave continuity equation

1 Introduction

The shallow water equations (SWE) are used to model free surface flow in the deep ocean, coastal ocean, estuaries, rivers, open channels, and coastal floodplain. Many flow processes are described by these equations including tides, the propagation of flood waves in rivers, tsunami waves, and storm surges associated with hurricanes. Often coupled with these hydrodynamic phenomena are the transport of quantities such as pollutants, salinity and sediment. The simulation of these types of processes typically requires solving the SWE over large, geometrically complex spatial domains for long periods of time. Such a situation necessitates the use of large-scale computational models that can accurately and efficiently obtain numerical solutions to the SWE.

Over the past three decades, various finite element based SWE models have been developed for these purposes; see, for example, [1, 9, 17, 19–22, 24, 28, 29]. Finite element methods are a judicious choice as the base algorithm for SWE models because of the flexibility they provide, through the use of unstructured meshes, for accurately resolving a wide range of flow scales and geometrically complex vertical and horizontal boundaries introduced by both the bathymetry/topography and the coastline. Historically, one of the more widespread continuous Galerkin (CG) finite element approaches that has been used for the solution of the SWE involves solving a reformulation of the continuity equation—the so-called "wave continuity equation". This approach was first proposed by Gray and Lynch [12], was extended for the solution of a generalized wave continuity equation (GWCE) by Kinnmark [14], and forms the basis of the advanced circulation (ADCIRC) model [22] developed by the fourth author and a number of collaborators. While this approach has been successfully used for a number of finite element studies (see, for example, [10, 12, 25, 29–31]), it is not without its disadvantages. Most notably perhaps, the performance and mass conservation properties of the method are highly dependent on a user-selected GWCE weighting parameter; see, for example, [5, 14–16].

In part, to address the issues associated with GWCE-based CG models, discontinuous Galerkin (DG) finite element models have recently been developed for the solution of the SWE; see, for example, [1, 9, 17, 21, 28]. DG methods possess a number of favorable properties for solving the SWE such as the ability to model highly advective flows including problems with hydraulic jumps or tidal bores (discontinuities), the ability to vary polynomial approximations element-by-element, and the ability to employ non-conforming meshes, i.e. meshes with so-called hanging nodes. These last two facts simplify the implementation of automatic h (mesh) and p (polynomial order) adaptive strategies. Moreover, DG methods inherently possess local (element-by-element) mass conservation properties, which has been shown to be an important factor when coupling flow and transport models [7]. While DG methods have a number of advantageous properties, one major drawback that has often been cited in the literature is the larger number of degrees of freedom that are involved in comparison to CG methods; see, for example, [6, 8, 13]. More specifically, spatial discretization of the two-dimensional SWE using a CG method with linear triangular elements results in $3N_n$ total degrees of freedom where N_n is the number of nodes (vertices) in a given computational mesh. A full DG discretization of the SWE (i.e., both the continuity and momentum equations are discretized using a DG method) on the same computational mesh using linear elements results in $9N_e$ total degrees of freedom where N_e is the number of elements in the mesh. For many typical unstructured meshes, the ratio $N_e/N_n \approx 2$. Thus, a DG method using linear triangular elements will typically have 6 times as many degrees of freedom compared to a corresponding CG method on a given mesh.

This increase in the numbers of degrees of freedom will obviously have a negative impact on the relative efficiency of DG methods compared to CG methods. Indeed, a preliminary comparative study of CG, DG, and hybrid DG/CG finite element SWE models confirmed that the DG approach was on the order of 4 to 5 times more expensive than the CG approach on a per time step basis on serial machines [8]. At the same time, preliminary studies have also suggested that the DG methodology offers significant improvements in accuracy and robustness over their CG counterparts [1, 17]; however a detailed study comparing model run times and error levels has been lacking to properly assess the *net* computational cost of the two approaches. That is, while CG based models may perform faster serially than DG models they could have larger error levels on a given mesh and/or converge to the true solution at a slower rate upon subsequent mesh refinements thus actually making the DG approach more efficient in terms of achieving a specified error level for a given computational cost. Furthermore, and perhaps more importantly, given the rising availability and use of massively parallel computing resources, the parallel scalability of algorithms is becoming an increasingly important factor in the assessment of computational models. While the DG methods involve more "work" per element than CG methods, and may be deemed computationally inefficient on serial or small-scale parallel machines, the reported high parallelizability of DG methods (see, for example, [3, 4]) indicates that DG based models may be significantly more scalable on large-scale parallel clusters.

The goal of this paper is to provide a detailed comparative study of CG and DG finite element SWE models to investigate these issues. Specifically, the models will be applied to a set of test problems where they will be compared with respect to accuracy, convergence rates, serial and parallel run times, and efficiency. The models will be tested on both linear and nonlinear idealized problems as well as a full-scale application. The CG model that is investigated here is the previously mentioned GWCE-based ADCIRC model—hereafter referred to as ADCIRC–CG—that uses continuous, piecewise linear approximations on triangular elements. The DG model that is used employs the approach outlined in [17]. It allows the use of arbitrary polynomial approximations over elements, but herein, for purposes of comparison to ADCIRC–CG, we limit our investigation to linear approximations. We will refer to this model as ADCIRC–DG due to the fact that, although it uses a completely different solution strategy than ADCIRC–CG, it has been integrated into the ADCIRC framework in the sense that it uses the same I/O, domain decomposition for parallelization, and physical parameterizations, e.g. bottom friction and surface stress formulations, etc.

The rest of this paper is as follows. In the next section, we give summaries of the CG and DG models that are used in this study. We begin by presenting the governing equations that are discretized by the models and then give brief summaries of the numerical approaches used in the models including the parallelization strategies that are employed. In Sect. 3, we present the results of our extensive numerical testing, and we make some concluding remarks in Sect. 4.

2 CG and DG Model Summaries

2.1 Governing Equations

Vertical integration of the Navier–Stokes equations along with the assumptions of hydrostatic pressure and a vertically uniform horizontal velocity profile results in the SWE, which consist of the depth-averaged continuity equation $\mathcal{L}$ and the momentum equations $\mathcal{M}$:

$$\mathcal{L} \equiv \frac{\partial \zeta}{\partial t} + \nabla \cdot \mathbf{q} = 0, \tag{2.1}$$

$$\mathcal{M} \equiv \frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \big(\mathbf{q}\mathbf{q}/H\big) + \tau_{bf}\mathbf{q} + \mathbf{f}_c \times \mathbf{q} + gH\nabla\zeta - \varepsilon\Delta\mathbf{q} - \mathbf{F} = \mathbf{0}. \tag{2.2}$$

In the equations above, ζ is the free surface water elevation relative to the geoid (positive upward), $H = \zeta + h_b$ is the total depth of the water column where h_b is the bathymetric depth relative to the geoid (positive downward), $\mathbf{q} \equiv (Hu, Hv)$ where u and v are the depth averaged velocities in the x and y directions, respectively, τ_{bf} is the bottom friction factor, $\mathbf{f}_c$ is the Coriolis parameter, g is the gravitational constant, ε is the eddy viscosity coefficient, and $\mathbf{F}$ is meant to represent additional body and surface forces, which may be due to, for example, surface wind stress, variable atmospheric pressure, and tidal potential forcing. A quadratic friction factor is generally used, i.e. $\tau_{bf} = C_f\sqrt{u^2 + v^2}/H$ where C_f is the bottom friction coefficient, the exception being when the linearized form of the SWE are solved, in which case τ_{bf} is taken as a constant. The variables ζ and $\mathbf{q}$ are functions of the horizontal coordinates x and y and time t.

These equations are solved over a spatial domain Ω in $\mathbb{R}^2$ for time $t > 0$. Initial free surface elevations ζ_0 and velocities $\mathbf{u}_0$ are specified at time $t = 0$, and along $\partial\Omega$, the boundary of Ω, ζ and/or $\mathbf{q}$ are specified for all time t. To apply a finite element method, the domain Ω is first partitioned into a set of non-overlapping elements Ω_e. Here we only consider triangular elements.

2.2 ADCIRC–CG Methodology

As previously mentioned, the ADCIRC–CG model solves a reformulation of the depth-averaged continuity equation (2.1). Using the notation of the previous section, this reformulated equation, the GWCE, can be written as:

$$\frac{\partial \mathcal{L}}{\partial t} + \nabla \cdot \mathcal{M} + \tau_0\mathcal{L} = 0, \tag{2.3}$$

where τ_0 is the so-called GWCE weighting parameter. The GWCE is solved in conjunction with the momentum equations $\mathcal{M}$ to yield ζ and $\mathbf{q}$. The main motivation behind this reformulation is that a CG discretization of the linear forms of these equations gives rise to a monotonic dispersion relationship, which yields non-oscillatory solutions. This is in contrast to the folded dispersion relationship obtained from a CG discretization of the primitive equations using equal-order interpolation spaces; see, for example, [2, 14]. However, as mentioned previously, one of the drawbacks of this approach is that the performance of the method depends on the selection of the GWCE weighing parameter τ_0. As a general guideline, τ_0 is typically set equal to the largest value of an equivalent linear friction factor (e.g., for linear friction $\tau_0 = \tau_{bf}$, for quadratic friction $\tau_0 = \max(C_f\sqrt{u^2 + v^2}/H)$); see [15].

To apply a CG method to (2.3) and (2.2), a weak formulation is first obtained by multiplying each equation by a sufficiently smooth test function v and integrating over the domain Ω, where certain terms of (2.3) are integrated by parts. In ADCIRC–CG, ζ and $\mathbf{q}$ are approximated by ζ_h and $\mathbf{q}_h$ and the test function is set to $v = v_h$—all three of which are continuous, piecewise linear functions. The momentum equation (2.2) is integrated explicitly in time, and the mass matrix associated with the time derivative term is mass-lumped; therefore no linear systems are solved to compute the velocity. The GWCE (2.3) is discretized using three time levels, mostly centrally weighted. A linear system involving the mass matrix is solved at each time step to update the elevation. This system is sparse, symmetric and positive definite. A Jacobi preconditioned conjugate gradient method is used to solve it. For more details concerning the implementation of ADCIRC–CG; see [22].

2.3 ADCIRC–DG Methodology

ADCIRC–DG directly solves (2.1) and (2.2) without reformulation. In this case, a weak formulation is obtained by multiplying each equation by a sufficiently smooth test function v and integrating over a *single* element where the divergence terms of (2.1) and (2.2) are integrated by parts, which generates a boundary integral of the flux terms. In our implementation, these boundary flux terms are approximated using either the so-called local Lax Friedrichs (LLF) or Roe flux [17].

With a DG method, the test functions and approximations of ζ and $\mathbf{q}$ are all chosen to be discontinuous, piecewise polynomials of some degree k, defined on each element. Furthermore, an explicit Runge-Kutta method is used to integrate in time. In our implementation, the degree of approximation k is arbitrary, and we have tested up to $k = 7$ with good results [17]. We note that this has been implemented using the so-called Dubiner basis—the orthogonality of which results in a "matrix-free" algorithm. In this study, we restrict our attention to $k = 1$ and second-order Runge-Kutta time-stepping for sake of comparison to ADCIRC–CG. More details of the DG implementation that is used can be found in [17].

2.4 Parallelization

For parallel simulations with both ADCIRC–CG and ADCIRC–DG, the finite element partition of the computational domain Ω is subdivided into a set of N subdomains $\{\omega_j\}_{j=1}^{N}$ using METIS [18]. Each subdomain is assigned its own processor (core). Let $\partial\omega_j$ denote the portion of the boundary of ω_j that is not part of $\partial\Omega$. Elements belonging to ω_j that have at least one vertex on $\partial\omega_j$ also belong to the neighboring subdomain(s), i.e., neighboring subdomains share a "layer" of elements; see Fig. 1 where this is shown for two neighboring subdomains. The set of elements that ω_i shares with a neighboring subdomain ω_j is denoted by $\bar{\omega}_{ij}$, i.e. $\bar{\omega}_{ij} \equiv \omega_i \cap \omega_j$ where $i \neq j$.

Given the local structure of the DG method, it is simplest to begin the discussion with the description of the ADCIRC–DG parallelization strategy. To this end, consider a subdomain ω_i and one of its neighboring subdomains ω_j as illustrated in Fig. 1. As noted above, these subdomains will share a common set of elements $\bar{\omega}_{ij}$. In the DG method, all calculations are local to an element with the exception of the computation of the boundary flux integrals, which involves the neighboring element to an edge. Thus, considering subdomain ω_i, those elements of ω_i that have an edge on $\partial\omega_i$ require information from the neighboring subdomain ω_j. Initially, dummy values are used for this missing information and the degrees of freedom for each element of ω_i are computed as usual. Corrected degrees of freedom for elements with edges on $\partial\omega_i$ can then be obtained from the neighboring subdomain ω_j via

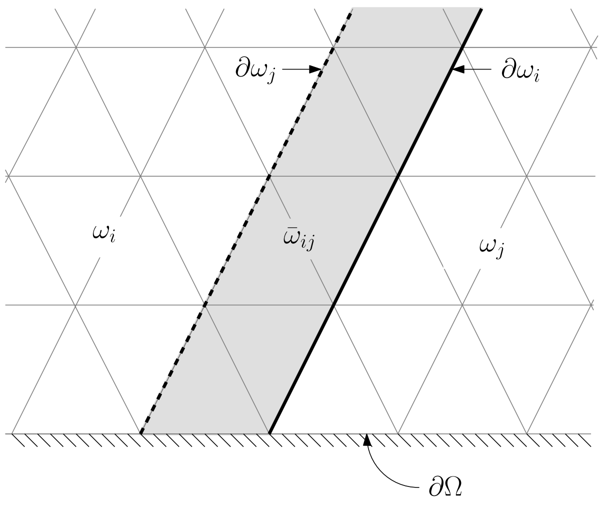


Fig. 1 Overlap of two neighboring subdomains for parallelization

message passing, given the fact these elements and their edge neighbors are wholly contained within ω_j and will therefore have correct degrees of freedom. We note that this local communication between neighboring subdomains is the only communication required for the parallelization of the DG method. This communication occurs at the end of each Runge-Kutta stage in the time-stepping algorithm.

The parallelization of ADCIRC–CG follows the same structure. Instead of elemental degrees of freedom, nodal degrees of freedom at vertices on $\partial\omega_i$ are passed from ω_j to ω_i. This exchange of information occurs after each conjugate gradient iteration in the linear solution step. We remark that since we are using a simple Jacobi preconditioner, the preconditioning matrix is independent of the domain decomposition. Generally 5 to 15 iterations of the linear solver are required per time step to reach a solver tolerance of 10^{-6}. Furthermore, two dot products (global sums) and one matrix-vector multiply must be performed during each conjugate gradient iteration, which hinders the parallel performance of the algorithm.

3 Numerical Tests

In this section, we present the results of our numerical tests. All simulations were performed on the Texas Advanced Computing Center's (TACC) Dell Linux Cluster, Lonestar, which consists of 5,840 cores within 1,460 Dell PowerEdge 1955 compute blades (nodes). Each compute node has two processors—each a Xeon 5100 series 2.66 GHz dual–core processor. Reported serial and parallel run times, denoted T_s and T_p, respectively, are wall clock times to the nearest second. Reported times are typically an average of 3 identical simulations.

3.1 Accuracy, Convergence, and Serial Run Times

In this subsection, we consider a set of test cases for the linear SWE with analytic solutions proposed by Lynch and Gray [23]. Examination of these problems allows us to systematically investigate and rigorously assess the accuracy, convergence and relative computational costs of the CG and DG models. We begin with a brief description of these problems.

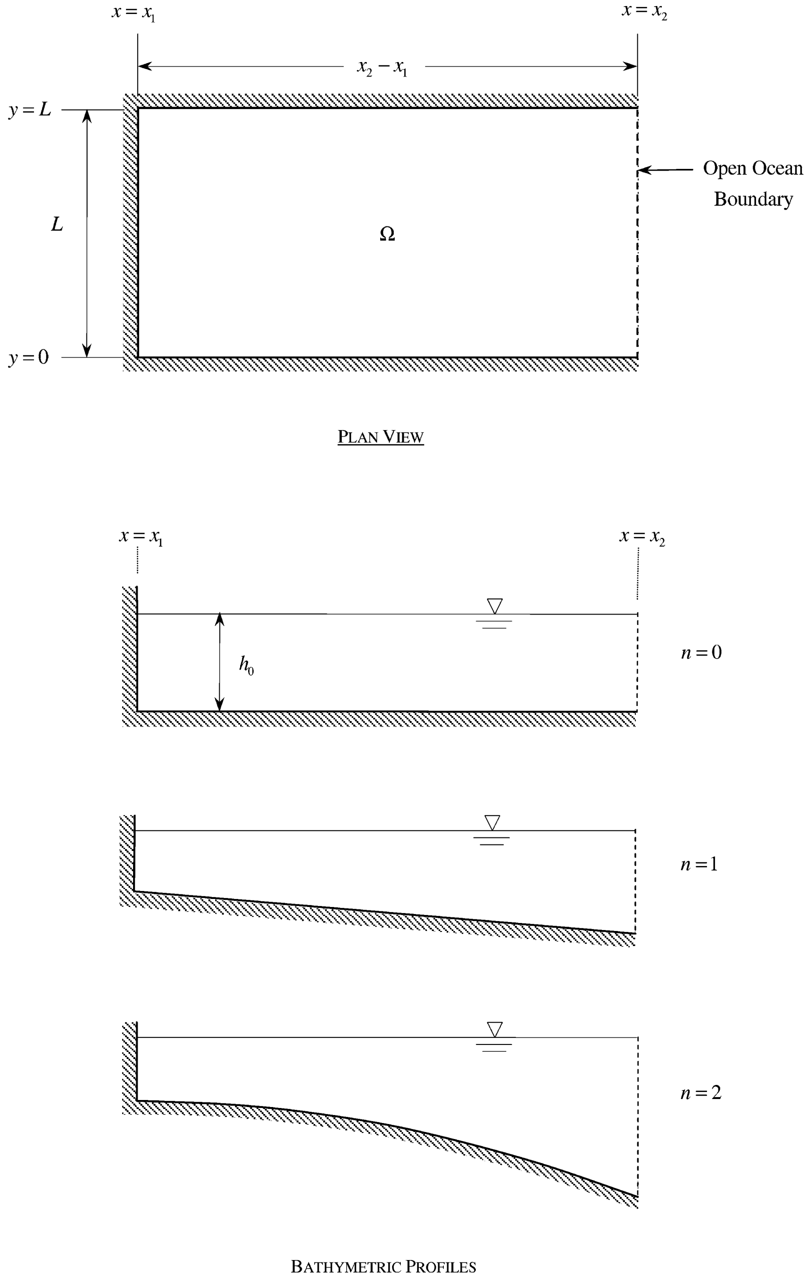


Fig. 2 Domains of the test cases of Lynch and Gray

The domains of the test cases are as defined in Fig. 2. The boundaries at $y = 0$, $y = L$, and $x = x_1$ are land or no-normal flow boundaries, and at $x = x_2$ a surface elevation is

Table 1 Bathymetry parameters for test cases of Lynch and Gray

n	h_0	$h_b(x_1)$	$h_b(x_2)$
0	3	3	3
1	1.00×10^{-4}	6	15
2	1.11×10^{-9}	4	25

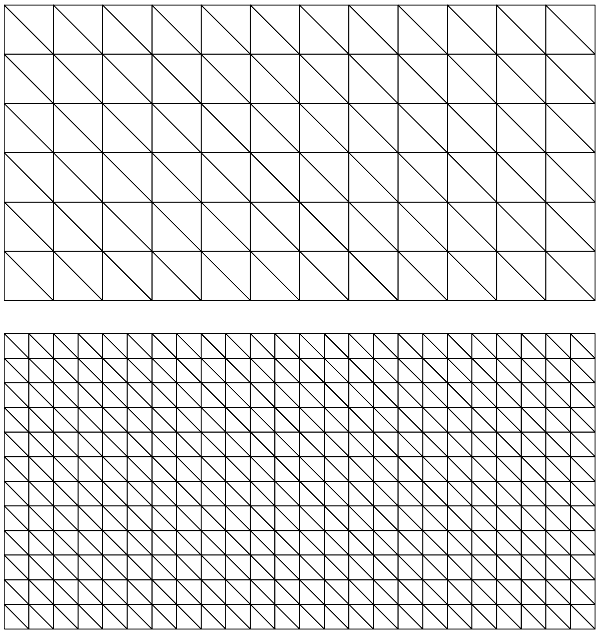


Fig. 3 Computational meshes h_1 and h_2 used in the convergence studies

prescribed by a periodic tidal forcing function of the general form:

$$\zeta(x_2, y, t) = \mathrm{Re}\left\{\zeta_0(y)e^{i\omega t}\right\} \tag{3.1}$$

where ζ_0 is, in general, a complex function representing the tidal forcing amplitude and phase, ω is the frequency of the tidal forcing function, and $i = \sqrt{-1}$. The bathymetry of the domain is defined by:

$$h_b(x, y) = h_0 x^n \tag{3.2}$$

where h_0 and n are specified constants with $h_0 > 0$; see Fig. 2. For our numerical tests, we take the width of the domain in the y-direction to be $L = 45$ km. In the x-direction, we take $x_2 - x_1 = 2L$. We consider flat, linear, and quadratic bathymetric profiles—$n = 0$, 1, and 2, respectively, in (3.1); see Table 1. A tidal amplitude of $\zeta_0 = 0.5$ m is used with a frequency of $\omega \approx 0.0001405$ s^{-1}, which corresponds to an M_2 tide, and a linear friction factor of $\tau_{bf} = 0.0001$ s^{-1} is used.

The problems are solved on the four computational meshes. The coarsest mesh has a node-to-node spacing in both the x and y directions of $h = 7500$ m and has 144 elements. The three finer meshes are obtained by applying successive 1:4 refinements of this mesh with the finest mesh thus having a node spacing of $h = 937.5$ m and consisting of 9216 ($= 144 \times 4^3$) elements (Fig. 3 shows the first two meshes). We denote these meshes as h_1

Table 2 Mesh summaries, serial run times (T_s) and relative costs (Γ) of the models

Mesh	# of Elements	# of Nodes	ADCIRC-CG T_s	ADCIRC-DG T_s	Γ
h_1	144	91	119	378	3.17
h_2	576	325	348	1168	3.36
h_3	2304	1225	1208	4415	3.65
h_4	9216	4753	4547	21577	4.75

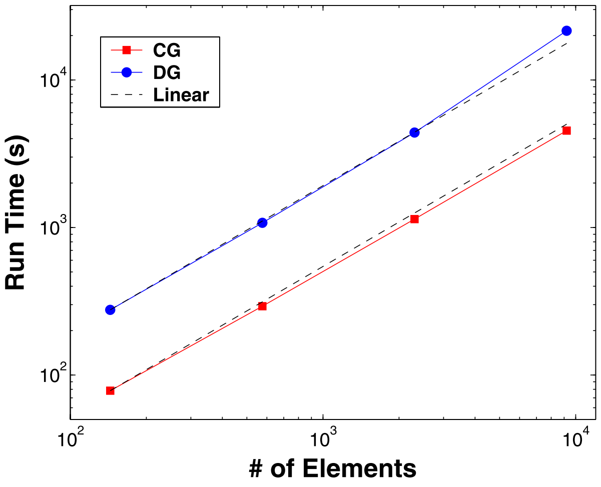


Fig. 4 Serial run times as a function of the number of elements

through h_4 from coarsest to finest. Simulations are started from time $t = 0$ and run for a period of $T = 10$ days using a time step of $\Delta t = 1$ s, which is well below the maximum allowable time step estimated from the CFL condition for all the meshes. Initial conditions are computed for the models from the exact solutions. Absolute element errors are computed at the barycenter of each element of the mesh as the absolute value of the difference between the exact and numerical solutions at the final time of the simulation. The L^1 errors reported in the tables that follow are then computed by multiplying the element errors by the area of the element, summing these products over the whole domain and dividing by the total area of the domain. The L^∞ errors reported are the maximum of the element errors over the whole domain.

First, we examine the serial run times T_s and the relative costs of the two models on the four computational meshes. The relative costs are reported as the ratio Γ of the ADCIRC–CG and ADCIRC–DG run times. Thus, $\Gamma < 1$ indicates that the CG model is more efficient than the DG model, and $\Gamma > 1$ indicates the DG model is more efficient. The mesh data and run times are summarized in Table 2. As shown in Fig. 4, model run times scale close to linearly with respect to the number of elements up to mesh h_3 for the DG model and up to h_4 for the CG model. From h_3 to h_4, the DG model shows a slightly greater increase in run time probably due to cache effects (a similar trend is observed with the CG model run times in going from h_4 to a mesh that is a uniform refinement of h_4). As expected, the DG model takes, on average, approximately 4 times ($4.06\times$) longer to run than the CG model.

Before comparing the errors and convergence rates of the two models for this series of problems, we remark on the performance of the CG model with respect to the selection of the GWCE weighting factor τ_0. Following the general guideline mentioned in Sect. 2.2, we initially take τ_0 equal to the linear friction factor, i.e. $\tau_0 = 0.0001$. Using this value, errors in the surface elevation initially increase in refining from mesh h_1 to mesh h_2; see Table 3. Beyond mesh h_2, the surface elevation solutions begin to converge slowly with mesh refinement, but at best only first-order in the L^1 and L^∞ error norms (this is also the case for the velocity solutions). By increasing τ_0 by an order of magnitude, the error in surface elevation for mesh h_1 increases slightly from what was previously obtained, however convergence is now observed with all further mesh refinements albeit at less than optimal rates—again only around first-order in both surface elevation and velocity in refining from mesh h_3 to h_4. By increasing τ_0 again by an order of magnitude ($\tau_0 = 0.01$ now), nearly optimal convergence rates are finally observed in the L^1 and L^∞ error norms for the surface elevation and in L^1 for the velocity solutions. The first-order convergence observed for the velocity solutions in L^∞ may be due to the way the open ocean boundary condition is enforced as the maximum element errors consistently occur in elements near this boundary. We remark that the improved performance of the models observed by increasing τ_0 (up to a point) is consistent with previous observations made relating τ_0 to local mass errors, i.e., local mass errors, which are an indicator of local solution errors, generally decrease when τ_0 is increased; see [15].

Having established a suitable value for τ_0, we now compare the errors and convergence rates of the CG and DG models. The results for the three bathymetric profiles are summarized in Tables 4, 5 and 6. It can be observed that error levels for both of the models are relatively small at these levels of resolution, however the DG errors are consistently lower than the CG errors, generally by an order of magnitude. Additionally, optimal second-order convergence is observed for the DG surface elevation and velocity solutions in both L^1 and L^∞ for the three test cases. However, as noted above, the CG solutions only exhibit first-order convergence rates in L^∞ for the velocity. The remaining CG convergence rates are near optimal, although there does appear to be a slight degradation in the convergence rates with the introduction of spatially varying bathymetry, e.g. CG convergence rates for the surface elevation in L^1 for refinement from h_3 to h_4 are 1.91, 1.86, and 1.77 for the $n = 0$, 1, and 2 bathymetry test cases, respectively.

Although the errors for both of the models are quite low at these levels of resolution, one can still do a simple cost versus accuracy analysis using the above data. For example, for the $n = 0$ test case, the CG L^∞ errors in surface elevation as a function of h can be approximated by:

$$E(h)_{CG} \approx e^{1.92\log h - 22.85}. \tag{3.3}$$

Similarly, the CG run times as a function of h can be estimated by the relation:

$$T(h)_{CG} \approx e^{-1.76\log h + 20.38}. \tag{3.4}$$

Using the first relationship, the mesh spacing h required for the CG solution to achieve the same level of error as the $h_1 = 7500$ mesh DG solution is $h \approx 3000$; see Fig. 5. Using (3.4), this gives an estimated CG run time of $T_{CG} \approx 556$; see Fig. 5. Thus, the CG model would take about 1.5 times longer than the DG model to achieve the same level of error in the surface elevation for this test case. Indeed, a simple numerical test verifies this estimate where a mesh with $h = 3000$ gives an error in surface elevation of 5.7810×10^{-4} (compare to the h_1 L^∞ DG error of Table 4 and takes 1.6 times longer to run than the h_1 DG simulation. Estimates of the CG run times required to achieve the DG error levels of the first two meshes are computed in a similar manner and are summarized in Table 7. It can be seen that in the

Table 3 ADCIRC-CG errors and convergence rates as a function of τ_0

ADCIRC-CG									
τ_0	Mesh	L^∞ Error		Conv. rate		L^1 Error		Conv. rate	
		ξ	u	ξ	u	ξ	u	ξ	u
10^{-4}	h_1	1.9117×10^{-3}	5.0851×10^{-2}	–	–	9.0982×10^{-4}	1.0931×10^{-2}	–	–
	h_2	2.4885×10^{-3}	2.5071×10^{-2}	−0.3804	1.0203	1.2762×10^{-3}	4.3624×10^{-2}	−0.4883	1.3252
	h_3	1.8801×10^{-3}	1.2150×10^{-2}	0.4045	1.0451	1.1790×10^{-3}	2.0087×10^{-3}	0.1143	1.1188
	h_4	9.0946×10^{-4}	6.1010×10^{-3}	1.0477	0.9938	5.9531×10^{-4}	8.9221×10^{-4}	0.9859	1.1708
10^{-3}	h_1	3.4074×10^{-3}	2.2568×10^{-2}	–	–	1.4222×10^{-3}	2.6922×10^{-3}	–	–
	h_2	8.4169×10^{-4}	1.2162×10^{-2}	2.0173	0.8919	4.0096×10^{-4}	9.9530×10^{-4}	1.8266	1.4356
	h_3	2.8325×10^{-4}	6.3533×10^{-3}	1.5712	0.9368	1.5865×10^{-4}	3.9903×10^{-4}	1.3376	1.3186
	h_4	1.3095×10^{-4}	3.1956×10^{-3}	1.1130	0.9914	6.9034×10^{-5}	1.5515×10^{-4}	1.2004	1.3629
10^{-2}	h_1	3.4179×10^{-3}	1.7295×10^{-2}	–	–	1.3192×10^{-3}	2.4400×10^{-3}	–	–
	h_2	8.9072×10^{-4}	9.3977×10^{-3}	1.9401	0.8799	3.2599×10^{-4}	7.6839×10^{-4}	2.0168	1.6670
	h_3	2.3366×10^{-4}	4.8297×10^{-3}	1.9306	0.9604	8.2312×10^{-5}	2.2328×10^{-4}	1.9856	1.7830
	h_4	6.2735×10^{-5}	2.4356×10^{-3}	1.8970	0.9877	2.1882×10^{-5}	6.3205×10^{-5}	1.9114	1.8208

Table 4 ADCIRC-CG and ADCIRC-DG errors and convergence rates for the flat bathymetry ($n = 0$) case

ADCIRC-CG

Bath.	Mesh	Absolute L^∞ error		Conv. rate		Absolute L^1 error		Conv. rate	
Type		ξ	u	ξ	u	ξ	u	ξ	u
$n = 0$	h_1	3.4179×10^{-3}	1.7295×10^{-2}	–	–	1.3192×10^{-3}	2.4400×10^{-3}	–	–
	h_2	8.9072×10^{-4}	9.3977×10^{-3}	1.9401	0.8799	3.2599×10^{-4}	7.6839×10^{-4}	2.0168	1.6670
	h_3	2.3366×10^{-4}	4.8297×10^{-3}	1.9306	0.9604	8.2312×10^{-5}	2.2328×10^{-4}	1.9856	1.7830
	h_4	6.2735×10^{-5}	2.4356×10^{-3}	1.8970	0.9877	2.1882×10^{-5}	6.3205×10^{-5}	1.9114	1.8208

ADCIRC-DG

Bath.	Mesh	Absolute L^∞ error		Conv. rate		Absolute L^1 error		Conv. rate	
Type		ξ	u	ξ	u	ξ	u	ξ	u
$n = 0$	h_1	5.7185×10^{-4}	1.8290×10^{-3}	–	–	3.8273×10^{-4}	1.0795×10^{-3}	–	–
	h_2	1.3346×10^{-4}	4.6262×10^{-4}	2.0885	1.9831	9.8798×10^{-5}	2.7108×10^{-4}	1.9538	1.9935
	h_3	3.2727×10^{-5}	1.1618×10^{-4}	2.0386	1.9935	2.5064×10^{-5}	6.7869×10^{-5}	1.9789	1.9979
	h_4	8.0874×10^{-6}	2.9096×10^{-5}	2.0167	1.9974	6.3107×10^{-6}	1.6977×10^{-5}	1.9897	1.9992

Table 5 ADCIRC-CG and ADCIRC-DG errors and convergence rates for the linear bathymetry ($n = 1$) case

ADCIRC-CG

Bath.	Mesh	Absolute L^∞ error		Conv. rate		Absolute L^1 error		Conv. rate	
Type		ξ	u	ξ	u	ξ	u	ξ	u
$n = 1$	h_1	4.6505×10^{-3}	1.0266×10^{-2}	–	–	1.3156×10^{-3}	2.0020×10^{-3}	–	–
	h_2	1.3560×10^{-3}	5.9361×10^{-3}	1.7780	0.7903	3.4063×10^{-4}	5.9192×10^{-4}	1.9494	1.7580
	h_3	3.8905×10^{-4}	3.2524×10^{-3}	1.8013	0.8680	9.0002×10^{-5}	1.6688×10^{-4}	1.9202	1.8266
	h_4	1.1153×10^{-4}	1.7724×10^{-3}	1.8025	0.8758	2.4796×10^{-5}	4.5822×10^{-5}	1.8598	1.8647

ADCIRC-CG

Bath.	Mesh	Absolute L^∞ error		Conv. rate		Absolute L^1 error		Conv. rate	
Type		ξ	u	ξ	u	ξ	u	ξ	u
$n = 1$	h_1	5.6973×10^{-4}	1.2210×10^{-3}	–	–	2.9715×10^{-4}	2.6201×10^{-4}	–	–
	h_2	1.5028×10^{-4}	3.4353×10^{-4}	1.9226	1.8296	7.5456×10^{-5}	6.6428×10^{-5}	1.9775	1.9798
	h_3	3.8591×10^{-5}	9.1131×10^{-5}	1.9613	1.9144	1.9003×10^{-5}	1.6725×10^{-5}	1.9894	1.9898
	h_4	9.7833×10^{-6}	2.3470×10^{-5}	1.9799	1.9571	4.7704×10^{-6}	4.1949×10^{-6}	1.9940	1.9953

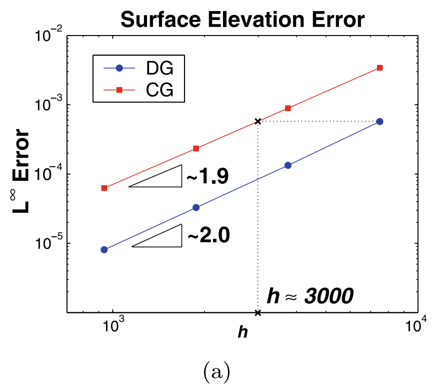
Table 6 ADCIRC-CG and ADCIRC-DG errors and convergence rates for the quadratic bathymetry ($n = 2$) case

ADCIRC-CG

Bath.	Mesh	Absolute L^∞ error		Conv. rate		Absolute L^1 error		Conv. rate	
Type		ξ	u	ξ	u	ξ	u	ξ	u
$n = 2$	h_1	3.7214×10^{-3}	1.4205×10^{-2}	–	–	6.7052×10^{-4}	2.2034×10^{-3}	–	–
	h_2	1.1474×10^{-3}	7.1825×10^{-3}	1.6975	0.9838	1.7417×10^{-4}	6.2764×10^{-4}	1.9448	1.8117
	h_3	3.4472×10^{-4}	4.2391×10^{-3}	1.7348	0.7607	4.7113×10^{-5}	1.7648×10^{-4}	1.8863	1.8304
	h_4	1.0292×10^{-4}	2.3519×10^{-3}	1.7439	0.8499	1.3838×10^{-5}	4.7835×10^{-5}	1.7675	1.8834

ADCIRC-CG

Bath.	Mesh	Absolute L^∞ error		Conv. rate		Absolute L^1 error		Conv. rate	
Type		ξ	u	ξ	u	ξ	u	ξ	u
$n = 2$	h_1	1.3239×10^{-3}	3.0708×10^{-3}	–	–	5.0031×10^{-4}	1.0183×10^{-3}	–	–
	h_2	3.4749×10^{-4}	8.5656×10^{-4}	1.9297	1.8420	1.2470×10^{-4}	2.5506×10^{-4}	2.0043	1.9972
	h_3	8.9044×10^{-5}	2.2647×10^{-4}	1.9644	1.9192	3.1103×10^{-5}	6.3796×10^{-5}	2.0034	1.9993
	h_4	2.2539×10^{-5}	5.8238×10^{-5}	1.9821	1.9593	7.7663×10^{-6}	1.5954×10^{-5}	2.0018	1.9996



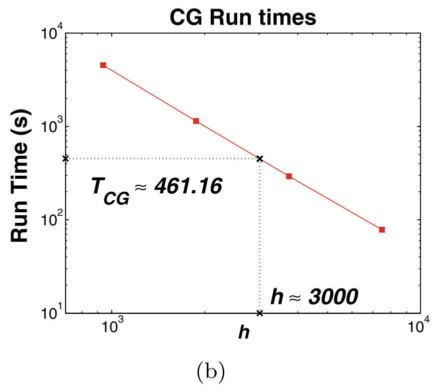


Fig. 5 (**a**) CG and DG errors and (**b**) CG serial run times as a function of h

Table 7 Estimated ratio Γ of the CG to DG run times required for ADCIRC-CG to achieve errors equivalent to the ADCIRC-DG errors of the first two meshes

Bath.	Mesh	L^∞ Error		L^1 Error	
Type		ξ	u	ξ	u
$n = 0$	h_1	1.50	19.94	0.87	0.68
	h_2	1.83	83.16	0.94	0.88
$n = 1$	h_1	2.31	25.21	1.14	2.14
	h_2	2.75	113.12	1.30	2.61
$n = 2$	h_1	0.85	6.39	0.38	0.62
	h_2	1.07	28.54	0.45	0.75

majority of cases the DG model is actually more efficient ($\Gamma > 1$) than the CG model, in terms of the amount of compute time it takes to achieve a specified error level. In particular, it can be noted that the Γ factors of Table 7 related to the velocity errors in L^∞ are very large because the CG velocity solutions only exhibit first-order convergence in that case.

3.2 Parallel Performance

In this subsection, we investigate the parallel speedup, efficiency, and scalability of the two codes. We begin with some preliminary definitions. The parallel speedup S is defined as the ratio of the time of a single processor run T_s for a problem of fixed size over the time of a parallel run T_p for a problem of the same size. The parallel efficiency E is defined as the ratio of the speedup over the number of processors (cores).

For the parallel testing, we use the four meshes of the previous subsection along with two successive, uniform refinements of mesh h_4 denoted h_5 and h_6 (approximately 37,000 and 150,000 elements, respectively). We solve the $n = 0$ test case described in the previous subsection—this time solving the full nonlinear equations. Two scenarios are investigated. In the first, we fix the problem size, using mesh h_6, and solve the problem on 1, 4, 16, 64, 256 and 1024 processors. In the second, we change the problem size while increasing the number of processors such that the number of elements per processor remains (approximately) constant. This is investigated by solving mesh h_1 on a single processor, mesh h_2 on 4 processors, etc. Again, we test on up to 1024 processors.

Table 8 Parallel speedup: Wall clock times, parallel efficiency, and DG to CG wall clock time ratio for a fixed problem size

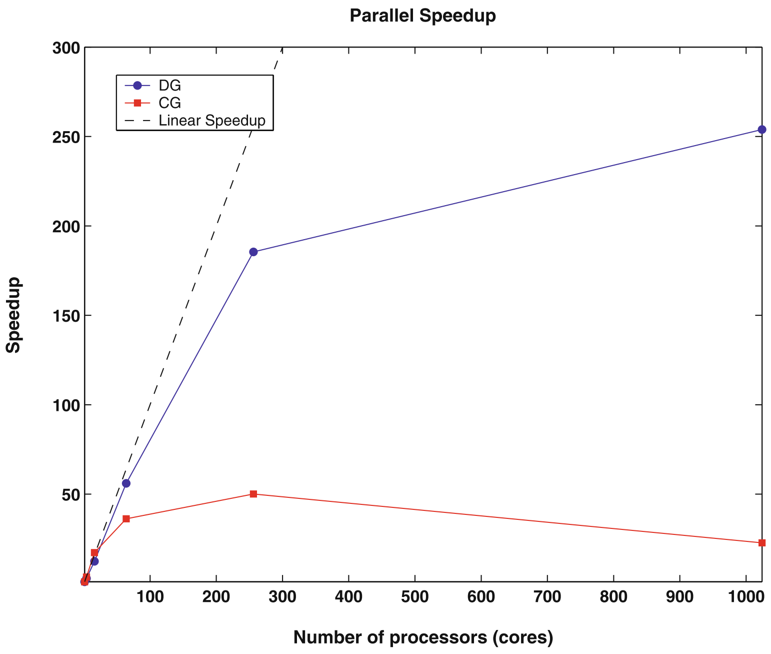
CPUs	ADCIRC-CG		ADCIRC-DG		Γ
	T_p (s)	E (%)	T_p (s)	E (%)	
1	8913	...	37829	...	0.24
4	2428	90.65	13234	71.46	0.19
16	515	108.17	3040	77.77	0.17
64	246	56.61	675	87.57	0.36
256	178	19.56	204	72.44	0.87
1024	392	2.22	149	24.79	2.63

The results of the first scenario are summarized in Table 8 and Fig. 6. For a problem of this size (again mesh h_6 has approximately 150,000 elements), the ADCIRC-CG code shows reasonably good parallel speedup and efficiency (>55%) on up to 64 processors. Beyond this point, it shows significant drops in parallel efficiency—down to 20% and 2% for 256 and 1024 processors, respectively. Furthermore, in going from 256 to 1024 processors, the parallel run times actually increase by a factor greater than 2 because message-passing has now become the dominant factor of the run time.

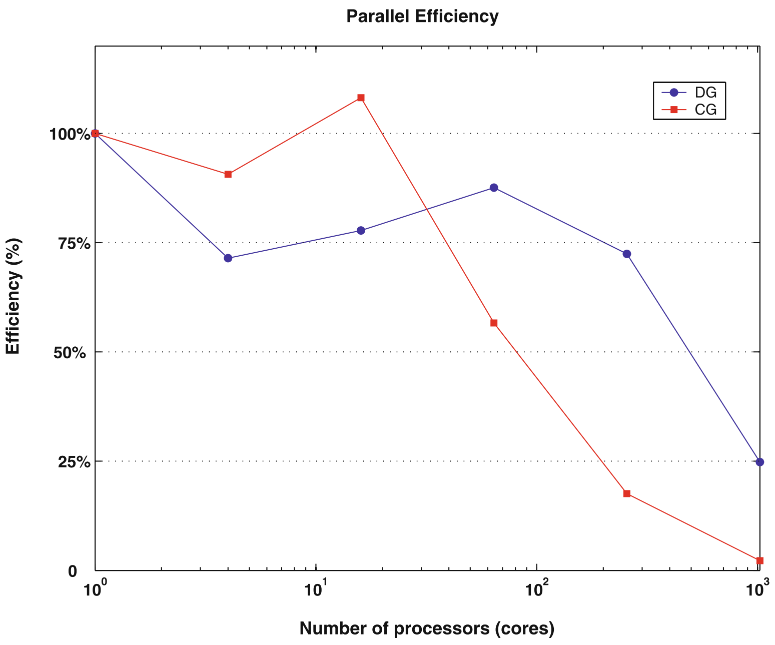
On the other hand, ADCIRC-DG maintains close to linear speedup with efficiencies greater than 70% on up to 256 processors; see Fig. 6. On 1024 processors, parallel efficiency shows a significant drop; however, as opposed to the ADCIRC-CG code, a decrease in actual compute time is still observed in going from 256 to 1024 processors. Of course, it is important to point out that in terms of wall clock time, the ADCIRC-CG code is faster than the ADCIRC-DG code on up to 256 processors for a problem of this size. However, with 1024 processors the DG code is more than 2.5 times faster. For smaller problems, this cross–over point in efficiency is observed at a smaller number of processors, while for larger problems, it is observed at a greater number of processors. Thus, for very large problem sizes, the number of processors required before the DG code is more efficient, in terms of wall clock time, than the CG code could be prohibitively large. We note that the problem size examined here (≈150,000 elements) is a typical size used for many regional coastal ocean studies as will be examined in the next section. Both models show improved parallel efficiency results as problem size increases.

With respect to the parallel efficiencies, we note that the ADCIRC-CG code shows a jump in parallel efficiency (exhibiting super linear speedup) in going from 4 to 16 processors. This corresponds to going from approximately 36,000 elements to 9,000 elements per processor, i.e. going from a mesh the size of h_5 to a mesh the size of h_4 on each processor. Similarly, the ADCIRC-DG code exhibits a significant jump in efficiency in going from 16 to 64 processors (mesh h_4 to h_3). These results, which are most likely due to cache effects, are consistent with the observations made in the previous subsection for the serial run times (see the comments to this effect in Sect. 3.1).

The results of the second scenario, where the problem size changes in proportion to the number of processors such that each processor has approximately 150 elements, are summarized in Table 9. With each case, the DG code shows greater parallel efficiency than the CG code, with the differences in parallel efficiency becoming increasingly greater as the number of processors increases—for example, with 64 processors the CG and DG codes show parallel efficiencies of 8.38% and 48.28%, respectively. It can also be observed that the DG code is more efficient in terms of wall clock time on 64, 256, and 1024 processors in this case. These trends are due to fact that the overhead time associated with message passing for the CG code grows at a greater rate than the DG code as the number of processors increases.

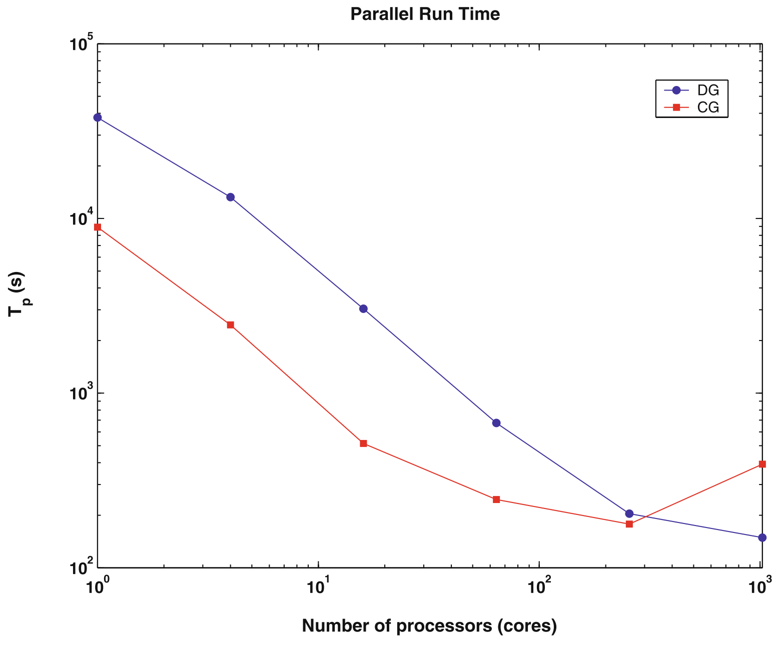


(a)



(b)

Fig. 6 (Color online) (**a**) Parallel speedup, (**b**) efficiency, and (**c**) run times for ADCIRC-CG (*red squares*) and ADCIRC-DG (*blue circles*)



(c)

Fig. 6 (*Continued*)

Table 9 Parallel scaling: Wall clock times, parallel efficiency, and DG to CG wall clock time ratio for problem size scaled with number of processors

CPUs	ADCIRC-CG		ADCIRC-DG		Γ
	T_p (s)	E (%)	T_p (s)	E (%)	
1	88	···	337	···	0.26
4	144	61.11	482	69.92	0.30
16	455	19.34	610	55.25	0.75
64	1050	8.38	698	48.28	1.50
256	1875	4.69	920	36.63	2.04
1024	4499	1.96	1385	24.33	3.25

This is because the CG code requires *global* communication among all processors to solve the resulting system of equations, while the DG code only requires *local* communication between neighboring subdomains.

3.3 Modeling Tides in Chesapeake Bay

With the final test case, we compare the two models on a full-scale application. The finite element mesh that is used is shown in Fig. 7(a). The mesh covers a large portion of the western north Atlantic with open ocean boundaries introduced at a longitude of 63° W and a latitude of 28° N. The remaining boundaries along the east coast of the US and Canada are treated as land or no-normal flow boundaries. The mesh has approximately 160,000 elements, around 95% of which are in the area of Chesapeake Bay, which will serve as

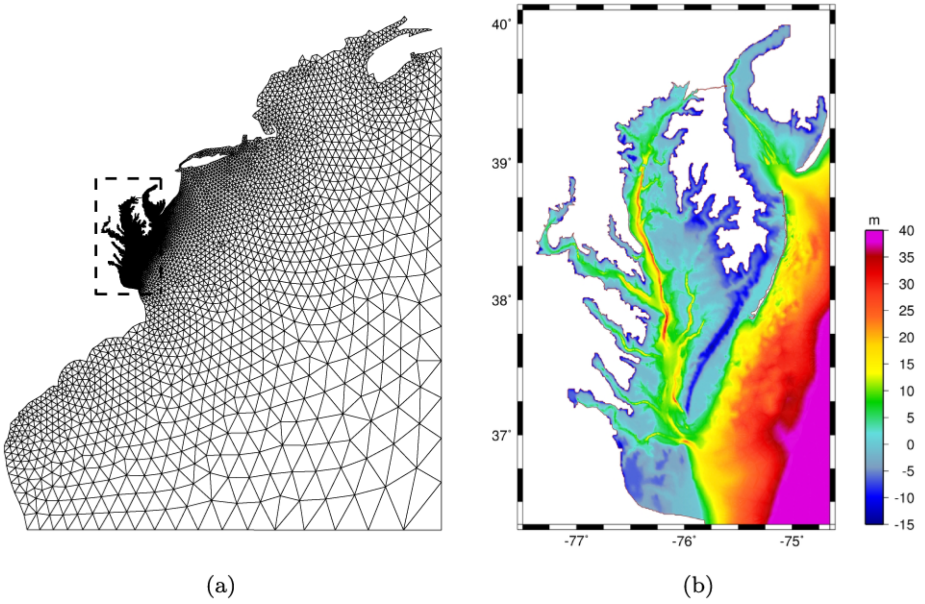


Fig. 7 (**a**) Finite element mesh used for tidal simulations in Chesapeake Bay; (**b**) Closeup of the Chesapeake Bay area showing bathymetric/topographic contours

the area of interest for our model comparison; see Fig. 7(b). Note that the use of such a large scale unstructured mesh for this area, with open ocean boundaries set predominately in the deep ocean, as opposed to closer to the area of interest, simplifies boundary condition specification and mitigates a number of physical and numerical issues related to boundary conditions with little additional computational overhead.

Located throughout the area of Chesapeake Bay, the National Ocean Service (NOS) has several tidal data stations with harmonic constituent data, which generally consist of amplitude and phase data for 37 different tidal constituents; see [27]. Model tidal amplitudes and phases for the O_1, K_1, Q_1, M_2, N_2, S_2, and K_2 constituents—obtained by harmonically analyzing the numerical solutions over a period of time—will be compared to NOS data at 10 different stations located throughout the area. The locations of these stations, along with their corresponding NOS station identification numbers, are shown in Fig. 8.

As boundary conditions along the open ocean boundaries, tidal signals are constructed using amplitudes and phases of the seven constituents mentioned above, which are obtained from a tidal data base of the East Coast that was created using ADCIRC-CG; see [26]. Internal tidal potential forcing is also included in the models using these same tidal constituents (this forcing term is accounted for in the force term **F** of (2.2)). Simulations begin from initial conditions of rest and the tidal forcing is applied gradually, building up to full amplitude over a period of 5 days. Simulations that are run to obtain model tidal amplitudes and phases are run for a period of 45 days on 256 processors. The surface elevation solutions at each of the stations are harmonically analyzed using data every 300 s over the last 30 days of the simulation. Simulations to obtain timings on 64, 256, and 1024 processors are only run for a period of 5 days because of computational time restrictions on the parallel cluster.

Average amplitude and phase errors over the 10 stations shown in Fig. 8 are summarized in Table 10 for each of the seven harmonic constituents listed above. Graphical displays

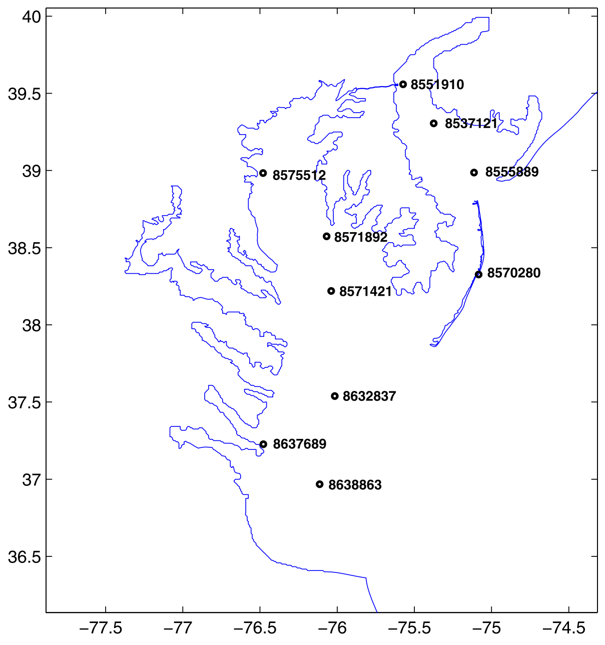


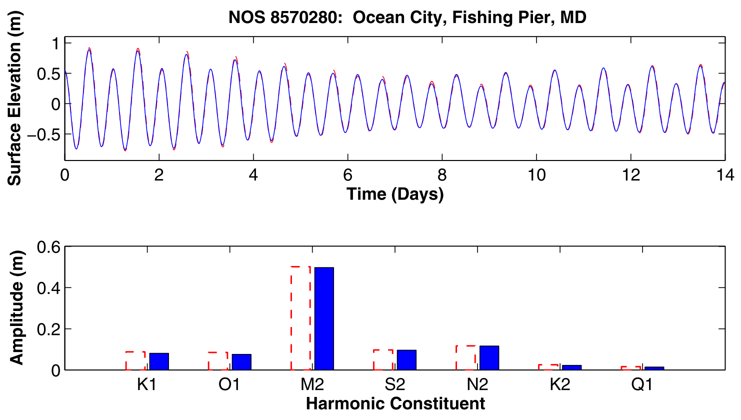
Fig. 8 NOS station locations used for tidal validation of the models

Table 10 Average amplitude and phase errors for the Chesapeake Bay test case

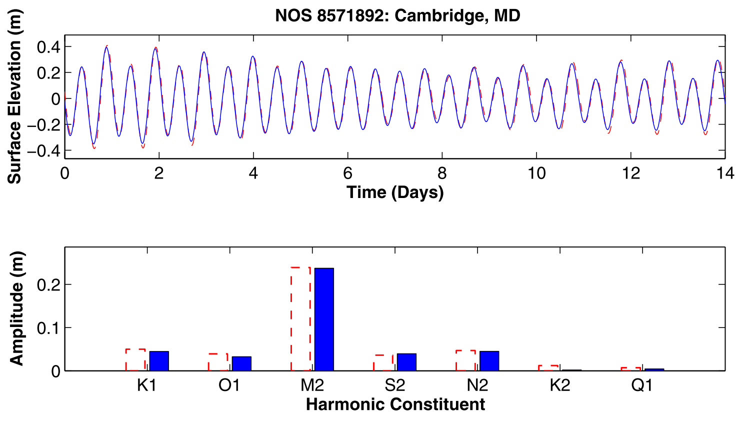
	Tidal constituent:	K_1	O_1	M_2	S_2	N_2	K_2	Q_1
ADCIRC-CG	Amplitude (m)	0.0064	0.0066	0.0378	0.0103	0.0084	0.0119	0.0017
	Phase (°)	0.2049	0.8303	0.7432	0.1890	0.1244	1.3887	0.1972
ADCIRC-DG	Amplitude (m)	0.0074	0.0085	0.0396	0.0074	0.0098	0.0115	0.0022
	Phase (°)	0.1935	0.9399	0.7524	0.1685	0.1263	1.5025	0.2539

of the NOS data compared to the ADCIRC-CG and ADCIRC-DG results are shown for 3 of these stations in Figs. 9 and 10, respectively (the tidal signals shown in these figures are reconstructed from the seven harmonic constituents). As can be observed from these Figures and Table 10, both of the models accurately reproduce the tides in this area (in some cases the field data and computed results are indistinguishable graphically). The amplitude and phase errors of the two models are very similar.

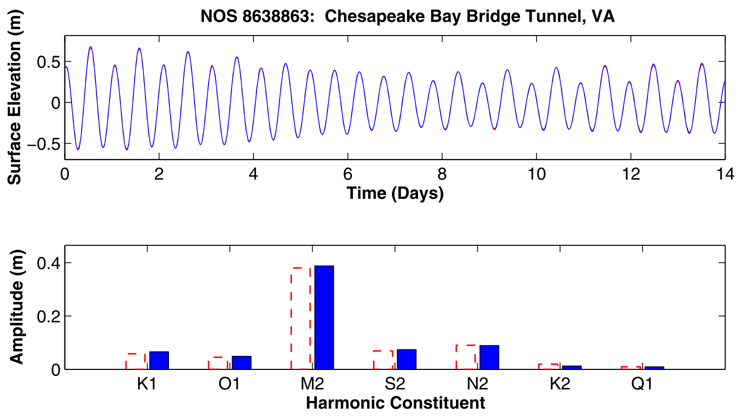
Wall clock times for simulations on 64, 256, and 1024 processors are reported in Table 11. When using 64 processors (≈ 2500 elements/processor) the CG code is approximately 1.8 times faster than the DG code. However the DG code is more efficient in terms of both parallel efficiency and run time when using 256 and 1024 processors—by factors of 1.9 and 9.0, respectively. We note that in this case, in contrast to the problem of the previous section, which was of similar size, the DG code is more efficient than the CG code in terms of compute time on 256 processors. It was observed that this was due to the fact that the solver used in ADCIRC-CG typically required a greater number of iterations (≈ 15 as opposed to 7), and thus a greater number of global communications, for this problem than the previous one.



(a)

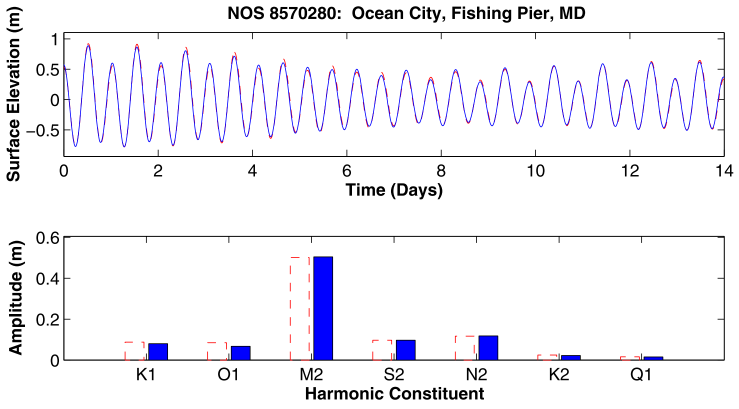


(b)

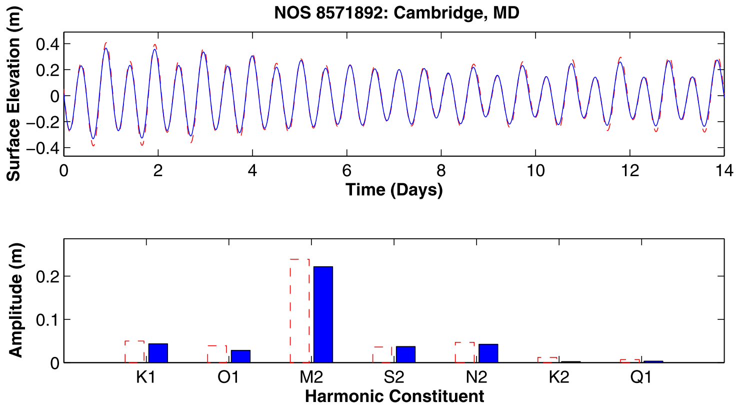


(c)

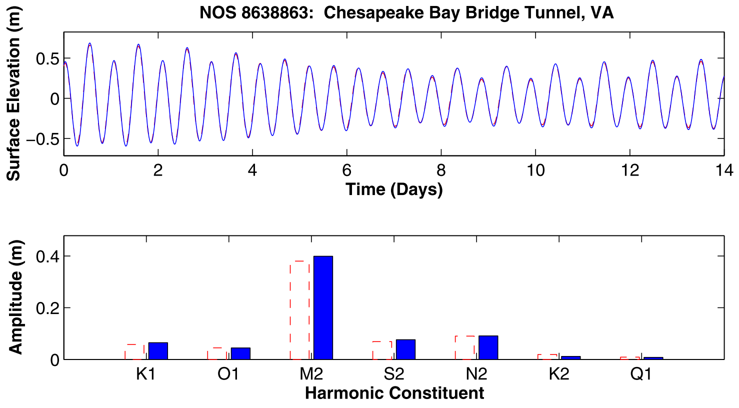
Fig. 9 (Color online) Comparison of field data (*dashed red lines*) and ADCIRC-CG results (*solid blue lines*) for the time history of tidal signals and tidal amplitudes at three select stations



(a)



(b)



(c)

Fig. 10 (Color online) Comparison of field data (*dashed red lines*) and ADCIRC-DG results (*solid blue lines*) for the time history of tidal signals and tidal amplitudes at three select stations

Table 11 Wall clock times and efficiencies for 64, 256, and 1024 processors for Chesapeake Bay

CPUs	ADCIRC-CG		ADCIRC-DG		Γ
	T_p (s)	E (%)	T_p (s)	E (%)	
64	13535	...	24967	...	0.54
256	14109	23.98	7318	85.29	1.93
1024	37709	2.24	4183	37.30	9.01

4 Discussion and Conclusions

In this paper, we have presented a detailed comparative study of two SWE finite element models—a GWCE-based CG model and a recently developed DG model. The models were compared on three sets of problems in terms of accuracy, convergence rates, serial and parallel run times, and efficiency. While the DG model was observed to be less efficient in terms of compute time on serial machines (generally by a factor of 4) because of the larger number of degrees of freedom associated with the method, it was also observed that the CG solution errors were greater than the corresponding DG solution errors—generally by an order of magnitude. A simple cost versus accuracy analysis based on data obtained from a number of runs demonstrated that the mesh spacing required by the CG model to achieve the same error levels of the DG model for a given mesh typically resulted in a greater compute time than the DG model. Additionally, the DG model consistently displayed better convergence rates than the CG model. In terms of parallel performance, it was observed that the DG model generally showed greater parallel efficiency both for problems of fixed size and for problem sizes that increased in proportion with the number of processors. However, for a problem size of approximately 150,000 elements, the CG model was more efficient than the DG model in terms of actual compute time on up to 256 processors. Results for problems of different fixed sizes (not shown here) demonstrated that this crossover point in efficiency occurred at a lower number of processors for smaller problems and a higher number of processors for larger problems. Thus, which model would be more efficient for a given parallel simulation ultimately depends on both the size of the problem and the computing resources one has available. A full-scale application in the Chesapeake Bay area demonstrated that both models accurately reproduce the tides in that area and displayed very similar errors with the DG model showing significantly better parallel efficiency and resulting in lower compute times on both 256 and 1024 processors in that case.

Finally, we have limited our focus here to piecewise linear approximations over triangular elements for the DG model. It should be mentioned that for certain classes of problems significant gains in efficiency can be gained using p-refinement for DG methods as demonstrated in [17]. The DG parallelization strategy described here can also be used without change for higher-order DG methods due to the fact that the size of the element stencil does not change for a given order. Additionally, using a combination of triangular and quadrilateral elements in the DG model, which can be implemented with relative ease due to the fact that C^0 continuity is not required across element edges, could make the DG model even more efficient than the CG model given that the element to node ratio would go down with the introduction of quadrilateral elements. Lastly, in future work we will investigate the performance of CG and DG models for full three-dimensional simulations.

Acknowledgements This work was supported by National Science Foundation grants OCI-0746232 and OCI-0749015 and by the Office of Naval Research, Award Number: N00014-06-1-0285.

References

1. Aizinger, V., Dawson, C.: A discontinuous Galerkin method for two-dimensional flow and transport in shallow water. Adv. Water Resour. **25**, 67–84 (2002)
2. Atkinson, J.H., Westerink, J.J., Luettich, R.A.: Two-dimensional dispersion analyses of finite element approximations to the shallow water equations. Int. J. Numer. Methods Fluids **45**, 715–749 (2004)
3. Bey, K.S., Patra, A., Oden, J.T.: hp-version discontinuous Galerkin methods for hyperbolic conservation laws—a parallel adaptive strategy. Int. J. Numer. Methods Eng. **38**, 3889–3908 (1995)
4. Biswas, R., Devine, K.D., Flaherty, J.E.: Parallel, adaptive finite element methods for conservation laws. Appl. Numer. Math. **14**, 255–283 (1994)
5. Blain, C.A., Massey, T.C.: Application of a coupled discontinuous–continuous Galerkin finite element shallow water model to coastal ocean dynamics. Ocean Model. **10**, 283–315 (2005)
6. Buffa, A., Hughes, T.J.R., Sangalli, G.: Analysis of a multiscale discontinuous galerkin method for convection-diffusion problems. SIAM J. Numer. Anal. **44**, 1420–1440 (2006)
7. Dawson, C., Sun, S., Wheeler, M.F.: Compatible algorithms for coupled flow and transport. Comput. Methods Appl. Mech. Eng. **193**, 2565–2580 (2004)
8. Dawson, C., Westerink, J.J., Feyen, J.C., Pothina, D.: Continuous, discontinuous and coupled discontinuous–continuous Galerkin finite element methods for the shallow water equations. Int. J. Numer. Methods Fluids **52**, 63–88 (2006)
9. Eskilsson, C., Sherwin, S.J.: A triangular spectral/hp discontinuous Galerkin method for modelling 2D shallow water equations. Int. J. Numer. Methods Fluids **45**, 605–623 (2004)
10. Foreman, M.G.G.: A comparison of tidal models for the southwest coast of Vancouver Island. In: Celia, M. (ed.) Computational Methods in Water Resources: Proceedings of the VII International Conference. Elsevier, Amsterdam (1988)
11. Gray, W.G.: A finite element study of tidal flow data for the North Sea and English Channel. Adv. Water Resour. **12**, 143–154 (1989)
12. Gray, W.G., Lynch, D.R.: Time–stepping schemes for finite element tidal model computations. Adv. Water Resour. **1**, 83–95 (1977)
13. Hughes, T.J.R., Scovazzi, G., Bochev, P.B., Buffa, A.: A multiscale discontinuous Galerkin method with the computational structure of a continuous Galerkin method. Comput. Methods Appl. Mech. Eng. **195**, 2761–2787 (2006)
14. Kinnmark, I.P.E.: The Shallow Water Wave Equations: Formulations, Analysis and Application. Lecture Notes in Engineering, vol. 15. Springer, Berlin (1986)
15. Kolar, R.L., Westerink, J.J., Cantekin, M.E., Blain, C.A.: Aspects of nonlinear simulations using shallow-water models based on the wave continuity equation. Comput. Fluids **23**, 523–528 (1994)
16. Kolar, R.L., Gray, W.G., Westerink, J.J.: Boundary conditions in shallow water models–an alternative implementation for finite element codes. Int. J. Numer. Methods Fluids **22**, 603–618 (1996)
17. Kubatko, E.J., Westerink, J.J., Dawson, C.: hp discontinuous Galerkin methods for advection dominated problems in shallow water flow. Comput. Methods Appl. Mech. Eng. **196**, 437–451 (2006)
18. Karypis, G., Kumar, V.: 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 (1998)
19. Le Provost, C., Vincent, P.: Finite element for modeling ocean tides. In: Parker, B. (ed.) Tidal Hydrodynamics, pp. 41–60. Wiley, New York (1991)
20. Le Roux, D.Y., Lin, C.A., Staniforth, A.: A semi-implicit semi-Lagrangian finite-element shallow-water ocean model. Mon. Weather Rev. **128**, 1384–1401 (2000)
21. Li, H., Liu, R.: The discontinuous Galerkin finite element method for the 2D shallow water equations. Math. Comput. Simul. **56**, 223–233 (2001)
22. Luettich, R.A., Westerink, J.J., Scheffner, N.W.: ADCIRC: An advanced three-dimensional circulation model for shelves, coasts and estuaries, Report 1: Theory and methodology of ADCIRC-2DDI and ADCIRC-3DL. In: Dredging Research Program Technical Report DRP-92-6, US Army Engineers Waterways Experiment Station, Vicksburg, MS (1992)
23. Lynch, D.R., Gray, W.G.: Analytic solutions for computer flow model testing. J. Hydraul. Div. **104**, 1409–1428 (1978)
24. Lynch, D.R., Gray, W.G.: A wave equation model for finite element tidal computations. Comput. Fluids **7**, 207–228 (1979)
25. Lynch, D.R., Werner, F.E., Molines, J.M., Fornerino, M.: Tidal dynamics in a coupled ocean/lake system. Estuar. Coast. Shelf Sci. **31**, 319–343 (1990)
26. Mukai, A.Y., Westerink, J.J., Luettich, R.A., Mark, D.: Eastcoast 2001, a tidal constituent database for Western North Atlantic, Gulf of Mexico, and Caribbean Sea. TR ERDC 01-x, US Army Engineer, Engineer Research and Development Center, Vicksburg, MS (2001)

27. National Oceanic and Atmospheric Administration: Tides and currents database. Online, available: http://tidesandcurrents.noaa.gov/
28. Schwanenberg, D., Kiem, R., Kongeter, J.: Discontinuous Galerkin method for the shallow water equations. In: Cockburn, B., Karniadakis, G.E., Shu, C.-W. (eds.) Discontinuous Galerkin Methods, pp. 289–309. Springer, Heidelberg (2000)
29. Walters, R.A.: A model for tides and currents in the English Channel and southern North Sea. Adv. Water Resour. **10**, 138–148 (1987)
30. Werner, F.E., Lynch, D.R.: Harmonic structure of English Channel/Southern Bight tides from a wave equation simulation. Adv. Water Resour. **12**, 121–142 (1989)
31. Westerink, J.J., Luettich, R.A., Baptista, A.M., Scheffner, N.W., Farrar, P.: Tide and storm surge predictions in the Gulf of Mexico using a wave-continuity equation finite element model. In: Spaulding, M.L. (ed.) Estuarine and Coastal Modeling: Proceedings of the 2nd International Conference. American Society of Civil Engineers, New York (1992)