Available online at www.sciencedirect.com
SCIENCE DIRECT
Journal of Computational Physics 216 (2006) 179–194
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Self-adaptive time integration of flux-conservative equations with sources

Y.A. Omelchenko, H. Karimabadi *

SciberQuest Inc., Computational Physics, 777 South Highway 101, Suite 108, Solana Beach, CA 92075-2623, United States

Received 22 April 2005; received in revised form 14 December 2005; accepted 18 December 2005
Available online 25 January 2006

Abstract

We present a novel, flux-conserving, asynchronous method for the explicit time integration of multi-scale, flux-conservative partial differential equations with source terms. Unlike the conventional explicit and implicit integration schemes, it is based on a discrete-event simulation paradigm, which describes time advance in terms of increments to physical quantities and causality rules rather than time stepping. This method exerts self-adaptive control over local update rates of solution by predicting and correcting changes to simulation variables in accordance with local physical scales. The discrete-event simulation paradigm is independent of the underlying spatial mesh and thus can be incorporated into block-structured and unstructured mesh refinement techniques. The effectiveness and robustness of the new method is demonstrated on a number of one-dimensional, uniform mesh models based on diffusion–convection–reaction equations. The event-driven integration reduces numerical approximation errors due to large local time derivatives, prevents explosive numerical instabilities in locally super-Courant calculations and automatically reduces the CPU overhead associated with stiff terms and inactive regions in computation space.

Keywords: Flux-conservative; Conservation laws; Multi-scale; PDE; Discrete-event simulation; Asynchronous; Adaptive

1. Introduction

Multiple temporal and spatial scales commonly occur in large-scale, non-linear physical systems. Methods for their numerical treatment overcome enormous challenges and drive advances in simulation technology [1]. Many scientific and engineering applications involve solution of time-dependent flux-conservative partial differential equations (PDEs). Numerical techniques for such systems commonly assume that the governing equations and simulation variables are properly discretized in space on a suitable spatial mesh composed of discrete control elements (also referred to as "cells"). A mesh can be either uniform or adaptive to solution spatial scales. The discretized equations are normally "time stepped", i.e., solution values in all mesh elements are synchronously updated in discrete time intervals determined by the choice of a constant or variable global

* Corresponding author. Tel.: +1 8587937063; fax: +1 8587775684.
E-mail address: homak@sciberquest.com (H. Karimabadi).

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.12.008

time increment, Δt. Numerical time-stepping integration techniques for PDEs generally fall into two categories: *explicit* and *implicit* schemes. In the explicit schemes, all unknown variables are computed at the current time level from quantities already available (computed in the past). The implicit schemes employ unknown variables for evaluating (some or all) spatial terms of the underlying finite-difference equations, which leads to a set of coupled linear or non-linear equations for each discrete solution variable. Below we briefly discuss limitations of these traditional formulations.

The Neumann stability analysis of explicit schemes leads to the famous Courant–Friedrichs–Levy stability criterion, $\Delta t < (\Delta x/V)_{\min}$ (also referred to as the Courant or CFL condition), where V and Δx are the local physical speed and mesh size, respectively. This condition guarantees that the explicit time integration always resolves the fastest transient change in the system state. In practice, physical systems can develop multiple time scales either due to different (multi-physics) processes or spatial inhomogeneities. Numerical difficulties associated with multi-physics issues are usually addressed by using time-averaged descriptions for fast processes or sub-cycling disparate processes with different update frequencies [1]. On the other hand, multiple time scales associated with spatial inhomogeneity may be a consequence of either a specific problem set-up (e.g., non-uniform external factors, initial conditions) or inherent non-linearities in the governing equations. As a result, to avoid explosive numerical instabilities, explicit time-stepping integration techniques have to employ the minimum time-step size typically determined by the most restrictive (global) CFL condition in the system. In contrast to this, unconditionally stable implicit methods introduce some sort of numerical smoothing, which enables use of larger time increments. Alternatively, implicit–explicit formulations [2] treat only stiff (fast evolving) terms of the model equations implicitly and approximate the other terms in an explicit fashion.

With the advent of fast iterative Krylov–Newton–Schwartz algorithms, implicit (as well as implicit–explicit) formulations have quickly become important tools for studying multi-scale phenomena described by non-linear PDEs, e.g., radiation diffusion [3–6], reaction–diffusion [7,8], magnetohydrodynamics [9] and radiation hydrodynamics [10] systems. In order to correctly describe the global system evolution, implicit solvers employ physics-based strategies for selecting suitable time-step sizes that are commensurate with the dominant time scale in the problem. CPU-efficient time-step sizes can often be obtained by applying a Courant condition [4]. Despite recent progress in this field, large-scale implicit models demonstrate significant CPU costs resulting from a large number of iterations necessary to converge non-linearities to within small tolerances. More importantly, implicit integrations conducted with large time increments are unable to correctly represent local physical phenomena with characteristic process frequencies higher than the inverse of the time-step size. These issues render them inefficient for modeling long-time behavior of large-scale systems (e.g., the Earth's magnetosphere, climate models, laser–plasma interactions, etc.), where one typically needs to resolve strongly varying time scales with equal accuracy over long periods of time.

To address the above problems, a number of asynchronous explicit time-stepping approaches have been developed for flux-conservative systems. These can be classified into the following groups:

1. *Adaptive mesh refinement*. Berger and Oliger [11] proposed the adaptive mesh refinement (AMR) technique for block-structured meshes. In this technique, refined (daughter) meshes overlap regions covered by the coarser (parent) mesh so that the global (composite) mesh is made of a hierarchy of nested levels of logically rectangular patches. Each patch is still updated with a constant time increment restricted by the patch cell size and the maximum velocity magnitude. In the case of explicit algorithms for conservation laws, the inter-patch synchronization involves only corrections to the coarse mesh solution: the coarse-mesh data is replaced by the volume-weighted average of the fine-mesh data and the fluxes at the fine–coarse interfaces are corrected by interpolating the fine-mesh solution values. This technique has successfully been applied to simulate physical processes in various inhomogeneous media [12–15].
2. *Adaptive time refinement*. This method [16,17] implements local time stepping by allowing solution values in different mesh elements (cells) to be updated with different time increments, which are usually selected to be fractions of the global time-step size in order to satisfy local CFL conditions. Special care is taken to preserve flux conservation across cell interfaces in all updates. This "telescopic" method requires ad hoc strategies for calculation of the global time increment at each time step. They may become difficult to implement for highly inhomogeneous or strongly non-linear large-scale systems, where fast solution updates may

invalidate initial assumptions made for slower evolving quantities (e.g., make their local CFL conditions more restrictive). In addition, this technique does not provide built-in recipes for the treatment of inactive regions.

3. *Domain decomposition*. This approach has been used to dynamically track action potential (described by a parabolic equation) in an excitable heterogeneous biological medium [18]. The solution is integrated by assigning individual time increments (multiples of the minimum time step size) to different sub-domains. The sub-domains are sorted by their update times with the use of a priority queue and an explicit integration method advances the solution in each "active" sub-domain. Boundary information is obtained by time interpolation of fluxes at sub-domain interfaces. This procedure violates flux consistency at sub-domain boundaries and therefore, an additional implicit procedure is applied to obtain an accurate global solution. Another asynchronous domain decomposition technique is based on a combination of implicit and explicit schemes [19].
4. *Variable time-stepping method.* In this method [20] time-step size is a continuous function of space. The order of cell updates is determined through a binary tree scheduling mechanism and individual time increments are chosen to follow the fastest local time scales. As in the previous (domain decomposition) approach, this method uses time interpolation for flux computation, which violates local conservative properties of the underlying model.
5. *Adaptive time–space discretization*. This technique (see [21] and references therein) introduces an extra finite dimension to the computational model: a separate *temporal* mesh. A discontinuous Galerkin solution is constructed in time–space by adapting the duration of each element to the local degree of spatial mesh refinement. In addition to introducing an extra time dimension, this method does not seem to present clear conditions for enforcing stability and defining efficient adaptive strategies on irregular space–time domains.

The existing adaptive techniques can significantly reduce CPU load in many problems of interest. Among these techniques, block-structured AMR is the most developed and widely used approach. This powerful and robust methodology significantly increases computational efficiency for many evolution problems [12–15]. In AMR applications, the maximum time-step size in each refinement patch is still limited by the local CFL condition, which may result in very restrictive configurations for problems where adaptation has to be made to non-linear physical processes with irregular or fast changing temporal and spatial scales (i.e., turbulent or strongly coupled reactive systems). More importantly, at the beginning of every global time step, all asynchronous time-stepping algorithms require global information for calculating the appropriate set of time steps to be taken in each patch (or element). To summarize, both the explicit and implicit time-stepping schemes share two important features that generally limit their efficiency:

1. At every time level, they update the whole system state without due regard to actual *changes* to simulation variables. In other words, the time-stepping models utilize the CPU resources uniformly, including parts of the computation space where no significant modification of system state variables (a.k.a *information*) is produced during the calculation. Moreover, the time-stepping methods are not "intelligent" enough to predict *when* and *where* the underlying model will generate such changes. As a result, to avoid numerical instabilities, the explicit schemes have to "blindly" assume that meaningful information is *always* available for processing within the stencil neighborhood of each cell (i.e., a group of cells used to approximate the spatial terms for a given cell). This lack of "intelligence" inevitably results in imposing a CFL-type restriction on the time step size.
2. In multi-scale problems (where the solution rate of change varies considerably throughout the system), time stepping inevitably results in advancing the system state with different degrees of accuracy in different parts of computation domain.

Recently a new time integration approach has been applied to equations of non-linear elastodynamics [22]. It is based on a discrete spacetime form of Hamilton's variational principle. Compared to the variable time-stepping method [20], this algorithm permits the selection of independent time steps in each mesh element so that the local time steps do not bear an integral relation to each other. The time advance is done by organizing computational elements into a priority queue based on their precomputed update times. The algorithm

permits updating each subsystem with a frequency dictated by its natural timescale, subject to solvability of the local time steps [22]. However, this approach is only applicable to Hamiltonian systems in which the Lagrangian is expressible as a sum of component sub-Lagrangians.

In this paper we present a self-adaptive, flux-conserving, asynchronous approach to the explicit time integration of flux-conservative equations with source terms. Unlike the asynchronous variational integration technique [22], this method is based on discrete-event simulation (DES) methodology [23–27], which is applicable to more general dynamical systems. It offers the following advantages over the traditional time-driven simulation (TDS) techniques:

1. *Robustness (stability and accuracy)*. The DES integration algorithm exerts adaptive, local control over the numerical update rate of solution. This reduces integration errors associated with large time derivatives and prevents explosive numerical instabilities.
2. *Performance efficiency*. Event-driven updates advance spatially distributed physical quantities *asynchronously*, in accordance with their natural temporal scales. This removes the global CFL restriction and eliminates the CPU overhead associated with idle (informationless) computation. In this sense, the concept of DES is similar to ideas widely exploited in video and audio data compression algorithms.

The remainder of the paper is structured as follows. In Section 2, we summarize basic ideas of the event-driven methodology for the time integration of flux-conservative equations (solution of PDEs in non-conservation form is not discussed in this paper). The DES method is described in detail in Section 3. Its application to a non-linear diffusion–advection–reaction equation in one dimension is presented in Section 4. We discuss results from test problems in Section 5 and present our conclusions in Section 6.

2. Discrete-event simulation

Discrete-event (or event-driven) simulations have their origin in operations research, management science, war games and telecommunications [24,25]. More recently, DES methodology has been extended to modeling simple continuous systems [26,27] and electrostatic plasma kinetic interactions [23].

In a discrete-event simulation, the temporal evolution of a global system is modeled by allowing the system to "jump" from one state to another at discrete moments in simulated time upon the occurrence of an "event", which represents an effective unit of information [24,25]. For example, typical events in a network simulation may represent arrival of a message at some node, forwarding a message to another node, etc. From a programming standpoint, each event is a simulation object characterized by a *process function* (a method for changing the object state) and a *time stamp* (a point in simulated time when the process function is to be called). Sequential DES programs typically operate with the following data structures [23–25]:

1. *The state variables*. These variables describe the instantaneous state of the system.
2. *The event list (queue)*. This is a priority queue containing all pending events that have been scheduled but have not yet been executed. The event list preserves causality by sorting all events by their time stamps in non-decreasing order so that the time stamp of the top event is always at least as large as the current simulation time.
3. *The simulation clock*. This data structure corresponds to the main loop of the traditional time-stepping simulation. The clock indicates how far in simulated time the discrete-event simulation has progressed. The actual loop of the DES program (*the engine*) repeatedly removes the top (smallest time-stamped) event from the event list and processes that event (Fig. 1). Processing an event may result in retracting previously scheduled events (i.e., removing them from the list of pending events) and scheduling new events.

In DES terminology, each spatial cell is assigned a number of discrete, time-dependent "states" characterized by different simulation variables. Each state "schedules" a unique event by delaying the execution of its process function until its predicted process time (Fig. 1). If the delay is infinite the state is said to be "idle". Therefore, at any point in simulated time the number of pending events for each cell can never exceed the total number of its states.

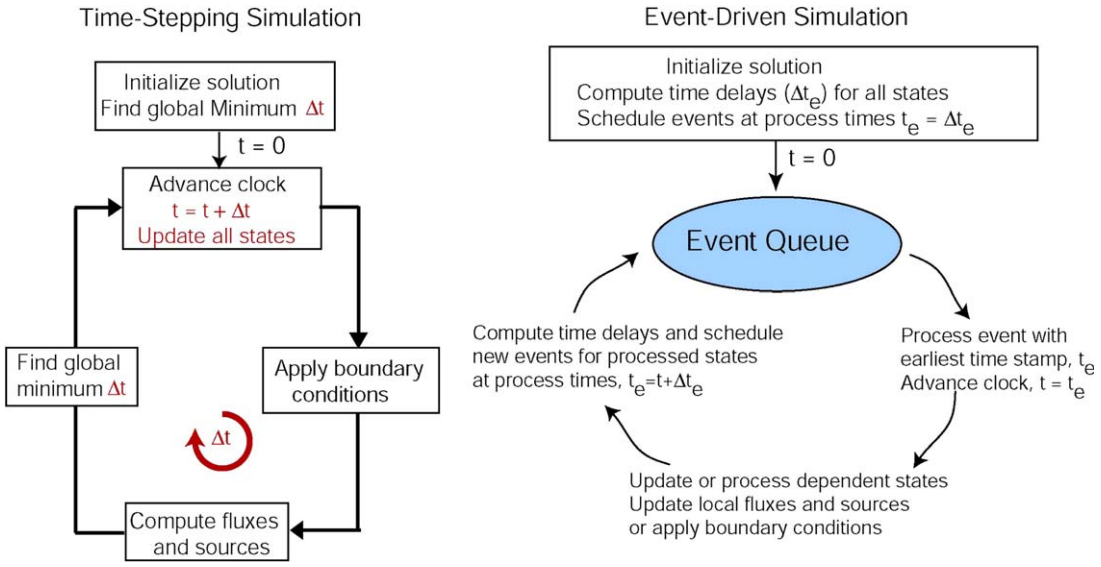

Fig. 1. Control flows in typical time-stepping and event-driven simulation models.

A phenomenological discrete-event approach was developed to simulate linear advection, diffusion and wave propagation [26]. That work independently introduced some of the concepts presented in this paper but did not offer a general methodology for the solution of flux-conservative equations.

The concept of event-driven time integration can be explained as follows. Suppose a quantity, f (e.g., a discrete field value or a particle) is integrated in time by solving an ordinary differential equation, $\mathrm{d}f/\mathrm{d}t = R(f)$. Then its individual measure of time advance, $\Delta t_f = \Delta f/|\mathrm{d}f/\mathrm{d}t|$ can be expressed in terms of a physically meaningful information unit, Δf and its current rate-of-change, $\mathrm{d}f/\mathrm{d}t$. In the philosophy of DES, f is considered to have been "processed" only when a change to its previous state (a.k.a. information) is found to exceed the minimum amount of information Δf (same as the "quantum" value in [26,27]). As we show below, this postulation leads to an algorithm where the accuracy of time integration can always be adaptively adjusted in accordance with local time scales. As a bonus, informationless ("idle") parts of the system (where $\mathrm{d}f/\mathrm{d}t \approx 0$) "warp" through simulated time to the extent that their local time increments become infinite.

Naturally, a successful application of this philosophy to the time integration of general flux-conservative equations must satisfy one important caveat: asynchronous time advance must produce a consistent (converging) numerical solution and preserve conservation laws embodied by the underlying equations. Higher order explicit time-stepping schemes (e.g., Runge–Kutta algorithms) are usually based on some sort of a predictor–corrector approach, i.e., they pre-advance physical quantities in time ("predictor") and carry out the final step with time-centered spatial terms ("corrector"). These techniques are known to produce accurate solutions when the time-step size is small enough to meet the numerical stability and accuracy constraints (usually it is set to be a fraction of the global CFL-limited value but additional constraints may arise when modeling systems with sources).

In contrast to the traditional time-stepping methodology, the DES paradigm is based on applying a "self-adaptive" predictor–corrector scheme to each state of the global computational model. For instance, given the current rate of change $\mathrm{d}f/\mathrm{d}t$ for state f at time t, the DES *predictor* schedules a corresponding event, e, to be executed at the future time, $t_e = t + \Delta f/[\mathrm{d}f/\mathrm{d}t]$, at which f is predicted to change by the "target" increment Δf. Note that Δf can be either preset to be constant during the simulation or automatically selected by the predictor algorithm based on the current state condition (see below). At this stage one assumes that f "ballistically" evolves in simulated time (i.e., $\mathrm{d}f/\mathrm{d}t$ remains unchanged). The DES corrector is responsible for modifying the predicted rate of change and processing ("waking up") the state earlier than its scheduled process time, should the conditions used to predict its ballistic trajectory undergo a significant change. Therefore, the state update rate is constrained by a set of threshold values and dynamic causality rules, i.e., the system

employs "self-intelligence" to predict and (if necessary) correct its behavior, as opposed to being forcefully "time-stepped". This "self-adaptive" paradigm guarantees that self-consistent computation always takes place *when* required by the governing laws. It can effectively prevent explosive numerical instabilities associated with growing errors in explicit schemes due to inconsistent updates.

Computational systems based on hyperbolic or parabolic equations are often modeled with equations in conservation form, i.e., expressed in terms of source and flux functions. As mentioned above, numerical preservation of the underlying conservation laws is an important aspect of any numerical algorithm for such systems. The synchronous explicit schemes automatically update adjacent cells with identical fluxes. On the other hand, implicit solvers commonly employ iterative techniques, which may not always satisfy finite differences to within round-off errors. The presence of residual errors is equivalent to introducing extraneous fluxes not accounted for by the governing equations and may result in a severe degradation of conservative properties unless the convergence criterion explicitly takes this into account. By the same token, non-conservative, asynchronous methods for the time integration of flux-conservative equations may generate inconsistent solutions. In the next section we present a technique that effectively resolves this issue.

3. Flux-conservative DES algorithm

3.1. Problem formulation

For simplicity, in this paper we focus on solving a one-dimensional, scalar, flux-conserved equation with a source term:

$$\frac{\partial f}{\partial t}+\frac{\partial}{\partial x}\Phi(f,x)=S(f,x). \tag{1}$$

Here $f(x)$ is the solution of interest on the domain, $x\in[0,L]$, $\Phi(f,x)$ and $S(f,x)$ are the flux and source functions, respectively. The spatial discretization of Eq. (1) on a uniform cell-centered mesh, $x_i=(i+1/2)\Delta x$ ($\Delta x=L/N$, $i=0,\ldots,N-1$) is shown in Fig. 2.

In general, we assume that a linear combination of Dirichlet and Neumann boundary conditions is applied at the domain "ghost" cells ($i=-1,N$):

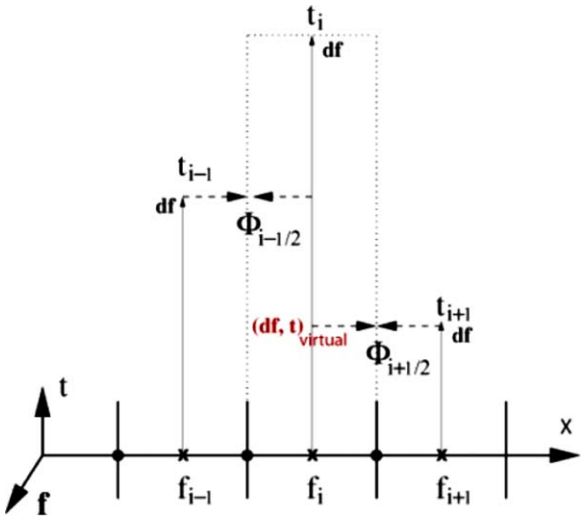

Fig. 2. The spatial discretization and asynchronous time integration of Eq. (1). The solution variables f_i and source term values S_i are defined at integer (cell-centered) locations, x_i. The flux variables $\Phi_{i+1/2}$ are defined at half-integer (node) positions, $x_{i+1/2}$. Note that the "virtual" (synchronization) update of solution f_i and flux $\Phi_{i+1/2}$ at time t_{i+1} is a result of processing an update event in neighboring cell $i+1$.

$$f_{-1} = 2r_{\mathrm{L}} + c_{\mathrm{L}} f_0, \quad f_N = 2r_{\mathrm{U}} + c_{\mathrm{U}} f_{N-1}, \tag{2}$$

where r_{L}, r_{U}, c_{L}, c_{U} are free parameters and subscripts L, U refer to the lower and upper domain boundaries, respectively. Let f_i represent the local solution in cell i, last updated at time t_i. Then the solution $\tilde{f}_i$ at the current simulation time, t_{clock} can be obtained with a forward Euler integrator:

$$\tilde{f}_i = f_i + R_i \tau, \tag{3}$$

where $\tau = t_{\mathrm{clock}} - t_i$ is the local time increment and

$$R_i = S_i - (\Phi_{i+1/2} - \Phi_{i-1/2})/\Delta x. \tag{4}$$

At the start-up time, $t_{\mathrm{clock}} = 0$ the simulation model is initialized as follows: (i) all system variables $\{f_i, S_i, \Phi_{i+1/2}, R_i\}$ are initialized and $t_i = t_{\mathrm{clock}}$, (ii) the proper events are scheduled for execution in the active interior cells (see Section 3.4). In addition, for each state f_i, we introduce a "flux capacitor" variable, δf_i ($\delta f_i(t_{\mathrm{clock}} = 0) = 0$). This auxiliary variable keeps track of the net change to f_i occurred since the time of its most recent scheduling operation ("predictor"). We also assume that each state, in principle, can be assigned an individual "target" increment, Δf_i^{tr}. An algorithm for selecting local target increments is described in Section 4.2.

An event-driven integration (Fig. 1) is carried out by continuously applying the following three computation phases (Sections 3.2–3.4) until the global simulation clock is advanced past the simulation finish time, T_{sim} ($t_{\mathrm{clock}} > T_{\mathrm{sim}}$).

3.2. Event processing

1. Select the event with the smallest timestamp, t_{e} and let $t_{\mathrm{clock}} = t_{\mathrm{e}}$. Identify the "active" cell p, corresponding to the event being processed.
2. Obtain the up-to-date solution $\tilde{f}_p$ using Eq. (3): $\tilde{f}_p = f_p + R_p(t_{\mathrm{clock}} - t_p)$. Let $t_p = t_{\mathrm{clock}}$.
3. Zero out the flux capacitor variable: $\delta f_p = 0$.
4. For each neighboring cell $s = p \pm 1$ call the event synchronization procedure (Section 3.3, see also Fig. 2).
5. Schedule a new event for cell p (Section 3.4).

3.3. Event synchronization

1. If this cell s is a not boundary cell, then proceed to step 2. Otherwise, apply an appropriate boundary condition (Eq. (2)), execute step 5 of this phase and return.
2. Let $\Delta f \equiv R_s(t_{\mathrm{clock}} - t_s)$. Update the local flux capacitor (Eq. (3)): $\delta\tilde{f}_s = \delta f_s + \Delta f$.
3. Update the local solution (Eq. (3)): $\tilde{f}_s = f_s + \Delta f$. Let $t_s = t_{\mathrm{clock}}$.
4. If $|\delta\tilde{f}_s| \geqslant \Delta f_s^{\mathrm{tr}}$, then retract the corresponding pending event, execute steps 3–5 of the process function for state s (see Section 3.2) and return. Otherwise, proceed to step 5.
5. Update the flux $\tilde{\Phi}_{(p+s)/2}$ across the interface between this cell (s) and the cell that initiated the synchronization call (p), using the up-to-date solution values, $\tilde{f}_p$ and $\tilde{f}_s$.
6. Update $\tilde{R}_s$ (Eq. (4)) using the latest approximations, $\tilde{\Phi}_{(p+s)/2}$ and $\tilde{S}_s$.

3.4. Event scheduling

1. Update $\tilde{R}_p$ (Eq. (4)) using the latest approximations, $\tilde{\Phi}_{p\pm1/2}$ and $\tilde{S}_p$.
2. Compute the local target increment Δf_p^{tr} (optional, see Section 4.2).
3. Compute the next event time delay, $\Delta t_p = \Delta f_p^{\mathrm{tr}}/|\tilde{R}_p|$.
4. Schedule the next event for state p at the process time, $t_{\mathrm{e}} = t_{\mathrm{clock}} + \Delta t_p$.

It should be emphasized that the DES time-integration algorithm possesses the following important properties:

1. For any given pair of adjacent cells i, $i+1$, the *same* discrete (piecewise constant in time) flux $\Phi_{i+1/2}$ is used to compute time advances in both cells at *any* point in simulation time. As a result, the numerical flux integrals are always conserved to within computer precision round-off errors (provided all cells have been integrated to the same time).
2. A proper spatial range of locally synchronous corrections in the neighborhood of an active cell is self-adaptively determined through a sequence of synchronization updates (Section 3.3). Note that step 4 of Section 3.3 for a cell being synchronized takes into account changes to its local solution due to updates in the neighboring cells occurred since this cell was last scheduled. For the end cells in the synchronization range, this procedure may result in obtaining fluxes through the different faces at different times. However, since this flux inaccuracy is restricted to $\sim O(\Delta f)$, it can only lead to a numerical approximation error $\sim O(\Delta t)$, which is consistent with the temporal approximation order of Eq. (3).
3. The synchronization updates compute the fluxes through the synchronization faces only. In multi-dimensional problems the majority of CPU time is spent on flux computation. Therefore, the synchronization-driven (“virtual”) calculations will be much less CPU consuming than the event-driven updates.

4. Diffusion–convection–reaction equation

Stiff (multi-scale) diffusion–convection–reaction equations form foundations of many computational models in various scientific disciplines [2,3,7,20]. As we show below, the self-adaptive DES algorithm is a viable alternative to the time-stepping techniques for the numerical integration of such systems.

4.1. Flux discretization

We represent the flux function in Eq. (1) as a sum of non-linear diffusion and linear advection components:

$$\Phi(f,x) = \Phi_{\mathrm{d}} + \Phi_{\mathrm{a}}, \tag{5}$$

$$\Phi_{\mathrm{d}} = -D(f,x)\frac{\partial f}{\partial x}, \quad \Phi_{\mathrm{a}} = uf(x). \tag{6}$$

Here $u > 0$ is a constant advection velocity and $D(f,x)$ is a generally variable diffusion coefficient. The diffusion component of flux Φ_{d} is discretized in space with the usual central scheme:

$$\Phi_{\mathrm{d},i+1/2} = -D_{i+1/2}\frac{f_{i+1}-f_i}{\Delta x}, \quad D_{i+1/2} = D(f_i, f_{i+1}, x_{i+1/2}). \tag{7}$$

The advection flux Φ_{a} is handled via the first-order upwind scheme:

$$\Phi_{\mathrm{a},i+1/2} = uf_i. \tag{8}$$

Note that scheme (8) is known to be very diffusive and chosen here for its simplicity. More appropriate schemes will be considered in the future. Using Eqs. (5), (7) and (8) we rewrite Eq. (4) in the following form:

$$R_i = S_i + \left[\frac{D_{i-1/2}}{\Delta x^2} + \frac{u}{\Delta x}\right] f_{i-1} - \left[\frac{u}{\Delta x} + \frac{D_{i-1/2}+D_{i+1/2}}{\Delta x^2}\right] f_i + \frac{D_{i+1/2}}{\Delta x^2} f_{i+1}. \tag{9}$$

A standard stability analysis of Eqs. (3) and (9) results in the following *local* CFL condition:

$$\tau_i \leqslant \tau_i^{\mathrm{CFL}} = \frac{1}{u/\Delta x + (D_{i-1/2}+D_{i+1/2})/\Delta x^2}. \tag{10}$$

For the interior cells located next to the domain boundaries ($x_i = \Delta x/2, L - \Delta x/2$) expression (10) is modified by taking into account boundary conditions (2):

$$\tau_0 \leqslant \tau_0^{\mathrm{CFL}} = \frac{1}{(1-c_{\mathrm{L}})u/\Delta x + (1-c_{\mathrm{L}})D_{-1/2}/\Delta x^2 + D_{1/2}/\Delta x^2}, \tag{11}$$

$$\tau_{N-1} \leqslant \tau_{N-1}^{\mathrm{CFL}} = \frac{1}{u/\Delta x + (1-c_{\mathrm{U}})D_{N-1/2}/\Delta x^2 + D_{N-3/2}/\Delta x^2}. \tag{12}$$

Incidentally, the CFL condition guarantees positivity of all coefficients in the explicit scheme described by Eqs. (3) and (9). Therefore, if the solution is constrained by conditions (10)–(12), then the above scheme satisfies the maximum (monotonicity) principle, which automatically preserves solution non-negativity, $f_i \geqslant 0$.

4.2. Target increment control

As mentioned above, the algorithm described in Sections 3.2–3.4 is not limited to the use of a constant value of Δf. Different strategies for determining local values of Δf may influence both the global accuracy and CPU performance of DES integration. In particular, for models with physically smooth, non-negative solutions, one should be concerned with the preservation of solution monotonicity and non-negativity (we define the latter as $f_i \geqslant -\varepsilon$, where ε is a positive constant taken to be smaller than the typical numerical precision round-off error). As was previously noted [23], the satisfaction of the local CFL condition is not *necessary* for numerical stability as long as one properly *controls* propagation of discrete information on the mesh. Indeed, in the extreme case one can set event delays, $\tau \propto \infty$ for certain states, effectively deactivating them "on-the-fly" without causing stability problems. However, in order to preserve both the solution monotonicity and numerical non-negativity, and take a full advantage of the local (asynchronous) nature of DES updates, one needs to specify a mechanism for selecting proper local target increments Δf_p^{tr} (step 2 of Section 3.4). Below we describe a technique that satisfies these criteria.

Let $f_{\min}, f_{\max}$ the minimum and maximum solution values computed in the stencil-wide neighborhood of the active cell p. Then step 2 of the event scheduling procedure (Section 3.4) may be carried out as follows ($\Delta f_{\max}$, ω_{CFL}, $\lambda_{\min}$, $\omega_{\min}$ are control parameters described below):

1. Compute $\Delta f_p^{\mathrm{CFL}} = |R_p|\omega_{\mathrm{CFL}}\tau_p^{\mathrm{CFL}}$.
2. If $\Delta f_p^{\mathrm{CFL}} < \varepsilon$, then let $\Delta t_p = \infty$ and return (do not proceed with steps 3 and 4 in Section 3.4). Otherwise, let $\Delta f_p^{\mathrm{tr}} = \Delta f_p^{\mathrm{CFL}}$.
3. Compute $\lambda = \min(f_{\min}/\Delta f_p^{\mathrm{tr}}, \lambda_{\min})$.
4. If $\lambda > 1$, then let $\Delta f_p^{\mathrm{tr}} = \max[\Delta f_p^{\mathrm{tr}}, \min(f_{\min}/\lambda, \omega_{\mathrm{lim}}(f_{\max} - f_{\min}))]$.
5. Compute $\Delta f_p^{\mathrm{tr}} = \min(\Delta f_{\max}, \Delta f_p^{\mathrm{tr}})$.

Here $\Delta f_{\max} > 0$ is the maximum target increment allowed in the simulation, ω_{CFL} is the characteristic Courant number ($0 < \omega_{\mathrm{CFL}} \leqslant 1$), and $\lambda_{\min}$, ω_{lim} are the factors ($\lambda_{\min} > 1, 0 < \omega_{\mathrm{lim}} \leqslant 1$) that permit local Courant numbers larger than ω_{CFL} without compromising the solution positivity and monotonicity (steps 3–4). Step 2 of the above algorithm deactivates idle states ($\Delta f_p^{\mathrm{CFL}} < \varepsilon$). It should be noted that even though the above algorithm provides a satisfactory performance in our test cases (see below), it may not represent the optimum strategy for enabling larger (super-Courant) time increments for all problems. However, it does maintain solution positivity and monotonicity even when the local CFL condition is violated.

5. DES examples

We validate the DES algorithm on several test models based on Eqs. (1), (5), (6). These examples represent ubiquitous physical phenomena and demonstrate the power and versatility of the DES paradigm. In each example we compare the event-driven solution, f_{DES} to either its time-stepped counterpart, f_{TDS} (obtained with the same temporal discretization scheme) or the known analytical solution, f_{a}. For a simulation of duration, T_{sim} with the total number of cells, N_{cell} ($\Delta x = L/N_{\mathrm{cell}}$) and total number of processed events, N_{event} we introduce the following effective measure of DES performance efficiency:

$$Q = \Delta t_{\mathrm{DES}}^{\mathrm{eff}}/\Delta t_{\mathrm{TDS}} \equiv \frac{N_{\mathrm{cell}} \times T_{\mathrm{sim}}}{N_{\mathrm{event}} \times \Delta t_{\mathrm{TDS}}}. \tag{13}$$

Here Δt_{TDS} is the constant time increment in the corresponding time-stepping (time-driven) simulation. DES incurs CPU overhead due to event queueing and synchronization operations. In one-dimensional problems considered in this paper the amount of computation per cell is relatively small compared to this overhead. As a result, the actual CPU speed-up factor, Q_{CPU} in our tests is somewhat lower, $Q_{\mathrm{CPU}} \approx Q/5$. However, we expect this ratio to significantly improve with further programming optimization and introduction of higher spatial dimensions.

Let u and v be discrete solutions to Eq. (1). As a measure of their quantitative closeness, we choose the relative L_2 norm error, $\sigma(u,v) = \|u - v\|_2/\|v\|_2$. Accordingly, we define three error metrics: $\sigma_{\mathrm{DES}} \equiv \sigma(f_{\mathrm{DES}},f_{\mathrm{a}})$, $\sigma_{\mathrm{TDS}} \equiv \sigma(f_{\mathrm{TDS}},f_{\mathrm{a}})$ and $\sigma \equiv \sigma(f_{\mathrm{DES}},f_{\mathrm{TDS}})$. All simulations are performed with double precision. Unless specifically stated otherwise, in DES runs we employ the monotonicity enforcing algorithm (Section 4.2) with the following control parameters:

$$\Delta f_{\max} = 10^{-3}, \quad \lambda_{\min} = 10, \quad \omega_{\lim} = 0.25, \quad \omega_{\mathrm{CFL}} = 1, \quad \varepsilon = 0.5 \times 10^{-14}. \tag{14}$$

The important configuration parameters for all test cases are summarized in Table 1. In DES runs, we introduce the local update rate, $\nu_{\mathrm{DES}}(x) = \Delta t_{\mathrm{TDS}}/\Delta t_{\mathrm{DES}}(x)$, where Δt_{TDS} is the time-step size used in the time-stepping integration and $\Delta t_{\mathrm{DES}}(x_i) \equiv \Delta t_i$ is the local DES time increment. Typically, we set $\Delta t_{\mathrm{TDS}} = \min \tau_i^{\mathrm{CFL}}$. The only exception is the "linear convection" case where Δt_{TDS} is assumed to be limited by special accuracy considerations (see below).

The solution profiles in all but one ("heat wave") cases are initialized to be Gaussian, $f(x, t = 0) = f_0 f_{\mathrm{G}}(x)$, $f_{\mathrm{G}}(x) = \exp(-(x - x_0)^2/d_0^2)$, with $f_0 = 1$, $d_0 = L/20$, $x_0 = L/2$ ($x_0 = L/5$ in the "linear convection" case). Homogeneous Dirichlet boundary conditions ($R_{\mathrm{L}} = R_{\mathrm{U}} = 0, C_{\mathrm{L}} = C_{\mathrm{U}} = -1$) are applied in the steady-state, "linear diffusion–reaction" problem (LD–LR) and homogeneous Neumann boundary conditions ($R_{\mathrm{L}} = R_{\mathrm{U}} = 0, C_{\mathrm{L}} = C_{\mathrm{U}} = 1$) are assumed in all other cases.

5.1. Steady-state, linear diffusion–reaction ($D = 10$, $S = 0.01$, $u = 0$)

Assuming homogeneous Dirichlet boundary conditions, the steady-state solution of the linear diffusion–reaction equation is easily recovered:

$$f_{\mathrm{a}}(x) = \frac{S}{2D}(Lx - x^2). \tag{15}$$

Fig. 3(a) shows a close match between the numerical DES and exact solutions. In this case, in the absence of inhomogeneity and non-linearity, DES is not expected to offer a performance advantage over explicit time stepping. However, the DES performance factor, $Q = 3.4$ is still above unity due to the ability of the self-adaptive algorithm to apply local time increments that exceed the CFL limited time-step size. Note that the DES integration is inherently robust with respect to numerical blow-up instabilities that take place in explicit time-stepping simulations when the time-step size exceeds the CFL limit. Fig. 3(b) illustrates this point by comparing two event-driven solutions obtained with different constant values of the target increment Δf. It shows that even for unreasonably large values of Δf, the DES integration still remains numerically stable, i.e., it produces a bounded (albeit noisy) solution that is still close to the exact one.

Table 1
Summary of configuration parameters for different test cases: (i) linear diffusion–reaction (LD–LR), (ii) non-linear diffusion (ND), (iii) linear convection (LC), (iv) non-uniform diffusion–linear convection (ND–LC), (v) linear diffusion–non-linear reaction (LD–NR or "heat wave")

	Δx	N_{cell}	T_{sim}	Δt_{TDS}
LD–LR	0.5	200	800.0	1.25e − 2
ND	5e − 3	200	25.0	2.5e − 3
LC	0.25	1200	50.0	2e − 2
ND–LC	5e − 3	200	1.25	1.25e − 5
LD–NR	0.00625–0.05	120–960	1.0	1e − 4

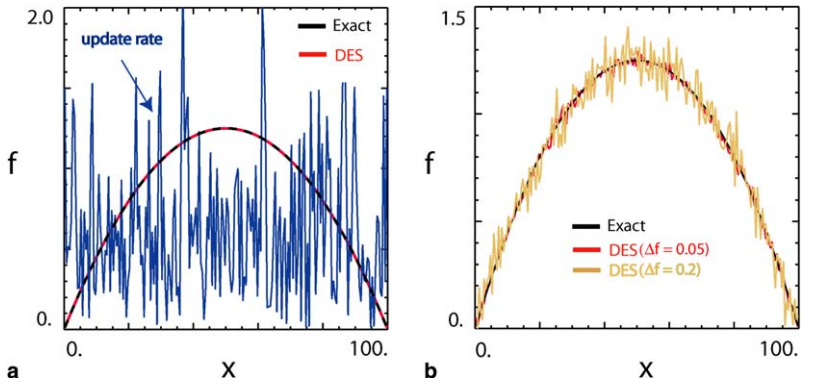

Fig. 3. The steady-state, linear diffusion–reaction (LD–LR) model: (a) the DES solution with $Q = 3.4$, $\sigma_{\mathrm{DES}} = 5 \times 10^{-4}$ (red), the exact solution (dashed black), the normalized DES solution update rate, $\Delta t_{\mathrm{CFL}}/\Delta t_{\mathrm{DES}}$ (blue); (b) two bounded DES profiles obtained with large, constant values of the target increment Δf (red and brown curves), the exact solution (black). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2. Non-linear diffusion ($D(f) = 5 \times 10^{-3} f^{3/2}$, $S = 0$, $u = 0$)

This problem is characterized by a non-linear dependence of the diffusion coefficient. In this case, the traditional explicit time-stepping techniques have to apply the smallest time-step size determined by the most restrictive CFL condition in the system, $\Delta t \leqslant \tau_{\mathrm{CFL}} = K\Delta x^2/D_{\max}$ ($K = 1/2$ for the forward Euler scheme), which may result in a numerically "stiff" integration for strongly non-linear systems. As a result, implicit techniques are often considered to be the only viable candidates for solving such equations. On the other hand, the DES algorithm proceeds free of the global CFL restriction by adaptively adjusting the local update rate of solution to the local time scale. This effectively alleviates the "stiffness" imposed by the system non-linearity. Fig. 4 further illustrates this point. Note that the update rate envelope matches the spatial variation in the diffusion coefficient profile, with short-scale

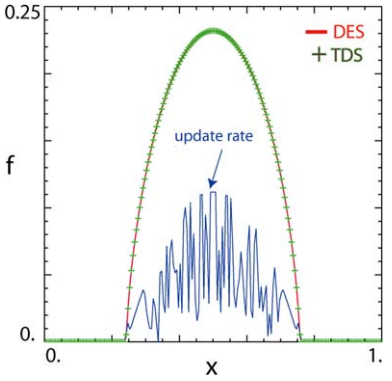

Fig. 4. The non-linear diffusion (ND) model: the DES solution with $Q = 44$, $\sigma = 8.9 \times 10^{-4}$ (red), the time-stepping solution (green crosses) and the normalized DES solution update rate, $\Delta t_{\mathrm{TDS}}/\Delta t_{\mathrm{DES}}$ (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Summary of relative errors and characteristic time-step sizes for the DES and TDS solutions of the Fischer equation (LD–NR) in the following format, respectively: $\sigma_{\mathrm{DES}}(\Delta t^{\mathrm{eff}}_{\mathrm{DES}})$ and $\sigma_{\mathrm{TDS}}(\Delta t_{\mathrm{TDS}})$

N_{cell}	120	240	480	960
DES-R1	0.399 (1e − 2)	0.068 (1.2e − 2)	0.086 (3e − 3)	0.097 (8.5e − 4)
DES-R2	0.362 (5e − 3)	0.034 (8e − 3)	0.019 (1.3e − 3)	0.017 (3e − 4)
TDS-R1	0.418 (1e − 4)	0.210 (1e − 4)	0.057 (1e − 4)	0.014 (1e − 4)
TDS-R2	0.422 (5e − 5)	0.216 (5e − 5)	0.071 (5e − 5)	0.012 (5e − 5)

Table 3
Values of the performance factor, Q for DES solutions of the Fischer equation (LD–NR) obtained with different mesh resolutions

N_{cell}	120	240	480	960
DES-R1	100	120	30	8.5
DES-R2	50	80	13	3

oscillations being a consequence of adaptive flux adjustments. The regions of zero diffusion automatically remain free of computation. As a result, the DES run performs at a considerable speed-up, $Q = 44$.

5.3. Linear diffusion–non-linear reaction ($D = 0.01$, $S = \gamma f(1 - f^2)$, $\gamma = 100$, $u = 0$)

In this test problem we solve the Fisher equation, which describes a general class of "pulled front" problems that typically occur in diffusion–reaction systems, where steep wave fronts enter spatial regions corresponding

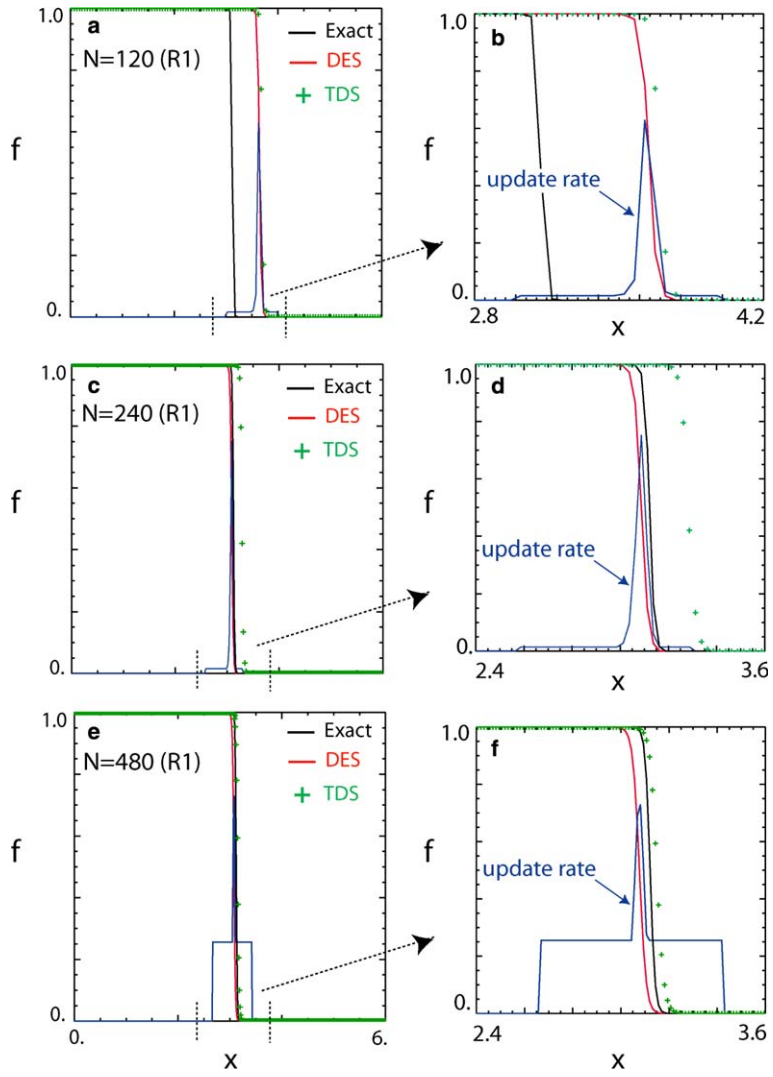

Fig. 5. Comparison of solutions of the Fisher equation (case LD–NR) for different mesh resolutions in run series R1 (Table 2): the DES solution (red), the corresponding time-stepping solution (green crosses), the exact solution (black), the normalized DES solution update rate, $\Delta t_{TDS}/\Delta t_{DES}$ (blue). Panels (b), (d), (f) represent the zoomed-in portions of plots (a), (c), (e), respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to unstable equilibrium states [6,28]. The Fisher equation has a smooth analytic solution in the form of a propagating "heat" wave [28]:

$$f_a(x,t) = (1 + e^{\beta(x-1-\alpha t)})^{-1}, \quad \beta = (1/2)(2\gamma/D)^{1/2}, \quad \alpha = (3/2)(2\gamma D)^{1/2}. \tag{16}$$

We have conducted a number of DES and TDS runs on meshes of different resolutions and compared numerical solutions to exact solution (16). Table 2 contains relative errors and characteristic time increments for all simulations. Note that each DES run is characterized by the effective time-step size Δt_{DES}^{eff} (defined by Eq. (13)), which is varied in run series, R1 and R2 by setting the maximum Courant number ω_{CFL} to 0.05 and 0.01, respectively. The only exception is $N_{cell} = 240$ case, where we were able to choose ω_{CFL} to be 5 times as large and still achieve an acceptable combination of simulation accuracy and performance.

Separately, in Table 3 we summarize the values of Q obtained in all DES runs. For all meshes considered in this test problem, the event-driven time integration achieves both a better efficiency and similar or better accuracy than the equivalent explicit time-stepping scheme. The DES advantage in this case is due to self-adaptive adjustment of the local computation update rate in accordance with the moving solution front. This can be observed by comparing update rate profiles in Figs. 5 and 6 with corresponding values of Q in Table 3. Note that in order to maintain proper accuracy, the event-driven "front tracking" mechanism automatically adjusts the update window (the region of non-zero update rate) around the moving position of the wave front.

Interestingly, for the finest mesh $N_{cell} = 960$ (Fig. 6), decreasing the Courant number ω_{CFL} alone (series R2) does not result in any improvement in accuracy until the positivity control parameter ε is adjusted to a value much smaller (by a factor of 3) than the double precision round-off value. This adjustment results in expanding the update window on both sides of the front, which leads to a more accurate solution (Fig. 6(b)). The apparent sensitivity of the solution is a consequence of the wave speed being determined by asymptotically small solution values ahead of the wave front. This argument has also led to development of flux-limited higher-order spatial discretization algorithms for similar problems [28]. For the same reason, both the event-driven and time-stepping simulations performed on the coarsest mesh ($N_{cell} = 120$) fail to converge to the analytical solution (Fig. 5(a) and (b)).

5.4. Linear convection ($D = 0$, $S = 0$, $u = 1$)

This example demonstrates the DES solution of a simple linear convection equation. Note that the reduction of the maximum value of the convected Gaussian profile (Fig. 7) is caused by the diffusive nature of scheme (8).

The difference between the TDS and DES solutions in this case is very small, $\sigma = 2.8 \times 10^{-3}$. As in the first example (linear diffusion–reaction), asynchronous solution updates in this test problem are purely accuracy

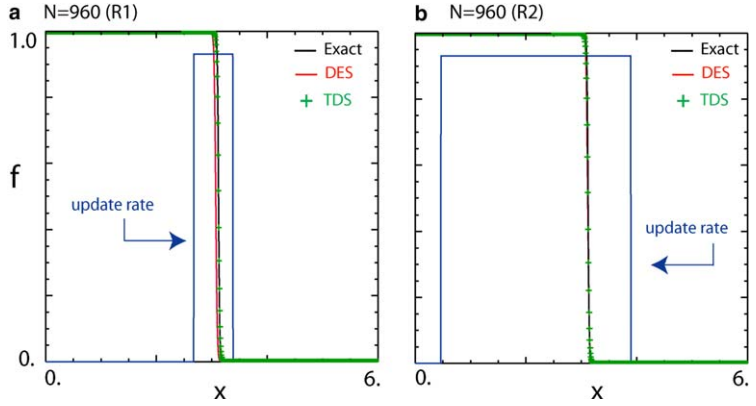

Fig. 6. Comparison of the finest mesh solutions of the Fisher equation (case LD–NR) in runs R1 (a) and R2 (b): the DES solution (red), the corresponding time-stepping solution (green crosses), the exact solution (black), the normalized DES solution update rate, $\Delta t_{TDS}/\Delta t_{DES}$ (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

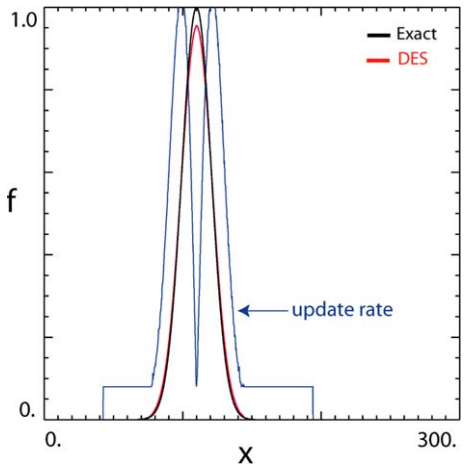

Fig. 7. The linear convection (LC) model: the DES solution with $Q = 7.5$, $\sigma_{\mathrm{DES}} = 4.3 \times 10^{-2}$ (red), the exact solution (black), the normalized DES solution update rate, $\Delta t_{\mathrm{TDS}}/\Delta t_{\mathrm{DES}}$ (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

driven, i.e., local time steps are automatically selected to enforce proper solution increments in all updates. Accordingly, we calculate the performance factor Q with respect to the minimum time increment obtained in the DES run, $\Delta t_{\mathrm{TDS}} = \min \Delta t_{\mathrm{DES}}$, which corresponds to the maximum spatial derivative of the solution profile (Fig. 7). Note that the "pedestal" and central minimum values of the local update rate v_{DES} correspond to the CFL limited time-step size τ_{CFL}.

5.5. Non-uniform diffusion–linear convection ($D(x) = f_G(x)$, $S = 0$, $u = 1$)

This example combines non-uniform diffusion and linear advection fluxes to demonstrate the flexibility with which the self-adaptive DES method can handle more complex systems. Fig. 8 illustrates a close match between the corresponding event-driven and time-stepping solutions.

In this case, a considerable DES performance gain, $Q = 35$ is due to the alleviation of solution stiffness and the elimination of unnecessary computation.

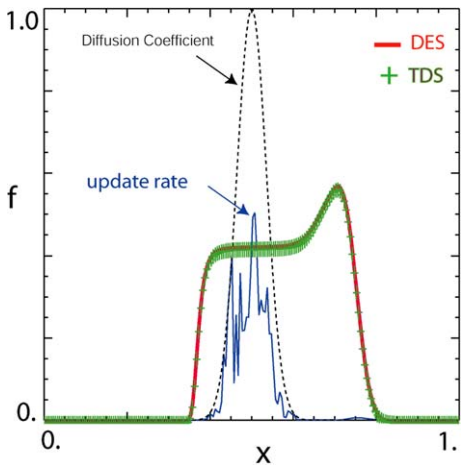

Fig. 8. The non-uniform diffusion–linear convection (ND–LC) model: the DES solution with $Q = 35$, $\sigma = 2.7 \times 10^{-3}$ (red), the time-stepping solution (green crosses), the diffusion coefficient profile (dotted black), the normalized DES solution update rate, $\Delta t_{\mathrm{CFL}}/\Delta t_{\mathrm{DES}}$ (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6. Conclusion

We have developed a novel, self-adaptive paradigm for the explicit time integration of multi-scale flux-conservative equations. In contrast to the traditional explicit and implicit time-stepping techniques, this method is based on discrete-event simulation technology [23,24]. The DES method presented in this paper enforces adaptive control over the system evolution by predicting and correcting local increments to the solution. This results in CPU-efficient, asynchronous, flux-conserving time integration of conservation laws.

As with any explicit integration, DES *cannot* perform asynchronous calculations with time increments *always* violating the local CFL condition. However, time increments in event-driven updates *may* exceed local CFL-limited Δt's without causing instability. The advantage of self-adaptive DES is in its ability to project and correct changes to the local solution "on-the-fly". For instance, if a local solution state has been scheduled to be updated with a super-Courant time increment but in the interim (before its scheduled execution time) receives a "synchronization call" from one of its neighbors, it may "decide" to fully update itself earlier, permitting information to propagate without violating causality, which prevents explosive numerical instability (see Section 5.1). Formal investigation of the numerical stability of DES will constitute the subject of future research.

In addition to this and our previous work on particle-in-cell electrostatics [23], we have successfully applied the DES method to even more complex systems, such as self-consistent hybrid particle-in-cell simulations [29], where a system of coupled particle motion and Maxwell's equations is integrated asynchronously via closely interacting discrete events. All these simulations have demonstrated the robustness (i.e., stability, accuracy) and efficiency of the DES paradigm. We are currently extending this method in several directions, including multiple spatial dimensions, coupled systems of diffusion–reaction equations, conservative Navier–Stokes models and higher order discretizations of the governing equations. In addition, one of our priorities is to combine DES with adaptive mesh techniques involving block-structured, stretched and unstructured meshes.

An important issue for modern large-scale computational algorithms is their scalability to parallel processors. The adaptive nature of all asynchronous algorithms naturally leads to a highly heterogeneous work load, which poses a significant challenge to effective parallelization of multi-scale simulations. The *conservative* and *time warp* parallel synchronization techniques for conventional discrete-event applications constitute a special field of computer science [25]. We applied similar synchronization approaches to particle-in-cell simulations [30] and developed a novel *preemptive event processing* (PEP) method for distributed physics-based DES systems, in general. The description of the latter technique, along with a discussion of load-balancing strategies for parallel PDE and particle simulations, will be the subject of a separate paper.

Acknowledgments

This research was supported by the National Science Foundation Information Technology Research (ITR), Grant Nos. 0325046 and 0529919. The authors appreciate useful discussions with Jonathan Driscoll on DES engine development issues. Discussions with Professors Richard Fujimoto and Kalyan S. Perumalla are acknowledged.

References

[1] J. Brackbill, B. Cohen (Eds.), Multiple Time Scales, Academic Press, Orlando, 1985.
[2] U.M. Ascher, S.J. Ruuth, B.T.R. Wetton, Implicit–explicit methods for time dependent partial differential equations, SIAM J. Numer. Anal. 32 (3) (1995) 797.
[3] W.J. Rider, D.N. Knoll, G.L. Olson, A multigrid Newton–Krylov method for multimaterial equilibrium radiation diffusion, J. Comput. Phys. 152 (1999) 164.
[4] W.J. Rider, D.N. Knoll, Time step size selection for radiation diffusion calculations, J. Comput. Phys. 152 (1999) 790.
[5] R.B. Lowrie, A comparison of implicit time integration methods for nonlinear relaxation and diffusion, J. Comput. Phys. 196 (2004) 566.
[6] C.C. Ober, J.N. Shadid, Studies on the accuracy of time-integration methods for the radiation diffusion equations, J. Comput. Phys. 195 (2004) 743.
[7] J.G. Verwer, B.P. Sommeijer, An implicit–explicit Runge–Kutta–Chebychev scheme for diffusion–reaction equations, SIAM J. Sci. Comput. 25 (5) (2004) 1824.

[8] W. Hundsdorfer, J. Jaffre, Implicit–explicit time stepping with spatial discontinuous finite elements, Appl. Numer. Math. 45 (2003) 231.
[9] L. Chacón, D.A. Knoll, J.M. Finn, An implicit, nonlinear reduced resistive MHD solver, J. Comput. Phys. 178 (2002) 15.
[10] C. Baldwin, P.N. Brown, R. Falgout, F. Graziani, J. Jones, Iterative linear solvers in a 2D radiation-hydrodynamics code: methods and performance, J. Comput. Phys. 154 (1999) 1.
[11] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484.
[12] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1989) 64.
[13] G.L. Bryan, Fluids in the universe: adaptive mesh refinement in cosmology, Comput. Sci. Eng. 1 (2) (1999) 46.
[14] J.B. Bell, M.S. Day, C.A. Rendleman, S.E. Woosley, M.A. Zingale, Adaptive low Mach number simulations of nuclear flame microphysics, J. Comput. Phys. 195 (2004) 677.
[15] E.M. Cherry, H.S. Greenside, C.S. Henriquez, Efficient simulation of three-dimensional anisotropic cardiac tissue using an adaptive mesh refinement method, Chaos 13 (3) (2003) 853.
[16] C. Dawson, R. Kirby, High resolution schemes for conservation laws with locally varying time steps, SIAM J. Sci. Comput. 22 (6) (2001) 2256.
[17] R. Kirby, On the convergence of high resolution methods with multiple time scales for hyperbolic conservation laws, Math. Comput. 72 (243) (2002) 1239.
[18] W. Quan, S.J. Evans, H.M. Hastings, Efficient integration of a realistic two-dimensional cardiac tissue model by domain decomposition, IEEE Trans. Biomed. Eng. 45 (3) (1998) 372.
[19] D. Amitai, A. Averbuch, M. Israeli, S. Itzikowitz, Implicit–explicit parallel asynchronous solver of parabolic PDEs, SIAM J. Sci. Comput. 19 (4) (1998) 1366.
[20] N.F. Otani, Computer modeling in cardiac electrophysiology, J. Comput. Phys. 161 (2000) 21.
[21] R. Abedi, S.-H. Chung, J. Erickson, Y. Fan, M. Garland, D. Guoy, R. Haber, J.M. Sullivan, S. Thite, Y. Zhou, Spacetime meshing with adaptive refinement and coarsening, in: Proceedings of the 20th Annual Symposium on Computational Geometry, Brooklyn, New York, USA, 2004, pp. 300–308.
[22] A. Lew, J.E. Mardsen, M. Ortiz, M. West, Asynchronous variational integrators, Arch. Rational Mech. Anal. 167 (2003) 85.
[23] H. Karimabadi, J. Driscoll, Y.A. Omelchenko, N. Omidi, A new asynchronous methodology for modeling of physical systems: breaking the curse of Courant condition, J. Comput. Phys. 205 (2) (2005) 755.
[24] J. Banks (Ed.), Handbook of Simulation, Wiley, New York, 1998.
[25] R.M. Fujimoto, Parallel and Distributed Simulation Systems, Wiley Interscience, New York, 2000.
[26] J. Nutaro, Parallel discrete event simulation with application to continuous systems, Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Arizona, 2003.
[27] J. Nutaro, B.P. Zeigler, R. Jammalamadaka, S. Akerkar, Discrete event simulation of gas dynamics within the DEVS framework, Lect. Note Comp. Sci. 2660 (2003) 319.
[28] W. Hundsdorfer, C. Montijn, A note on flux limiting for diffusion discretizations, IMA J. Numer. Anal. 24 (4) (2004) 635.
[29] Y.A. Omelchenko, H. Karimabadi, Event-driven hybrid particle-in-cell simulation: a new paradigm for multi-scale plasma modeling, J. Comp. Phys. 216 (1) (2006) 153–178.
[30] Karimabadi H., J. Driscoll, J. Dave, Y. Omelchenko, K. Perumalla, R. Fujimoto, N. Omidi, Parallel discrete event simulations of grid-based models: asynchronous electromagnetic hybrid code, in: Workshop on State-of-the-art in Scientific Computing, Springer LNCS Proceedings, 580, 2005.