



ELSEVIER

Available online at www.sciencedirect.com

Journal of Computational Physics xxx (2007) xxx–xxx

JOURNAL OF
COMPUTATIONAL
PHYSICSwww.elsevier.com/locate/jcp

A time-accurate explicit multi-scale technique for gas dynamics

Y.A. Omelchenko, H. Karimabadi *

SciberQuest, Inc., Solana Beach, CA 92075, United States

Received 18 July 2006; received in revised form 16 January 2007; accepted 5 April 2007

Abstract

We present a new time-accurate algorithm for the explicit numerical integration of the compressible Euler equations of gas dynamics. This technique is based on the discrete-event simulation (DES) methodology for nonlinear flux-conservative PDEs [Y.A. Omelchenko, H. Karimabadi, Self-adaptive time integration of flux-conservative equations with sources, *J. Comput. Phys.* 216 (1) (2006) 179–194]. DES enables adaptive distribution of CPU resources in accordance with local time scales of the underlying numerical solution. It distinctly stands apart from multiple (local) time-stepping algorithms in that it requires neither selecting a *global synchronization time step* nor pre-determining a *sequence of time-integration operations* for individual parts of a heterogeneous numerical system. In this paper we extend the DES methodology in three important directions: (i) we apply DES to a system of coupled *gas dynamics* equations discretized via a central-upwind scheme [A. Kurganov, E. Tadmor, New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations, *J. Comput. Phys.* 160 (2000) 241–282; A. Kurganov, S. Noelle, G. Petrova, Semidiscrete central-upwind schemes for hyperbolic conservation laws and Hamilton–Jacobi equations, *SIAM J. Sci. Comput.* 23 (3) (2001) 707–740]; (ii) we introduce a new *Preemptive Event Processing* (PEP) technique, which automatically enforces synchronous execution of events with sufficiently close update times; (iii) we significantly improve the accuracy of the previous algorithm [Y.A. Omelchenko, H. Karimabadi, Self-adaptive time integration of flux-conservative equations with sources, *J. Comput. Phys.* 216 (1) (2006) 179–194] by applying locally *second-order-in-time* flux-conserving corrections to the solution obtained with the forward Euler scheme. The performance of the new technique is demonstrated in a series of one-dimensional gas dynamics test problems by comparing numerical solutions obtained in event-driven and equivalent time-stepping simulations.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Asynchronous; Explicit; Discrete-event simulation; Event-driven; Gas dynamics; Euler equations; Central-upwind scheme; Adaptive; Multi-scale; PDE; Time-accurate integration; Multiple time-stepping

1. Introduction

In recent years computational scientists have been paying increased attention to adaptive techniques for multi-scale heterogeneous systems. The primary reason for this interest is obvious: despite continuing

* Corresponding author. Tel.: +1 858 7937063; fax: +1 858 7775684.

E-mail addresses: yurio@sciberquest.com (Y.A. Omelchenko), homak@sciberquest.com (H. Karimabadi).

33 advances in computer architectures, high-resolution simulations in many scientific fields still remain compu-
34 tationally prohibitive. Examples of such multi-scale systems are global models of the Earth's magnetosphere
35 [27], porous and combustion flows [12,19,24,30] and weather phenomena [28,29], to name a few. In general,
36 CPU efficiency of explicit multi-scale simulations is constrained by the global Courant–Friedrich–Levy (CFL)
37 condition. Without resorting to implicit and timestep-splitting techniques, it can only be increased via spatial
38 and/or temporal refinement of the numerical solution. As a result, a number of explicit multiple time-stepping
39 (MTS) methods have been developed for hyperbolic conservation laws [6,7,10,15,35,38]. The common strategy
40 is to use each cell's maximum stable timestep rather than the value limited by the global CFL condition. The
41 difficulty is in ensuring that asynchronous information is correctly and efficiently propagated in a time-accu-
42 rate fashion. In order to address this problem, these references suggest various time integration and synchro-
43 nization schemes. Some of them do not enforce conservation of numerical fluxes. This degrades the accuracy
44 and stability of asynchronous computation, despite the fact that such schemes may be formally higher-order
45 accurate in time (e.g. see Ref. [35]). More importantly, all MTS techniques require careful selection of global
46 time steps, at which conditions for determining new local time steps need to be recomputed (unless these con-
47 ditions do not change in time, as in linear models [8]). As a result, between two successive global synchroni-
48 zations points in time, MTS methods have to rely on a pre-determined hierarchy of interleaved temporal
49 updates, where cells with smaller timesteps ("faster" cells) are typically integrated before cells with larger time-
50 steps ("slower" cells). This causality requirement, however, is violated in those iterations where "slower" cells
51 are advanced prior to their "faster" neighbors in order provide flux information at cell interfaces for further
52 time interpolation. For efficiency, MTS methods employ fixed values of local time-step sizes throughout a sin-
53 gle global synchronization step. The sizes of local timesteps and their sequence are determined based on the
54 minimum acceptable time-step size, which is evaluated *at the beginning* of each global time step. For strongly
55 nonlinear systems, however, these conditions may change during a single global integration step significantly
56 enough, so that *a priori* computed local timesteps may exceed *current* permissible time-step sizes [6]. In that
57 case the solution becomes numerically unstable. MTS techniques are also known to lead to "resonance" (non-
58 linear) instabilities in molecular dynamics simulations [5]. Thus, the multiple time-stepping paradigm does not
59 seem to allow a flexible approach to merging local time steps or offer a robust strategy for selecting local
60 updates that would automatically satisfy causality and accuracy constraints imposed by the underlying physics
61 and geometry. In addition, time-stepping techniques, in general, have difficulty adaptively deactivating non-
62 informative parts of the computational domain, as this can easily trigger explosive numerical instabilities [26].

63 In structured adaptive mesh refinement (SAMR) methods [2–4], temporal refinement is usually achieved by
64 choosing hierarchical time steps for finer patches in accordance with their mesh refinement ratios and CFL
65 conditions. Therefore, this type of time integration can also be considered as a subset of MTS. SAMR pre-
66 serves conservation laws by imposing flux corrections (in a time-integrated sense) at coarse-fine patch inter-
67 faces. Being an MTS method, the SAMR integration algorithm suffers from the same deficiencies pointed
68 out above. Furthermore, the original formulation of SAMR [2–4] assumes that spatial refinement should
69 be based on error estimation. Application of standard error estimation procedures (e.g. Richardson's extrap-
70 olation) requires the knowledge of the approximation orders of the governing finite-difference equations.
71 However, in stochastic or multi-physics systems (e.g. reactive flows) accurate estimates may be difficult or
72 expensive to obtain. As a result, practical mesh refinement algorithms are often based on more physical (intu-
73 itive), rather than numerical (error-based) assumptions. In that case, however, hierarchical MTS updates may
74 produce uncontrolled solution errors due to inferior accuracy of coarse-patch calculations. On the other hand,
75 in multiple-timescale simulations, where the solution is fairly smooth in space, applying SAMR may result in
76 overrefining the computational domain. Therefore, for such applications local time integration alone (if prop-
77 erly implemented) would result in substantial savings in computation time. For instance, to integrate efficiently
78 from the subsonic, sub-Alfvénic regime through the solar corona into the super-Alfvénic solar wind [34], one is
79 faced with a significant computational challenge that the integration is essentially controlled by the fast time
80 scales characteristic of the turbulent MHD plasma in the subsonic regime. Clearly, to achieve optimum
81 numerical resolution and performance, mesh and temporal refinements should be considered as two separate
82 numerical issues. The former is required to reduce approximation errors due to solution gradients in config-
83 uration space, the latter being sought to maximize the use of CPU resources in accordance with local physical
84 time scales and chosen mesh density.

85 It was pointed out earlier [21,23] that temporal refinement of PDE-based simulations does not need to be
86 necessarily carried out in the form of predetermined (hierarchical) updates. In fact, a heterogeneous numerical
87 system may proceed in simulated time by adaptively applying local time increments, which lead to desirable
88 *solution increments*, satisfying the appropriate accuracy and causality requirements. This important capability
89 is achieved via self-adaptive Discrete-Event Simulation (DES), which abandons the concept of uniform time
90 progression in favor of enabling individual time lines for system micro-states [13,22,23]. In DES, the solution
91 variables predict and synchronize their temporal trajectories through enforcement of local causality and accu-
92 racy constraints, formulated in terms of changes to these variables.

93 Recently, a number of multiple-timescale algorithms have emerged for specific applications across various
94 scientific and engineering disciplines [8,18,20,21,25,32]. Some of these methods [18,21,32] are based on classical
95 discrete-event methodology [1,39]. In order to enable robust and accurate asynchronous integration of non-
96 linear particle-in-cell (PIC) and PDE models, two additional principles of event-driven simulation were pro-
97 posed [13,22,23]: (i) self-adaptive event synchronization; and (ii) conservative flux transfer between mesh
98 elements.

99 In this paper, when applying DES to Euler's equations of gas dynamics, we introduce two new features: (i)
100 Preemptive Event Processing (PEP); (ii) second-order-in-time-accurate integration. PEP bridges the gap
101 between time-stepping and event-driven computations by forcing events closely spaced in time continuum
102 to synchronize at time levels, which are adaptively determined during the simulation. The second-order-in-
103 time flux correction, presented in this paper, significantly improves the accuracy of DES compared to the
104 first-order forward Euler scheme [23]. It should be also noted that this work extends the DES machinery to
105 high-resolution schemes for coupled hyperbolic conservation laws, following the asynchronous treatment of
106 diffusion and reactive terms [23].

107 In Section 2 we summarize the most essential features of DES and compare them to those of MTS methods.
108 A discrete gas dynamics model is introduced in Section 3. In Section 4 we present a new time-integration tech-
109 nique (DES-PEP). Section 5 provides descriptions of discrete-event and time-stepping algorithms used to
110 perform gas dynamics simulations. Section 6 compares event-driven and time-stepping solutions to several
111 one-dimensional test problems. Section 7 presents results from a temporal convergence study conducted for
112 one of the test problems considered. Concluding remarks and general directions for future work are given
113 in Section 8. Additional details concerning the new DES algorithm are provided in the [Appendix](#).

114 2. Discrete-event simulation

115 DES has its origin in operations research and management science, war games and telecommunications
116 [1,11,39]. An event-driven simulation progresses in time by allowing the global system to “jump” from one global
117 state to another at irregular (asynchronous) moments upon the occurrence of “events”, which represent effective
118 units of information in the system. This automatically eliminates two well-known causes of CPU inefficiency
119 (“degeneracy”) in simulations with spatially inhomogeneous time scales: (i) “idle” performance due to the waste
120 of CPU time on updating inactive (noninformative) parts of the system state; (ii) “stiff” performance caused by
121 the presence of relatively small parts of the system undergoing faster changes compared to the rest of the system.

122 In DES, each event is a simulation object characterized by its process function, which is responsible for
123 changing the system state and a timestamp, indicative of when the process function is scheduled to be executed
124 in simulated time. DES programs typically operate with the following data structures: (1) *The state variables*.
125 These variables describe the solution of the system. (2) *The event list (queue)*. The priority queue contains
126 events, which are sorted by their timestamps in non-decreasing order so that the timestamp of the top event
127 corresponds to the earliest execution time in the system. (3) *The simulation clock*. This data structure corre-
128 sponds to the main loop of a traditional time-stepping simulation. The clock indicates how far in time the sim-
129 ulation has progressed. The DES cycle proceeds by repeatedly removing the top event from the event list and
130 executing (processing) it by calling its process function. In DES terminology, a discrete “micro-state” (simply
131 referred to as a “state”) corresponds to an independent computational variable. Each state schedules its
132 update by delaying the execution of a corresponding event until its due time. The main advantage of event-
133 driven time integration may be explained as follows. Suppose we integrate a computational quantity, f by solv-
134 ing an ordinary differential equation:

135
137

$$df/dt = R(f). \quad (1)$$

138 In DES, the traditional numerical measure of time advance (time-step size), Δt , is replaced by a physically
139 meaningful information unit, Δf , whose choice is usually dictated by considerations of local accuracy and sta-
140 bility [21–23]. For a given Δf , an individual time increment Δt for every state is found by solving the inverse of
141 Eq. (1):

$$dt/df = 1/R(f). \quad (2)$$

145 As a result, heterogeneous parts of the solution may “warp” through simulated time to the extent that time
146 increments for inactive states ($R(f) = 0$) may effectively become infinite.

147 The robust application of discrete-event methodology to nonlinear flux-conservative equations was made
148 possible with the advent of self-adaptive integration [22,23]. Here DES is effectively equivalent to applying
149 a self-adaptive “predictor–corrector” scheme to each computational element (state). The “predictor” sched-
150 ules an event with a time delay, Δt_e , during which the value of a corresponding state is estimated to change
151 by a local *target* amount, Δf_e . The scheduled state is assumed to “ballistically” evolve for the duration of time,
152 Δt_e along a predicted time trajectory, specified by the chosen discretization scheme and current system param-
153 eters. Accordingly, the “corrector” ensures that a pending event is processed earlier than its scheduled process
154 time (preempted), should the causality constraints used to predict its ballistic trajectory undergo a significant
155 change (e.g. the predicted change has already been achieved via synchronization updates [23]). This is accom-
156 plished by requiring that each state always synchronize itself with its dependent (“neighboring”) states when
157 an event associated with that state is either processed directly (at the top of the event queue), or preempted
158 during a synchronization call. As a result, the DES code automatically determines appropriate spatial
159 synchronization ranges for local updates. This ability of self-adaptive DES to *predict* and *correct* local com-
160 putations constitutes its fundamental difference compared to other asynchronous (time-stepping and discrete-
161 event) techniques.

162 In our previous paper on the self-adaptive integration of flux-conservative equations with reactive terms
163 [23] we described an algorithm, which synchronizes update events via causality constraints only. In principle,
164 it does not prevent events with arbitrarily close execution times from being processed *progressively*, at slightly
165 different times. Naturally, this may result in unnecessary flux evaluations and synchronization operations. A
166 similar efficiency problem arises in MTS (and SAMR) techniques, where it is commonly addressed by round-
167 ing all timesteps down to the next lower fractional power of two (e.g. see Refs. [10,20]). In this paper, we show
168 that efficient synchronization of events with close time stamps can be achieved *adaptively* via a new (PEP) tech-
169 nique (Section 4).

170 The DES algorithm described in Refs. [22,23] is based on the forward Euler scheme, which is formally first-
171 order accurate in time. It assumes that numerical fluxes at cell interfaces remain piecewise-constant between
172 two successive synchronization acts. This is by no means a general restriction for DES. Indeed, one may use
173 local time integrators, which could be formally second-order (or higher) accurate in time and would still pre-
174 serve numerical fluxes in a time-averaged sense. This can be achieved by representing flux trajectories as linear
175 functions (or piecewise-polynomials) in time and solving Eq. (2) for local Δt 's (with given Δf 's). In this paper,
176 however, we describe a simpler DES algorithm, where fluxes are evaluated *a posteriori* using the second-order
177 Euler correction (Section 3). It should also be noted that for certain numerical systems, such as flux-conser-
178 vative equations discretized in space via the Cauchy–Kowalevski procedure, it is possible to derive matching
179 higher-order-in-time asynchronous integrators even with the forward Euler scheme [8]. Below we summarize
180 the most essential properties of DES–PEP by grouping them in three important categories:

181 2.1. Robustness (self-adaptivity)

182

- 183 1. No need for global synchronization time steps.
- 184 2. Stability (correct propagation of information) is always maintained: at any point in simulated time compu-
185 tational elements are continuously updated in a physically driven, self-adaptive order by means of event
186 sorting and synchronization operations.

187 3. Individual time increments are selected based on local physical thresholds, estimated through stability and
 188 accuracy considerations.
 189

190 2.2. Efficiency

191
 192 1. Local time increments need not bear any integer multiple relations with each other (e.g. fractional powers
 193 of two or integer multiples), as events with close execution times are adaptively synchronized via PEP.
 194 2. No need to explicitly group elements with similar rates of change in blocks. The DES algorithm also
 195 ensures that fluxes are computed once per face.
 196 3. Computational elements are adaptively deactivated (become CPU-idle) and reactivated without causing
 197 numerical instability.
 198

199 2.3. Accuracy

200
 201 1. DES is flux-conservative: common interfaces are always updated with the same fluxes.
 202 2. Physical thresholds for changes to variables per update are automatically observed. This property is espe-
 203 cially important for reactive systems.
 204 3. Locally higher-order time integration is possible.
 205

206 3. Numerical model for Euler's equations

207 Our discrete model is based on one-dimensional Euler's equations for compressible inviscid gas dynamics
 208 written in conservation form:
 209

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{U})}{\partial x} = 0, \quad (3)$$

$$\mathbf{U} = \begin{bmatrix} \rho \\ M \\ E \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} M \\ \rho v^2 + p \\ (E + p)v \end{bmatrix}, \quad E = \frac{p}{(\gamma - 1)} + \frac{\rho v^2}{2}. \quad (4)$$

212 Here, \mathbf{U} and \mathbf{f} represent the solution and flux vectors; ρ , v , $M = \rho v$, p , E are the density, velocity, momentum,
 213 pressure and total energy of gas, respectively and γ is the gas adiabatic index.

214 This model is discretized on a uniform mesh, $i = -1, \dots, N_{\text{cell}}$ using the following notation: $x_i = i\Delta x$ (cell
 215 centers) and $x_{i\pm 1/2} = (i \pm 1/2)\Delta x$ (cell faces). Here Δx is the cell size and indices $i = -1$, $i = N_{\text{cell}}$ correspond
 216 to the left and right boundary cells, respectively. As in previous work [23], cell-centered discrete solution states
 217 $u_i(t_i) := \mathbf{U}(x_i, t_i)$ are defined at times t_i , corresponding to the most recent updates of these states. Eqs. (3)–(4)
 218 are discretized with a second-order semi-discrete central-upwind scheme, with finite differences being expressed
 219 in a scalar (component-wise) form [16,17]:
 220

$$\tilde{u}_i \equiv u_i(t_i + \Delta t_i) = u_i + R_i \Delta t_i, \quad (5)$$

$$R_i = -\frac{1}{\Delta x} [F_{i+1/2} - F_{i-1/2}], \quad (6)$$

$$F_{i-1/2} = \frac{a_{i-1/2}^+ f(u_{i-1}^R) - a_{i-1/2}^- f(u_i^L)}{a_{i-1/2}^+ - a_{i-1/2}^-} + \frac{a_{i-1/2}^+ a_{i-1/2}^-}{a_{i-1/2}^+ - a_{i-1/2}^-} [u_i^L - u_{i-1}^R], \quad (7)$$

$$a_{i-1/2}^+ = \max[\lambda_{\max}(u_{i-1}^R), \lambda_{\max}(u_i^L), +\varepsilon], \quad a_{i-1/2}^- = \min[\lambda_{\min}(u_{i-1}^R), \lambda_{\min}(u_i^L), -\varepsilon], \quad (8)$$

223 where ε is a small numerical constant of the order of machine precision (roundoff); $\lambda_{\max}(u) = v + c$,
 224 $\lambda_{\min}(u) = v - c$ are the maximum and minimum eigenvalues of the Jacobian $\partial \mathbf{f} / \partial \mathbf{U}$ ($c = \sqrt{\gamma p / \rho}$ is the speed
 225 of sound) and u_i^L , u_i^R are the left and right solution values obtained by performing a TVD-limited linear
 226 reconstruction:

$$229 \quad u_i^R (\equiv u_i^{i+1/2}) = u_i + \frac{1}{2} \bar{A}_i, \quad u_i^L (\equiv u_i^{i-1/2}) = u_i - \frac{1}{2} \bar{A}_i. \quad (9)$$

230 In this paper we use the following approximation for the slopes \bar{A}_i [40]:

$$233 \quad \bar{A}_i = \frac{\max[A_{i-1/2} A_{i+1/2}, 0]}{A_i}, \quad A_{i-1/2} = u_i - u_{i-1}, \quad A_{i+1/2} = u_{i+1} - u_i, \quad A_i = \frac{1}{2}(u_{i+1} - u_{i-1}). \quad (10)$$

234 Interestingly enough, flux approximation (7) coincides with the well-known HLLC formulation [9]. It becomes
 235 purely upwind in the case of super sonic flow:

$$236 \quad v - c > 0: \quad a_{i-1/2}^- = -\varepsilon, \quad a_{i-1/2}^+ = |v| + c, \quad F_{i-1/2} = f(u_{i-1}^R), \quad (11a)$$

$$237 \quad v + c < 0: \quad a_{i-1/2}^+ = +\varepsilon, \quad a_{i-1/2}^- = -|v| - c, \quad F_{i-1/2} = f(u_i^L). \quad (11b)$$

238 Eqs. (5)–(10) are stable under the following local CFL condition [9]:

$$241 \quad \Delta t_i < \Delta t_i^{\text{CFL}} = \frac{\Delta x}{2 \max(|a_{i-1/2}^-|, |a_{i-1/2}^+|, |a_{i+1/2}^-|, |a_{i+1/2}^+|)}. \quad (12)$$

242 The time integration in a synchronous time-driven simulation (TDS) is constrained by the global minimum
 243 CFL timestep, $\Delta t_{\min}^{\text{CFL}}$ computed in the interior of the computational domain, $i = 0, \dots, N_{\text{cell}} - 1$. On the other
 244 hand, a corresponding event-driven simulation may proceed asynchronously by choosing proper local time
 245 increments Δt_i .

246 The solution \tilde{u} , integrated using Eq. (5), is first-order accurate in time. However, a formally second-order-
 247 in-time solution $\tilde{\tilde{u}}$ can easily be constructed without violating flux conservation by performing a local Euler
 248 correction:

$$249 \quad \tilde{\tilde{u}}_i(t_{\text{clock}}) = \tilde{u}_i(t_{\text{clock}}) + \Delta u_i, \quad \Delta u_i = \Delta u_{i-1/2} - \Delta u_{i+1/2}, \quad (13)$$

$$251 \quad \Delta u_{i+1/2} = \frac{1}{2\Delta x} [F_{i+1/2}(\tilde{u}_i) - F_{i+1/2}(u_i)] \Delta t_{i+1/2}, \quad \Delta t_{i+1/2} = t_{\text{clock}} - t_{i+1/2}^{\text{sync}}, \quad (14)$$

252 where $t_{i+1/2}^{\text{sync}}$ is the time at which states u_i and u_{i+1} were last synchronized (i.e., the time at which flux $F_{i+1/2}$ was
 253 last evaluated). Eqs. (13)–(14) are also applicable to TDS. However, in this case, the second-order correction is
 254 applied globally by explicitly recomputing all rate-of-changes, $R_i(\tilde{u}_i)$:

$$257 \quad \tilde{\tilde{u}}_i(t_{\text{clock}}) = u_i(t) + \hat{R}_i \Delta t, \quad \hat{R}_i = (R_i(u_i) + R_i(\tilde{u}_i))/2. \quad (15)$$

258 4. Preemptive event processing (PEP)

259 Traditional (“single-event mode”) DES algorithms generally assume that the global simulation clock is
 260 advanced upon processing and rescheduling each event in the event queue. As mentioned above, this type
 261 of event-driven computation may result in unnecessary inter-element synchronizations and flux evaluations
 262 in parts of the computational domain, where the solution properties are fairly homogeneous. This would incur
 263 additional CPU overhead compared to synchronous computation (carried out on a subdomain basis). More-
 264 over, efficient parallelization of single-event DES models may easily become a challenging problem since opti-
 265 mistic and conservative strategies proposed for traditional parallel discrete-event simulations [1,11,14,33] may
 266 become difficult to optimize for strongly nonlinear systems. Thus, processing and scheduling single events
 267 without taking into account their temporal proximity to other (pending) events, may lead to inefficiencies
 268 in both serial and parallel DES. To circumvent this problem, we have developed an alternative, “batch” mode
 269 for DES–Preemptive Event Processing (PEP). In this mode, events with sufficiently close timestamps are pro-
 270 jected onto synchronous time levels, which are determined *adaptively* by the program. This generally improves

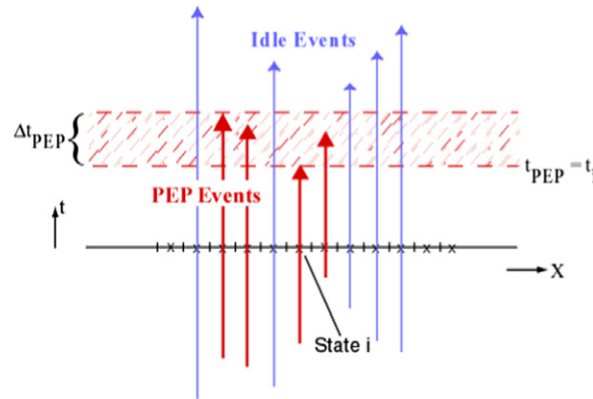


Fig. 1. Illustration of preemptive event processing (PEP). In each PEP loop, events with timestamps, $t_e \leq t_{PEP} + \Delta t_{PEP}$ (red) are synchronously executed at the time level $t = t_{PEP}$, corresponding to the earliest timestamp (here t_i). The PEP “time window” size Δt_{PEP} is dynamically adjusted as events are processed in timestamp order. Events with timestamps falling outside the PEP window (blue) remain CPU-idle. (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

271 the efficiency of event processing and makes it possible to parallelize DES programs via conventional message
 272 passing. In principle, different DES models may implement PEP in different ways, as conditions for event pre-
 273 emption may vary by model. Here we describe an algorithm, which we have found to be simple, flexible and
 274 efficient. The proximity of events in simulated time can be characterized by a finite time window extending into
 275 the future from the global lower bound on timestamp (LBTS), which always corresponds to an event with the
 276 earliest process time. Events with timestamps, t_e^{proc} lying within the bounds of this window are executed at the
 277 current simulation time, t_{clock} coinciding with the LBTS (Fig. 1).

278 Importantly, at each time level t_{clock} the size of this window, Δt_{PEP} can be adaptively adjusted (independ-
 279 ently on each processor when run in parallel) by minimizing a number of preempted states, whose next pro-
 280 cess times are predicted to fall within the current time window. In each PEP loop events are processed at
 281 $t = t_{\text{clock}}$ in their *timestamp order* as follows. Let us initialize $\Delta t_{PEP} = \infty$. Then an event, e with a characteristic
 282 time increment, Δt_e (see below) is preempted only if its timestamp t_e^{proc} satisfies the following (PEP) condition:

$$285 \quad t_e^{\text{proc}} \leq t_{\text{clock}} + \Delta t_{PEP}, \quad \Delta t_{PEP} = \min(\Delta t_{PEP}, R_{PEP} \Delta t_e), \quad (16)$$

286 where R_{PEP} is a positive PEP parameter (usually $R_{PEP} \leq 1$). Note that condition (16) implies that during the
 287 PEP loop Δt_{PEP} is dynamically reevaluated, with the synchronous event processing stopping automatically
 288 when this condition is violated. Despite its apparent simplicity, this algorithm allows a number of different
 289 strategies, depending on a particular choice of Δt_e in Eq. (16). For instance, for event e one may select Δt_e
 290 to be equal to its predicted (target) time increment, $\Delta t_e^{\text{tr}} \equiv t_e^{\text{proc}} - t_{\text{clock}}$, where t_e^{proc} is the event process time,
 291 estimated when the event is scheduled for execution. However, we experimentally discovered that in most cases
 292 DES-PEP models run faster if Δt_e is set to the actual time period, $\Delta t_e^{\text{proc}} \equiv t_{\text{clock}} - t_e^{\text{last}}$, passed between the cur-
 293 rent simulation time t_{clock} and the event’s last processing time, t_e^{last} . An in-depth description of the DES-PEP
 294 algorithm for gas dynamics Eqs. (3)–(4) is given in the following section.

295 5. Algorithm implementation

296 The pseudocode of our gas-dynamics DES model is shown in Table 1, with more pseudocode (for individ-
 297 ual functions) given in Tables A1–A4 found in the Appendix. Note that the pseudocode is written in an object-
 298 oriented programming (OOP) style, which emphasizes the object-oriented nature of discrete-event models
 299 [1,39]. In particular, the OOP notation (e.g. *object.ObjectType::method()*) seems to be more appropriate for
 300 describing event-related operations than the usual functional notation (e.g. *method(ObjectType object)*).
 301 Throughout the pseudocode we use the “//” symbol to provide additional comments at the ends of selected
 302 lines.

303 5.1. Data structures and state handling

304 The solution variables (states) are stored in a global data container, *DESFab*. States u_i ($u(i)$) are linked with
 305 their corresponding event objects, e_i ($e(i)$). States, which are updated in a single PEP loop, are grouped in a
 306 linked list structure, *PEPStack*, reinitialized empty before entering each PEP loop. All states are logically
 307 marked as being “valid” or “invalid”, based on their current status. A state is “invalidated” when its associ-
 308 ated event is processed (see *Event::process()* in Table A2) or preempted (see *Event::synchronize()* in Table A2).
 309 An “invalid” state needs to be rescheduled upon the completion of the PEP loop, in which it was invalidated.
 310 The scheduling procedure (*Event::schedule()* in Table A4) may add a new event to an event queue,
 311 *EventQueue*. This automatically “validates” and “activates” its associated state. Alternatively, the scheduling
 312 procedure may “deactivate” an invalidated state if its next execution time is deemed to be “infinite” for the
 313 purpose of this simulation (see details in *Event::schedule()* in Table A4). Note that second-order flux correc-
 314 tions (line 8 in Table 1) are allowed only at cell interfaces, which separate “active” states (see *Event::correct()*
 315 in Table A3). Accordingly, for programming convenience, boundary states and states being synchronized are
 316 always assigned the “active” status (see *Event::synchronize()* in Table A2). A logical variable, *Finished* is
 317 introduced in order to force all “active” events to synchronize at the simulation end time, t_{END} . An input
 318 parameter, *TimeOrder* (equal to 1 or 2) corresponds to the temporal approximation order adopted in a given
 319 run.

Table 1

Pseudocode of the gas-dynamics DES algorithm

```

function run_des()
1: initialize  $u$ ,  $Finished = \text{false}$ 
2: for  $u_i$  in DESFab:
3:    $\delta u_i = 0$ ;  $t_{\text{clock}} = t_i = t_i^{\text{last}} = t_{i+1/2}^{\text{sync}} = 0$ ; invalidate and deactivate  $u_i$ 
4:   add  $e_i$  to PEPStack
5: endif
6: for  $e$  in PEPStack:  $e.reconstruct()$ 
7: if (TimeOrder > 1):
8:   forein PEPStack:  $e.correct()$ 
9:   for  $e$  in PEPStack:  $e.reconstruct()$ 
10: endif
11: if ( $Finished$  is true): return // after correcting solution
12: for  $e$  in PEPStack:
13:    $e.dfdt()$  // compute rate-of-change and update local state time  $t_e$ 
14:   if ( $e$  is not valid):  $e.schedule()$  // compute  $t_e^{\text{ptoc}}$ ,  $\Delta t_e$  and schedule new event
15: endif
16: if (EventQueue is empty): // set clock time to end time
17:    $t_{\text{clock}} = t_{\text{END}}$ 
18: else // set clock time to earliest timestamp
19:    $t_{\text{clock}} = \min(t_e^{\text{ptoc}}, t_{\text{END}})$ 
20: endif
21: if ( $t_{\text{clock}} == t_{\text{END}}$ ):  $Finished = \text{true}$ 
22: clear PEPStack;  $\Delta t_{\text{PEP}} = \infty$ 
23: while (EventQueue is not empty): // do PEP-loop
24:    $e = \text{EventQueue.top}()$  // get earliest valid event
25:    $\Delta t_{\text{PEP}} = \min(\Delta t_{\text{PEP}}, R_{\text{PEP}} \Delta t_e)$ 
26:   if ( $t_e^{\text{ptoc}} > t_{\text{clock}} + \Delta t_{\text{PEP}}$  and  $Finished$  is false): break // out of PEP loop
27:    $e.process()$  // execute event  $e$ 
28:   EventQueue.pop() // discard event  $e$ 
29:   remove invalidated events from EventQueue
30: endwhile
31: goto line 6 // continue DES
endfunction

```

The simulation starts at $t = 0$ and finishes at $t = t_{\text{END}}$. Parts of the numerical solution u are self-adaptively updated via PEP at time levels, $t = t_{\text{clock}}$. Event-related operations (event methods) are described in the Appendix.

320 It should be emphasized that at each PEP time level t_{clock} , the contents of *PEPStack* represent an instan-
 321 taneous snapshot of the *actively changing* simulation phase space, dynamically assembled by processing and
 322 synchronizing solution states during the PEP loop (see *Event::process()* and *Event::synchronize()* in Table A2).
 323 Note that the global simulation time t_{clock} still progresses at the fastest rate of change present in the system.
 324 However, the computational efficiency of DES comes from the fact that at any time the phase space volume
 325 stored in *PEPStack* constitutes only a *fraction* of the whole simulation phase space stored in the *DESFab*
 326 container.

327 5.2. DES–PEP integration cycle

328 The DES–PEP algorithm is outlined in Table 1 (for more details, we suggest reading Ref. [23]). At $t = 0$ all
 329 simulation variables u_i are properly initialized (line 1). All states are also marked as “inactive” and “invalid”,
 330 and their corresponding event objects are added to *PEPStack* (lines 2–5). The DES cycle begins on line 6,
 331 where the left and right interface solutions (Eq. (9)) are reconstructed for each *PEPStack* state (*Event::recon-*
 332 *struct()* in Table A1). If one selects *TimeOrder* > 1 , then the *PEPStack* states are corrected (*Event::correct()* in
 333 Table A3) and reconstructed again. Numerical fluxes are corrected (*Event::correct()* in Table A3) and recom-
 334 puted (*Event::dfdt()* in Table A1) at the cell interfaces synchronized during the preceding PEP loop (note that
 335 at $t = 0$ all interfaces are considered to be synchronized, and *Event::correct()* does nothing since all states are
 336 initialized as “inactive”). The simulation ends when *Finished* is found to be true (line 11). Otherwise (lines 12–
 337 15), the rate-of-changes of all *PEPStack* states are updated (*Event::dfdt()* in Table A2) and “invalid” states
 338 are rescheduled for execution (*Event::schedule()* in Table A4). Lines 16–20 determine the next global clock
 339 time, which is set to either the earliest event timestamp (if there are “valid” events in *EventQueue*), or the sim-
 340 ulation end time t_{END} , whichever is earlier. The variable *Finished* is set to true if the simulation end time is
 341 reached (line 21). *PEPStack* is emptied on line 22. On lines 23–30 the DES code executes PEP (Section 4)
 342 by processing and popping top events from *EventQueue* (lines 24, 27–28). Processing event e_i results in updat-
 343 ing its state u_i , incrementing the “flux capacitor” variable δu_i and synchronizing u_i with its neighboring states
 344 (see *Event::process()* and *Event::synchronize()* in Table A2). In turn, neighboring states preempt their pending
 345 events if the values of their “flux capacitors” exceed their target thresholds, Δu_i^{tg} [23], or if they are located next
 346 to physical boundaries. Events e_i , popped from *EventQueue*, are executed at $t = t_{\text{clock}}$ as long as their time-
 347 stamps, t_i^{proc} fall within the bounds of the PEP window (see Eq. (16)). In our tests we found that the simulation

Table 2
Pseudocode of the gas-dynamics TDS algorithm

```

Function run_tds()
1:  $t_{\text{clock}} = 0$  ; initialize  $u$ 
2: while ( $t_{\text{clock}} \leq t_{\text{END}}$ ):
3:   if ( $\text{TimeOrder} > 1$ ):  $u^{\text{save}} = u$ 
4:   for  $\text{Pass} = 1, \text{TimeOrder}$ :
5:     for  $\forall i$ :  $u(i).reconstruct()$ 
6:     for  $\forall i$ : compute fluxes,  $F_{i-1/2}$  // see Eq. (7)
7:     if ( $\text{Pass} == 1$ ):  $\Delta t = \omega_{\text{CFL}} \Delta t_{\text{min}}^{\text{CFL}}$ 
8:     for  $\forall i$  :
9:       compute  $R_i$  // see Eq. (6)
10:      if ( $\text{TimeOrder} > 1$  and  $\text{Pass} == 1$ ):  $R_i^{\text{save}} = R_i$ 
11:    endfor
12:    if ( $\text{Pass} == 2$ ): for  $\forall i$  :  $\{R_i = (R_i + R_i^{\text{save}})/2; u_i = u_i^{\text{save}}\}$  // see Eq. (15)
13:    for  $\forall i$ :  $u_i = u_i + R_i \Delta t$ 
14:    apply boundary conditions for  $u$ 
15:  endfor
16:   $t_{\text{clock}} = t_{\text{clock}} + \Delta t$ 
17: endwhile
endfunction

```

The simulation starts at $t = 0$ and finishes at $t = t_{\text{END}}$. The whole numerical solution u is synchronously updated at adaptively selected time levels, $t = t_{\text{clock}}$.

348 accuracy may be somewhat increased if, in addition to condition (16), events e_i are also preempted when the
349 virtual advance of their states’ “flux capacitors” (δu_i) to the current time t_{clock} results in their “overflow”:

$$351 \quad \|\delta u_i + R_i(t_{\text{clock}} - t_i)\| \geq \Delta u_i^{\text{lg}} \quad (17)$$

352 Finally (line 29), we discard event objects, corresponding to “invalid” states having been preempted during
353 synchronization calls. Normally, such events constitute a small fraction of the total number of events pro-
354 cessed. We find this procedure to be more CPU-efficient, compared to immediate removal of preempted
355 events. Once the PEP loop finishes (either on line 23 or 26), the DES simulation continues (line 31) by pro-
356 ceeding to line 6. For comparison, Table 2 shows the pseudocode of the TDS algorithm, where the global
357 timestep Δt is adaptively selected at each time level based on the minimum CFL-limited timestep, $\Delta t_{\text{min}}^{\text{CFL}}$.

358 6. Test problems and results

359 For simplicity, we validate the DES–PEP technique in a series of one-dimensional test problems. Most of
360 them are well documented in the literature (e.g. see Refs. [16,17]). The computational speedup due to DES is
361 approximately proportional to the degree of numerical stiffness and inversely proportional to the relative num-
362 ber of active micro-states in a given system. As a result, the actual CPU time gain is highly application depen-
363 dent. Therefore, while still emphasizing the CPU efficiency of DES integration, in this paper we primarily
364 focus on verifying the computational accuracy and robustness of the new algorithm.

365 In all tests DES solutions are matched against those obtained in corresponding time-driven simulations
366 (TDS), assuming the second order of temporal approximation ($TimeOrder=2$). To evaluate the relative per-
367 formance of a given DES run with respect to an equivalent TDS run, we introduce two metrics: (i) a “theo-
368 retical” speed-up, $Q_E = (N_{\text{TDS}}/N_{\text{DES}})(N_{\text{cell}}/N_e)$, where N_{TDS} and N_{DES} are the numbers of synchronous TDS
369 and DES (PEP) time levels, respectively and N_e is the average number of processed events (invalidated states)
370 per PEP loop in the DES run; (ii) a CPU speed-up, $Q_{\text{CPU}} = t_{\text{TDS}}/t_{\text{DES}}$, where t_{TDS} and t_{DES} are the actual CPU
371 times measured in these TDS and DES runs, respectively. These metrics are summarized in Table 3 for all test
372 problems. Note that in general $Q_{\text{CPU}} \neq Q_E$ because other contributing factors, notably the event synchroniza-
373 tion and priority queue overheads, influence the actual CPU performance of each simulation. Not surpris-
374 ingly, since our DES and TDS models employ similar rules for selecting time increments, the numbers of
375 PEP levels observed in DES runs with $R_{\text{PEP}} = 1$ coincide with the numbers of time levels obtained in corre-
376 sponding TDS runs (Table 3).

377 In our numerical experiments we approximate gas dynamics Eqs. (3)–(4) on a computational domain,
378 $x \in [0, 1]$, choose $\gamma = 1.4$ (air), use double precision and set the global (TDS) and local (DES) CFL numbers,
379 $\omega_{\text{CFL}} = \Delta t/\Delta t_{\text{CFL}}$ to the same value, $\omega_{\text{CFL}} = 0.5$. If not stated otherwise, boundary conditions at both physical
380 boundaries are assumed to be Neumann. For convenience, when initializing the solution, we use an alternative
381 definition of the state vector: $\bar{\mathbf{U}} \equiv (\rho, \mathbf{v}, \mathbf{p}/\rho)$. To facilitate comparison between time-stepped and even-driven
382 simulations, we introduce the normalized theoretical and observed computational update rates,
383 $f_{\text{CFL}}(x_i) = \Delta t_0/\Delta t_i^{\text{CFL}}$ and $f_{\text{CPU}}(x_i) = \Delta t_0/\Delta t_i$, respectively, where $\Delta t_0 = \min \Delta t_i^{\text{CFL}}(t = 0)$. Since event preempt-
384 ion is controlled with a number of conditional statements involving floating-point variables, instantaneous
385 profiles of $f_{\text{CPU}}(x_i)$ may contain insignificant spurious features, sensitive to input parameters of particular
386 DES runs (including the underlying machine precision). In DES runs, selecting smaller values of R_{PEP}
387 typically results in a closer match between f_{CPU} and f_{CFL} . However, it also leads to a larger number of PEP

Table 3
Summary of performance metrics obtained in the TDS and DES tests

Run	N_{cell}	N_{TDS}	N_e	N_{DES}	Q_E	Q_{CPU}
Advection	2000	2000	122	2000	2000	22
WC ($t = 0.01$)	800	1661	63	4739	1661	2.3
WC ($t = 0.038$)	800	5043	137	14764	5043	1.1
Sod	800	1154	70	3254	1154	2.2
Blowoff	2000	2691	172	7939	2691	4.0

Double-columned DES entries combine results obtained for $R_{\text{PEP}} = 0.4$ (left column) and $R_{\text{PEP}} = 1.0$ (right column).

388 synchronization levels. Therefore, some experimentation may be needed to guarantee the maximum perfor-
 389 mance of DES–PEP programs (especially on parallel computer architectures).

390 6.1. Advection test

391 The first test represents a simple advection problem, with $\Delta x = 5 \times 10^{-4}$ ($N_{\text{cell}} = 2000$). The initial solution
 392 is represented by a “step-wave” in the computational domain, $x \in [0, 1]$:

$$\bar{\mathbf{U}}(x, t = 0) = \langle u_L[0, 0.4], u_M[0.4, 0.6], u_R[0.6, 1] \rangle, \quad (18a)$$

$$394 \quad u_L = (0.05, 0.5, 0)^T, \quad u_M = (1, 0.5, 0)^T, \quad u_R = (0.05, 0.5, 0)^T. \quad (18b)$$

395 Despite the obvious simplicity of this test, it clearly demonstrates an important DES feature, namely adaptive
 396 allocation of CPU resources to the “active” parts of the computational domain [21]. This property alone may
 397 boost CPU performance in a number of simulation fields, including fire propagation [19] and levelset dynam-
 398 ics [26]. Indeed, as shown in Fig. 2, at any time during the simulation the DES solution is only updated in the
 399 close vicinity of the front and back of the moving pulse. The rest of the computational domain is automatically
 400 “deactivated” without causing instability. Naturally, this leads to a significant CPU speedup, $Q_{\text{CPU}} = 22$.

401 6.2. Woodward–Colella test

402 This test simulates the interaction of two blast waves [37]. We apply solid wall boundary conditions at both
 403 ends of the computational domain and choose $\Delta x = 1.25 \times 10^{-3}$ ($N_{\text{cell}} = 800$).

404 The initial solution is represented by a superposition of three state vectors:

$$\bar{\mathbf{U}}(x, t = 0) = \langle u_L[0, 0.1], u_M[0.1, 0.9], u_R[0.9, 1] \rangle, \quad (19a)$$

$$406 \quad u_L = (1, 0, 1000)^T, \quad u_M = (1, 0, 0)^T, \quad u_R = (1, 0, 100)^T. \quad (19b)$$

407 The DES and TDS solutions are plotted in Fig. 3 for two different simulation times. Clearly, they are of compa-
 408 rable numerical accuracy. However, compared to TDS, DES achieves faster or similar CPU times (see Table 3).

409 6.3. Sod test

410 This classic “shock tube” problem was proposed by Sod [31]. It is a Riemann problem, which tests a gas
 411 dynamics code’s ability to capture shocks and contact discontinuities, as well as produce the correct density

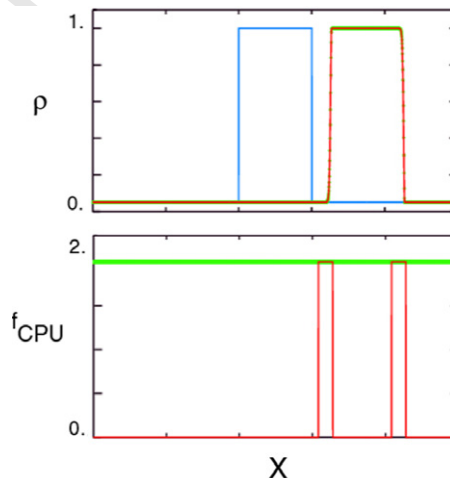


Fig. 2. Advection test: $t_{\text{END}} = 0.2$, $R_{\text{PEP}} = 1$. Shown are three solutions: $u(t = 0)$ (blue), $u_{\text{DES}}(t_{\text{END}})$ (red) and $u_{\text{TDS}}(t_{\text{END}})$ (green). DES updates are localized in the vicinity of the front and back of the moving pulse. (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

412 profile of a rarefaction wave. In this test we assume $\Delta x = 1.25 \times 10^{-3}$ ($N_{\text{cell}} = 800$). The initial solution is rep-
 413 resented by two state vectors:

$$\bar{\mathbf{U}}(x, t = 0) = \langle u_L[0, 0.5], u_R[0.5, 1] \rangle, \quad (20a)$$

$$415 \quad u_L = (1, 0, 1)^T, \quad u_R = (0.125, 0, 0.8)^T. \quad (20b)$$

416 Again, the event-driven and time-stepping solutions are found to match perfectly (Fig. 4), with DES running
 417 faster than TDS (in this case mainly due to the presence of inactive regions).

418 6.4. “Blowoff” test

419 Our final test simulates the propagation of two step-like gas perturbations, which are initialized moving
 420 away (with finite initial velocities) from their common interface. Assuming $\Delta x = 5 \times 10^{-4}$ ($N_{\text{cell}} = 2000$), the
 421 solution profile has the following form:

$$\bar{\mathbf{U}}(x, t = 0) = \langle u_L[0, 0.35], u_{LM}[0.35, 0.5], u_{RM}[0.5, 0.55], u_R[0.55, 1] \rangle, \quad (21a)$$

$$423 \quad u_L = (0.05, 0, 0)^T, \quad u_{LM} = (0.5, -0.5, 0.2)^T, \quad u_{RM} = (1, 0.5, 0.4)^T, \quad u_R = (0.05, 0, 0)^T. \quad (21b)$$

424 As in the previous tests, there is an excellent agreement between the DES and TDS solutions (Fig. 5). Notably,
 425 the DES run outperforms the TDS run by a significant factor, $Q_{\text{CPU}} = 4$. As shown in Fig. 5, choosing
 426 $R_{\text{PEP}} = 1$ leads to piecewise constant CPU rates in the active parts of the computational domain. This illus-
 427 trates the ability of DES-PEP to automatically synchronize updates characterized by quasi-uniform time
 428 scales.

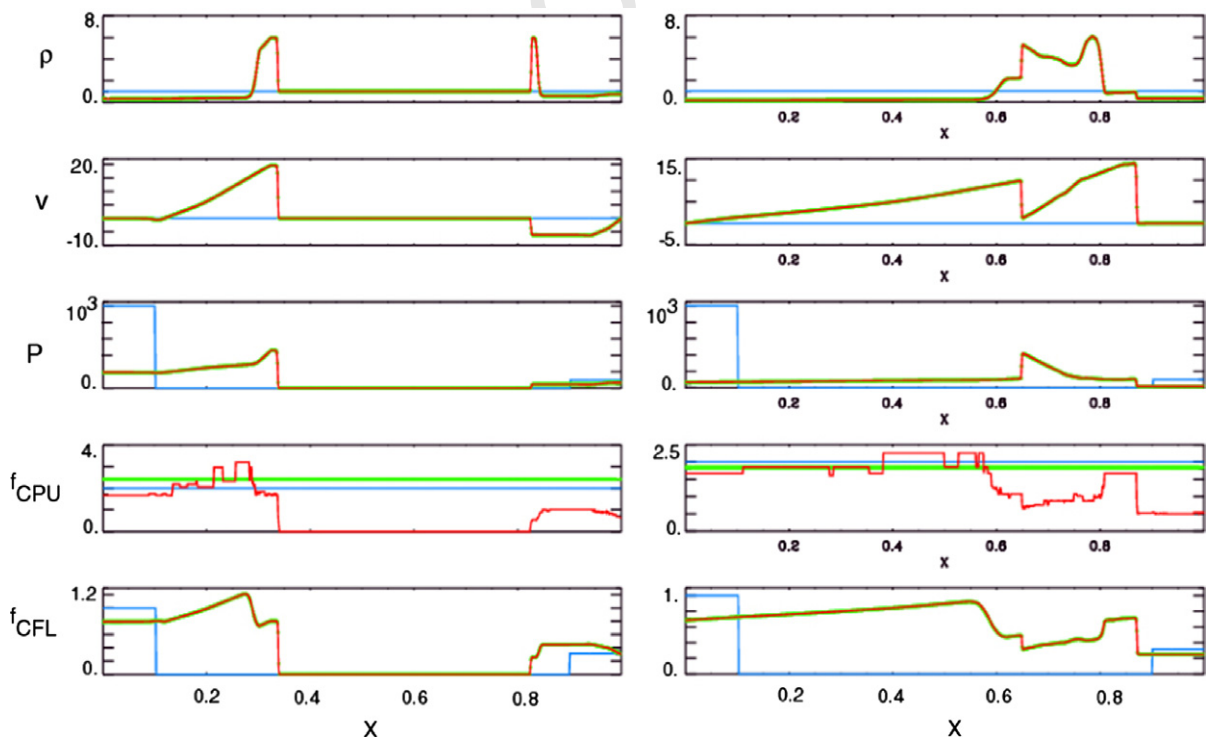


Fig. 3. Woodward–Colella test: $t_{\text{END}} = 0.01$ (left) and $t_{\text{END}} = 0.038$ (right), $R_{\text{PEP}} = 0.4$. Shown are three solutions: $u(t = 0)$ (blue), $u_{\text{DES}}(t_{\text{END}})$ (red) and $u_{\text{TDS}}(t_{\text{END}})$ (green). DES chooses update rates in accordance with local CFL conditions in the active regions of the computational domain. (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

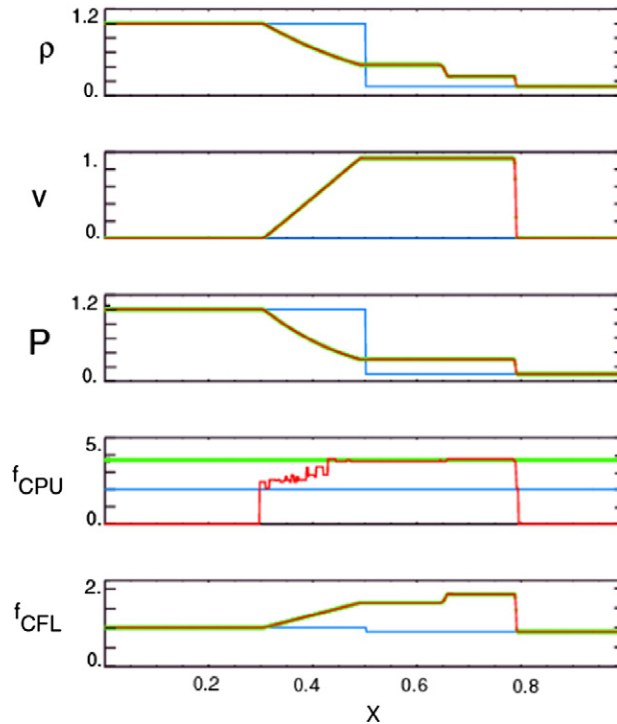


Fig. 4. Sod test: $t_{\text{END}} = 0.01644$, $R_{\text{PEP}} = 0.4$. Shown are three solutions: $u(t = 0)$ (blue), $u_{\text{DES}}(t_{\text{END}})$ (red) and $u_{\text{TDS}}(t_{\text{END}})$ (green). DES chooses update rates in accordance with local CFL conditions in the active regions of the computational domain. (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

429 7. Convergence study

430 In order to examine the temporal convergence of the proposed technique (i.e., the sensitivity of simulation
 431 errors to the magnitudes of local time increments), we conducted an additional series of runs for the “blowoff”
 432 case. In these runs we used the same grid spacing (given in Section 6.4), but varied the CFL number ω_{CFL} and
 433 the temporal approximation order $TimeOrder$ (p_T). An “exact” (reference) solution, f_{ref} was obtained in the
 434 TDS run with $\omega_{\text{CFL}} = 0.05$ and $TimeOrder = 2$. Relative numerical errors, η_L ($L = \infty, 2$) were computed
 435 for the density ρ (with other quantities behaving similarly) with respect to this reference solution using the
 436 maximum ($\|f\|_{\infty}$) and quadratic ($\|f\|_2$) norms (Tables 4 and 5):

$$438 \quad \eta_L = \|f - f_{\text{ref}}\|_L / \|f_{\text{ref}}\|_L, \quad (22)$$

439 The numerical rate of convergence, p_N was computed by assuming $\eta_L \sim \Delta t^{p_N}$ and employing two solutions ob-
 440 tained with two different (successive) values, $\omega_{\text{CFL},1}$ and $\omega_{\text{CFL},2}$:

$$442 \quad p_N = \ln(\eta_{L,2}/\eta_{L,1}) / \ln(\omega_{\text{CFL},2}/\omega_{\text{CFL},1}). \quad (23)$$

443 All DES runs were performed for two values of the PEP parameter, $R_{\text{PEP}} = 0.5$ and $R_{\text{PEP}} = 1$. As R_{PEP} de-
 444 creases, spatial profiles of the CPU rate f_{CPU} begin to better reflect the local nature of DES integration (Fig. 6).

445 The results of our convergence study (summarized in Tables 4 and 5) illustrate the ability of DES-PEP to
 446 produce accurate results for different values of R_{PEP} . Numerical rates, p_N , with which DES solutions converge,
 447 are found to be consistent with expected formal orders of temporal approximation, p_T . The accuracy of sec-
 448 ond-order DES calculations exceeds that observed in corresponding first-order DES runs by one-two orders of
 449 magnitude. Note that the magnitudes of errors observed both in the TDS and TDS runs are only meaningful
 450 in the context of studies conducted with the same slope limiter (in our case given by Eq. (10)). Use of different
 451 slope limiters may result in differences between solutions of order or greater than the magnitudes of errors

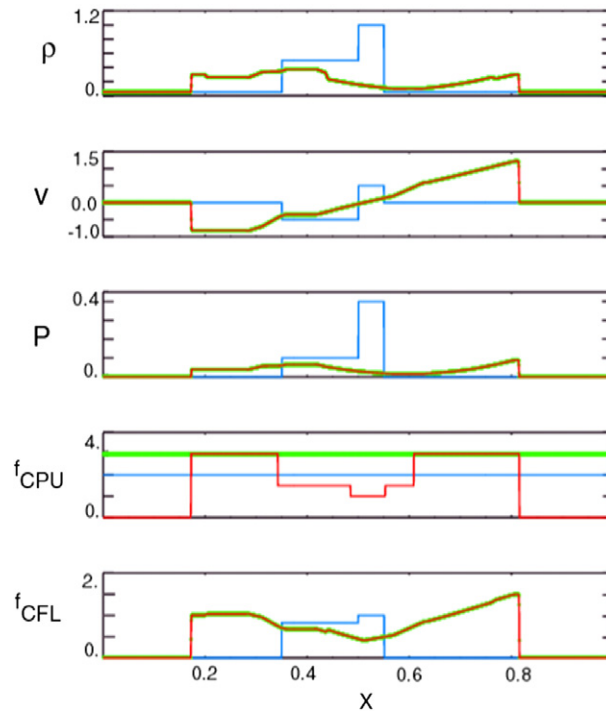


Fig. 5. “Blowoff” test: $t_{\text{END}} = 0.18$, $R_{\text{PEP}} = 1$. Shown are three solutions: $u(t = 0)$ (blue), $u_{\text{DES}}(t_{\text{END}})$ (red) and $u_{\text{TDS}}(t_{\text{END}})$ (green). The DES–PEP algorithm updates the numerical solution with piecewise constant update rates in the active regions of the computational domain. (For interpretation of the references in colour in this figure legend, the reader is referred to the web version of this article.)

452 given in Tables 4 and 5. A negative effect of slope limiters on convergence rate was also noted in formally sec-
453 ond-order MTS methods [7].

454 The demonstration of comparable accuracy achieved in DES–PEP runs with different values of R_{PEP} (Table
455 4) has important implications for boosting the scalability of future parallel DES–PEP implementations.
456 Indeed, as seen in Fig. 6, increasing the value of R_{PEP} leads to more uniform spatial distributions of CPU
457 resources (flatter profiles of f_{CPU}). Even though it also results in generating more computations per each
458 PEP loop, corresponding PEP synchronization rates proportionally decrease (Table 3). This particular flexi-
459 bility of DES–PEP should help automate fine-tuning of future DES–PEP applications to their optimum per-
460 formance on massively parallel computer architectures.

Table 4

Summary of relative errors, η_L and convergence rates, p_N for “blowoff” DES runs ($N_{\text{cell}} = 2000$) conducted with different values of the CFL number ω_{CFL} and the formal temporal approximation order p_T (TimeOrder)

p_T/ω_{CFL}	$\eta_{\infty}(p_N)$	$\eta_2(p_N)$	$\eta_{\infty}(p_N)$	$\eta_2(p_N)$
1/0.8	1.11E–01 (–)	2.68E–02 (–)	1.05E–01 (–)	2.21E–02 (–)
1/0.4	5.68E–02 (0.97)	1.00E–02 (1.42)	6.08E–02 (0.79)	9.79E–03 (1.17)
1/0.2	2.88E–02 (0.98)	4.22E–03 (1.24)	3.28E–02 (0.89)	4.03E–03 (1.28)
1/0.1	1.42E–02 (1.02)	2.02E–03 (1.06)	1.84E–02 (0.83)	1.93E–03 (1.06)
2/0.8	1.52E–02 (–)	9.41E–04 (–)	1.10E–02 (–)	7.43E–04 (–)
2/0.4	2.55E–03 (2.58)	1.68E–04 (2.49)	2.68E–03 (2.04)	1.79E–04 (2.05)
2/0.2	6.27E–04 (2.02)	4.12E–05 (2.03)	6.20E–04 (2.11)	4.36E–05 (2.04)
2/0.1	1.10E–04 (2.51)	1.04E–05 (1.99)	1.24E–04 (2.32)	1.20E–05 (1.86)

The error analysis was performed for two values of the PEP parameter R_{PEP} : $R_{\text{PEP}} = 0.5$ (the leftmost two error data columns) and $R_{\text{PEP}} = 1.0$ (the rightmost two columns).

Table 5

Summary of relative errors, η_L and convergence rates, p_N for “blowoff” TDS runs ($N_{\text{cell}} = 2000$) conducted with different values of the CFL number ω_{CFL} and the formal temporal approximation order p_T (TimeOrder)

p_T/ω_{CFL}	$\eta_{\infty}(p_N)$	$\eta_2(p_N)$
1/0.8	1.07E-01 (-)	2.10E-02 (-)
1/0.4	6.06E-02 (0.82)	8.48E-03 (1.19)
1/0.2	3.27E-02 (0.89)	3.71E-03 (1.03)
1/0.1	1.84E-02 (0.83)	1.82E-03 (1.18)
2/0.8	1.05E-02 (-)	6.84E-04 (-)
2/0.4	2.62E-03 (2.00)	1.63E-04 (2.07)
2/0.2	6.05E-04 (2.11)	3.83E-05 (2.09)
2/0.1	1.21E-04 (2.32)	7.97E-06 (2.26)

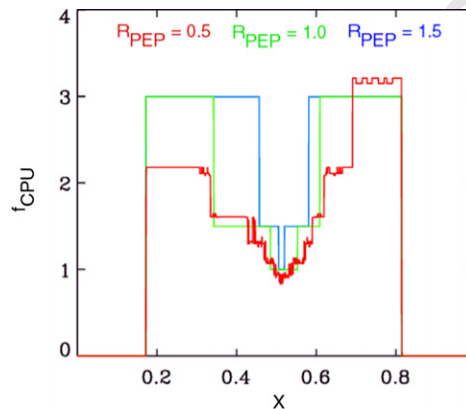


Fig. 6. Instantaneous spatial profiles of the normalized rate f_{CPU} observed at the end of “blowoff” DES runs performed with $\omega_{\text{CFL}} = 0.5$ and different values of the PEP parameter, R_{PEP} .

461 8. Summary

462 As mentioned in Section 1, large-scale gas-dynamics simulations may struggle with a disparity of spatially
 463 inhomogeneous time scales imposed by the governing physics and underlying mesh geometry. In particular,
 464 modeling multi-scale physical phenomena with explicit time-stepping schemes forces the global simulation
 465 to evolve with the smallest timestep allowed by the global CFL condition, despite the fact that large parts
 466 of the system may not need to be updated as frequently. Multiple time-stepping schemes (including hierarchi-
 467 cal time integration in SAMR) rely on selecting global synchronization time steps, which have to be *a priori*
 468 subdivided in hierarchical groups of local time steps based on flow (stability/accuracy) conditions existing at
 469 the beginning of each global time step. As local computations progress, these conditions may be violated. As a
 470 result, an initially chosen time-integration sequence may become inaccurate or unstable before the simulation
 471 reaches the end of the global synchronization step. Repairing such time-stepping hierarchies (by reducing
 472 unstable timesteps) may not always be efficient (in terms of CPU time) or robust (accuracy/stability-wise),
 473 especially when integrating highly nonlinear (e.g. reactive, turbulent) systems. In addition, time-stepping
 474 methods, in general, suffer from their inability to automatically detect noninformative (“idle”) parts of the
 475 simulation phase space, where they continue to waste CPU resources without producing meaningful changes
 476 to the solution.

477 To address these issues, we presented a second-order-in-time DES algorithm for gas dynamics, which, by
 478 tracking incremental changes to local flow variables, enables a physically intuitive approach to the time inte-
 479 gration of nonlinear systems with heterogeneous time scales. The self-adaptive sequence of event-driven
 480 updates in such systems ensures physical causality and local accuracy required, while preserving numerical
 481 fluxes in a time-averaged sense. We also showed that the new preemptive event processing (DES-PEP)

algorithm enables automatic synchronization of spatially distributed elements, which are characterized by close frequencies of numerical updates. This allows CPU-efficient asynchronous integration without having to pregroup such elements in timestep bins. Also, parallelization of DES–PEP codes (to be discussed elsewhere) becomes a more straightforward task, compared to standard optimistic and conservative techniques for DES systems [11].

It should be noted that the algorithm presented in this paper is especially well suited for inhomogeneous flows with a broad range of dynamic time scales, as well as numerical systems discretized on adaptive unstructured meshes. For instance, multi-phase porous flows [12] and combustion models [19,24,30,36] are often characterized by the presence of localized high-speed flows, fast diffusion and detailed chemistry reactions. For such problems, the DES–PEP approach is expected to produce a significant impact by enabling high-resolution explicit simulations, which are currently considered to be prohibitive.

In some compressible gas-dynamics applications (e.g. [28,29]) fast acoustic or gravity waves may dominate characteristic material flow velocities in large parts of the computational domain. Consequently, in its present form, DES would still have to account for these fast waves by satisfying the stiffest local CFL conditions, unless the forcing terms in the governing equations are artificially modified to speed up explicit computation (e.g. see Ref. [36]). In the long run this difficulty may be overcome by allowing slower parts of reacting and flux terms to be updated not as frequently as local (wave-driven) fluctuations. Moreover, it seems possible to envision a DES-based algorithm, which would make use of local averaging of fast time scales, similar to the global method of averages introduced in Ref. [28]. In addition, previous results from multi-physics event-driven simulations [23] suggest that aside from increased efficiency, the DES machinery may produce more robust (stable) and accurate solutions than analogous time-stepping codes.

The DES algorithm presented in this paper is mainly targeted towards semi-discrete ($O(\Delta t)$) numerical discretizations. For this type of schemes convergence of higher-order-in-time asynchronous integration can be achieved by making *a posteriori* flux corrections. It is also possible to devise DES algorithms for second-order-in-time unsplit discretizations of compressible Navier–Stokes equations (e.g. unsplit Riemann, Lax–Wendroff, central schemes, etc.). As already mentioned in the Introduction, such algorithms could be naturally built by assuming piecewise-linear flux trajectories at cell interfaces, which would in turn necessitate solving a quadratic equation for each state's Δt , given an appropriate threshold value, Δf . On the other hand, for finite-volume systems discretized via the ADER approach [8], higher temporal resolutions may be obtained even with the forward Euler scheme. Finally, extending the DES algorithm to multiple dimensions is deemed to be straightforward, as long as the underlying multi-dimensional spatial discretization scheme is adequate for a time-evolution problem in question. Work on the parallelization and extension of DES–PEP to multiple dimensions, nonuniform (structured and unstructured) meshes, higher-order spatial discretizations and MHD, will be reported in our future publications.

Acknowledgments

This research was supported by NSF ITR program Grant 0529919, NASA Grant NNG06GE65G, and NASA Supporting Research and Technology Grant in Solar and Heliospheric Physics to the Goddard Space Flight Center. The authors are grateful to Dr. M. Goldstein and Dr. A. Usmanov for their valuable advice and strong encouragement of this work.

Appendix

This appendix contains the pseudocode for a number of functions used in the DES and TDS algorithms illustrated in Tables 1 and 2 (Section 5), respectively. The prefix *Event::* precedes function names and indicates that these functions represent methods of the class *Event*, i.e., they operate on (have access to) data encapsulated in an object of the class *Event*, $e(i) \equiv \text{This}$. Table A1 presents update methods. Table A2 illustrates processing and synchronization methods. Table A3 contains a method implementing the second-order correction described by Eqs. (13)–(14). Finally, Table A4 formulates a scheduling procedure, which predicts event process times based on local CFL condition (12). Below we assume that during each PEP loop flux synchronization interfaces, $S_{(i+k)/2}$ (cell faces, which act as information “gateways” between adjacent cells, i and k) are

Table A1

Pseudocode of three *Event* class methods: *reconstruct()* constructs the left and right interface solutions (for better accuracy one-sided derivatives $A_{i\pm 1/2}$ in Eq. (10) are evaluated with solution states computed at the current clock time t_{clock}); *dfd* $t()$ computes the local rate-of-change; *update()* integrates the cell-centered solution u and the “flux capacitor” variable δu

```

function Event::reconstruct()
1: compute  $\bar{A}_i$  at  $t = t_{\text{clock}}$  // see Eq. (10)
2: for  $k = i + l, l = \pm 1$ :
3:   if ( $S_{(i+k)/2}$  is synchronized):
4:     reconstruct  $u_i^{(i+k)/2}$  // see Eq. (9)
5:     if ( $k$  is boundary): reconstruct  $u_k^{(i+k)/2}$ 
6:   endif
7: enddo
endfunction

function Event::dfd $t()$ 
1: for  $k = i + l, l = \pm 1$ :
2:   if ( $S_{(i+k)/2}$  is synchronized and not flux-computed):
3:     compute  $F_{(i+k)/2}$  // see Eq. (7)
4:   endif
5: enddo
6: compute  $R_i$  // see Eq. (6)
7:  $t_i = t_{\text{clock}}$  // reset internal timer
endfunction

function Event::update()
1:  $\Delta u_i = R_i(t_{\text{clock}} - t_i)$  // compute change since last update
2:  $u_i = u_i + \Delta u_i$  // increment solution
3:  $\delta u_i = \delta u_i + \Delta u_i$  // increment “flux capacitor”
endfunction

```

Table A2

Pseudocode of two *Event* class methods: *process()* executes the top event in *EventQueue*; *synchronize()* initiates a self-adaptive synchronization sequence, which enforces physical causality. States adjacent to physical boundaries are always synchronized with their neighboring states regardless of their “flux capacitor” values (see line 12 in *synchronize()*)

```

function Event::process()
1: if (This is not in PEPStack): // not already synchronized
2:   add This to PEPStack
3:   This.update()
4: endif
5: This.synchronize() // synchronize this state with neighbors
endfunction

function Event::synchronize()
1: invalidate This //  $This = e(i)$ 
2: if (This is not active): activate This
3:  $\delta u_i = 0$  // flush the flux capacitor for this state
4: for  $k = i + l, l = \pm 1$ :
5:   if ( $S_{(i+k)/2}$  is synchronized):
6:     continue // for-loop
7:   else if ( $k$  is boundary):
8:     apply boundary conditions for  $u_k$ 
9:   else if ( $e(k)$  is not in PEPStack): // if  $e(k)$  is not already synchronized
10:    add  $e(k)$  to PEPStack
11:     $e(k).update()$ 
12:    if ( $\|\delta u_k\| \geq \Delta u_k^{\text{tg}}$  or  $k \pm 1$  is boundary):  $e(k).synchronize()$  // preempt  $e(k)$ 
13:   endif
14: endfor
endfunction

```

Table A3

Pseudocode of the *Event* class method, *correct()*

```

function Event::correct()
1: if (This is not active): return // This = e(i)
2: for  $k = i + l, l = \pm 1$ :
3:   Syncok =  $S_{(i+k)/2}$  is synchronized and not flux-corrected
4:   if (Syncok and e(k) is active):
5:     compute  $\Delta u_{(i+k)/2}$  // see Eq. (14)
6:      $u_i = u_i - l \Delta u_{(i+k)/2}$ 
7:     if (This is valid):  $\delta u_i = \delta u_i - l \Delta u_{(i+k)/2}$ 
8:     if (k is not boundary):
9:        $u_k = u_k + l \Delta u_{(i+k)/2}$ 
10:      if (e(k) is valid):  $\delta u_k = \delta u_k + l \Delta u_{(i+k)/2}$ 
11:      endif
12:    endif
13:  endfor
14: for  $k = i + l, l = \pm 1$ :
15:   if (k is boundary): apply boundary conditions for  $u_k$ 
16: endfor
endfunction

```

This method implements the asynchronous second-order Euler correction (Eqs. (13)–(14)). Flux corrections are only allowed at cell interfaces connecting solution states marked as “active”.

Table A4

Pseudocode of the *Event* class method *schedule()*

```

function Event::schedule()
1: compute  $|\Delta u_i^{\text{CFL}}| = |R_i| \omega_{\text{CFL}} \Delta t_i^{\text{CFL}}$  // see Eq. (12)
2: if ( $|\Delta u_i^{\text{CFL}}| < \varepsilon$ ):  $\{\Delta u_i^{\text{tr}} = \varepsilon; \Delta t_i^{\text{tr}} = \infty\}$  else:  $\{\Delta u_i^{\text{tr}} = |\Delta u_i^{\text{CFL}}|; \Delta t_i^{\text{tr}} = \Delta t_i^{\text{tr}} / |R_i|\}$ 
3: if ( $t_{\text{clock}} == 0$ ):  $\Delta t_i = \Delta t_i^{\text{tr}}$  else:  $\Delta t_i = t_{\text{clock}} - t_i^{\text{last}}$  // compute  $\Delta t_i$  for PEP
4:  $t_i^{\text{last}} = t_{\text{clock}}$ 
5: if ( $\Delta t_i^{\text{tr}} \geq t_{\text{END}}$ ): // this is problem-dependent condition
6: deactivate This
7: else:
8: if (This is not active): activate This
9:  $t_i^{\text{proc}} = t_{\text{clock}} + \Delta t_i^{\text{tr}}$  // estimate next process time
10: add This to EventQueue
11: endif
endfunction

```

This particular implementation uses local CFL condition (12) to compute the target solution increment Δu_i^{tr} (line 2). The small parameter ε is taken to be of the order of machine roundoff.

530 logically flagged. Additional flags are also used to avoid making duplicate flux corrections (see Tables A1 and
 531 A3). Note that in the time-stepping algorithm (Table 2) fluxes are automatically computed/corrected once per
 532 face since they are evaluated in a sequential, face-centered order, as opposed to an arbitrary (physically dri-
 533 ven), cell-centered order, characteristic of DES.

534 References

- 535 [1] J. Banks (Ed.), Handbook of Simulation, John Wiley & Sons Inc., 1998.
 536 [2] J. Bell, M. Berger, J. Saltzman, M. Welcome, Three-dimensional adaptive mesh refinement for hyperbolic conservation laws, SIAM J.
 537 Sci. Comput. 15 (1) (1994) 127–138.
 538 [3] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484.
 539 [4] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1989) 64.
 540 [5] J.J. Biesiadecki, R.D. Skeel, Dangers of multiple time step methods, J. Comput. Phys. 109 (1993) 318–328.
 541 [6] A.J. Crossley, N.G. Wright, Time accurate local timestepping for the unsteady shallow water equations, Int. J. Numer. Methods
 542 Fluids 48 (2005) 775–799.

- 543 [7] C. Dawson, R. Kirby, High resolution schemes for conservation laws with locally varying time steps, *SIAM J. Sci. Comput.* 22 (6)
544 (2001) 2256.
- 545 [8] M. Käser, M. Dumbser, An arbitrary high order discontinuous Galerkin method for elastic waves on unstructured meshes V: local
546 time stepping and p -adaptivity, *Geophys. J. Int.*, submitted for publication.
- 547 [9] B. Einfeldt, On Godunov-type methods for gas-dynamics, *SIAM J. Numer. Anal.* 25 (1988) 357–393.
- 548 [10] J.E. Flaherty, R.M. Loy, M.S. Shephard, B.K. Szymanski, J.D. Teresco, L.H. Ziantz, Adaptive local refinement with octree load
549 balancing for the parallel solution of three-dimensional conservation laws, *J. Parallel Distrib. Comput.* 47 (1997) 139–152.
- 550 [11] R.M. Fujimoto, *Parallel and Distributed Simulation Systems*, Wiley Interscience, 2000.
- 551 [12] M.G. Gerritsen, L.J. Durlofsky, Modeling fluid flow in oil reservoirs, *Annu. Rev. Fluid Mech.* 37 (2005) 211–238.
- 552 [13] H. Karimabadi, J. Driscoll, Y.A. Omelchenko, N. Omidi, A new asynchronous methodology for modeling of physical systems:
553 breaking the curse of Courant condition, *J. Comput. Phys.* 205 (2) (2005) 755–775.
- 554 [14] H. Karimabadi, J. Driscoll, J. Dave, Y. Omelchenko, K. Perumalla, R. Fujimoto, N. Omidi, Parallel discrete event simulations of
555 grid-based models: asynchronous electromagnetic hybrid code, *Lect. Notes Comput. Sci.* 3732 (2006) 573–582.
- 556 [15] W.L. Kleb, J.T. Batina, M.H. Williams, Temporal adaptive Euler/Navier–Stokes algorithm involving unstructured dynamic meshes,
557 *AIAA J.* 32 (9) (1994) 1926–1928.
- 558 [16] A. Kurganov, E. Tadmor, New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations,
559 *J. Comput. Phys.* 160 (2000) 241–282.
- 560 [17] A. Kurganov, S. Noelle, G. Petrova, Semidiscrete central-upwind schemes for hyperbolic conservation laws and Hamilton–Jacobi
561 equations, *SIAM J. Sci. Comput.* 23 (3) (2001) 707–740.
- 562 [18] A. Lew, J.E. Madsen, M. Ortiz, M. West, Asynchronous variational integrators, *Arch. Rational Mech. Anal.* 167 (2003) 85.
- 563 [19] R.R. Linn, P. Cunningham, Numerical simulations of grass fires using a coupled atmosphere–fire model: basic fire behavior and
564 dependence on wind speed, *J. Geophys. Res.* 110 (2005) D13107, doi:10.1029/2004JD00559.
- 565 [20] J. Makino, P. Hut, M. Kaplan, H. Saygun, A time-symmetric block time-step algorithm for N -body simulations, *New Astron.* 12
566 (2006) 124–133.
- 567 [21] J. Nutaro, B.P. Zeigler, R. Jammalamadaka, S. Akerkar, Discrete event solution of gas dynamics within the DEVS framework, *Lect.*
568 *Notes Comput. Sci.* 2660 (2003) 319–328.
- 569 [22] Y.A. Omelchenko, H. Karimabadi, Event-driven hybrid particle-in-cell simulation: a new paradigm for multi-scale plasma modeling,
570 *J. Comput. Phys.* 216 (1) (2006) 153–178.
- 571 [23] Y.A. Omelchenko, H. Karimabadi, Self-adaptive time integration of flux-conservative equations with sources, *J. Comput. Phys.* 216
572 (1) (2006) 179–194.
- 573 [24] E.S. Oran, J.P. Boris, *Numerical Simulation of Reactive Flows*, second ed., Cambridge University Press, 2001, Naval Research
574 Laboratory.
- 575 [25] J.M. Owen, J.V. Villumsen, P.R. Shapiro, H. Martel, Adaptive smoothed particle hydrodynamics: methodology. II, *Astrophys. J.*
576 *Suppl.* 116 (1998) 115–209.
- 577 [26] D. Peng, B. Merriman, S. Osher, H. Zhao, M. Kang, A PDE-based fast local level set method, *J. Comput. Phys.* 155 (1999) 410–438.
- 578 [27] J. Raeder, Global geospace modeling: tutorial and review, in: J. Büchner, C.T. Dam, M. Scholer (Eds.), *Space Plasma Simulation*,
579 *Lecture Notes in Physics*, vol. 615, Springer-Verlag, Berlin, 2003.
- 580 [28] J.M. Reisner, S. Wynne, L. Margolin, R.R. Linn, Coupled atmospheric–fire modeling employing the method of averages, *Month.*
581 *Weather Rev.* 128 (2000) 3683–3691.
- 582 [29] J.M. Reisner, V.A. Mousseau, A.A. Wyszogrodzki, D.A. Knoll, An implicitly balanced hurricane model with physics-based
583 preconditioning, *Month. Weather Rev.* 133 (2005) 1003–1022.
- 584 [30] V. Sankaran, S. Menon, LES of spray combustion in swirling flows, *J. Turbul.* 3 (2002) 1–23.
- 585 [31] G. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, *J. Comput. Phys.* 27
586 (1978) 1–30.
- 587 [32] Y.-G. Tao, W.K. den Otter, J.K.G. Dhont, W.J. Briels, Isotropic–nematic spinodals of rigid long thin rodlike colloids by event-driven
588 Brownian dynamics simulations, *J. Chem. Phys.* 124 (2006) 134906.
- 589 [33] Y.R. Tang, K.S. Perumalla, R.M. Fujimoto, H. Karimabadi, J. Driscoll, Y. Omelchenko, Optimistic simulations of physical systems
590 using reverse computation, *Trans. Soc. Model. Simul. Int.* 82 (1) (2006) 61–73.
- 591 [34] A.V. Usmanov, M.L. Goldstein, A tilted-dipole MHD model of the solar corona and solar wind, *J. Geophys. Res.* 108 (A9) (2003)
592 1354.
- 593 [35] H. van der Ven, B.E. Niemann-Tuitman, A.E.P. Veldman, An explicit multi-time-stepping algorithm for aerodynamic flows, *J. Comp.*
594 *Appl. Math.* 82 (1997) 423–431.
- 595 [36] Y. Wang, A. Trouvé, Artificial acoustic stiffness reduction in fully compressible, direct numerical simulation of combustion, *Combust.*
596 *Theory Model.* 8 (2004) 633–660.
- 597 [37] P. Woodward, P. Colella, The numerical solution of two-dimensional fluid flow with strong shocks, *J. Comput. Phys.* 54 (1988) 115–
598 173.
- 599 [38] X.D. Zhang, J.-Y. Trépanier, M. Reggio, R. Camarero, Time-accurate local time stepping method based on flux updating, *AIAA J.*
600 30 (8) (1994) 1980–1985.
- 601 [39] B.P. Zeigler, H. Praehofer, T.G. Kim, *Theory of Modeling and Simulation*, second ed., Academic Press, 2000.
- 602 [40] U. Ziegler, A central-constrained transport scheme for ideal magnetohydrodynamics, *J. Comput. Phys.* 196 (2004) 293–416.
- 603