JOURNAL OF COMPUTATIONAL PHYSICS 118, 9–23 (1995)

The Product Formula Algorithm Applied to Linear and Radiation Diffusion*

FRANK R. GRAZIANI

Lawrence Livermore National Laboratory, P.O. Box 808, L-35, Livermore, California 94550

Received August 11, 1994

We extend an unconditionally stable, explicit algorithm due to DeRaedt and Richardson, Farrell, and Long to include various linear and radiation diffusion problems in one and two dimensions with open and/or reflecting boundary conditions. We consider the ramifications of the ordering ambiguity problem (a feature that arises in the product formula scheme). In order to improve accuracy, we introduce a new type of subcycling based on the Lie–Trotter product formula. We consider a one-dimensional test problem which consists of a slab of material with a constant driving temperature source on one side. We compare the analytic and numerical results for the time evolution of the temperature profile in the linear and radiation diffusion problems as a function of Courant factor (α). We find excellent agreement except when $\alpha \gg 1$. For large α, the transient temperature profiles exhibit a "staircase' like behavior. However, we show (albeit, not rigorously) that all solutions regardless of α approximately converge to the correct steady state solution. We also present results for a two-dimensional problem consisting of a constant driving temperature source on one side of a slab of material with an optically thick region interior to the slab.

I. INTRODUCTION

Recently, there has been a resurgence of interest in a new generation of explicit algorithms and their application to a variety of physics problems [1–4]. Like their predecessors, these algorithms are easy to code. However, unlike their predecessors, these schemes tend to have improved stability characteristics. Currently, there are several algorithms available that are both explicit and unconditionally stable. One is the symplectic integrators which have enjoyed success in N body simulations of various plasma and astronomical phenomena [2]. These schemes tend to be applied to Hamiltonian systems where it is important that the finite difference algorithm preserve the Poincaré invariants. Another is the algorithm due to Livne and Glasner [3]. It is a scheme which evaluates neighboring zones at the old time step and the zone of interest at the new time step. This globally explicit but locally implicit algorithm has been applied to a variety of heat conduction problems [3]. It possesses the unusual feature of being non-energy conserving. Energy conservation is enforced by hand by adding a source term to the finite difference equations. The final scheme, and the one of interest in this paper, is the product formula (PF) algorithm due to DeRaedt [1]. As far as we are aware, the algorithm has essentially been ignored until Richardson, Farrell, and Long (RFL) [4] realized its importance in light of the existence of new computer architectures (massively parallel computers). DeRaedt's original focus was on the time dependent Schrodinger equation with periodic boundary conditions. He was able to show up to a factor of 10 speedup (at a comparable accuracy) compared with Crank–Nicholson. Because DeRaedt's focus was on linear equations, RFL extended the algorithm to include non-linearities (in particular, the one-dimensional Burger's equation). Although nothing concrete was done in their paper concerning multiple dimensions or unstructured meshes, they do discuss these points.

At present, there exists a large number of unresolved issues concerning the PF scheme. First is the practical realization of a two-dimensional problem. Second is the application of the PF scheme to a variety of boundary conditions (open, reflecting, and periodic) with the possibility of external sources. Third is the accuracy as a function of the Courant factor and the approach to steady state. Although one normally would not think of running an explicit scheme at large Courant factors (accuracy and not stability being the limiting factor in the new generation of explicit algorithms), it is still useful to understand the behavior of the PF scheme in this regime. Finally, the ordering

* Work Performed under the auspices of the Department of Energy at Lawrence Livermore National Laboratory under Contract W-7403-ENG-48.

0021-9991/95 $6.00

ambiguity problem (which we make precise in the next section) is an issue which needs to be addressed. In this paper we attempt to answer these problems. We apply the PF algorithm to both linear and non-linear diffusion problems (the latter being the transport of radiation energy density) in one and two dimensions with reflecting and open boundary conditions. In addition, arbitrary time dependent external sources are capable of being applied at the open boundaries.

II. THE LINEAR DIFFUSION PROBLEM

a. *Theory*

In order to understand the PF algorithm and the relevant issues raised in the Introduction, it is sufficient to consider the one-dimensional linear diffusion equation,

$$\frac{\partial}{\partial t}T(x,t)=\frac{\partial}{\partial x}\left(D\frac{\partial}{\partial x}T(x,t)\right), \tag{2.1}$$

where D is the diffusion coefficient (a constant) and T is a temperature. We wish to solve Eq. (2.1) for either open or reflecting boundary conditions. We present two methods for solving Eq. (2.1) when external sources are present. The first choice is an obvious one and involves spatially discretizing Eq. (2.1) and forming an N (N = number of zones) component matrix $\hat{T}$ which corresponds to the zone centered temperature, we may rewrite Eq. (2.1) as

$$\frac{d}{dt}=\frac{D}{\Delta x^2}\hat{L}\hat{T}+\hat{S}, \tag{2.2}$$

where Δx is the zone size and $\hat{S}$ is the source rate. For open–open (OO) (our convention is that the first label denotes the boundary type for the left-hand side while the second applies to the right-hand side), reflecting–reflecting (RR), (RO), and (OR) boundary conditions the matrix $\hat{L}$ can be written

$$\hat{L}=\begin{bmatrix} -1-\Theta_L & 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & \dots & 0 & 1 & -1-\Theta_R \end{bmatrix}, \tag{2.3}$$

where boundary flags Θ_L and Θ_R are chosen to be $\Theta = 1$ if a boundary is open and if it is reflecting, $\Theta = 0$.

Formally, the solution of Eq. (2.2) (for constant $\hat{L}$ only!) is given by

$$\hat{T}(t+\Delta t)=\exp(\alpha\hat{L})\,T(t) + \int_0^{\Delta t} d\tau \exp\left(\frac{D\tau}{\Delta x^2}\hat{L}\right)\hat{S}(t+\Delta t-\tau). \tag{2.4}$$

Where $\alpha = (D\,\Delta t)/(\Delta x^2)$ is the Courant factor and the second term (the source integral) in Eq. (2.4) is present only for open boundary conditions with an external source. This is a generalization of the solution presented by RFL. Note that the traditional explicit stability limit is $\alpha = \frac{1}{2}$. For an OO type problem, for example, the source function $\hat{S}$ is an N component vector given by

$$\hat{S}(t)=\frac{D}{\Delta x^2}\begin{vmatrix} T_L(t) \\ 0 \\ \dots \\ \dots \\ 0 \\ T_R(t) \end{vmatrix} \tag{2.5}$$

T_L and T_R are the left- and right-hand side external sources, respectively. It should be noted that the inclusion of internal sources can be made simply by replacing, in the above column matrix, the appropriate zero by a source temperature. This will not be of interest in the present paper and, hence, we will assume that the source term is of the form Eq. (2.5).

Traditional explicit and implicit algorithms rely on constructing rational approximations to the time step operator $\exp(\alpha\hat{L})$ (the so-called Padé approximant) [5]. The PF algorithm, however, offers an alternative approximation to the time step operator. First, the matrix $\hat{L}$ can be decomposed into a sum of two matrices, each of which is block diagonal (up to the bordering rows and columns). Following DeRaedt's convention, we call these $\hat{L}_e$ and $\hat{L}_o$. They are given by

$$\hat{L}_e=\begin{bmatrix} -1 & 1 & 0 & 0 & \dots & \dots & 0 \\ 1 & -1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & -1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & -1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & 0 & -1 & 1 \\ 0 & 0 & \dots & \dots & \dots & 1 & -1 \end{bmatrix},$$

$$\hat{L}_o=\begin{bmatrix} -\Theta_L & 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 1 & -1 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & -\Theta_R \end{bmatrix}. \tag{2.6}$$

Using the Campbell–Baker–Hausdorf formula [6], the time step operator (which involves the sum of block diagonal matri-

ces) can be written approximately as a product of exponentials of $\hat{L}_e$ and $\hat{L}_o$. This latter fact is useful, since the exponential of a block diagonal matrix can be evaluated exactly. In fact, it is straightforward to show that if we can decompose $\hat{L}$ into a sum of terms each having the form

$$\begin{vmatrix} a_1 & b_1 & 0 & 0 & \dots & \dots & 0 \\ c_1 & d_1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & a_2 & b_2 & 0 & \dots & 0 \\ 0 & 0 & c_2 & d_2 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & a_n & b_n \\ 0 & 0 & 0 & 0 & 0 & c_n & d_n \end{vmatrix}, \tag{2.7}$$

where each 2×2 block has eigenvalues λ_i and Λ_i (not necessarily distinct) with the block index i going from one to N then we can compute the exponential of the above matrix by using Putzer's method [7]. Let $\hat{A}_i$ represent a particular block, then the exponential consists of the same block diagonal structure with

$$\hat{A}_i \rightarrow \exp(\lambda_i)\frac{(\hat{A}_i - \Lambda_i \hat{I})}{(\lambda_i - \Lambda_i)} + \exp(\Lambda_i)\frac{(\hat{A}_i - \lambda_i \hat{I})}{(\Lambda_i - \lambda_i)}, \tag{2.8}$$

if the eigenvalues are distinct, and

$$\hat{A}_i \rightarrow \exp(\lambda_i)\,(\hat{I} + (\hat{A}_i - \lambda_i \hat{I})), \tag{2.9}$$

if the eigenvalues are equal ($\hat{I}$ is the identity matrix). All of the expressions contained in RFL concerning exponentiation of a particular block diagonal matrix can be derived simply by using Eq. (2.8) or Eq. (2.9). A more practical and efficient method for exponentiating matrices involves using Mathematica [8]. We have found it a useful tool for not only checking our PF code but it also offers useful insight into the mathematical properties of the PF algorithm. Details concerning the use of Mathematica and its applications to the PF algorithm are presented in the next section. Exponentiating Eq. (2.6) yields

$$\exp(\alpha\hat{L}_e) = \begin{vmatrix} \Delta_1 & \Delta_2 & 0 & 0 & \dots & \dots & 0 \\ \Delta_2 & \Delta_1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & \Delta_1 & \Delta_2 & 0 & \dots & 0 \\ 0 & 0 & \Delta_2 & \Delta_1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & 0 & \Delta_1 & \Delta_2 \\ 0 & 0 & \dots & \dots & 0 & \Delta_2 & \Delta_1 \end{vmatrix} \tag{2.10}$$

and

$$\exp(\alpha\hat{L}_o) = \begin{vmatrix} \exp(-\alpha\Theta_L) & 0 & 0 & 0 & \dots & \dots & 0 \\ 0 & \Delta_1 & \Delta_2 & 0 & 0 & \dots & 0 \\ 0 & \Delta_2 & \Delta_1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \Delta_1 & \Delta_2 & \dots & 0 \\ \dots & \dots & \dots & \Delta_2 & \Delta_1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & \dots & 0 & \exp(-\alpha\Theta_R) \end{vmatrix}, \tag{2.11}$$

where $\Delta_{1(2)} = (1 \pm \exp(2\alpha))/2$. The second-order accurate [1, 3] time advanced solution with external sources present becomes

$$\hat{T}(t + \Delta t) = \exp(\alpha\hat{L}_e)\exp(\alpha(\hat{L}_o/2))\exp(\alpha\hat{L}_e) \left(\hat{T}(t) + \frac{\Delta t}{2}\hat{S}(t)\right) + \frac{\Delta t}{2}\hat{S}(t + \Delta t) \tag{2.12}$$

with the exponentials given by Eq. (2.10) and Eq. (2.11). We have approximated the integral using the trapezoidal rule. Equation (2.12) is evidence of what we call the ordering ambiguity problem. It is possible to come up with an entirely different approximation to the time step operator which is also second-order accurate simply by making the transformation $\hat{L}_e \leftrightarrow \hat{L}_o$. In addition we can also form a symmetrized sum which is also second-order accurate; that is, $(\exp(\alpha\hat{L}_e)\exp(\alpha\hat{L}_o) + \exp(\alpha\hat{L}_o)\exp(\alpha\hat{L}_e))/2$. We discuss in the numerical section the differences and similarities between these various choices. Unfortunately, the explicit algorithm as presented above, which is unconditionally stable due to the boundedness of each term making up the product formula, has had an instability introduced via the approximate form of the source integral. Whereas $\exp(\alpha\hat{L}_e)$ and $\exp(\alpha\hat{L}_o)$ and its various products are bounded as the time step goes to infinity, the same is not true for any numerical form of the source integral whose error remainder is some power of the time step. Hence, the time advanced temperature is no longer bounded from above. In numerical simulations we have performed of a linear diffusion problem consisting of a slab heated from one side with a constant source, at a sufficiently large time step (usually on the order of the Courant condition), the temperature in the interior zones can become higher than the source temperature!—a completely unacceptable result. Besides the trapezoidal rule discussed above, we have also tried Gaussian quadrature. This method also has its problems, since for large α the integrand of the source integral is highly peaked about the lower limit of integration. Hence, one would have to make sure that the integrand was appropriately sampled. Fortunately, there exists a simpler alternative which we discuss below. It should be stressed that the first method is entirely satisfactory if sufficiently small time steps are used. The advantages are a minor extension of the

presently known PF algorithm. The disadvantages are, of course, mentioned above. Since one of the issues we are interested in in this paper is the behavior of the PF algorithm at large time steps, we will adopt a different technique.

The second method is based on increasing number of zones to $N + 2$ with two phony zones (their purpose is to enforce the boundary conditions) on either side of the first and last real zone. Our temperature vector now consists of the following entries:

$$\hat{T}(t) = \begin{vmatrix} T_L(t)\,\Theta_L + T_1(1-\Theta_L) \\ T_1 \\ T_2 \\ \cdots \\ T_N \\ T_R(t)\,\Theta_R + T_N(1-\Theta_R) \end{vmatrix}. \tag{2.13}$$

For reflecting boundaries ($\Theta_{L(R)} = 0$) the above temperature vector implies a redundant set of equations for the phony zones.

For open boundaries ($\Theta_{L(R)} = 1$) the meaning of Eq. (2.13) is that we are supplementing our original diffusion equation with user-supplied evolution equations for the sources. For example, if the left boundary is open we write the source rate as

$$\frac{d}{dt}T_L(t) = \left(\frac{d}{dt}\log T_L(t)\right) T_L(t). \tag{2.14}$$

The sources can either be included explicitly or implicitly into the expanded time step operator. For example, to first order in the time step, Eq. (2.14) can be written

$$T_L(t+\Delta t) = (1 + \Delta t\gamma_L(t))\, T_L(t) \text{ (explicit)},$$

$$T_L(t+\Delta t) = \frac{T_L(t)}{(1-\Delta t\gamma_L(t))} \text{ (implicit)},$$

where $\gamma_L(t)$ is the time derivative of the logarithm of the external temperature source applied at the left boundary evaluated at the previous time step (analogous expressions hold for the right hand side source temperatures). If we define Γ as the factor multiplying $T(t)$ in the above explicit or implicit source equations, then we have for the full diffusion matrix with sources

$$\hat{L} = \begin{bmatrix} \Gamma_L\Theta_L & -1+\Theta_L & 1-\Theta_L & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \Theta_L & -1-\Theta_L & 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ \cdots & 0 & 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & 0 & 0 & 1 & -2 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & -1-\Theta_R & \Theta_R \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & 1-\Theta_R & -1+\Theta_R & \Gamma_R\Theta_R \end{bmatrix} \tag{2.15}$$

which is more complicated than the matrices encountered in either DeRaedt or RFL but which still possesses a simple structure when decomposed and exponentiated. The time advanced solution to the linear diffusion equation is given by Eq. (2.12) without the source integral. That is,

$$\hat{T}(t+\Delta t) = \exp(\alpha\hat{L})\hat{T}(t). \tag{2.16}$$

Therefore, we have transformed the task of exponentiating an $N \times N$ matrix with a source integral into exponentiating an $(N+2)\times(N+2)$ matrix. Decomposing Eq. (2.15) into almost block diagonal parts yields

$$\hat{L}_e \approx \begin{bmatrix} 0 & -1+\Theta_L & 1-\Theta_L & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & -1 & 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & -1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \cdots & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 & 1-\Theta_R & -1+\Theta_R & 0 \end{bmatrix}, \tag{2.17}$$

$$\hat{L}_o = \begin{bmatrix} \Gamma_L\Theta_L & 0 & 0 & 0 & \dots & \dots & \dots & \dots & 0 \\ \Theta_L & -\Theta_L & 0 & 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & \dots & 0 & -\Theta_R & \Theta_R \\ 0 & 0 & 0 & \dots & \dots & \dots & \dots & 0 & \Gamma_R\Theta_R \end{bmatrix}. \tag{2.18}$$

Unlike previous works on the PF algorithm, the decomposition is not unique. The impact of this feature on the time step operator is at present unknown. Using Mathematica to exponentiate the above matrices yields

$$\exp(\alpha\hat{L}_e) = \begin{bmatrix} 1 & -\Delta_2(1-\Theta_L) & \Delta_2(1-\Theta_L) & 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & \Delta_1 & \Delta_2 & 0 & 0 & 0 & \dots & \dots & 0 \\ 0 & \Delta_2 & \Delta_1 & 0 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & \Delta_1 & \Delta_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \Delta_2 & \Delta_1 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & 0 & \Delta_1 & \Delta_2 & 0 \\ 0 & 0 & 0 & \dots & \dots & 0 & \Delta_2 & \Delta_1 & 0 \\ 0 & 0 & 0 & \dots & \dots & 0 & \Delta_2(1-\Theta_R) & -\Delta_2(1-\Theta_R) & 1 \end{bmatrix}, \tag{2.19}$$

$$\exp(\alpha\hat{L}_o) = \begin{bmatrix} \exp(\Gamma_L\Theta_L) & 0 & 0 & 0 & \dots & \dots & \dots & \dots & 0 \\ \frac{\alpha\zeta_L}{\alpha+\Gamma_L} & \exp(-\alpha\Theta_L) & 0 & 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & \Delta_1 & \Delta_2 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & \Delta_2 & \Delta_1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & 0 & \Delta_1 & \Delta_2 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \Delta_2 & \Delta_1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \dots & 0 & \exp(-\alpha\Theta_R) & \frac{\alpha\zeta_R}{\alpha+\Gamma_R} \\ 0 & 0 & 0 & 0 & \dots & \dots & \dots & 0 & \exp(\Gamma_R\Theta_R) \end{bmatrix}, \tag{2.20}$$

where $\zeta_{L(R)} = \exp(\Gamma_{L(R)}\Theta_{L(R)}) - \exp(-\alpha\Theta_{L(R)})$. Note that for a constant source, the corner elements of Eq. (2.20) are just one. For a time dependent source, the Γ's change and, hence, the corner elements of the exponential of the odd matrix must be revaluated each cycle. Hence (using a particular ordering choice), the temperature profile is advanced according to

$$\hat{T}(t+\Delta t) = \exp(\alpha\hat{L}_e)\exp(\alpha(\hat{L}_o/2))\exp(\alpha\hat{L}_e)\hat{T}(t). \tag{2.21}$$

As mentioned earlier, the above expression is second-order accurate in the sense that the remainder term is proportional

to $O[\alpha^2]$. We introduce a new way of improving the accuracy of Eq. (2.21) without necessarily going to a higher order PF. This method is based on the Lie–Trotter product formula [9] and is defined thusly,

$$\exp(\alpha(\hat{L}_e + \hat{L}_o)) = \lim_{s\to\infty}\left(\exp\left(\frac{\alpha\hat{L}_e}{s}\right)\exp\left(\frac{\alpha\hat{L}_o}{2s}\right)\exp\left(\frac{\alpha\hat{L}_e}{s}\right)\right)^s. \tag{2.22}$$

The reader can think of this as a type of subcycling. If s is finite, the error involved is $O[(\alpha/s)^2]$. The use of the Lie–Trotter product formula is helpful when accuracy needs to be maintained at large time steps. For the linear problem, the number of subcycles, s, is calculated by taking the Courant factor (which is specified by the time step, zone size, and diffusion coefficient) and dividing by a Courant factor at which we wish to subcycle. Before discussing our numerical calculations using Eq. (2.22) and, consequently, some of the issues raised in the Introduction, it is important to mention how to solve problems using the PF scheme in higher dimensions. The simplest implementation of PF to problems in more than one dimension involves operator splitting. That is, we treat each direction as a separate one-dimensional problem. For example, given an x–y structured mesh, we first perform a gather operation and form 1D temperature strips by sweeping along the x direction. The temperatures in a given x strip are advanced half a cycle by applying the product formula given by Eq. (2.21). After this has been done for the whole mesh, the advanced temperatures are gathered into 1D strips in the y direction. The final updated temperature is then arrived at by applying Eq. (2.21) to each temperature strip. It should be mentioned that applying the results of this section to 1D strips in the y direction requires that we make the replacement *left* $\rightarrow$ *bottom* and *right* $\rightarrow$ *top*.

b. *Numerical*

Once the vector temperature strips are formed via a "gather" operation, the time advanced solution arises by a sequence of (almost) tridiagonal matrix–vector multiply operations. Define $V(n)$ as a temporary vector, where n is between one and $N + 2$. Define $V'(n)$ as the result of multiplying $V(n)$ either by Eq. (2.19) or by Eq. (2.20). In component form we have

$$V'(n) = a(n-1)V(n-1) + b(n)V(n) + c(n+1)V(n+1), \tag{2.23}$$

where

$$2 \le n \le N+1$$

$$V'(1) = b(1)V(1) + c(2)V(2) + \Omega(3)V(3) \tag{2.24}$$

$$V'(N+2) = \Omega(N)V(N) + a(N+1)V(N+1) + b(N+2)V(N+2). \tag{2.25}$$

In this format, once a, b, and c are given, the matrix–vector multiply is in fully vectorized form. If the matrix is $\exp(\alpha\hat{L}_e)$ then the coefficients are given by $a(N+1) = -\Delta_2(1-\Theta_R)$ and $a(n) = 0$ for all other odd n, $a(n) = \Delta_2$ for even n; $b(1) = b(N+2) = 1$ and $b(n) = \Delta_1$ for all other n; $c(2) = -\Delta_2(1-\Theta_L)$ and $c(n) = 0$ for all other even n and $c(n) = \Delta_2$ for odd n; $\Omega(3) = \Delta_2(1-\Theta_L)$ and $\Omega(N) = \Delta_2(1-\Theta_R)$. For the exponential of the odd matrix the coefficients are similarly defined.

We have written a code using the PF algorithm for solving linear and non-linear diffusion problems. The code possesses the following properties:

a. 2D planar geometry with reflecting and open boundary conditions

b. Time dependent external sources

c. Sub-cycling based on the Lie–Trotter product formula.

The code has additional properties when it is solving a non-linear diffusion problem using the PF method. These will be discussed in the next section.

Although we have looked at a variety of 1D linear diffusion problems using our code, we present results only from a representative example. We consider a 1D linear diffusion problem with a constant driving temperature on the left-hand side of a domain of length s and a fixed temperature of zero on the right-hand side. The initial temperature profile is taken to be zero. For our calculations, we took the length of the slab to be 5 cm and the temperature source to be 1 kev. The diffusion coefficient was taken to be 1 cm^2/s. For simplicity, we only consider second-order accurate representations of the time step operator. The number of zones is taken to be 100. Our results are presented in Figs. 1–3. Figures 1a–d show the time developing profile for four different values of the Courant factor (0.25, 0.5, 1., 10.) with an even–odd–even ordering for the TSO. The dashed lines represent the analytic solution. Figures 2a and 2b show curves at Courant factors of 0.25 and 10. each representing the temperature difference between odd–even–odd ordering and even–odd–even ordering for the TSO. It should be noted that we have run cases with the symmetric ordering. These will not be shown for the sake of brevity. We do mention, however, that the symmetric ordering shows behavior similar to odd–even–odd and even–odd–even. In addition, differences between all three orderings show up as a "phase" difference that grows with increasing Courant factor (see Figs. 2a and 2b). Figure 3 shows the result of subcycling at a Courant factor of 0.5 when the time step was chosen such that the running Courant factor was 10.

It is apparent that as α grows the accuracy decreases. Although the solutions are always stable, the inaccuracies show up as a "staircase" behavior and as a time delay in the propagation of the diffusion "front." Interestingly enough, this behavior is similar to the ripple that appeared at large time steps in the temperature profiles of Livne and Glasner [3]. As $\alpha \rightarrow 0$, the numerical solutions converge to the exact solution regardless of

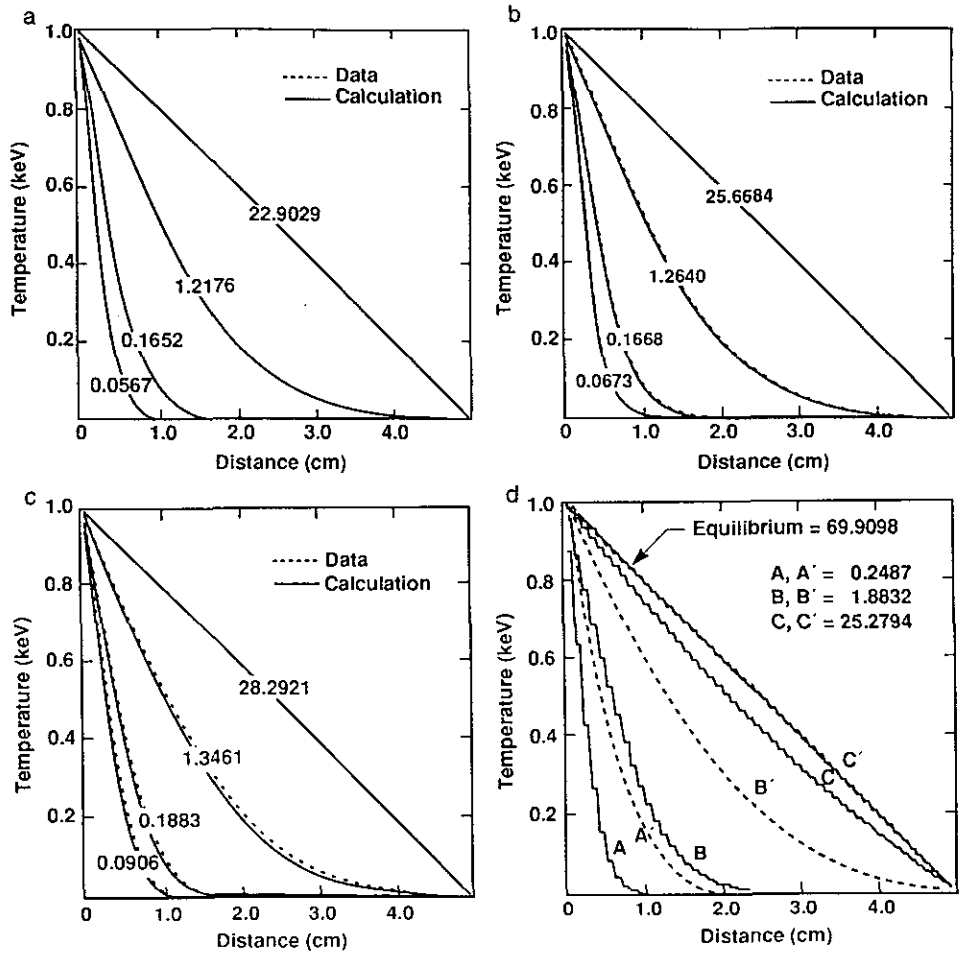

FIG. 1. (a) Snapshots of a temperature profile at a Courant factor of 0.25 evolving via the one-dimensional linear diffusion equation. The initial conditions are a cold slab of length 5 cm heated on one side with a constant temperature source of 1 kev. Solid lines represent the numerical solution using an even–odd–even ordering, while dashed lines represent the analytic solution. (b) Same as (a), with Courant factor 0.5. (c) Same as (a), with Courant factor 1.0. (d) Same as (a), with Courant factor 10.

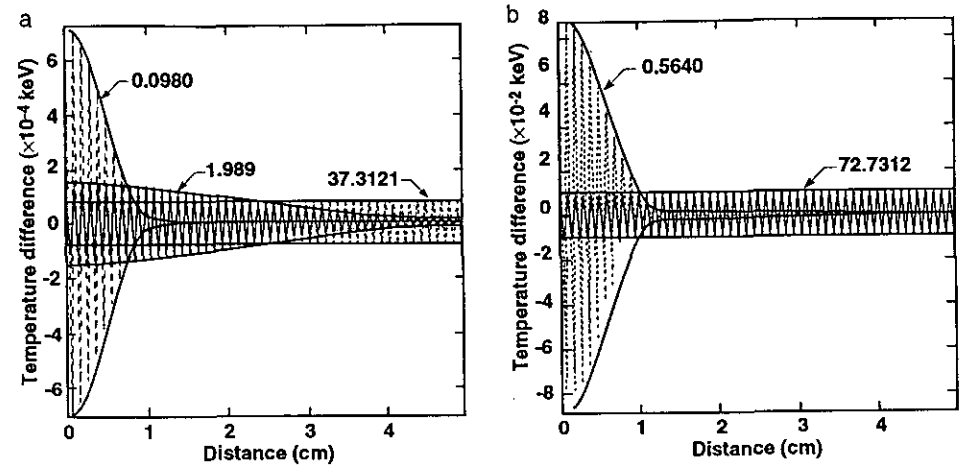

FIG. 2. (a) Snapshots of the temperature difference between temperature profiles calculated with odd–even–odd versus even–odd–even orderings. Courant factor is 0.5. The initial conditions are a cold slab of length 5 cm heated on one side with a constant temperature source of 1 kev. (b) Same as (a), with Courant factor 10.

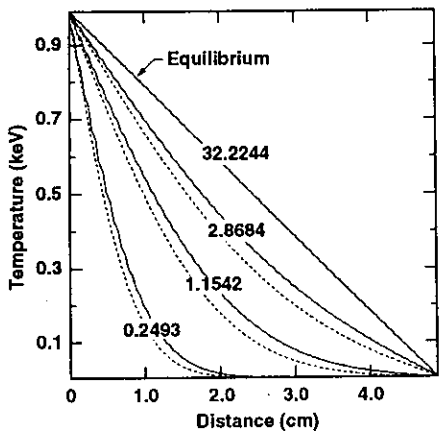

FIG. 3. Snapshots of a temperature profile at a Courant factor of 10.0 evolving via the one-dimensional linear diffusion equation. Subcycling is performed at a Courant factor of 0.5, using the Lie–Trotter product formula discussed in the text. The initial conditions are a cold slab of length 5 cm heated on one side with a constant temperature source of 1 kev. Solid lines represent the numerical solution, using an even–odd–even ordering, while dashed lines represent the analytic solution.

the TSO ordering. However, for $\alpha \gg 1$, the three orderings we consider have transient solutions which diverge from each other and the exact solution.The most striking feature of this difference is that while all three yield a staircase like profile, the "steps" are offset when we compare, for example, even–odd–even versus odd–even–odd. What was somewhat unexpected, was the fact that given enough time, regardless of the Courant factor, the time evolving solution will approximately converge to the steady state regardless of ordering. For example, note that in Figs. 2a and 2b that the peak temperature difference between the profiles decreases as steady state is reached. We originally discovered this fact using Mathematica. In fact we will show an example (without sources), where in fact the numerical and analytical solutions converge exactly to the correct steady state regardless of the Courant factor. This will be discussed in the next section.

c. *Mathematica and the Product Formula Algorithm*

In this section, we wish to show the usefulness of Mathematica [8] (or possibly other symbolic software systems [11]) as tools for understanding the mathematical features of the product formula algorithm. In addition, we have found Mathematica to be a useful debugging tool and indispensable when it comes to constructing exponentials of the decomposed matrices. As an example, we consider linear diffusion with reflecting boundary conditions and six zones (four real and two phony). What we present is an example of a Mathematica session with our comments in italic. The prompts will not be shown. All Mathematica-supplied functions begin with a capital letter. All functions, both user- and Mathematica-supplied require that the input be enclosed within square brackets. The even and odd matrices are entered as a list of lists. That is, $\{\{a, b, c\}, \{d, e, f\}, \{g, h, i\}\}$ is equivalent to

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}. \tag{2.26}$$

We first define the even and odd matrices for a problem where different zones have different diffusion coefficients. This is done to illustrate the usefulness of Mathematica for constructing the TSO. We will specialize to a constant diffusion later on. This is done to make the presentation more compact.

```
even = {{0,-D2,D2,0,0,0}, {0,-D2,D2,0,0,0},
        {0,-D2,-D2,0,0,0},{0,0,0,-D4,D4,0},
        {0,0,0,D4,-D4,0},{0,0,0,D4,-D4,0}}
odd = {{0,0,0,0,0,0},{0,0,0,0,0,0},{0,0,D3,D3,0,0},
       {0,0,D3,D3,0,0},{0,0,0,0,0,0},{0,0,0,0,0,0}}
```

The diffusion coefficients refer to zone number. For example, D2 *is the coefficient for zone two. We now perform the exponentiation of the even and odd matrices and put the result in matrix format.*

```
MatrixForm[MatrixExp[even]]
```

Mathematica yields

$$\begin{bmatrix} 1 & -\Delta_2 2 & \Delta_2 2 & 0 & 0 & 0 \\ 0 & \Delta_1 2 & \Delta_2 2 & 0 & 0 & 0 \\ 0 & \Delta_2 2 & \Delta_1 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta_1 4 & \Delta_2 4 & 0 \\ & & & \Delta_2 4 & \Delta_1 4 & 0 \\ 0 & 0 & 0 & \Delta_2 4 & -\Delta_2 4 & 1 \end{bmatrix}$$

where for simplicity, we have used the notation $\Delta_{1(2)}2 = (1 \pm \exp(-2D2))/2$ *for zone two and similarly for other zones. Mathematica of course will write a matrix whose elements consist of the full expression. For the odd matrix we have*

```
MatrixForm[MatrixExp[odd]]
```

Mathematica yields

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta_1 3 & \Delta_2 3 & 0 & 0 \\ 0 & 0 & \Delta_2 3 & \Delta_1 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The usefulness of Mathematica should be apparent. For the types of matrices which we encounter in this paper which are not quite block-diagonal, Mathematica easily computes the exponentials of the even and odd matrices. The result is then used in our code. We now specialize to the case where all of the diffusion coefficients are one. The problem we choose is the diffusion of a heat source interior to a cold material (boundary conditions are taken to be reflective). The initial temperature profile consists of all real zones zero except real zones three and four. We will follow the initial temperature profile cycle by cycle for various values of the Courant factor. We first define the function "matxe" which is the exponential of the even matrix times a variable Courant factor divided by a factor n

matxe[n_,cour_] := MatrixExp[(cour/n)*even]]

where "even" is the previously defined matrix with all diffusion coefficients equal to one. Likewise, for the odd matrix, we define

matxo[n_,cour_] := MatrixExp[(cour/n)*odd]

The second-order time step operator ("propso") and the exact time step operator ("propx") can now be constructed. Matrix matrix multiplies are denoted by ".".

propso[cour_] := matxe[2,cour].matxo[1,cour].matxe[2,cour]

full = even + odd

propx[cour_] := MatrixExp[cour*full]

The initial temperature profile is given by (remember, the first and last zones are the phony zones whose temperatures are equal to the first and last real zones)

temp = {0,0,0,1,1,0,0,0}

The new second-order temperature profile is given by "propso" acting on a column matrix (call it foo). Similarly for the exact temperature profile.

tempnewso[cour_,foo_] := propso[cour].foo

tempnewx[cour_,foo_] := propx[cour].foo

In the following we will use the Mathematica convention of "%" to denote the previous expression. For a Courant factor of 0.5, *we have*

tempnew[.5,temp]

Mathematica yields

{.0672564,.0672564,.258957,.673787,.673787,.258957, .0672564,.0672564}

Further time advancing the profile yields

tempnewx[.5,%]

Mathematica yields

{.157691,.157691,.316738,.525571,.525571,.316738,.157691, .157691}

After 29 *cycles the temperature profile is*

{.333333,.333333,.333333,.333333,.333333,.333333, .333333,.333333}

which is the equilibrium solution. One can even verify this by operating with "tempnewx" on this profile and proving that it does not change. The above profile is the fixed point for the exact TSO.

The second-order scheme at a Courant factor of 0.5 *yields*

tempnewso[.5,temp]

Mathematica yields

{.06218,.06218,.25388,.68394,.68394,.25388,.06218,.06218}

which is not too bad when one considers that we are operating at the stability limit. After one more cycle,

tempnewso[.5,%]

Mathematica yields

{.151855,.151855,.31205,.536095,.536095,.31205,.151855, .151855}

After 31 *cycles, the equilibrium solution is reached. That is,*

{.333333,.333333,.333333,.333333,.333333,.333333, .333333,.333333}

Again, repeated applications of "tempnewso" prove that this is the steady state. Note that the second-order solution converges to the correct equilibrium solution (albeit in slightly more cycles). This is suggestive of the fact that the product formula algorithm (applied to systems without external sources) used to construct various time step operators possess the same fixed point as the exact time step operator. We have run this problem at Courant factors ranging from 0.1 *to* 100 *for first- and second-order time step operators. All results point to the fact that the product formula solutions eventually converge or at worst approximately converge (in the case of an external source) to the correct steady state solution even though the transient profiles may not be accurate. As an example, consider the steady state behavior at Courant factors of* 1, 5 *and* 100. *We find that the second-order solution at a Courant factor of* 1 *converges to the steady state solution in* 17 *cycles while the exact solution converges in* 15 *cycles. At a Courant factor of* 5 *we obtain convergence in* 11 *cycles for the second-order solution and three cycles for the exact solution. Finally, at a Courant factor of* 100, *the number of cycles required for convergence to steady state is* 12 *for the second-order solution while it is just* 1 *for the exact solution. At the Courant factor of* 5, *the number of cycles required to reach steady state has*

pretty much saturated. The explanation is simple; the exponential factors in the TSO rapidly approach zero for $\alpha \gg 1$, *and the TSO becomes a constant matrix independent of* α.

We now illustrate the above observations by computing the error defined by

$$\text{error} = \sqrt{\sum_{i=\text{zone}} (T_{\text{exact}}(i) - T_{\text{so}}(i))^2} \qquad (2.27)$$

using Mathematica. We first define functions "fx" and "fso" with arguments given by the Courant factor, cycle number, and initial temperature profile. The output from these functions will be the temperature profile at a given Courant factor and after a given number of cycles. Therefore, we have in Mathematica syntax

```
fx[cour_,cyc_,foo_] := MatrixPower[propx[cour],cyc].foo
fso[cour_,cyc_,foo_] := MatrixPower[propso[cour],cyc].foo
```

The MatrixPower function is a Mathematica supplied function that takes a given matrix (first argument) and raises it to the power cyc (second argument). The error function can now be defined using "fx" and "fso" thusly,

```
error[cour_,cyc_,foo_] := Sqrt[Apply[Plus,(fx[cour,cyc,foo]-
fso[cour,cyc,foo])^2]]
```

Most of the syntax used above is self explanatory. The "Apply" function applies the "Plus" operation to the second argument consisting of a list of squares of differences between the exact and the second-order solutions. Given a Courant factor and an initial profile ("foo"), Mathematica will plot "error [cour_,cyc_,foo_]" as a function of cycle number. The results are shown in Figs. 4a–d. *Note that all the cases run show the error going to zero as the cycle number increases regardless of the Courant factor (although not shown, the Courant factor of* 100 *was run and those results were essentially identical to the Courant factor equal to five). It is interesting to note that for small Courant factors, the error peaks beyond cycle one. On the other hand, the error is largest at cycle one for those cases run with a large Courant factor. The fact that the large Courant factor results tend to converge faster to the equilibrium solution is only an artifact of the error being plotted versus the cycle number (a large Courant factor implies large time step). Our results show that for Courant factors less than or on the order of one, the error is never worse than* 10% *throughout the evolution of the problem. Large Courant factors, on the other hand, show large errors* (~30%) *early. But these rapidly die out. The number of cycles required to converge is typically on the order of the square of the number of zones of the problem, divided by the Courant factor.*

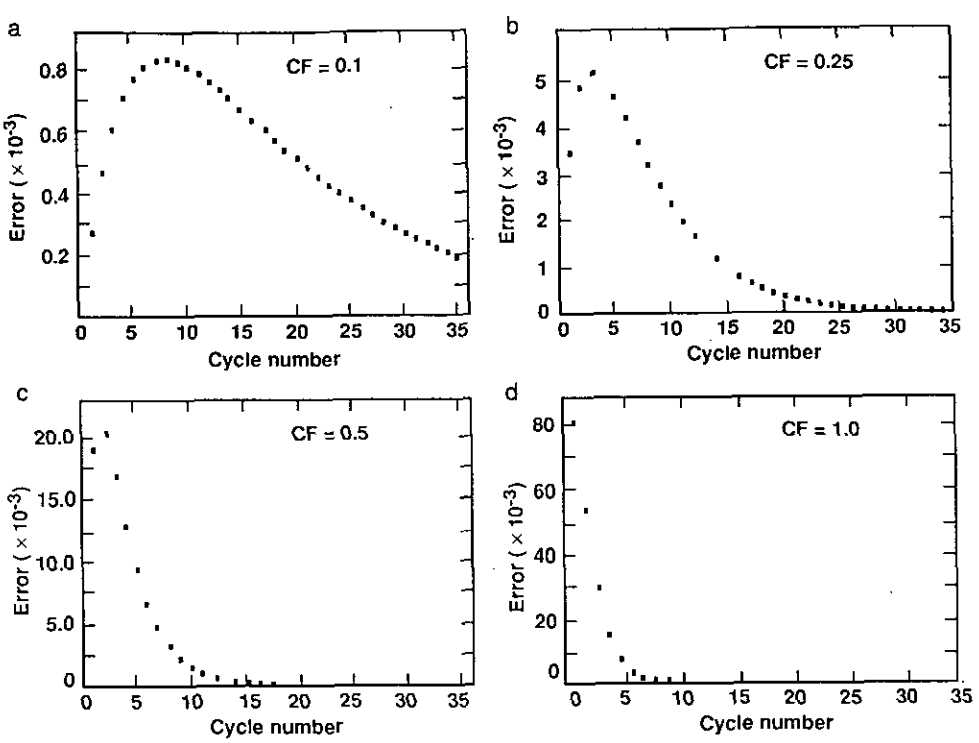

FIG. 4. (a) Error function (see Eq. (2.27)) calculated using Mathematica for a linear one-dimensional diffusion problem consisting of reflecting boundaries and four real zones. The two most interior zones were given a temperature of 1 kev. The Courant factor is 0.1. (b) Same as (a), with Courant factor 0.25. (c) Same as (a), with Courant factor 0.5. (d) Same as (a), with Courant factor 1.0.

III. THE RADIATION DIFFUSION PROBLEM

a. *Theory*

In this section, we will develop the theoretical framework for applying the PF algorithm to a particular non-linear diffusion problem with open and reflecting boundary conditions. We consider the diffusion of radiation energy density [12] in the limit where matter energy density is negligible. If T_R is the radiation temperature and $\lambda_R(\rho, T_R)$ is the Rosseland mean free path [12] then the radiation energy density given by aT_R^4 evolves according to

$$\frac{\partial}{\partial t} aT_R^4 = \nabla \cdot \left(\frac{ac}{3} \lambda_R(\rho, T_R) \nabla T_R^4 \right). \tag{3.1}$$

We choose a simple analytic form for the Rosseland mean free path [12] given by

$$\lambda_R(\rho, T) = \lambda_0 \left(\frac{\rho_0}{\rho} \right)^{\nu} \left(\frac{T}{T_0} \right)^{\gamma} \tag{3.2}$$

We will use the second method presented in Section II for solving Eq. (3.1).

Most of the results of Section II can be simply extended to include the non-linear case by making the proviso that each entry of the decomposed diffusion matrix (Eq. (2.17) and Eq. (2.18)) have a zonal index. However, what has not been appreciated by previous authors is the fact that Eq. (2.4) is not the solution to the matrix form of the diffusion equation if the diffusion matrix is time dependent (a feature of the non-linear problem). Because we operator split each direction, it is enough to consider the one-dimensional version of Eq. (3.1). As done previously, we discretize the equation in space. In addition, because the scheme is explicit, the Rosseland mean is evaluated at the old time step and, hence, is a known time dependent function. Writing the energy density as $\varepsilon = aT^4$, Eq. (3.1) can be written in the matrix form

$$\frac{d}{dt} \hat{\varepsilon} = \frac{\hat{\Lambda}(t)}{\Delta x^2} \hat{\varepsilon}, \tag{3.3}$$

where $\hat{\varepsilon}$ is a column matrix analogous to Eq. (2.13) and $\hat{\Lambda}(t)$ is a time dependent matrix given by

$$\begin{bmatrix}
\Gamma_L\Theta_L & (-1+\Theta_L)D_2 & (1-\Theta_L)D_2 & 0 & \dots & \dots & \dots & \dots & 0 \\
\Theta_L D_2 & -(1+\Theta_L)D_2 & D_2 & 0 & 0 & \dots & \dots & \dots & 0 \\
0 & D_2 & -T_{23} & D_3 & 0 & 0 & \dots & \dots & 0 \\
0 & 0 & D_3 & -T_{34} & D_4 & 0 & 0 & \dots & 0 \\
\dots & 0 & 0 & D_4 & -T_{45} & D_5 & 0 & \dots & 0 \\
\dots & \dots & 0 & 0 & D_5 & -T_{56} & D_6 & \dots & 0 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & \dots & 0 & D_N & -(1+\Theta_R)D_N & \Theta_R D_N \\
0 & 0 & 0 & \dots & \dots & \dots & (1-\Theta_R)D_N & (-1+\Theta_R)D_N & \Gamma_R\Theta_R
\end{bmatrix}. \tag{3.4}$$

D_j is the diffusion coefficient evaluated at the interface between zones j and $j+1$ and $T_{ij} = D_i + D_j$. As an aside, the problem of how to define an interface diffusion coefficient in terms of the corresponding zonal quantity given by Eq. (3.2) has been around for a long time in radiation diffusion problems [13]. We choose a fairly simple algorithm that seems to work quite well for a variety of test problems we have run, including the classic Marshak wave [12] [for which an analytic solution exists). Define the zonal diffusion coefficient by

$$DZ_j = \frac{ac}{3} \left(\frac{\rho_0}{\rho_j} \right)^{\nu} \left(\frac{\varepsilon_j}{\varepsilon_0} \right)^{\gamma/4}. \tag{3.5}$$

The diffusion coefficient at the interlace between j and $j+1$ is defined by

$$\frac{1}{D_j} = \left(\frac{1}{DZ_j} + \frac{1}{DZ_{j+1}} \right) \Bigg|_{\mathrm{MAX}(\varepsilon_j \varepsilon_{j+1})}. \tag{3.6}$$

The fact that the average is evaluated at the maximum zonal temperature prevents cold zones from giving the major contribution to the above sum and thereby inhibiting radiation flow.

In general the solution of Eq. (3.3) is not as simple as Eq. (2.2). For arbitrary time step, the time advanced solution of Eq. (3.3) is given by

$$\hat{\varepsilon}(t + \Delta t) = \hat{M}(t) \hat{\varepsilon}(t), \tag{3.7}$$

where $\hat{M}(t)$ is the so-called matrizant [14] of the differential equation and is given by

$$\hat{M}(t) = I + \frac{\int_t^{(t+\Delta t)} \hat{\Lambda}(\tau)\, t\tau}{\Delta x^2} + \frac{\int_t^{(t+\Delta t)} \hat{\Lambda}(\tau) \int_\tau^{(\tau+\Delta t)} \hat{\Lambda}(s)\, ds\, d\tau}{(\Delta x^2)^2} + \cdots. \tag{3.8}$$

This is the time step operator for an arbitrary Δt. However, in the limit of small Δt, the integral can be approximately evaluated and the form used in RFL is obtained. That is,

$$\hat{M}(t) \approx \left(I + \frac{\hat{\Lambda}(t)}{\Delta x^2}\Delta t + \frac{1}{2}\left(\frac{\hat{\Lambda}(t)}{\Delta x^2}\right)^2 (\Delta t)^2 + \cdots\right) = \exp\left(\frac{\hat{\Lambda}(t)}{\Delta x^2}\Delta t\right). \tag{3.9}$$

Therefore, when dealing with non-linear problems or linear problems with time dependent coefficients within the PF formalism, we see that the accuracy of the solution is restricted both by the order of the product formula used and the approximation of the matrizant given by Eq. (3.8). Which is the more restrictive? We do not yet know.

Assuming that the time step operator has the simplified form given by Eq. (3.9), we can now decompose $\hat{\Lambda}(t)$ into approximately block diagonal pieces. Each part can be exponentiated just like the linear problem and an approximate time step operator given by Eq. (2.21) is then constructed. It should be stressed that unlike the linear problem, where the second-order time step operator needs only be calculated once and then applied repeatedly to advance the initial profile, the non-linear problem requires that the second-order time step operator be updated every cycle. For example, given the initial ε profile, the diffusion coefficients are evaluated. The second-order time step operator is evaluated using this information and then applied to the profile to obtain a time advanced solution. A new cycle is begun by using this updated information to recalculate the diffusion coefficients and, hence, the time step operator, using the new profile. The new time step operator is again applied and the process is repeated until equilibrium or a preset time has been reached.

b. *Numerical*

We present the results of two radiation diffusion problems. The first is a finite slab of material at a constant density $\rho = \rho_0$ and initially zero temperature with the top and bottom boundaries reflecting and the right and left boundaries open with the left boundary held at a fixed temperature $T = T_0$ and the right boundary held at $T = 0$. This is effectively a one-dimensional situation which we will call the Marshak problem and the resulting flow we call a Marshak wave. Technically, this is a generalization of what some authors call a Marshak wave (namely a semi-infinite one-dimensional slab with a temperature pulse applied to the boundary). The other problem of interest is a two-dimensional problem consisting of a finite slab of material with initially zero temperature but with a dense region ($\rho_{\text{interior}} > \rho_0$ interior to the slab. Three of the four boundaries are held at zero temperature while the fourth boundary is held at a fixed temperature T_0.

For problem one, an approximate solution has been obtained using a variety of methods [12]. We will merely quote the results (the details can be found in the literature). Given a partial differential equation defined on the interval $[0, \infty]$ of the form

$$\frac{\partial}{\partial t}\Phi^m(x,t) = \kappa\frac{\partial^2}{\partial x^2}\Phi^{n+4}(x,t) \tag{3.10}$$

with $\Phi(x,0) = 0$, $\Phi(0,t) = \Phi_0$, and $\Phi(\infty,t) = 0$ an approximate solution (valid to several percentages) is given by

$$\Phi(x,t) = \Phi_0\left(1 - \frac{x}{\xi(t)}\right)^p, \tag{3.11}$$

where

$$\xi^2(t) = 2\kappa\Phi_0^{n+4-m}\left(\frac{n+4}{n+4-m}\right)\left(\frac{m}{n+4-m}\right)t, \qquad p = \frac{1}{n+4-m}. \tag{3.12}$$

Since our slab in finite, the above solution is valid inside the slab (that is, valid for times such that the radiation front has not reached the end of the slab). The steady state solution of Eq. (3.10) in the interval $[0, L]$ is simply given by

$$\Phi(x) = \Phi_0\left(1 - \frac{x}{L}\right)^{1/n}. \tag{3.13}$$

Figures 5a,b show the time evolution of the Marshak wave for the Courant factors of 0.5 and 10. Since the diffusion coefficient varies spatially, these numbers refer to a maximum value. Subcycling is performed by scanning the mesh for the largest Courant factor and dividing by the value at which we wish to subcycle. This number is the value of s (number of subcycles) used in Eq. (2.22). The number of zones chosen was 100 in the horizontal direction and three in the vertical direction. The right –left boundaries were declared open with a fixed temperature of 1 kev applied on the left boundary and 0 kev applied at the right boundary. The top and bottom boundaries were declared reflecting. The initial temperature in the slab was chosen to be 0 kev. All results were run on a Cray YMP. For simplicity, dimensionless variables were used. Assuming a constant density of 1 g/cc and parameters appropriate for aluminum ($\nu = 1.9$ and $\gamma = 5.6$), we can define a dimensionless time and length. That is $\tau = (ct)/\lambda_0$ and $\xi = x/\lambda_0$, where $\lambda_0 = 5$ cm. All data in Figs. 5a,b and 6a,d use these definitions.

Again, note the good agreement between theory and experiment for Courant factors less than one. For large Courant factors, the staircase behavior seen in the linear case makes its appearance and gives rise to an inaccurate time evolving solution. We note, however, that the large Courant factor solution converges to the steady state answer. Figure 5c shows the case

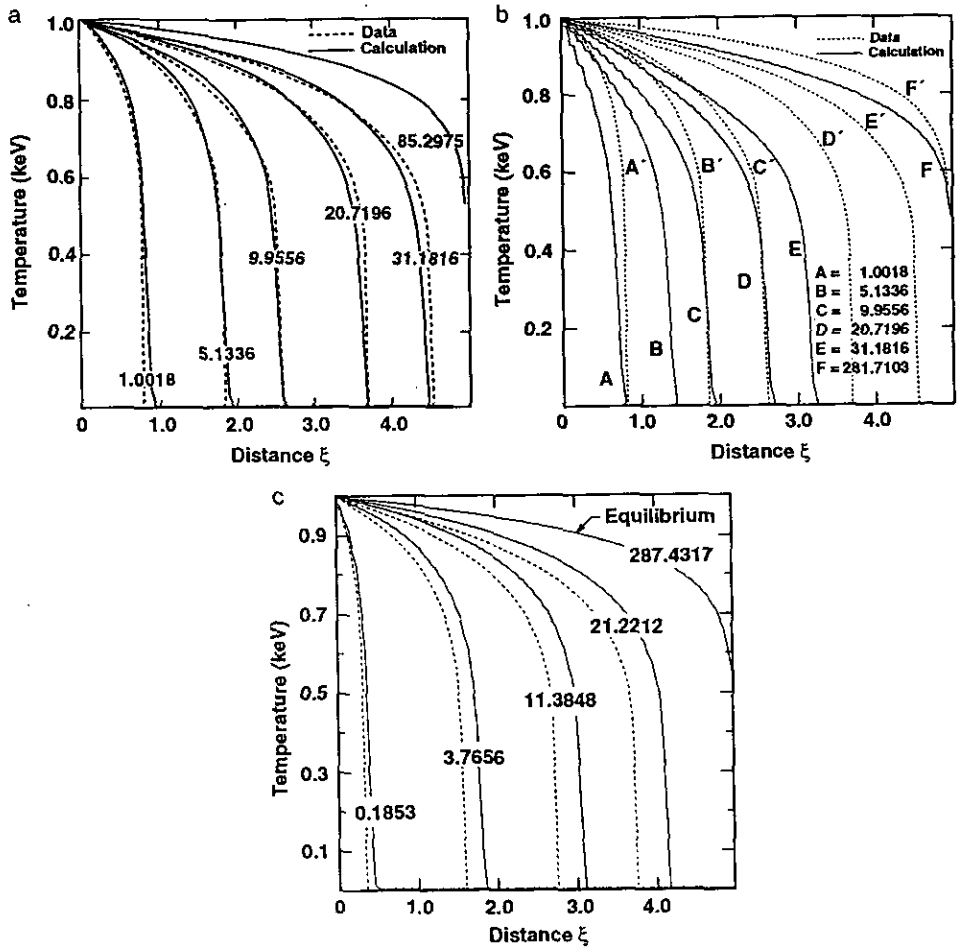

FIG. 5. (a) The Marshak wave problem. Snap shots of a temperature profile at a Courant factor of 0.5 evolving via the one-dimensional radiation diffusion equation. The initial conditions are a cold slab of length $\xi = 5$ heated on one side with a constant temperature source of 1 kev. Opacity information is given in the text. Solid lines represent the numerical solution, using an even–odd–even ordering, while dashed lines represent the analytic solution. (b) Same as (a), with Courant factor 10.0. (c) The Marshak wave problem. Snapshots of a temperature profile at a Courant factor of 10.0 evolving via the one-dimensional radiation diffusion equation. Subcycling is performed at a Courant factor of 0.5 using the Lie–Trotter product formula discussed in the text. The initial conditions are a cold slab of length $\xi = 5$ heated on one side with a constant temperature source of 1 kev. Solid lines represent the numerical solution using an even–odd–even ordering while dashed lines represent the analytic solution.

run with a Courant factor of 10, but with subcycling performed at a Courant value of 0.5. Note the improved accuracy. We do not show the results for different ordering schemes but merely state that the conclusions drawn from the linear diffusion problem hold in the non-linear case as well.

For problem two (a two-dimensional radiation diffusion problem), we chose a slab given by 100 by 100 zones. Figures 6a–d show the time evolution of the radiation front run at a Courant factor of 0.5. The initial conditions consisted of a constant applied temperature source of 1.0 kev at the left boundary. The bottom, right, and top boundaries were fixed at 0 kev. The density of the slab is constant in time. It consists of 1.0 g/cc everywhere except a section whose density is 100 g/cc interior to the slab. Refer to the drawing for the location of this dense section. The calculation took 156 s of CRAY YMP time. The same problem using a back substitution algorithm for solving the implicit equations took 436 s. The results of both algorithms were virtually identical. The time difference is due primarily to the fact that the back substitution scheme, due to its recursive nature, was not entirely vectorizable. Hence the factor of 3 speedup.

IV. CONCLUSIONS

With the advent of massively parallel computer architectures, explicit algorithms have started enjoying a resurgence of interest. Unlike their predecessors, the new generation of explicit algorithms are unconditionally stable. In light of these facts, we have extended the product formula algorithm of DeRaedt [1] and Richardson, Ferrell, and Long [4] to include reflecting and open boundary conditions with external sources in one and two dimensions. In addition, in order to improve accuracy, we

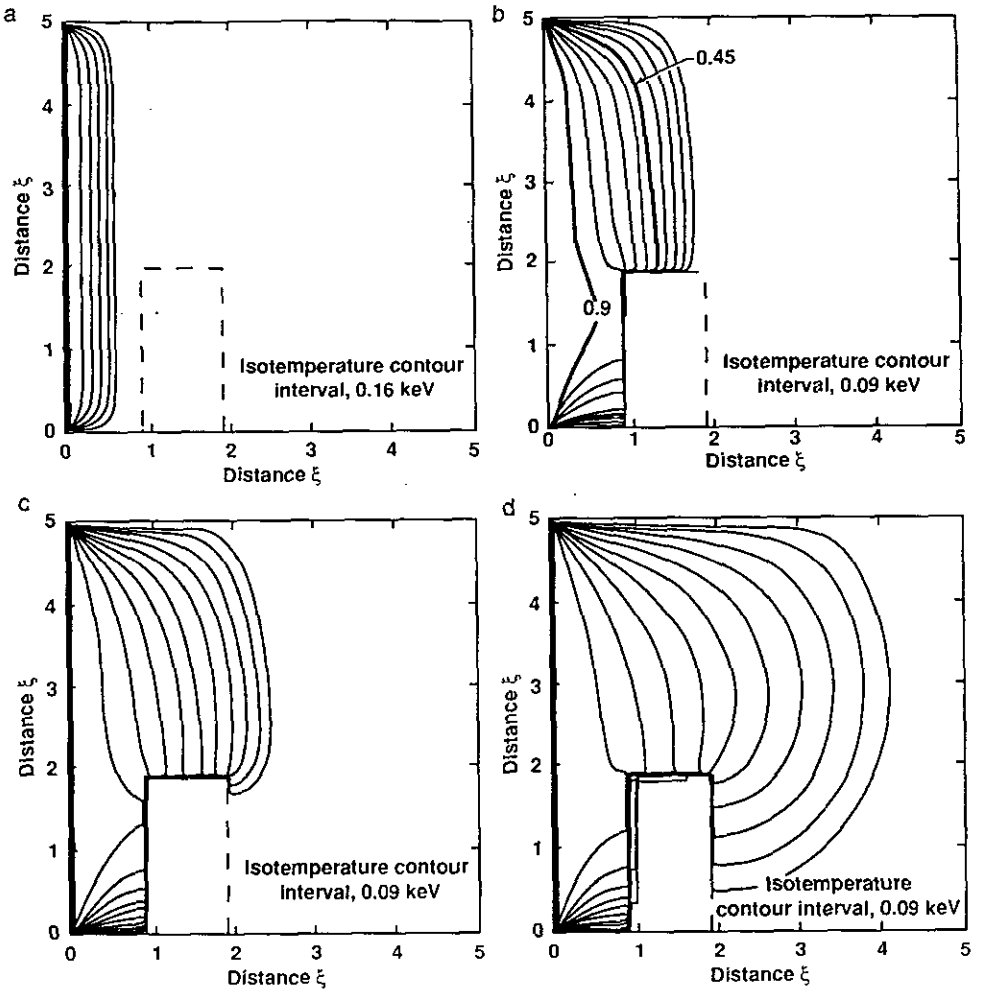

FIG. 6. (a) A two-dimensional radiation diffusion problem run at a Courant factor of 0.5 consisting of a slab of Al (opacity data is given in text) heated on the left boundary with a constant temperature of 1.0 kev. All remaining boundaries are fixed at 0 kev. The slab density is constant in time at 1.0 g/cc. There is a dense, and hence optically thick, region (denoted by the dashed line), where the density is 100.0 g/cc interior to the slab. All lengths and times are dimensionless. Scalings are given in text. Plot shows temperature contours in kilo electron volts. The time is 0.7. (b) Same as (a), with time 5.2. (c) Same as (a), with time 10.2. (d) Same as (a), with time 35.7.

have introduced a new type of sub-cycling based on the Lie–Trotter product formula. We have applied the extension of the PF scheme to various linear and radiation diffusion problems. In particular, we have considered one- and two-dimensional versions of a cold slab heated from one side by a constant temperature source. The answers are consistent with results computed via ADI and, where possible, analytic calculations. Computationally, we have found the algorithm easy to code and vectorize. We have as yet not implemented the scheme on a massively parallel computer.

Since the one-dimensional linear and non-linear solutions have analytic solutions, we have been able to address the ordering ambiguity problem and the accuracy as a function of Courant factor α. Our conclusions are that the accuracy is reasonable for Courant factors less than or on the order of one. However, for large Courant factors, the inaccuracy can become extreme, exhibiting itself as a staircase behavior in the profile. Interestingly enough, unlike traditional explicit schemes, the profile at large α does approach the steady state value in at worst an approximate fashion. The ordering ambiguity issue implies that a profile computed using one ordering scheme can be out of "phase" with a profile calculated using another scheme. The differences grow as the Courant factor grows. This difference might be minimized by using alternating orderings each cycle, thereby reducing any biasing that might appear due to one group of orderings versus another. The PF algorithm seems to be a promising scheme that is flexible enough to be applied to a variety of multidimensional non-linear problems with complicated boundary conditions.

Some interesting issues remain. One is the applicability of the algorithm to unstructured meshes. Another is a timing/accuracy comparison study on the massively parallel computers between an implicit solver such as ICCG [15] and the PF scheme. Finally, a useful application would be to a system of

equations involving multiple time scales. An example is radiation hydrodynamics [12] where fluid flow and radiation flow can have very different time scales.

ACKNOWLEDGMENTS

I thank many of my colleagues for useful discussions and help. These include Gary Carlson, Larry Carson, Chris Clouse, Frank McMahon, Charles McMillan, Ivan Otero, Todd Palmer, and Robert Tipton. I owe a special note of thanks to Joseph Bauer for useful discussions and for first introducing me to the product formula algorithm. In addition, Jim LeBlanc first introduced me to the concept of unconditionally stable explicit algorithms. His guidance and help were invaluable. The work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

REFERENCES

1. H. DeRaedt, *Comput. Phys. Rep.* **7,** 1 (1987).
2. J. R. Cary and I. Doxas, *J. Comput. Phys.* **107,** 98 (1993).
3. E. Livne and A. Glasner, *J. Comput. Phys.* **58,** 59 (1985).
4. J. L. Richardson, R. C. Ferrell, and L. N. Long, *J. Comput. Phys.* **104,** 69 (1993).
5. E. H. Twizell, *Computational Methods for Partial Differential Equations* (Ellis Horwood, Chichester, 1984).
6. R. M. Wilcox, *J. Math. Phys.* **8,** 962 (1967).
7. T. M. Apostol, *Calculus, Vol. II* (Xerox College, Waltham, MA, 1969).
8. S. Wolfram, *Mathematica* (Addison–Wesley, Redwood City, CA, 1991).
9. M. Reed and B. Simon, *Functional Analysis* (Academic Press, San Diego, 1980).
10. R. D. Richtmyer and K. W. Morton, *Difference Methods for Initial Value Problems* Wiley, New York, 1967).
11. See, for example, Maple or Macsyma.
12. D. Mihalas and B. Weibel-Mihalas, *Foundations of Radiation Hydrodynamics* (Oxford Univ. Press, New York, 1984).
13. R. Bowers and J. Wilson, *Numerical Modeling in Applied Physics and Astrophysics* (Jones & Bartlett, Boston, 1991).
14. A. S. Deif, *Advanced Matrix Theory for Scientists and Engineers* (Abacus, New York, 1982).
15. D. Kershaw, *J. Comput. Phys.* **26,** (1978).