INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 25, 455–473 (1997)

PARALLEL ALGORITHMS FOR SEMI-LAGRANGIAN ADVECTION

A. V. MALEVSKY[1] AND S. J. THOMAS[2]*

[1] *Centre de Recherche en Calcul Appliqué, 5160 Blvd. Décarie, Montréal, Québec H3X 2H9, Canada*
[2] *Recherche en Prévision Numérique, Environment Canada, 2121 Route Transcanadienne, Dorval, Québec H9P 1J3, Canada*

SUMMARY

Numerical time step limitations associated with the explicit treatment of advection-dominated problems in computational fluid dynamics are often relaxed by employing Eulerian–Lagrangian methods. These are also known as semi-Lagrangian methods in the atmospheric sciences. Such methods involve backward time integration of a characteristic equation to find the departure point of a fluid particle arriving at a Eulerian grid point. The value of the advected field at the departure point is obtained by interpolation. Both the trajectory integration and repeated interpolation influence accuracy. We compare the accuracy and performance of interpolation schemes based on piecewise cubic polynomials and cubic B-splines in the context of a distributed memory, parallel computing environment. The computational cost and interprocessor communication requirements for both methods are reported. Spline interpolation has better conservation properties but requires the solution of a global linear system, initially appearing to hinder a distributed memory implementation. The proposed parallel algorithm for multidimensional spline interpolation has almost the same communication overhead as local piecewise polynomial interpolation. We also compare various techniques for tracking trajectories given different values for the Courant number. Large Courant numbers require a high-order ODE solver involving multiple interpolations of the velocity field. © 1997 by John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids, **25**: 455–473 (1997).

No. of Figures: 2. No. of Tables: 10. No. of References: 35.

KEY WORDS: Eulerian–Lagrangian methods; domain decomposition; B-splines

1. INTRODUCTION

Numerical time step limitations associated with the explicit treatment of advection-dominated problems in computational fluid dynamics are often relaxed by employing variants of the method of characteristics. Characteristic-based schemes for advection–diffusion problems are known as Eulerian–Lagrangian methods (ELMs). Baptista[1] presents a broad overview of ELM-type methods with applications to advection–diffusion transport problems. These methods involve backward time integration of a characteristic equation in order to determine the location of the departure point of a fluid 'particle' in the Lagrangian frame of reference. After the departure point is found, the upstream value of the advected field is obtained by interpolation. This value is later used as an initial condition to integrate the diffusion equation in the Eulerian frame of reference. The diffusion equation is then solved by means of conventional finite element, finite difference or spectral methods. Oliveira and

* Correspondence to: S. J. Thomas, Recherche en Prévision Numérique, Environment Canada, 2121 Route Transcanadienne, Dorval, Québec H9P 1J3, Canada.

CCC 0271–2091/97/040455–19 $17.50
© 1997 by John Wiley & Sons, Ltd.
Received 12 April 1996
Revised 11 October 1996

Baptista[2] distinguish between two types of ELMs, namely interpolation- and quadrature-based methods. Interpolation-based methods simply assign values at the feet of characteristic curves to the nodes of a computational grid and use these nodal values to solve the diffusion equation in the Eulerian frame of reference. Quadrature-based methods are associated with finite element discretizations and track quadrature points backwards instead of tracking nodes.

Finite difference ELM's require repeated interpolation of the advected field and thus introduce numerical diffusion. Such schemes are in general not conservative. The ability to control numerical dissipation is critical to the implementation of interpolation-based ELMs and, for a given order of accuracy, methods based on splines exhibit the best conservation properties. Characteristic schemes of different flavours have been designed to model a wide spectrum of transport problems.[3] The so-called ELLAM methods designed for flow in porous media[4,5] are mass-conserving. Characteristic–Galerkin finite element methods for the Navier–Stokes equations[6–8] can also be shown to conserve mass when inner products are evaluated exactly. Related methods which employ B-spline basis functions are due to Bermejo[9] and the spline–characteristic method for the simulation of thermal convection.[10] In the atmospheric sciences these numerical methods are known as semi-Lagrangian schemes. Staniforth and Côté[11] present an overview of their use in atmospheric models.

The use of characteristics to integrate hyperbolic transport problems circumvents the Courant–Freidrich–Lewy (CFL) stability bound associated with Eulerian schemes. The CFL condition ensures that the domain of dependence of the analytical solution of a hyperbolic PDE is contained in the domain of dependence of the numerical solution.[12] Semi-Lagrangian methods shift the numerical domain of dependence, in the form of a finite difference stencil, to the grid cell containing the upstream departure point of the fluid particle trajectory in the Lagrangian reference frame.[13] The general form of the advection equation for a scalar field $f(\mathbf{x}, t)$ is

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial t} + (\mathbf{u} \cdot \nabla) f = 0, \qquad f(\mathbf{x}, t) = f(\mathbf{x}(t_0), t_0), \tag{1}$$

where the velocity vector is

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{u}(\mathbf{x}, t). \tag{2}$$

The value of f is constant along trajectories described by (2). For passive advection a one-step semi-Lagrangian method computes the material or substantial derivative along the trajectory originating at $(\mathbf{x}^*, t_{n-1})$ and terminating at $(\mathbf{x}, t_n)$,

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{f(\mathbf{x}, t_n) - f(\mathbf{x}^*, t_{n-1})}{\Delta t} = 0, \tag{3}$$

which is equivalent to $f(\mathbf{x}, t_n) - f(\mathbf{x}^*, t_{n-1}) = 0$. Numerical interpolation is employed to obtain $f(\mathbf{x}^*, t_{n-1})$ at the foot of the characteristic. In essence, such a method depends on the accurate backward integration of (2) to obtain the previous position of a fluid 'particle',

$$\mathbf{x}^* = \mathbf{x} + \int_{t_n}^{t_{n-1}} \mathbf{u}(x, t)\mathrm{d}t. \tag{4}$$

Trajectory integration is discussed in Reference 14. The velocity $\mathbf{u}(\mathbf{x}, t)$ is assumed to be known at time levels t_{n-1} and t_n and a variety of schemes developed for ordinary differential equations (ODEs) are available for the numerical integration of (2). For example, Malevsky[10] employs a second-order trapezoidal Runge–Kutta scheme, whereas Robert[15] proposes an iterative scheme based on the

 © 1997 by John Wiley & Sons, Ltd.

second-order-accurate midpoint Runge–Kutta method which iterates the displacement $\boldsymbol{\alpha} = \mathbf{x} - \mathbf{x}^*$ according to the rule

$$\boldsymbol{\alpha}^{[k+1]} = \Delta t \mathbf{u}(\mathbf{x} - \boldsymbol{\alpha}^{[k]}/2, t - \Delta t/2), \tag{5}$$

where the midpoint velocity is obtained by interpolation. Equation (2) is specified at every grid point, resulting in a set of independent ODEs which can be integrated in parallel.

Parallelization of a numerical model often requires modification of existing algorithms. Both the arithmetic operations and the interprocessor data exchange contribute to the overall cost of a distributed memory, parallel solver. Performance is influenced by data locality and overlapping computation/communication. These criteria may supersede the straightforward optimization of the number of floating point operations needed to obtain the solution. The two steps involved in semi-Lagrangian algorithms, i.e. integration of characteristic equations and interpolation at the feet of characteristic curves, determine the accuracy and computational cost of these schemes. In this study we analyse the accuracy and computational complexity of interpolation algorithms and trajectory integration schemes in a distributed memory, parallel environment.

2. INTERPOLATION ALGORITHMS

2.1. Lagrange Interpolation in 1D

When $u = u(x, t)$ is a constant and the Courant number $C < 1$, interpolation at grid points leads to well-known Eulerian finite difference schemes.[16] By adopting the standard notation $f_i^n = f(x_i, t_n)$, the numerical solution of (1) is written as

$$f_i^{n+1} = p(x_i - u\Delta t) = \sum_j l_j(x_i - u\Delta t) f_j^n, \tag{6}$$

where $\alpha = u\Delta t$ is the displacement, $p(x)$ represents the polynomial interpolating f at grid points and $l_j(x)$ is the Lagrange basis polynomial at a grid point j,

$$l_j(x) = \prod_{\substack{v \\ v \neq j}} \frac{x - x_v}{x_j - x_v}.$$

For example, if linear interpolation is employed and $C < 1$, it follows that

$$f_i^{n+1} = l_{i-1}(x_i - u\Delta t) f_{i-1}^n + l_i(x_i - u\Delta t) f_i^n. \tag{7}$$

If, on the other hand, $N < C \leqslant N + 1$, then $i \to i - N$ on the right-hand side of (7). On a uniform grid with mesh length $\Delta x = x_{i+1} - x_i$, expression (7) simplifies to

$$\begin{aligned} f_i^{n+1} &= \frac{x_i - u\Delta t - x_i}{x_{i-1} - x_i} f_{i-1}^n \frac{x_i - u\Delta t - x_{i-1}}{x_i - x_{i-1}} f_i^n \\ &= f_i^n - \frac{u\Delta t}{\Delta x}(f_i^n - f_{i-1}^n), \end{aligned} \tag{8}$$

which is a first-order *upwind* finite difference scheme.[17,18] Similarly, quadratic interpolation results in the well-known second-order Lax–Wendroff scheme

$$f_i^{n+1} = f_i^n - \frac{1}{2}\frac{u\Delta t}{\Delta x}(f_{i+1}^n - f_{i-1}^n) + \frac{1}{2}\left(\frac{u\Delta t}{\Delta x}\right)^2 (f_{i+1}^n - 2f_i^n + f_{i-1}^n). \tag{9}$$

In the more general case of time-varying $u(x, t)$, semi-Lagrangian schemes rely on the trajectory integration (4) to determine the upstream grid cell in which to interpolate.

© 1997 by John Wiley & Sons, Ltd.

2.2. Interpolation in higher dimensions

Consider semi-Lagrangian methods in 2D and 3D which are based on interpolation. The interpolating polynomial in 2D takes the form

$$p(x, y) = \sum_{n=0}^{N} \sum_{m=0}^{M} a_{nm} x^n y^m, \tag{10}$$

where $\{1, x, y, x^2, xy, y^2, \ldots, x^n y^m, \ldots\}$ is a set of basis polynomials. A unique polynomial can be constructed by choosing the coefficients a_{nm} such that they satisfy the $(M+1) \times (N+1)$ degrees of freedom $p(x_i, y_j) = f_{ij}$. Efficient algorithms for computing $p(x, y)$ rely on an appropriate set of basis polynomials (e.g. Lagrange, B-spline). A non-unique polynomial of lesser degree can be constructed by reducing the number of degrees of freedom. For example, the scalar field $f(x, y)$ can be expanded in terms of bilinear Lagrange basis polynomials $l_{ij}(x, y)$ at four points as

$$f(x, y) = \sum_{i=1}^{2} \sum_{j=1}^{2} l_{ij}(x, y) f(x_i, y_j). \tag{11}$$

Alternatively, the interpolating polynomial can be formed as the Cartesian product of 1D polynomials,

$$f(x, y) = \sum_{j=1}^{2} l_j(y) \sum_{i=1}^{2} l_i(x) f(x_i, y_j). \tag{12}$$

Advection schemes based on interpolation include higher-order cross-terms in the equivalent Taylor series expansion of f at the upstream point. Computational costs can be reduced by neglecting terms which do not affect the global truncation error. We illustrate this approach below by using the Newton form of the interpolating polynomial. The derivation of computationally efficient schemes from the Newton polynomial in the case of non-uniform grids is described in Reference 19.

2.3. Piecewise cubic Newton polynomial interpolation

Consider a regular Cartesian grid with uniform spacing. Given the upstream grid cell $[x_{i-1}, x_i]$ at the foot of the characteristic curve and the surrounding grid points (forming a stencil)

$$x_{i-2} < x_{i-1} < x < x_i < x_{i+1}, \tag{13}$$

then the piecewise cubic Newton polynomial interpolating f in this grid cell is

$$\begin{aligned} p(x) = {} & f_{i-1} + (x - x_{i-1}) f[x_{i-1}, x_i] + (x - x_{i-1})(x - x_i) f[x_{i-1}, x_i, x_{i+1}] \\ & + (x - x_{i-1})(x - x_i)(x - x_{i+1}) f[x_{i-2}, x_{i-1}, x_i, x_{i+1}]. \end{aligned} \tag{14}$$

Given $\Delta x = x_{i+1} - x_i$ and $\hat{x} = (x - x_{i-1})/\Delta x$, it is possible to simplify the above expression to

$$p(x) = f_{i-1}(1 - \hat{x}) + f_i \hat{x} + \beta_{i-1} f_{xx_{i-1}} + \beta_i f_{xx_i}, \tag{15}$$

where

$$f_{xx_i} = f_{i+1} - 2f_i + f_{i-1} \tag{16}$$

and the coefficients β_{i-1} and β_i are given by

$$\beta_{i-1} = -\tfrac{1}{6} \hat{x}(1 - \hat{x})(2 - \hat{x}), \qquad \beta_i = -\tfrac{1}{6} \hat{x}(1 - \hat{x})(1 + \hat{x}). \tag{17}$$

 © 1997 by John Wiley & Sons, Ltd.

Note that this scheme can be viewed as an extension of the Lax–Wendroff scheme (9) to four grid points. Indeed, the scheme contains second-order terms of the form $f_{i+1} - 2f_i + f_{i-1}$.

In two dimensions, interpolation in a 16×16 stencil of grid points requires the derivatives f_{xx}, f_{yy} and f_{xxyy}. In general it is desirable to construct lower-degree polynomials by dropping $\mathcal{O}(\Delta x^2 \Delta y^2)$ terms such as f_{xxyy}. This has the added benefit of significantly reducing the number of floating point operations (flops). Such a scheme is given below:

$$
\begin{aligned}
f_j^n &= (1-\hat{x}) f_{i-1,j}^n + \hat{x} f_{i,j}^n + \beta_{i-1,j} f_{xx_{i-1,j}}^n + \beta_{i,j} f_{xx_{i,j}}^n, \\
f_{j-1}^n &= (1-\hat{x}) f_{i-1,j-1}^n + \hat{x} f_{i,j-1}^n + \beta_{i-1,j-1} f_{xx_{i-1,j-1}}^n + \beta_{i,j-1} f_{xx_{i,j-1}}^n, \\
f_{yy_j}^n &= (1-\hat{x}) f_{yy_{i-1,j}}^n + \hat{x} f_{yy_{i,j}}^n, \\
f_{yy_{j-1}}^n &= (1-\hat{x}) f_{yy_{i-1,j-1}}^n + \hat{x} f_{yy_{i,j-1}}^n, \\
f_{ij}^{n+1} &= (1-\hat{y}) f_{j-1}^n + \hat{y} f_j^n + \beta_{j-1} f_{yy_{j-1}}^n + \beta_j f_{yy_j}^n.
\end{aligned}
$$

The computational complexity of the above scheme is simple to determine. Derivatives require $2 \times 3 = 6$ flops per grid point to compute. The local co-ordinates $(\hat{x}, \hat{y})$ take $2 \times 2 = 4$ flops to compute. The coefficient β_i require $2 \times (3 + 3 \times 2) = 14$ flops. Construction of the polynomial then requires $2 \times 7 + 2 \times 3 + 7 = 27$ flops. If several fields are to be interpolated, then some of these costs are shared. For example, the coefficients β_i need only be computed once for all fields. Thus the cost of the 2D scheme for interpolating n fields is $33n + 18$ flops. The operation counts in one, two and three dimensions for this polynomial interpolation scheme are summarized in Table I.

In a distributed memory, parallel environment, arithmetic operations and interprocessor data exchange both contribute to the total cost of a numerical scheme. Interprocessor communication is typically much slower than local arithmetic, where bandwidth and latency determine performance. The total amount of data to communicate and the number of communication steps are the two major concerns. The above scheme requires a local data exchange between adjacent subdomains residing on separate processors. Values at grid points adjacent to a subdomain boundary must be exchanged for each field which is to be interpolated. To minimize latency, the data for all fields should be combined in a single message.

The Courant number $C = |u|\Delta t/\Delta x$ determines the amount of data in the exchange. In particular, the Courant number indicates how many grid lengths a fluid particle may travel during one time step. Eulerian finite difference schemes are subject to the stability bound $C \leqslant 1$ and thus the amount of data 'overlap' exchanged between processors is determined by the finite difference stencil alone. For example, the first-order upwind finite difference scheme (8) would require a one-grid-point overlap to satisfy data dependences. A higher-order scheme such as (15) is based on a four-point stencil and would require a two-grid-point overlap at interprocessor boundaries. Semi-Lagrangian schemes remain stable for $C > 1$, implying that the length of particle trajectories may be greater than one grid length. Consequently, in a direct parallel implementation of a sequential semi-Lagrangian scheme the

Table I. Computational cost of piecewise polynomial interpolation scheme: flops per grid point; n fields to interpolate. Coords: local co-ordinates $\hat{x}$. Coeffs: coefficients β_i. Derivs: derivatives f_{xx}. Interp. cost of interpolation. Total: total flop count

	Coords	Coeffs	Derivs	Interp	Total
1D	2	7	$3n$	$7n$	$10n+9$
2D	4	14	$6n$	$27n$	$33n+18$
3D	6	21	$9n$	$79n$	$88n+27$

© 1997 by John Wiley & Sons, Ltd.

overlap is determined by C. For example, a semi-Lagrangian scheme based on (15) would require a fixed overlap region $\lceil C + 1 \rceil$ grid points wide around the local grid partition. In two dimensions each processor must send and receive four messages (two in the x-direction and two in the y-direction) after every time step in order to update these overlap regions.

2.4. Domain decomposition spline interpolation in 1D

Schemes based on piecewise polynomial interpolation are subject to numerical dissipation and are in general not conservative. Splines enforce continuity of derivatives across the entire grid and are far less dissipative. Indeed, it is known that cubic splines exhibit better conservation properties with respect to the L_2 and C^∞ norms than piecewise polynomials of equivalent order and Bermejo[9] demonstrates that cubic B-splines conserve mass. Thus semi-Lagrangian schemes based on splines can better preserve many important physical properties in fluid dynamics simulations. The use of splines implies global data dependences and would initially appear to limit the performance of a distributed memory implementation. In the present paper we propose a spline interpolation algorithm having practically the same communication cost as a purely local, piecewise polynomial scheme.

Given a 1D set of grid points $x_1, x_2, \ldots, x_n$, a cubic spline $S(x)$ on the interval (x_1, x_n) is a piecewise cubic polynomial that is continuous on (x_1, x_n) and has continuous first- and second-order derivatives. Each section of a cubic spline is a cubic polynomial and is described by four coefficients. The value of an interpolating spline $S(x)$ at the points $x_1, x_2, \ldots x_n$ is assumed to be known. These conditions together with the global continuity requirement result in a linear system to be solved for the coefficients of the basis polynomials which comprise the spline. Thus an interpolating spline is simply a linear combination of these basis splines. B-splines have compact support and form a basis in the space of splines[20]

$$S(x) = \sum_{i=1}^{n} c_i B_i(x). \tag{18}$$

It is possible to modify the cubic B-splines adjacent to the boundaries of an interval to satisfy different boundary conditions[10] or an additional B-spline can be added at either end of the interval. Cubic B-splines have compact support over four grid intervals, implying that four basis splines B_{i-1}, B_i, B_{i+1} and B_{i+2} overlap a grid interval (x_i, x_{i+1}). Thus the value of the interpolating spline in this interval is computed from a linear combination of these basis splines. A B-spline is a cubic polynomial within each grid interval and the spline interpolant is simply a sum of four basis splines multiplied by the associated B-spline representation coefficients or spline transform $\mathbf{c}$. Consequently, the construction of an interpolating spline requires two steps. First the B-spline representation coefficients c_i are computed. Next the value of the four basis splines must be evaluated at grid points, multiplied by the corresponding c_i and then added together.

The condition that $S(x)$ is equal to prescribed function values at grid points $j = 1, \ldots, n$ determines the B-spline representation coefficients c_i:

$$\sum_{i=1}^{n} c_i B_i(x_j) = S(x_j). \tag{19}$$

Since each B-spline has a non-zero value only at three grid points, equation (19) results in a tridiagonal linear system

$$\mathbf{A}\mathbf{c} = \mathbf{b}. \tag{20}$$

 © 1997 by John Wiley & Sons, Ltd.

The matrix **A** is later referred to as the spline transform matrix since it projects a vector from physical space onto spline representation space. For example, the spline transform matrix for an equispaced grid with periodic boundary conditions is

$$\mathbf{A} = \begin{vmatrix} 1 & \frac{1}{4} & & & & \frac{1}{4} \\ \frac{1}{4} & 1 & \frac{1}{4} & & & \\ & \frac{1}{4} & 1 & \frac{1}{4} & & \\ & & . & . & . & \\ & & & \frac{1}{4} & 1 & \frac{1}{4} \\ \frac{1}{4} & & & & \frac{1}{4} & 1 \end{vmatrix}. \tag{21}$$

Semi-Lagrangian advection requires repeated interpolation on the same grid and the spline transform matrix need only be inverted once for the lifetime of the grid. In fact, each interpolation corresponds to a matrix–vector multiplication $\mathbf{A}^{-1}\mathbf{b}$. However, multiplication is computationally expensive, because the inverse matrix $\mathbf{A}^{-1}$ is full, and $2n$ operations are required to obtain the exact value of each coefficient c_i. Arguments based on the concept of dual functions[21] lead to the conclusion that the absolute values of the elements of $\mathbf{A}^{-1}$ decay exponentially away from the main diagonal. Therefore the sum

$$c_i = \sum_{j=1}^{n} a_{ij}^{-1} b_j \tag{22}$$

can be truncated by retaining the left and right half-bands of the matrix $\mathbf{A}^{-1}$, each of size n_{hb}, and neglecting terms which are far away from the main diagonal:

$$c_i = c_i^- + c_i^+ + a_{ii}^{-1} b_i, \tag{23}$$

where

$$c_i^- = \sum_{j=1-1}^{i-n_{\mathrm{hb}}} a_{ij}^{-1} b_j, \qquad c_i^+ = \sum_{j=i+1}^{i+n_{\mathrm{hb}}} a_{ij}^{-1} b_j. \tag{24}$$

It has been observed that entries of the inverse spline transform matrix exhibit a rapid and monotone decay away from the main diagonal.[20] Our numerical experiments indicate that, regardless of the problem size, 21 entries per row of $\mathbf{A}^{-1}$ or $n_{\mathrm{hb}} = 10$ is sufficient to ensure an accuracy on the order of 10^{-6}. It is much less expensive to precompute an LU factorization of **A** and then obtain each c_i to full machine precision in 6 flops by performing forward and backward substitutions instead of the 41 operations needed to compute an approximate value using $\mathbf{A}^{-1}$. However, a direct LU solver is inherently sequential.

Let us assume that the values of **c** are known at both the left and right boundary points of all subdomains. Then the overall system (20) would reduce to a set of independent linear systems. These systems are solved separately on each processor by employing an economical LU solver, where boundary values are obtained from the truncated sum (24). A parallel algorithm along these lines for computing a spline transform in a distributed memory environment is described below. In this case each processor simultaneously performs the same sequence of operations. It is assumed that the subinterval or subdomain residing on a processor begins at the grid point $x_{i_{\mathrm{first}}}$ and ends at the point $x_{i_{\mathrm{last}}}$. The algorithm consists of the following steps.

© 1997 by John Wiley & Sons, Ltd.

1. Compute the right and left truncated sums (24) $c^+_{i_{\text{first}}-1}$ and $c^-_{i_{\text{last}}+1}$ for the boundary points of adjacent processors. These sums involve only the components of **b** residing on this processor, since the local grid size $i_{\text{last}} - i_{\text{first}} + 1$ is usually larger than n_{hb}.
2. Send these values to adjacent processors.
3. Compute the right and left truncated sums (24) $c^+_{i_{\text{first}}}$ and $c^-_{i\text{last}}$ for the boundary points of this processor.
4. Receive $c^-_{i_{\text{first}}}$ from the left and $c^+_{i_{\text{last}}}$ from the right.
5. Add these values to $c^+_{i_{\text{first}}}$ and $c^-_{i_{\text{last}}}$ to obtain $c_{i_{\text{first}}}$ and $c_{i_{\text{last}}}$ according to (23).
6. Boundary values are now known and the local components of **c** are obtained by solving the local LU system.

The local LU factorizations must be precomputed and the global spline transform matrix **A** must be inverted prior to performing the above algorithm. The cost of these operations is neglected since they are performed once for the lifetime of the computational grid. Only those rows of $\mathbf{A}^{-1}$ corresponding to boundary points need to be stored. The cost of the spline transform scheme is 6 flops per internal grid point plus 41 flops per local interface grid point (point on the boundary of a subdomain).

Once the spline transform **c** is known, it is possible to compute the spline interpolant at any point from a linear combination of four basis splines and this operation requires 24 flops. Two values of **c** come from adjacent processors and thus the algorithm requires two instances of interprocessor communication. The first boundary exchange updates the spline transform **c** and the second permits computation of the interpolating spline. To summarize, the proposed domain decomposition scheme combines the locality of piecewise polynomial interpolation with the accuracy achievable only with global spline interpolation.

2.5. *Spline interpolation in higher dimensions*

The fragments composing a multidimensional cubic spline are themselves cubic polynomials with respect to each of the variables. These pieces are welded together to preserve continuity of the derivatives

$$\frac{\partial^{i_1}}{\partial x^{i_1}}\left(\frac{\partial^{i_2}}{\partial x^{i_2}}\cdots\left(\frac{\partial^{i_r}}{\partial x^{i_r}}\right)\right),$$

where $i_1, i_2, \ldots, i_r$ is a multi-index representing derivatives of orders zero, one and two. A linear multidimensional spline space is constructed from a tensor product of 1D spline spaces,[20] where a multidimensional basis spline is the product of 1D basis splines. In higher dimensions, as in the 1D case, interpolation consists of two stages. First the spline transform of grid point data must be found. Then the spline interpolant between grid points is computed as a linear combination of basis splines. The spline transform matrix is a tensor product of 1D matrices and a multidimensional spline transform is computed from 1D transforms in each co-ordinate direction. Thus the 1D algorithm described above is applied to multiple data vectors in the x-direction, followed by the y-direction and so on. In fact, the procedure is completely analogous to a multidimensional Fourier transform.

The number of floating point operations required to compute a multidimensional spline transform depends on the number of subdomains. The cost for an internal point of a subdomain is 6 flops multiplied by the number of dimensions. The cost for an interface point is 41 flops multiplied by the number of dimensions, assuming that 21 entries are kept in the inverse approximation for each 1D spline transform matrix. Since the number of internal points is much larger than the number of interface points, any difference in cost is negligible. The number of basis spline fragments overlapping a grid interval in 1D, 2D and 3D in four, 16 and 64 respectively. Finally, computation of

 © 1997 by John Wiley & Sons, Ltd.

Table II. Computational cost of multidimensional cubic spline interpolation scheme: flops per grid point

	Spline transform	Interpolation	Total
1D	6	28	34
2D	12	112	124
3D	18	448	466

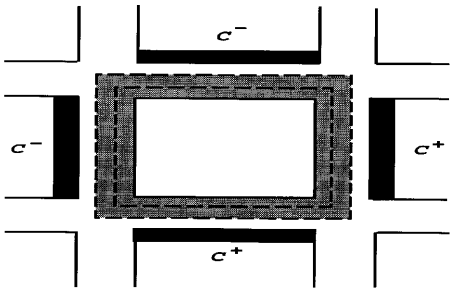

Figure 1. Communication pattern for distrubuted memory bicubic spline interpolation algorithm. Data blocks required for the spline transform are shown in black. A spline transform requires truncated sums (one number per interface point) from processors located to the north, south, east and west. After the spline transform is computed, a two-element-wide boundary layer of spline transform coefficients **c** (shown in grey) permits computation of the spline interpolant at any location. Unlike the spline transform, the communication pattern for the interpolant requires data from corner processors

the cubic polynomial at a particular point requires 6 flops. It is assumed that interpolation on a fixed grid is repeated many times and the cost of all preliminary operations is neglected, such as the inversion of **A** and LU factorization of the local submatrices. The operation counts in one, two and three dimensions for the above cubic spline interpolation scheme are summarized in Table II.

A distributed memory spline interpolation algorithm requires an explicit exchange of data between processors. The spline transform is therefore performed in each space dimension, but all non-local data are obtained in one step. The data required for the spline transform are a single number per interface point, the truncated sum (24). The second phase of the interpolation algorithm also requires a data exchange during which each processor must receive the spline transform coefficients from adjacent processors. In this case the overlap region is two grid points wide. The communication pattern for the distributed spline interpolation algorithm is illustrated in Figure 1.

3. TRAJECTORY INTEGRATION ALGORITHMS

The trajectory equation (2) is an ordinary differential equation (ODE). In the context of one-step Eulerian–Lagrangian methods the backward integration of (2) is represented by the initial value problem (4), where it is assumed that $\mathbf{u}(\mathbf{x}, t_n)$ is known. Given a vector $\mathbf{y}$ and dependent variable t, the general form of the initial value problem in ordinary differential equations is

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, t), \qquad \mathbf{y}(t_0) = \mathbf{y}_0, \tag{25}$$

where $\mathbf{f}$ is some non-linear function of $\mathbf{y}$ and the dependent variable t. There exist many well-known techniques for the numerical integration of (25) and these are described in the classic text by Gear.[22]

© 1997 by John Wiley & Sons, Ltd.

One-step methods of the Runge–Kutta type will be considered. A one-step integration scheme with step size h is written in the form

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + h\boldsymbol{\Phi}(\mathbf{y}(t_n), \Delta t), \tag{26}$$

where $\boldsymbol{\Phi}$ is known as the increment function. An explicit second-order Runge–Kutta method to integrate (25) from t_n to t_{n+1} has an increment function

$$\boldsymbol{\Phi}(\mathbf{y}(t_n), \Delta t) = \tfrac{1}{2}[\mathbf{f}(\mathbf{y}(t_n), t_n) + \mathbf{f}(\mathbf{y}(t_n) + h\mathbf{f}(\mathbf{y}(t_n), t_n), t_n)]. \tag{27}$$

The local truncation error $\mathbf{d}$ of the integration method is defined by

$$\mathbf{d}(\mathbf{y}_n, \Delta t) = \mathbf{y}_n - \mathbf{y}(t_n) + h\boldsymbol{\Phi}(\mathbf{y}(t_n), \Delta t), \tag{28}$$

where $\mathbf{y}_n$ is the numerical solution and $\mathbf{y}(t_n)$ is the actual solution at time t_n. If $\|\mathbf{d}\| \leqslant C\Delta t^{p+1}$, the method is said to be of order p. In the ODE (2) the velocity $\mathbf{u}$ is identified as the non-linear function $\mathbf{f}$ and $\mathbf{x}$ replaces $\mathbf{y}$. When the trapezoidal or Heun method (27) is applied to integrate (4), we obtain

$$\mathbf{x}^* = \mathbf{x} - \frac{\Delta t}{2}[\mathbf{u}(\mathbf{x}, t_n) + \mathbf{u}(\mathbf{x} - \Delta t\mathbf{u}(\mathbf{x}, t_n), t_n - \Delta t)].$$

The above approach can be extended to the classical fourth-order Runge–Kutta scheme.[22] For large values of the Courant number, time truncation errors can begin to dominate and a high-order integration method may be required. The global accuracy of a semi-Lagrangian scheme depends not only on the order of the chosen interpolation scheme but also on the order p of the trajectory integration algorithm.

When the value of $\mathbf{u}$ at time level t_n is not available, a numerical method to predict the velocity is required. The combination of prediction with backward integration is a predictor–corrector-type method. Malevsky[10] combines a quadratic extrapolation based on Adams predictor–corrector methods with a one-step trapezoidal Runge–Kutta method. In this case an $\mathcal{O}(\Delta t^2)$-accurate velocity is computed,

$$\mathbf{u}(\mathbf{x}, t_n) = \tfrac{23}{12}\mathbf{u}(\mathbf{x}, t_{n-1}) - \tfrac{16}{12}\mathbf{u}(\mathbf{x}, t_{n-2}) + \tfrac{5}{12}\mathbf{u}(\mathbf{x}, t_{n-3}), \tag{29}$$

and then the trapezoidal method (27) is employed as a corrector. For example, it was found that one corrector iteration provided sufficient accuracy in simulations of convective turbulence at high Prandtl number. Another issue which is not addressed here is the error introduced by interpolation to obtain $\mathbf{u}$ between grid points. In particular, this would be required in (27) to obtain $\mathbf{u}$ at $t_n - \Delta t$.

In the convection models of both Malevsky[10] and Robert,[23] cubic interpolation of the velocity was employed. For this reason a single corrector iteration such as (27) is recommended, since repeated cubic interpolation of $\mathbf{u}$ in 2D or 3D according to the iterative scheme (5) is very costly. For example, a second-order Adams-type predictor combined with a second-order Heun corrector resulted in a 50% improvement in the efficiency of the existing semi-Lagrangian advection scheme in the MC2 compressible mesoscale weather model.[24] Several authors have proposed alternative trajectory integration methods[25] and some of these are based on the Taylor series expansion of the position vector $\mathbf{x}(t)$ as

$$\mathbf{x}(t) = \mathbf{x}(t - \Delta t) + \sum_{k=1}^{\infty} \frac{(\Delta t)^k}{k!}\frac{\mathrm{d}^k\mathbf{x}}{\mathrm{d}t^k}(t - \Delta t).$$

Although we propose interpolation algorithms suitable for a distributed memory model of computation, the upstream trajectory integration implies a large data overlap for large Courant number C. Lie and Skålin[26] advocate an entirely different approach to the upstream departure point problem. They propose only tracking (integrating) the particle trajectories up to the interprocessor

 © 1997 by John Wiley & Sons, Ltd.

boundaries. The upstream value of a field at the intersection of a trajectory with an interprocessor boundary is then computed using an interpolation-in-time algorithm. This approach is intended to minimize the communication overhead and can be employed with either of the interpolation schemes described above. However, the advantage of such a scheme in the multidimensional case remains to be demonstrated.

4. ERROR ANALYSIS

Recent work by Falcone and Ferretti[27] and Falcone *et al.*[28] provides *a priori* $L^\infty(\Omega)$ and pointwise error estimates for the global rate of convergence of a class of finite element and polynomial interpolation-based semi-Langrangian advection schemes. In order to optimize the efficiency and numerical accuracy of such schemes, the authors characterize the relationship between the time step Δt and space step Δx for particular polynomial interpolation bases and trajectory integration methods. We adopt the notation of Falcone *et al.*[28] by defining a regular Cartesian grid with nodes $x_j, j \in Q = \{1, \ldots, q\}$, and let $\psi_j(x)_{j\in Q}$ be a family of bounded functions such that $\psi_j(x_i) = \delta_{ij}$, $\|\psi_j\|_\infty = 1$. These functions are then used to construct a local (finite-dimensional) representation of the approximate (semi-Lagrangian) solutions and such approximations can be based on finite element or polynomial bases.

It is assumed that the initial field $v(x, 0)$ belongs to the Sobolev space $W^{s,p}(\Omega)$ and thus has regularity s in the $L^p(\Omega)$ norm. According to Falcone and Ferretti,[27] the error of a semi-Lagrangian scheme is bounded according to

$$|v_j^n - v(x_j, n\Delta t)| \leqslant C(n\Delta t)\left(\Delta t^p + \frac{1}{\Delta t}E(\Delta x)\right), \tag{30}$$

where n is the number of time steps of length Δt, p is the order of the trajectory integration scheme (see equation (28) in Section 3), Δx is the space step and $E(\Delta x)$ is the interpolation error. Bermejo[29] derives a similar result (see equation (25) of Reference 9). Equation (30) may refer to either the global or local (in space) error. The global error depends on either the regularity of the solution or the interpolation error,[28] whereas the local error in smooth regions depends solely on the trajectory truncation error and the interpolation error. In our numerical tests, both global and local errors are measured in order to discriminate between these effects. If $\| \sum_m |\psi_m| \|_\infty = 1$, then the estimate (30) holds at a single node x_j provided that the functions ψ_j have compact support in a ball of radius $\mathcal{O}(\Delta x)$. This is true for finite elements and when $\Delta x = o(\Delta t)$. In this case, $E(\Delta x)$ represents the local interpolation error in the neighbourhood of x_j. For polynomial interpolation it is well-known that Lagrange basis polynomials $l_j(x)$ satisfy $l_j(x_j) = 1$, $l_j(x_m) = 0, m \neq j$, but the condition $\|l_j\|_\infty = 1$ is only valid when the nodes x_j are zeros of Legendre polynomials of degree q. Thus the interpolation error may not be bounded in the $L^\infty(\Omega)$ norm. Nevertheless, we can still apply local error analysis to Lagrange interpolation on equispaced grids. In the present paper we are primarily concerned with B-spline bases $B_j(x)$ which have compact support in an interval of length $4\Delta x$ and also satisfy $\|B_j\|_\infty = 1$ and $\| \sum_m |B_m| \|_\infty = 1$.[20]

To design the most efficient and accurate scheme for a given Δx and Δt, the global and local errors can be estimated given piecewise polynomial approximations (of degree r) with compact support in a ball of radius $\mathcal{O}(\Delta x)$ of the underlying basis functions. Moreover, we assume that the solution v has a finite global regularity s ($v \in W^{s,\infty}(\Omega)$) and is piecewise C^∞. A bound on the interpolation error is therefore

$$E(\Delta x) \leqslant C\Delta x^{\min(s, r+1)}. \tag{31}$$

© 1997 by John Wiley & Sons, Ltd.

Substitution of (31) into (30) results in the global error estimate

$$|v_j^n - v(x_j, n\Delta t)| \leqslant C\left(\Delta t^p + \frac{\Delta x^{\min(s,r+1)}}{\Delta t}\right) \tag{32}$$

and assuming that the solution is C^∞ in a small neighbourhood centred at x_j leads to the local error estimate

$$|v_j^n - v(x_j, n\Delta t)| \leqslant C\left(\Delta t^p + \frac{\Delta x^{r+1}}{\Delta t}\right). \tag{33}$$

In the latter case we assume that $\Delta x = o(\Delta t)$. It is then possible to express Δt as a function of Δx such that $\Delta t = \Delta x^\alpha$ and substituting this relation into (32) gives

$$|v_j^n - v(x_j, n\Delta t)| \leqslant C(\Delta x^{\alpha p} + \Delta x^{\min(s,r+1)-\alpha}). \tag{34}$$

The global convergence rate μ is controlled by the minimum of two exponents αp and $\min(s, r+1) - \alpha$ and the optimal convergence rate is attained at

$$\alpha = \frac{\min(s, r+1)}{p+1}. \tag{35}$$

The optimal convergence rate depends on the regularity of the solution. If the solution is not smooth enough ($s < r+1$), increasing the interpolation order r does not improve the rate μ. In regions where the solution is smooth the convergence rate may not improve, since the time truncation error does not depend on the local regularity.

For sufficiently smooth ($s \geqslant r+1$) advected fields the convergence rate is

$$\mu = \alpha p = \frac{p(r+1)}{p+1}. \tag{36}$$

If $r \geqslant p$, the condition $\Delta x = o(\Delta t)$ is no longer valid and thus the error may not depend solely on the local regularity of the solution. Consequently, for smooth enough solutions, we have the following.

1. When $r < p$, the local error estimate (33) holds, so (36) may apply.
2. When $r \geqslant p$, the convergence rate can be optimized by
   (a) choosing α to minimize the global $L^\infty(\Omega)$ error
   (b) choosing $0 \ll \alpha \leqslant 1$.

5. NUMERICAL RESULTS

In this section we perform two sets of tests. The first set of tests examines the mass and energy conservation properties of the semi-Lagrangian schemes described in this paper. Advection of a sharply peaked cone in a uniform rotational flow field in 2D is considered. The rotating cone problem has become a standard test case for numerical advection algorithms since it is perhaps the simplest problem with spatial variation of the flow field. The rotating cone problem has been employed by many authors to test numerical advection schemes and it is now part of many standard test suites in computational fluid dynamics (see the textbook by Vreugdenhil and Koren.[30] Spline-based advection schemes are described by Bermejo[9] and Bermejo and Staniforth.[31])

In a second set of tests we rotate a paraboloid which is smooth within a ball of radius $|\mathbf{x} - \mathbf{x}_0| < R$. This allows us to compute both global and local errors and analyse these according to the theoretical

 © 1997 by John Wiley & Sons, Ltd.

estimates of Falcone and Ferretti.[27] For solid body rotation at a constant angular rotation rate of ω radians per second the flow field is given by $\mathbf{u} = (-\omega y, \omega x)$, where

$$\mathbf{u} \cdot \nabla = -\omega y \frac{\partial}{\partial y} + \omega x \frac{\partial}{\partial x}.$$

In column vector notation this problem is equivalent to the integration of a linear system of ordinary differential equations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}.$$

5.1. Conservation tests

Our first two test problems are based on a low-resolution grid in Reference 9 and the high-resolution grid in Reference 31. In these tests we compute the field at the exact analytic departure points to isolate the effects introduced by repeated interpolation of the advected field from those of the trajectory integration method. In both cases the initial 2D scalar field is defined by

$$f(x, y) = \begin{cases} H(1 - |\mathbf{x} - \mathbf{x}_0|/R), & |\mathbf{x} - \mathbf{x}_0| \leqslant R, \\ 0, & \text{otherwise}, \end{cases} \tag{37}$$

on the domain $\Omega = [-0{\cdot}5, 0{\cdot}5] \times [-0{\cdot}5, 0{\cdot}5]$. R and H are the radius and height of the cone respectively and $\mathbf{x}_0 = (x_0, y_0)$ is the centre of the cone. The velocity components are $\mathbf{u} = (u, v) = (-\omega y, \omega x)$, with constant angular rotation rate $\omega = 0{\cdot}3636 \times 10^{-4}\ \mathrm{s}^{-1}$. This is now a standard test case in the meteorological literature, representing one rotation of the initial field every 2 days.[32] For example, typical synoptic-scale meteorological models based on semi-Lagrangian advection schemes use time steps of the order of $\Delta t = 1800$ s, corresponding to one revolution every 96 time steps. In the first case a 35×35 grid is employed with uniform mesh length $\Delta x = 1/35$, $H = 100$, $R = 4\Delta x$ and $(x_0, y_0) = (-8\Delta x, 0)$. The second case employs a 100×100 grid with uniform mesh length $\Delta x = 0{\cdot}01$, $H = 100$, $R = 8\Delta x$ and $(x_0, y_0) = (-14\Delta x, 0)$. The maximum Courant number $C_{\max}$ occurs near the boundary $\partial\Omega$, where

$$C_{\max} = \max\left\{0{\cdot}5\frac{\omega\Delta t}{\Delta x}, 0{\cdot}5\frac{\omega\Delta t}{\Delta y}\right\}.$$

Bermejo,[9] Priestley[32] and Bermejo and Staniforth[31] monitor the evolution of the normalized first (mass) and second (energy) moments of an advected scalar field f to determine the conservation properties of spline interpolation semi-Lagrangian schemes. The moments are defined by $\int f\mathrm{d}x / \int f_0\mathrm{d}x$ and $\int f^2\mathrm{d}x / \int f_0^2\mathrm{d}x$ respectively. The normalized maximum and minimum heights $\max f / \max f_0$ and $\min f / \max f_0$ monitor dissipation and monotonicity of the numerical scheme. Tests on 35×35 and 100×100 grids are reported using a time step of $\Delta t = 1800$ s and exact analytic departure points.[33] One full rotation of the cone corresponds to $N_{\mathrm{tot}} = 96$ time steps. For these tests a total of $N_{\mathrm{rot}} = 6$ full rotations of the cone are taken.

Results for the low-resolution 35×35 grid are presented in Tables III and IV, whereas the integrations on the high-resolution 100×100 grid are summarized in Tables V and VI. The domain decomposition B-spline with $n_{\mathrm{hb}} = 12$ exhibits better mass conservation properties than the piecewise cubic Newton interpolating polynomial on the 35×35 grid as indicated by the ratio $\int f\mathrm{d}x / \int f_0\mathrm{d}x$. The results are similar on the 100×100 grid but mass is not conserved up to machine precision. Further testing revealed that mass conservation improves as n_{hb} increases, i.e. $\int f\mathrm{d}x / \int f_0\mathrm{d}x$ remains close to 1·0. Relatively weak energy dissipation for cubic B-splines (see

© 1997 by John Wiley & Sons, Ltd.

Table III. 35×35 grid. $\Delta t = 1800$ s. $C_{\max} = 2{\cdot}29$. Exact analytic departure points. Piecewise cubic Newton polynomial

N_{tot}	N_{rot}	$\int f\,dx / \int f_0\,dx$	$\int f^2\,dx / \int f_0^2\,dx$	$\max f / \max f_0$	$\min f / \max f_0$
0	0	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}0000000 \times 10^{0}$
96	1	$0{\cdot}1001808 \times 10^{1}$	$0{\cdot}7283366 \times 10^{0}$	$0{\cdot}7028023 \times 10^{0}$	$-0{\cdot}2422089 \times 10^{-1}$
192	2	$0{\cdot}9956945 \times 10^{0}$	$0{\cdot}6171257 \times 10^{0}$	$0{\cdot}5829467 \times 10^{0}$	$-0{\cdot}2474830 \times 10^{-1}$
288	3	$0{\cdot}9817497 \times 10^{0}$	$0{\cdot}5495566 \times 10^{0}$	$0{\cdot}5125213 \times 10^{0}$	$-0{\cdot}2357593 \times 10^{-1}$
384	4	$0{\cdot}9682724 \times 10^{0}$	$0{\cdot}5021800 \times 10^{0}$	$0{\cdot}4643474 \times 10^{0}$	$-0{\cdot}2318519 \times 10^{-1}$
480	5	$0{\cdot}9583055 \times 10^{0}$	$0{\cdot}4663555 \times 10^{0}$	$0{\cdot}4285303 \times 10^{0}$	$-0{\cdot}2247365 \times 10^{-1}$
576	6	$0{\cdot}9523448 \times 10^{0}$	$0{\cdot}4379707 \times 10^{0}$	$0{\cdot}4004302 \times 10^{0}$	$-0{\cdot}2046848 \times 10^{-1}$

Table IV. 35×35 grid. $\Delta t = 1800$ s. $C_{\max} = 2{\cdot}29$. Exact analytic departure points. Domain decomposition cubic B-spline

N_{tot}	N_{rot}	$\int f\,dx / \int f_0\,dx$	$\int f^2\,dx / \int f_0^2\,dx$	$\max f / \max f_0$	$\min f / \max f_0$
0	0	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}0000000 \times 10^{0}$
96	1	$0{\cdot}9997961 \times 10^{0}$	$0{\cdot}9458537 \times 10^{0}$	$0{\cdot}9353862 \times 10^{0}$	$-0{\cdot}2123398 \times 10^{-1}$
192	2	$0{\cdot}9994493 \times 10^{0}$	$0{\cdot}9053355 \times 10^{0}$	$0{\cdot}8933336 \times 10^{0}$	$-0{\cdot}2664576 \times 10^{-1}$
288	3	$0{\cdot}1000237 \times 10^{1}$	$0{\cdot}8718465 \times 10^{0}$	$0{\cdot}8559584 \times 10^{0}$	$-0{\cdot}2922192 \times 10^{-1}$
384	4	$0{\cdot}1001367 \times 10^{1}$	$0{\cdot}8433061 \times 10^{0}$	$0{\cdot}8235943 \times 10^{0}$	$-0{\cdot}2992606 \times 10^{-1}$
480	5	$0{\cdot}1002408 \times 10^{1}$	$0{\cdot}8184590 \times 10^{0}$	$0{\cdot}7953780 \times 10^{0}$	$-0{\cdot}3128533 \times 10^{-1}$
576	6	$0{\cdot}1003185 \times 10^{1}$	$0{\cdot}7964808 \times 10^{0}$	$0{\cdot}7705087 \times 10^{0}$	$-0{\cdot}3382451 \times 10^{-1}$

Table V. 100×100 grid. $\Delta t = 1800$ s. $C_{\max} = 6{\cdot}54$. Exact analytic departure points. Piecewise cubic Newton polynomial

N_{tot}	N_{rot}	$\int f\,dx / \int f_0\,dx$	$\int f^2\,dx / \int f_0^2\,dx$	$\max f / \max f_0$	$\min f / \max f_0$
0	0	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}0000000 \times 10^{0}$
96	1	$0{\cdot}9999277 \times 10^{0}$	$0{\cdot}9683290 \times 10^{0}$	$0{\cdot}8474599 \times 10^{0}$	$-0{\cdot}1064930 \times 10^{-1}$
192	2	$0{\cdot}9998500 \times 10^{0}$	$0{\cdot}9438646 \times 10^{0}$	$0{\cdot}8218472 \times 10^{0}$	$-0{\cdot}1272116 \times 10^{-1}$
288	3	$0{\cdot}9997708 \times 10^{0}$	$0{\cdot}9218599 \times 10^{0}$	$0{\cdot}8017961 \times 10^{0}$	$-0{\cdot}1369425 \times 10^{-1}$
394	4	$0{\cdot}9996949 \times 10^{0}$	$0{\cdot}9017962 \times 10^{0}$	$0{\cdot}7828491 \times 10^{0}$	$-0{\cdot}1437081 \times 10^{-1}$
480	5	$0{\cdot}9996221 \times 10^{0}$	$0{\cdot}8833570 \times 10^{0}$	$0{\cdot}7647777 \times 10^{0}$	$-0{\cdot}1517552 \times 10^{-1}$
576	6	$0{\cdot}9995534 \times 10^{0}$	$0{\cdot}8663105 \times 10^{0}$	$0{\cdot}7476688 \times 10^{0}$	$-0{\cdot}1584693 \times 10^{-1}$

Table VI. 100×100 grid. $\Delta t = 1800$ s. $C_{\max} = 6{\cdot}54$. Exact analytic departure points. Domain decomposition cubic B-spline

N_{tot}	N_{rot}	$\int f\,dx / \int f_0\,dx$	$\int f^2\,dx / \int f_0^2\,dx$	$\max f / \max f_0$	$\min f / \max f_0$
0	0	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}1000000 \times 10^{1}$	$0{\cdot}000000 \times 10^{0}$
96	1	$0{\cdot}9999620 \times 10^{0}$	$0{\cdot}9951894 \times 10^{0}$	$0{\cdot}9120359 \times 10^{0}$	$-0{\cdot}9839442 \times 10^{-2}$
192	2	$0{\cdot}9999246 \times 10^{0}$	$0{\cdot}9919201 \times 10^{0}$	$0{\cdot}8961392 \times 10^{0}$	$-0{\cdot}1120558 \times 10^{-1}$
288	3	$0{\cdot}9998860 \times 10^{0}$	$0{\cdot}9890301 \times 10^{0}$	$0{\cdot}8842174 \times 10^{0}$	$-0{\cdot}1197218 \times 10^{-1}$
384	4	$0{\cdot}9998448 \times 10^{0}$	$0{\cdot}9863490 \times 10^{0}$	$0{\cdot}8749135 \times 10^{0}$	$-0{\cdot}1232583 \times 10^{-1}$
480	5	$0{\cdot}9998069 \times 10^{0}$	$0{\cdot}9837957 \times 10^{0}$	$0{\cdot}8676570 \times 10^{0}$	$-0{\cdot}1233632 \times 10^{-1}$
576	6	$0{\cdot}9997666 \times 10^{0}$	$0{\cdot}9813301 \times 10^{0}$	$0{\cdot}8619168 \times 10^{0}$	$-0{\cdot}1216845 \times 10^{-1}$

© 1997 by John Wiley & Sons, Ltd.

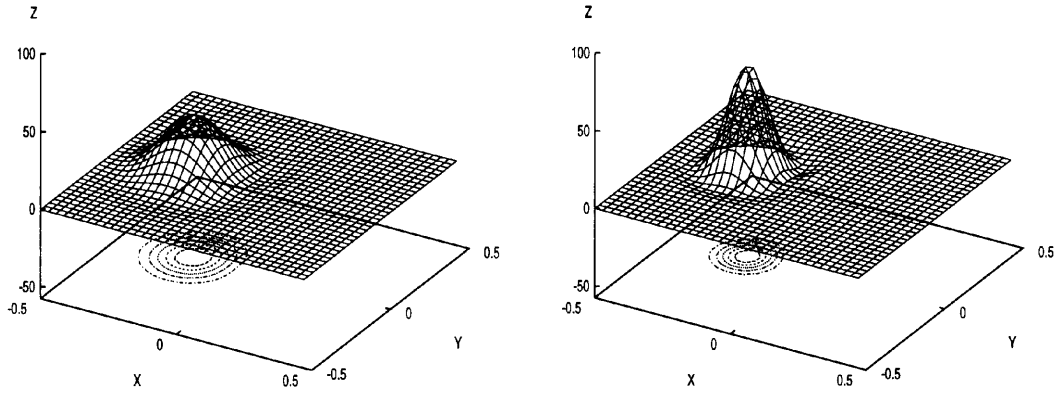

Figure 2. Advected cone after $N_{\text{rot}} = 6$ rotations. 35×35 grid. $\Delta t = 1800$ s. $C_{\max} = 2{\cdot}29$. Exact analytic departure points. Left: piecewise cubic Newton polynomial. Right: domain decomposition cubic B-spline

Figure 2) is indicated by slowly decreasing values of $\int f^2 \mathrm{d}x / \int f_0^2 \mathrm{d}x$ and $\max f / \max f_0$. We also observe that the initial height decreases by $\mathcal{O}(\Delta x)$ each time step. The method based on piecewise cubic polynomial interpolation tends to be more dissipative as indicated by a faster decrease in $\int f^2 \mathrm{d}x / \int f_0^2 \mathrm{d}x$ and $\max f / \max f_0$. For piecewise cubic interpolation we note that the amount of dissipation depends on the 'residual' Courant number of local co-ordinate $\hat{x}$.[34]

5.2. Convergence tests

In this subsection we compare numerical results with those reported by Falcone *et al.*[28] Experiments are designed so that α optimizes either the global L^∞ or local error. We consider numerical schemes with second-order Heun or fourth-order Runge–Kutta (RK4) time stepping coupled with domain decomposition cubic B-splines and piecewise cubic polynomials. The test case is a rotating paraboloid with rotation rate $\omega = 1$ in the domain $\Omega = [-0{\cdot}5, 0{\cdot}5] \times [-0{\cdot}5, 0{\cdot}5]$ with

$$f(x, y) = \begin{cases} H(1 - |\mathbf{x} - \mathbf{x}_0|^2 / R^2), & |\mathbf{x} - \mathbf{x}_0| \leqslant R, \\ 0, & \text{otherwise}, \end{cases} \tag{38}$$

where $\mathbf{x}_0 = (0{\cdot}2, 0{\cdot}2)$, $H = 1$ and $R = 0{\cdot}13$, so that the solution is globally Lipshitz continuous and C^∞ outside the curve $|\mathbf{x} - \mathbf{x}_0| = R$. The global and local errors are computed after one rotation at $t = 2\pi$. Table VII contains the global L^∞ errors $\|f - f_0\|_\infty$ for B-spline and piecewise cubic interpolation coupled with the Heun and Runge–Kutta methods. Three different time and space step sizes are employed, resulting in maximum Courant numbers ranging from $C_{\max} < 1$ up to the limit where trajectories intersect, $C_{\max} \approx 16$.[14,35] Both the error due to repeated interpolation (see Section 5.1) and that due to the trajectory integration method contribute to the total error of the semi-Lagrangian schemes reported in Tables VII and VIII. The RK4/B-spline scheme is better than the other three schemes in terms of both accuracy and low viscosity. In particular, the RK4/B-spline scheme performs far better at lower resolutions.

For the optimal α and relation $\Delta t = \Delta x^\alpha$ the global L^∞ and local errors at $t = 2\pi$ are listed in tabular form. The order of convergence is computed as

$$\frac{\log(\epsilon_1 / \epsilon_2)}{\Delta x_1 / \Delta x_2}, \tag{39}$$

© 1997 by John Wiley & Sons, Ltd.

Table VII. L^∞ errors $\|f - f_0\|_\infty$ at $t = 2\pi$ for rotating paraboloid (38). Grid sizes 25×25, 50×50, 100×100 correspond to step sizes $\Delta x = 0{\cdot}04, 0{\cdot}02, 0{\cdot}01$

	Fourth-order Runge–Kutta					
	Piecewise cubic			Cubic B-spline		
Δx	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$
0·04	0·106	0·136	0·277	0·0561	0·0696	0·1083
0·02	0·0658	0·0834	0·0970	0·0402	0·0486	0·0530
0·01	0·0425	0·0500	0·0600	0·0266	0·0303	0·0352
	Second-order Heun					
	Piecewise cubic			Cubic B-spline		
Δx	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$
0·04	0·338	0·171	0·277	0·346	0·119	0·1042
0·02	0·377	0·132	0·103	0·397	0·114	0·0608
0·01	0·401	0·115	0·0710	0·421	0·116	0·0478

where Δx_1 and Δx_2 are the largest and smallest two steps taken, with errors ϵ_1 and ϵ_2 respectively. In Tables IX and X the column heading L^∞ refers to the global error, whereas 'local' is the error obtained in the ball $|\mathbf{x} - \mathbf{x}_0| < R/2$ and the observed rate is computed according to (39). This problem has $W^{1,\infty}(\Omega)$ regularity and thus the optimal global error exponent is $\alpha = 1/p$. B-splines have order $r = 4$ and when coupled with a fourth-order Runge–Kutta method $p = 4$ we have $r \geqslant p$. Thus we can apply either 2(a) or 2(b) in Section 4 to optimize the local error rate. From Table IX we see that choosing $\Delta t = \Delta x^{3/5}$ optimizes the local convergence rate of the RK4/B-spline scheme and choosing $\alpha = \frac{1}{5}$ slightly underestimates the global convergence rate.

Table VIII. L^∞ norms $\|f\|_\infty$ at $t = 2\pi$ for rotating paraboloid (38). Grid sizes 25×25, 50×50, 100×100 correspond to step sizes $\Delta x = 0{\cdot}04, 0{\cdot}02, 0{\cdot}01$

	Fourth-order Runge–Kutta					
	Piecewise cubic			Cubic B-spline		
Δx	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$
0·04	0·910	0·849	0·686	0·969	0·976	0·950
0·02	0·999	1·002	1·010	1·000	1·002	1·001
0·01	1·000	1·000	1·000	1·000	1·000	1·000
	Second-order Heun					
	Piecewise cubic			Cubic B-spline		
Δx	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$	$\Delta t = \pi/10$	$\Delta t = \pi/20$	$\Delta t = \pi/40$
0·04	0·959	0·872	0·685	1·015	0·999	0·956
0·02	0·995	0·999	1·010	0·996	0·998	1·001
0·01	0·998	0·998	1·000	0·997	0·997	1·000

 © 1997 by John Wiley & Sons, Ltd.

Table IX. L^∞ and local errors for rotating paraboloid

	RK4/B-spline scheme			
	$\Delta t = \Delta x^{1/5}$		$\Delta t = \Delta x^{3/5}$	
Δx	L^∞	Local	L^∞	Local
0·04	0·04733	0·015042	0·07443	0·037560
0·02	0·04137	0·004885	0·05635	0·008721
0·01	0·02662	0·001072	0·04034	0·000536
Rate	0·42	1·91	0·44	3·06
	Heun/B-spline scheme			
	$\Delta t = \Delta x^{1/3}$		$\Delta t = \Delta x$	
Δx	L^∞	Local	L^∞	Local
0·04	0·40948	0·409478	0·11061	0·089810
0·02	0·30230	0·200723	0·08400	0·016482
0·01	0·20944	0·121656	0·05744	0·003791
Rate	0·48	0·87	0·47	2·28

Table X indicates that both the Runge–Kutta and Heun piecewise cubic schemes approach the optimal local convergence rate when $\Delta t = \Delta x^{3/5}$. The local error at high resolution $\Delta x = 0{\cdot}01$ is better in the case of B-splines. However, to achieve such a high convergence rate, the Courant number must remain small ($C < \pi$). We also find that these schemes can achieve a slightly higher global convergence rate but the global error for domain decomposition cubic B-splines is still smaller at the large Courant numbers resulting from $\Delta t = \Delta x^{1/5}$.

Table X. L^∞ and local errors for rotating paraboloid

	RK4/B-piecewise cubic scheme			
	$\Delta t = \Delta x^{1/5}$		$\Delta t = \Delta x^{3/5}$	
Δx	L^∞	Local	L^∞	Local
0·04	0·08871	0·066522	0·14189	0·135462
0·02	0·06400	0·006593	0·09845	0·026563
0·01	0·04255	0·001097	0·06641	0·002149
Rate	0·53	2·96	0·55	2·99
	Heun/piecewise cubic scheme			
	$\Delta t = \Delta x^{1/3}$		$\Delta t = \Delta x$	
Δx	L^∞	Local	L^∞	Local
0·04	0·40455	0·404545	0·33483	0·334834
0·02	0·29394	0·224590	0·15006	0·089693
0·01	0·20285	0·120935	0·10055	0·012611
Rate	0·50	0·87	0·87	2·36

© 1997 by John Wiley & Sons, Ltd.

6. CONCLUSIONS

We conclude that the domain decomposition B-spline interpolation method designed for MIMD distributed memory, parallel computation maintains a low communication overhead while still achieving the accuracy of a global B-spline interpolation scheme. In fluid dynamics models, piecewise cubic polynomial interpolation should only be employed for relatively short model runs owing to excessive numerical dissipation. In particular, the domain decomposition cubic B-spline interpolation scheme exhibited superior mass and energy conservation properties compared with the scheme based on piecewise cubic Newton polynomials. For domain decomposition B-splines there is a trade-off between mass conservation and the amount of computation at subdomain boundaries which is controlled by the half-bandwidth n_{hb}. Both methods have similar communication requirements and should prove to be scalable on distributed memory, parallel computers.

The global accuracy of an Eulerian–Lagrangian method depends on the regularity of the solution. Near steep gradients or fronts the solutions may not be smooth and increasing the interpolation order does not necessarily increase the global convergence rate of a semi-Lagrangian scheme. An optimal convergence rate can be achieved through the judicious choice of step sizes Δx and Δt. To maintain accuracy at large Courant numbers, a high-order numerical ODE solver is required. Another important issue which must be addressed is the computational complexity of the overall scheme. Runge–Kutta methods require evaluation of the velocity between grid points and this may require interpolation of the velocity $\mathbf{u}$. In 2D and 3D such a computation becomes very costly and alternative methods should be considered.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Jean Côté and Dr. Andrew Staniforth of Environment Canada for many stimulating discussions and their interest in this work. Maurizio Falcone kindly provided preprints of his recent papers on L^{∞} global convergence analysis. We would also like to thank the referees whose comments and suggestions greatly improved the manuscript.

REFERENCES

1. A. M. Baptista, 'Solution of advection-dominated transport by Eulerian–Lagrangian methods using the backwards method of characteristics', *Ph.D. Thesis*, Massachusetts Institute of Technology, Cambridge, MA, 1987.
2. A. Oliveira and A. M. Baptista, 'A comparison of integration and interpolation Eulerian–Lagrangian methods', *Int. j. numer. methods fluids*, **21**, 183–204 (1995).
3. P. J. Roache, *Computational Fluid Dynamics*, Hermosa, Alberquerque, NM, 1982.
4. R. R. Ewing, T. F. Russell and M. F. Wheeler, 'Convergence analysis of an approximation of miscible displacement in porous media by mixed finite elements and a modified method of characteristics', *Comput. Methods Appl. Mech. Eng.*, **47**, 73–92 (1984).
5. J. Douglas and T. F. Russell, 'Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures', *SIAM J. Numer. Anal.*, **19**, 871–885 (1982).
6. O. Pironneau, 'On the transport-diffusion algorithm and its applications to the Navier–Stokes equations', *Numer. Math.*, **38**, 309–332 (1982).
7. E. Süli, 'Convergence and nonlinear stability of the Lagrange–Galerkin method for the Navier–Stokes equations', *Numer. Math.*, **53**, 459–483 (1988).
8. K. Boukir, Y, Maday and B. Métivent, 'A high-order characteristics method for the incompressible Navier–Stokes equations', *Comput. Methods Appl. Meth. Eng.*, **116**, 211–218 (1994).
9. R. Bermejo, 'On the equivalence of semi-Lagrangian schemes and particle-in-cell finite element methods', *Mon. Weather Rev.*, **118**, 979–987 (1990).
10. A. Malevsky, 'Spline–characteristic method for simulation of convective turbulence', *J. Comput. Phys.*, **123**, 466–475 (1996).
11. A. Staniforth and J. Côté, 'Semi-Lagrangian integration schemes for atmospheric models—a review', *Mon. Weather Rev.*, **119**, 2206–2223 (1991).

 © 1997 by John Wiley & Sons, Ltd.

12. R. Courant, K. O. Friedrichs and H. Lewy, 'Uber die partiellen Differenzengleichungen der mathematischen Physik', *Math. Ann.*, **100**, 32–74 (1928).
13. P. J. Roache, 'A flux-based modified method of characteristics', *Int. j. numer. methods fluids*, **15**, 1259–1275 (1992).
14. P. K. Smolarkiewicz and J. A. Pudykiewicz, 'A class of semi-Lagrangian approximations for fluids', *J. Atmos. Sci.*, **49**, 2082–2096 (1992).
15. A. Robert, 'A stable numerical integration scheme for the primitive meteorological equations', *Atmos. Ocean.*, **19**, 35–46 (1981).
16. W. P. Crowley, 'Numerical advection experiments', *Mon. Weather Rev.*, **92**, 1–11 (1968).
17. R. J. LeVeque, *Numerical Methods for Conservation Laws*, Birkhaüser, Boston, MA, 1990.
18. R. Courant, E. Isaacson and H. Rees, 'On the solution of nonlinear hyperbolic differential equations by finite differences', *Commun. Pure Appl. Math.*, **5**, 243–255 (1952).
19. S. J. Thomas and J. Côté, 'Massively parallel semi-Lagrangian advection', *J. Simul. Pract. Theory*, **3**, 223–238 (1995).
20. C. de Boor, *A Practical Guide to Splines*, Springer, New York, 1978.
21. C. K. Chui, *An Introduction to Wavelets*, Academic, Boston, MA, 1992.
22. C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
23. A. Robert, 'Bubble convection experiments with a semi-implicit formulation of the Euler equations', *J. Atmos. Sci.*, **50**, 1865–1873 (1993).
24. S. J. Thomas, R. Benoit, M. Desgagne and A. V. Malevsky, 'A new dynamics kernel for the MC2 model. *Atmos. Ocean*., in press (1997)
25. J. L. McGregor, 'Economical determination of departure points for semi-Lagrangian models', *Mon. Weather Rev.*, **121**, 221–230 (1993).
26. I. Lie and R. Skålin, 'Parallelism in semi-Lagrangian transport', in G.-R. Hoffman and N. Kreitz (eds), *Parallel Supercomputing in Atmospheric Science*, World Scientific, Singapore, 1955, pp. 455–471.
27. M. Falcone and R. Ferretti, 'A class of fully discrete high-order schemes for advection equations', *SIAM J. Numer. Anal.*, in press.
28. M. Falcone, R. Ferretti and T. Manfroni, 'Optimal discretisation steps for a class of semi-Lagrangian schemes', in preparation.
29. R. Bermejo, 'An analysis of an algorithm for the Galerkin–characteristic method', *Numer. Math.*, **60**, 163–194 (1991).
30. C. B. Vreugdenhil and B. Koren, *Numerical Methods for Advection Diffusion Problems*, NNFM Vol. 45, Vieweg, Wiesbaden, 1993.
31. R. Bermejo and A. Staniforth, 'The conversion of semi-Lagrangian advection schemes to quasi-monotone schemes', *Mon. Weather Rev.*, **120**, 2622–2632 (1992).
32. A. Priestley, 'A quasi-conservative version of the semi-Lagrangian advection scheme', *Mon. Weather Rev.*, **121**, 621–629 (1993).
33. A. McDonald, 'Accuracy of multiply-upstream, semi-Lagrangian advection schemes', *Mon. Weather Rev.*, **112**, 1267–1275 (1984).
34. A. McCalpin, 'A quantitative analysis of the dissipation inherent in semi-Lagrangian advection', *Mon. Weather Rev.*, **116**, 2330–2336 (1988).
35. J. A. Pudykiewicz and A. Staniforth, 'Some properties and comparative performance of the semi-Lagrangian method of Robert in the solution of the advection–diffusion equation', *Atmos. Ocean.*, **22**, 283–308 (1984).

© 1997 by John Wiley & Sons, Ltd.