Parallel computing for solute transport models via alternating direction collocation

M. C. Curran and M. B. Allen III

Department of Mathematics, University of Wyoming, Laramie, WY 82070 U.S.A

We examine algorithmic aspects of M. Celia's alternating-direction scheme for finite-element collocation, especially as implemented for the two-dimensional advection-diffusion equation governing solute transport in groundwater. Collocation offers savings over other finite-element techniques by obviating the numerical quadrature and global matrix assembly procedures ordinarily needed in Galerkin formulations. The alternating-direction approach offers further saving in storage and serial runtime and, significantly, yields highly parallel algorithms involving the solution of problems having only one-dimensional structure. We explore this parallelism.

Key Words: Alternating-direction methods, collocation, parallel computing.

1. INTRODUCTION

Alternating-direction (AD) methods have been of interest in the numerical solution of partial differential equations since their introduction in 1955 by Peaceman and Rachford[1]. In 1970 Douglas and Dupont[2] developed an alternating-direction Galerkin method, variants of which have attracted the attention of several authors, including Dendy and Fairweather[3] and Hayes and Krishnamachari[4]. Analogous alternating-direction collocation (ADC) methods have also appeared in several papers, including those by Bangia *et al.*[5], Chang and Finlayson[6], Hayes[7], Celia *et al.*[8], Celia[9], and Celia and Pinder[10]. Reference 9, in particular, demonstrates the applicability of ADC to problems of practical importance in water resources engineering.

We examine Celia's ADC for the two-dimensional advection-diffusion equation for solute transport in a known velocity field. Of interest here are algorithmic features of ADC that enhance its efficiency in comparison with standard two-dimensional collocation, especially the amenability of ADC to implementation on parallel-architecture computers. The paper has the following structure: section 2 briefly reviews finite-element collocation using bicubic Hermite bases; section 3 discusses the AD method applied to collocation; in section 4 we discuss the method's performance on a parallel computer.

2. REVIEW OF FINITE-ELEMENT COLLOCATION

We begin by reviewing finite-element collocation for problems in two space dimensions. The primary aim of this review is to establish notation and terminology for the rest of the paper. Lapidus and Pinder[11] give an alternative, more detailed description of the methodology that may be more appropriate for those seeking an introduction.

Consider the following problem, posed on the rectangular spatial domain $\Omega = (a, b) \times (c, d)$:

$$\begin{aligned}&\text{(a)}\ \partial_t u + \mathbf{v} \cdot \nabla u - \nabla \cdot (D\nabla u) = 0, (x, y, t) \in \Omega \times (0, \infty),\\ &\text{(b)}\ u(x, y, 0) = u_I(x, y),\ (x, y) \in \Omega, \qquad (1)\\ &\text{(c)}\ u(x, y, t) = u_B(x, y, t),\ (x, y) \in \partial\Omega, t \geqslant 0.\end{aligned}$$

In equation (1a), $\mathbf{v} = \mathbf{v}(x, y)$ represents a known fluid velocity, which in applications might be the Darcy velocity computed using a groundwater flow model. $D = D(x, y)$ is a diffusion coefficient, which in underground flows could serve as a simple model of hydrodynamic dispersion. (For purposes of testing the efficiency of collocation algorithms, we neglect the possible tensorial nature of D and suppress explicit consideration of any dependence on the fixed velocity field $\mathbf{v}$.) The unknown function $u = u(x, y, t)$ represents a solute concentration. Equation (1b) gives the initial concentration field, while equation (1c) imposes Dirichlet boundary conditions. These boundary data are not the only ones to which the ADC method applies; in fact, one could just as well impose Neumann, Robin, or mixed boundary conditions.

We use finite-element collocation to discretize the spatial dimensions in the following class of semidiscrete analogs:

$$u^{n+1} - u^n + k[\mathbf{v} \cdot \nabla u^{n+\theta} - \nabla \cdot (D\nabla u^{n+\theta})] = 0, \quad n = 0, 1, 2, \ldots, \qquad (2)$$

where integer superscripts indicate time level. The notation $(\cdot)^{n+\theta}$ signifies a convex combination $\theta(\cdot)^{n+1} + (1-\theta)(\cdot)^n$ of the quantity $(\cdot)$ at successive time levels, where $0 \leqslant \theta \leqslant 1$, and k denotes the time step. In particular, the choice $\theta = 1/2$ yields a Crank-Nicolson scheme, for which we expect the local truncation error to be $\mathcal{O}(k^2)$.

Accepted September 1989. Discussion closes December 1990.

We begin by establishing a rectangular grid on Ω and a corresponding space of finite-element interpolating functions. Let $\Delta_x = \{a = x_0, \ldots, x_{N_x} = b\}$ and $\Delta_y = \{c = y_0, \ldots, y_{N_y} = d\}$ be grids on the x- and y-intervals (a, b) and (c, d), respectively, and call $h_x = \max_{1 \leqslant i \leqslant N_x}\{x_i - x_{i-1}\}$ and $h_y = \max_{1 \leqslant j \leqslant N_y}\{y_j - y_{j-1}\}$. The Hermite piecewise cubics on these one-dimensional grids are functions belonging to the spaces

$$\mathcal{M}_1^3(\Delta_x) = \{f \in C^1([a, b]) \mid f|_{[x_{i-1}, x_i]} \text{ is cubic, } i = 1, \ldots, N_x\},$$

$$\mathcal{M}_1^3(\Delta_y) = \{f \in C^1([c, d]) \mid f|_{[y_{j-1}, y_j]} \text{ is cubic, } j = 1, \ldots, N_y\},$$

Here $f|_{[x_{i-1}, x_i]}$ denotes the restriction of the globally defined function f to the subinterval $[x_{i-1}, x_i]$. Thus each function in either of these spaces agrees with some cubic polynomial on any subinterval in the grid, and these cubic 'pieces' connect in a manner that preserves global continuous differentiability. As Prenter[12] shows, each of these spaces has an interpolating basis $\{h_{0i}, h_{1i}\}_{i=0}^{N_x \text{ or } N_y}$, every element of which has support confined to at most two adjacent subintervals $[x_{i-1}, x_i]$ or $[y_{j-1}, y_j]$. Given any function $f \in \mathcal{M}_1^3(\Delta_x)$, for example, the representation of f with respect to this basis takes the form

$$f(x) = \sum_{i=0}^{N_x} [f(x_i)h_{0i}(x) + f'(x_i)h_{1i}(x)].$$

For the two-dimensional problem (1), we use these interpolating spaces to form a tensor-product interpolating space $\mathcal{M}_1^3(\Delta_x) \otimes \mathcal{M}_1^3(\Delta_y)$. This space has a basis in which each function is the product of a piecewise cubic basis function in $\mathcal{M}_1^3(\Delta_x)$ and one in $\mathcal{M}_1^3(\Delta_y)$. At each time level n, we compute an approximate solution $\hat{u}^n(x, y)$ belonging to the trial space

$$\mathcal{M} = \{v \in \mathcal{M}_1^3(\Delta_x) \otimes \mathcal{M}_1^3(\Delta_y) \mid v(x, y) = u_B(x, y)\ \forall\, (x, y) \in \partial\Omega\}.$$

As the notation indicates, each function in $\mathcal{M}$ automatically obeys the boundary conditions (1c) and has the form

$$\hat{u}^n(x, y) = \sum_{i=0}^{N_x}\sum_{j=0}^{N_y} [\hat{u}^n(x_i, y_j)H_{00ij}(x, y) + \partial_x\hat{u}^n(x_i, y_j)H_{10ij}(x, y) + \partial_y\hat{u}^n(x_i, y_j)H_{01ij}(x, y) + \partial_{xy}\hat{u}^n(x_i, y_j)H_{11ij}(x, y)],$$

where $H_{lmij}(x, y) = h_{li}(x)h_{mj}(y)$.

At $t = 0$ we form the initial approximate solution $\hat{u}^0$ by using the nodal values of the initial function u_I and its x-, y-, and xy-derivatives to form the projection of the true initial concentration onto $\mathcal{M}$. These criteria specify $\hat{u}^0$ completely. For subsequent time levels, the fact that every function in the trial space $\mathcal{M}$ satisfies the boundary conditions fixes the nodal values and tangential derivatives of the approximate solute concentration along the boundary $\partial\Omega$. A careful count will reveal that the boundary conditions determine $4(N_x + N_y + 1)$ of the $4(N_x + 1)(N_y + 1)$ nodal coefficients for each unknown function $\hat{u}^1$, $\hat{u}^2, \ldots$.

At each new time level $n + 1$, we use our knowledge of the most recently computed approximate solution $\hat{u}^n$ to determine the remaining $4N_xN_y$ degrees of freedom for $\hat{u}^{n+1}$. We first form the residual

$$R^{n+1} = \hat{u}^{n+1} - \hat{u}^n + k[\mathbf{v} \cdot \nabla\hat{u}^{n+\theta} - \nabla \cdot (D\nabla\hat{u}^{n+\theta})].$$

We then pick a collection $\{(\bar{x}_1, \bar{y}_1), (\bar{x}_1, \bar{y}_2), \ldots, (\bar{x}_{2N_x}, \bar{y}_{2N_y})\}$ of $4N_xN_y$ collocation points and force $R^{n+1}(\bar{x}_p, \bar{y}_q) = 0$ at each, thus enforcing precisely the correct number of conditions to determine $\hat{u}^{n+1}$. In particular, we choose $\bar{x}_p$ and $\bar{y}_q$ to be the two-point Gauss-quadrature abscissae on each subinterval $[x_{i-1}, x_i]$ or $[y_{j-1}, y_j]$. Since the spatial problem to be solved at each time level is elliptic we expect this choice of collocation points to yield optimal global error estimates of the form $\|u^n - \hat{u}^n\|_\infty = \mathcal{O}(h_x^4 + h_y^4)$ (see Refs 13 and 14).

3. THE ALTERNATING-DIRECTION METHOD

The aim of ADC is to modify the ordinary two-dimensional collocation procedure via an operator splitting. This splitting reduces the discrete problem to one involving a sequence of matrix equations, each of which has the same sparse structure as the one-dimensional collocation system. The following description of this splitting approach is essentially a review of the development presented by Celia and Pinder in Ref. 10.

We first perturb equation (2) by a term that is $\mathcal{O}(k^2)$ to get

$$u^{n+1} - u^n + k(\mathcal{L}_x + \mathcal{L}_y)u^{n+\theta} + k^2\theta^2(\mathcal{L}_x\mathcal{L}_y)(u^{n+1} - u^n) = 0, \quad (3)$$

where

$$\mathcal{L}_x = v_x\partial_x - \partial_x(D\partial_x); \qquad \mathcal{L}_y = v_y\partial_y - \partial_y(D\partial_y).$$

(Reference 10 treats the advection-diffusion equation in a slightly different fashion, splitting only the diffusive part of the spatial operator.) Rearranging equation (3) and factoring gives

$$(1 + k\theta\mathcal{L}_y)(1 + k\theta\mathcal{L}_x)(u^{n+1} - u^n) = -k(\mathcal{L}_x + \mathcal{L}_y)u^n.$$

Conceptually, we can solve $(1 + k\theta\mathcal{L}_y)z = -k(\mathcal{L}_x + \mathcal{L}_y)u^n$ for the intermediate unknown z, then solve $(1 + k\theta\mathcal{L}_x)(u^{n+1} - u^n) = z$ for the time increment in $\hat{u}$.

To see how this works algebraically, notice that substituting Hermite bicubic trial functions for $\hat{u}$ and collocating produces a matrix equation $\mathbf{K}\mathbf{u}^{n+1} = \mathbf{r}^n$, where $\mathbf{u}^{n+1}$ is the vector of time increments for the unknown nodal coefficients of $\hat{u}^{n+1}$. Consider a typical entry of the matrix $\mathbf{K}$:

$$\{[1 + k\theta(\mathcal{L}_x + \mathcal{L}_y) + k^2\theta^2(\mathcal{L}_x\mathcal{L}_y)]H_{lmij}\}(\bar{x}_p, \bar{y}_q), \quad (4)$$

where H_{lmij} is some basis function in the tensor-product interpolation space. Each $H_{lmij}(x, y) = h_{li}(x)h_{mj}(y)$, so we can expand the expression (4) and factor it to get

$$[h_{li}(\bar{x}_p) + k\theta(\mathcal{L}_xh_{li})(\bar{x}_p)] \cdot [h_{mj}(\bar{y}_q) + k\theta(\mathcal{L}_yh_{mj})(\bar{y}_q)].$$

This factoring of each matrix entry, together with Celia's scheme[9] for numbering and renumbering equations and unknowns, allows us to factor the entire matrix equation at each time level in a computationally attractive fashion. If we number the equations and unknowns 'vertically,' that is, consecutively along the lines $x = \bar{x}_p$, as shown in Fig. 1a, then the

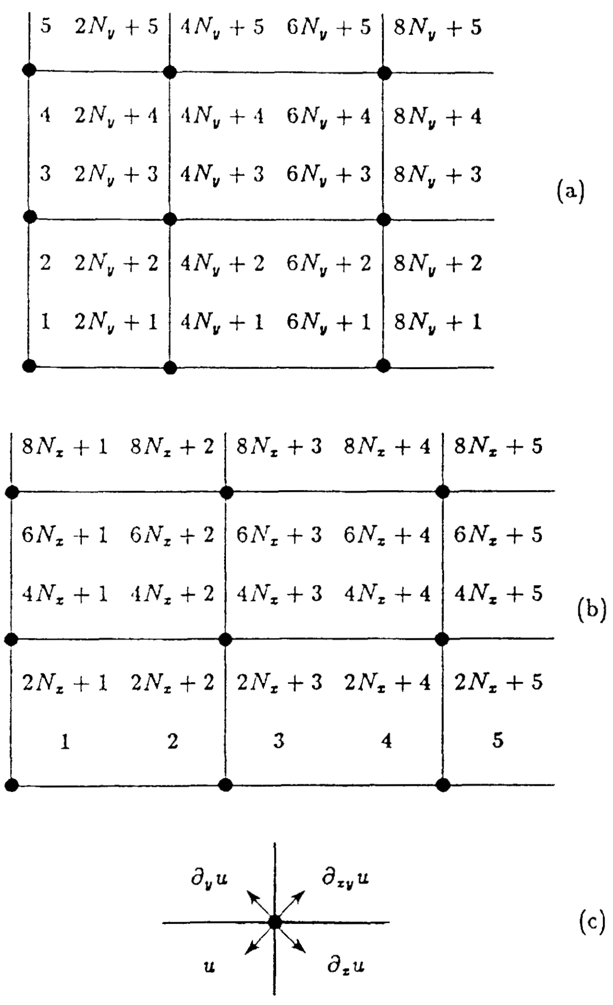
5 2Ny+5 4Ny+5 6Ny+5 8Ny+5
4 2Ny+4 4Ny+4 6Ny+4 8Ny+4
3 2Ny+3 4Ny+3 6Ny+3 8Ny+3
2 2Ny+2 4Ny+2 6Ny+2 8Ny+2
1 2Ny+1 4Ny+1 6Ny+1 8Ny+1
(a)
8Nx+1 8Nx+2 8Nx+3 8Nx+4 8Nx+5
6Nx+1 6Nx+2 6Nx+3 6Nx+4 6Nx+5
4Nx+1 4Nx+2 4Nx+3 4Nx+4 4Nx+5
2Nx+1 2Nx+2 2Nx+3 2Nx+4 2Nx+5
1 2 3 4 5
(b)
∂yu ∂xyu u ∂xu
(c)

Fig. 1. (a) Vertical numbering scheme for the equations used in the y-sweep. Equation numbers occupy the sites of corresponding collocation points; the symbols • indicate nodes in the grid. (b) Horizontal numbering scheme for the equations used in the x-sweep. (c) Association scheme for numbering nodal unknowns following a given numbering scheme for the collocation points surrounding the node.

$4N_xN_y \times 4N_xN_y$ matrix **K** factors as follows:

$$\mathbf{K} = \mathbf{YX} = \begin{bmatrix} \mathbf{Y}_{1,1} & & \\ & \ddots & \\ & & \mathbf{Y}_{2N_x,2N_x} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{1,1} & \cdots & \mathbf{X}_{1,2N_y} \\ \vdots & & \vdots \\ \mathbf{X}_{2N_x,1} & \cdots & \mathbf{X}_{2N_x,2N_y} \end{bmatrix}.$$

Each $2N_y \times 2N_y$ block $\mathbf{Y}_{p,p}$ has the five-band structure of a one-dimensional collocation matrix, shown in Fig. 2. Moreover, The entries in $\mathbf{Y}_{p,p}$ depend only on the y-coordinates of collocation points.

Now consider the matrix **X**. If we switch to the 'horizontal' numbering scheme for equations and unknowns, illustrated in Fig. 1b, then **X** transforms to a block-diagonal matrix that we denote as follows:

$$\mathbf{X}^* = \begin{bmatrix} \mathbf{X}^*_{1,1} & & \\ & \ddots & \\ & & \mathbf{X}^*_{2N_y,2N_y} \end{bmatrix}$$

(We use the superscript * to indicate the result of switching to the 'horizontal' numbering scheme.) Again, each $2N_x \times 2N_x$ block $\mathbf{X}^*_{q,q}$ has the five-band structure shown in Fig. 2.

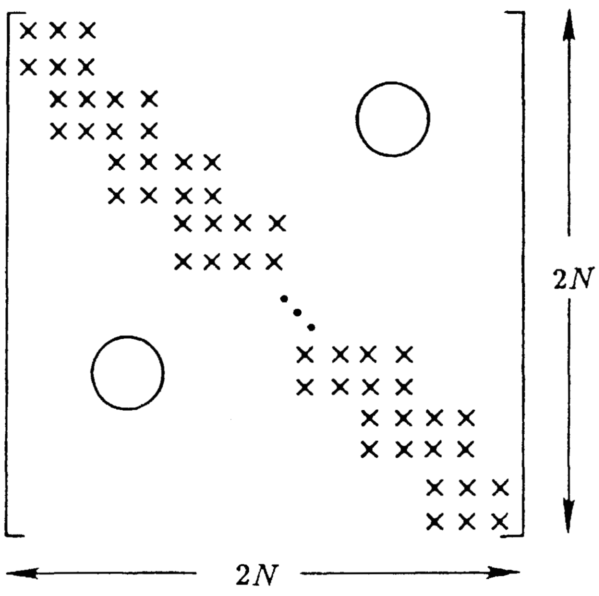

Fig. 2. Five-band zero structure associated with the matrix for standard one-dimensional collocation

In light of these observations, we can solve the two-dimensional matrix equation $\mathbf{Ku}^{n+1} = \mathbf{r}^n$ by the following procedure.

1. Adopt the 'vertical' numbering scheme, and solve $\mathbf{Yz} = \mathbf{r}^n$ for the intermediate vector **z** by solving the independent problems $\mathbf{Y}_{p,p}\mathbf{z}_p = \mathbf{r}^n_p$, $p = 1, \ldots, 2N_x$.
2. Renumber according to the 'horizontal' scheme, converting **z** to the reordered vector $\mathbf{z}^*$. This renumbering transforms **X** to the block-diagonal form $\mathbf{X}^*$.
3. Solve $\mathbf{X}^*\mathbf{u}^{n+1} = \mathbf{z}^*$ for the desired time increments by solving the independent systems $\mathbf{X}^*_{q,q}\mathbf{u}^{n+1}_q = \mathbf{z}^*_q$, $q = 1, \ldots, 2N_y$.

Thus each time step involves the solution of matrix equations that are at worst one-dimensional in structure.

At this point we can make some comments regarding the efficiency to be gained by the splitting scheme. For simplicity, let us assume that $N_x = N_y = N$. In the fully two-dimensional matrix problem $\mathbf{Ku}^{n+1} = \mathbf{r}^n$, there are then $4N^2$ unknowns, and the matrix **K** is asymmetric. If we order equations and unknowns to allow for row reduction without pivoting, **K** will have a bandwidth $B_2 = 8N + 16$ (see Ref. 15). Assuming that row reduction accounts for the bulk of the computational work in the sparse matrix solver used, we can expect the operation count for solving the fully two-dimensional equations at each time step to be roughly $4N^2B_2^2 = 256N^4$ for large N. By contrast, ADC calls for the solution of $4N$ matrix equations of bandwidth $B_1 = 5$ and order $2N$ at each time level. Thus an upper bound for the number of arithmetic operations required in the row reductions for ADC is $4N(2NB_1^2) = 200N^2$.

Furthermore, each of the 'one-dimensional' systems in steps 1 and 3 of ADC is independent of any other.

Therefore these steps can run concurrently, whereas there appears to be no such obvious parallelism in the standard solvers for the fully two-dimensional formulation. We explore the inherent parallelism of ADC in the next section.

4. IMPLEMENTATION ON A PARALLEL COMPUTER

We have implemented ADC on an Alliant FX/8 parallel processing computer. The Alliant has eight processors in a shared-memory configuration in which each processor is a vector-architecture machine. The Alliant allows users to control concurrency within a standard Fortran code through the use of compiler derectives. Since we are mainly interested in the general advantages to be gained through the shared-memory architecture and the concurrency controls furnished by the compiler, we shall not consider such other machine-specific features as size of the cache (high-speed memory), number of processors, or speed of the random-access memory.

The following is a description of the code outlined in Steps 1–3 of section 3. The compiler directives themselves begin with the flag CVD$ starting in the first column of code.

```
         Initialize $\hat{u}^0$, set $n = 0$
         Begin time level $n + 1$
CVD$L      CNCALL (Compiler directive to permit the concurrent execution of the following loop
                   containing a reference to an external procedure.)
           DO for each $p = 1, \ldots, 2N_x$
             CALL YSWEEP (Constructs the system $\mathbf{Y}_{p,p}\mathbf{z}_p = \mathbf{r}_p^n$, solves it, and
                          saves the results.)
           END DO
           CALL RENUM (Reorders $\mathbf{z}$ to get $\mathbf{z}^*$)
CVD$L      CNCALL
           DO for each $q = 1, \ldots, 2N_y$
             CALL XSWEEP (Constructs the system $\mathbf{X}_{q,q}^*\mathbf{u}_q^{n+1} = \mathbf{z}_q^*$, solves it and
                          updates the nodal coefficients of $\hat{u}$ to time level $n + 1$.)
           END DO
         End time step
           ⋮
CVD$R NOCONCUR (Directive to supress concurrency
                until the end of the subroutine.)
         SUBROUTINE YSWEEP
CVD$R NOCONCUR
         SUBROUTINE XSWEEP
```

The directive CNCALL forces the passes through a DO loop to execute in parallel, within the limitations of the machine's configuration. Thus, for example, if the algorithm calls for eight passes through the loop and there are eight processors, then CNCALL forces the operating system to map each pass onto a separate processor, allowing concurrent execution of the passes. If, on the same machine, the algortithm calls for nine passes through the loop, then the last pass must wait until one of the first eight terminates before the operating system can map the ninth onto a free processor. This logic implies that certain efficiencies accrue when the number of independent processes is an integer multiple of the number of processors in the machine being used.

The need for the directive NOCONCUR arises from the structure of the Alliant's optimizing compiler, which often must choose among several levels of parallelism in a code. By default, the compiler optimizes for parallelism at the finest level. Thus, for example, it will force independent processes *within* a subroutine to run concurrently, in preference to forcing independent calls to the subroutine itself to run concurrently. Inserting the directive NOCONCUR before the SUBROUTINE statement overrides the default level for optimization. This device allows the compiler to treat each call to the subroutine as an independent process, even if the potential for concurrency exists at a finer level inside the subroutine.

One measure of how well the algorithm makes use of the machine's parallel capabilities is the *speedup*. Speedup for n processors is the ratio of the CPU time needed by one processor to the time used by n processors to perform a set of tasks in parallel. For a perfectly parallel algorithm requiring no overhead to monitor or schedule the various processes and no storage of their results, the speedup for n processors would be n. Figure 3 shows the speedup curve for the ADC algorithm, implemented for the advection-diffusion problem on a 40×40-element spatial grid. The CPU time used to compute these ratios is actual clock time, excluding the processing required for initializing the code but including computational overhead required for scheduling and storage of intermediate results. The speedup curve is quite close to the ideal curve of unit slope, yielding a speedup of 7.27 for eight processors. Clearly, ADC makes very good use of the Alliant's shared-memory parallel architecture.

We caution against extrapolating these speedup results to much larger problems on the Alliant as configured. The size of the cache memory in any particular

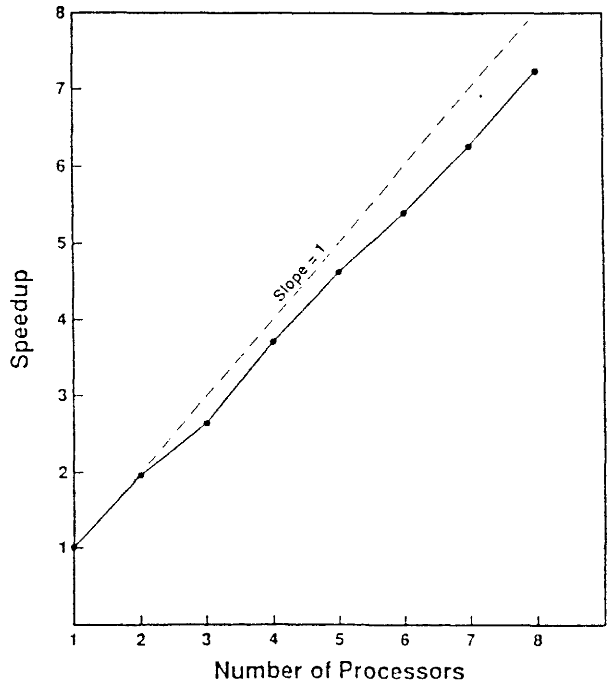

Fig. 3. Speedup curve for ADC using the Alliant FX/8 shared-memory architecture

computer clearly constrains that machine's ability to compute efficiently. What is important here is not the computational horsepower of the particular machine we have used but rather the natural parallelism inherent in the ADC algorithm. This parallelism can yield significant speedups on essentially any shared-memory parallel architecture.

To confirm that ADC gives useful approximations, Figs 4 and 5 show solution plots for two different problems. Figure 4 shows the results of a rotating plume problem on $\Omega = (-1, 1) \times (-1, 1)$, with $N_x = N_y = 40$ and $k = 0.004$. Here, $\mathbf{v} = 2\pi(-y, x)$ is a circular velocity field, $D = 0$, and the initial concentration plume $u_I(x, y)$ is a 'Gauss hill' with center at $(0, -0.6)$ and standard deviation $\sigma = 0.066$. This pure advection problem, while physically unrealistic, poses a fairly severe test of ADC's ability to approximate solutions with steep fronts in highly advective flow fields. In this case, the global error at $t = 1$, when the exact solution is identical to the initial condition, is less than $0.08 \| u \|_\infty$.

Figure 5 displays the results of an advection-diffusion

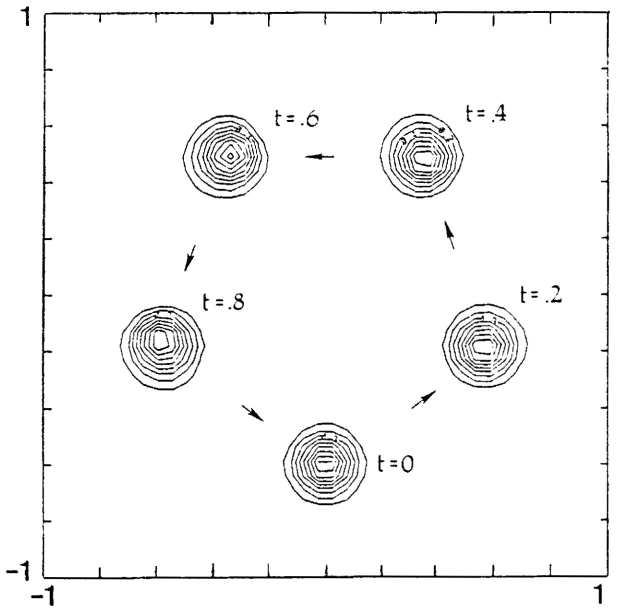

Fig. 4. Concentration contours for the purely advective rotating plume problem at various time levels. Contour interval is 0.1

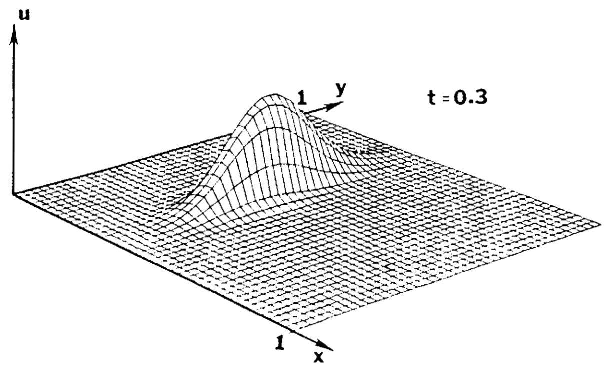

Fig. 5. Plot of concentration distribution at $t = 0.3$ for an advection-diffusion problem with Darcy flow

problem on $\Omega = (0, 1) \times (0, 1)$, with $N_x = N_y = 20$ and $k = 0.004$. The diffusion coefficient here is $D = 0.00385$. The velocity field is $\mathbf{v}(x, y) = 2e^{xy}(x, -y)$, which corresponds to the steady-state Darcy velocity $-K\nabla\Phi$ on Ω when the hydraulic conductivity is $K(x, y) = e^{xy}$ and the head obeys the boundary conditions $\Phi(x, y) = x^2 - y^2$ on $\partial\Omega$. The inital concentration distribution u_I for this problem is another 'Gauss hill,' with $\sigma = 0.05$ and center $(0.75, 0.25)$.

5. CONCLUSIONS

From operation counts alone, it has been clear for some time that ADC offers distinct efficiencies over standard methods for two-dimensional collocation in a serial computing environment. With the advent of practical parallel computers, ADC holds even more promise, since the splitting scheme converts a fully two-dimensional problem into a sequence of 'one-dimensional' problems that are amenable to concurrent processing. Similar observations should hold for other alternating-direction methods, including techniques for multidimensional finite-difference and Galerkin aproximations.

ACKNOWLEDGMENTS

The Wyoming Water Research Center supported this work. We also received support from NSF grant RII-8610680, EPA cooperative agreement CR813928-01-0, and ONR contract 0014-88-K-0370.

REFERENCES

1 Peaceman, D. W. and Rachford, H. H. The numerical solution of parabolic and elliptic equations, *SIAM J.* 1955, **3**, 28–41

2 Douglas, Jr., J. and Dupont, T. Alternating-direction Galerkin methods on rectangles, *Numerical Solution of Partial Differential Equations, Vol. 2,*' B. Hubbard, ed., Academic, New York, 1971, 133–214

3 Dendy, J. and Fairweather, G. Alternating-direction Galerkin schemes for parabolic and hyperbolic problems on rectangular polygons, *SIAM J. Numer. Anal.* 1975, **2**, 144–163

4 Hayes, L. J. and Krishnamachari, S. V. Alternating direction along flow lines in a fluid flow problem, *Comp. Meth. App. Mech. and Engg* 1989, **47**, 187–203

5 Bangia, V. K., Bennett, C. and Reynolds, A. Alternating direction collocation for simulating reservoir performance, 53rd annual fall conference, Society of Petroleum Engineers, Houston, 1978

6 Chang, P. W. and Finlayson, B. A. Orthogonal collocation on finite elements for elliptic equations, *Math. Comp. Simulation*, 1978, 83–92

7 Hayes, L. J. An alternating-direction collocation method for finite element approximations on rectangles, *Comput. Math. Appl.*, 1980, **6**, 45–50

8 Celia, M. A., Pinder G. F. and Hayes, L. J. Alternating direction collocation simulation of the transport equation, *Proceedings Third Int. Conf. Finite Elements in Water Resources*, S. Y. Wang *et al.*, eds., University of Mississippi, Oxford, MS, 1980, 3.36–3.48

9 Celia, M. A., *Collocation on deformed finite elements and alternating direction collocation methods*, Ph.D. Dissertation, Princeton University, 1983

10 Celia, M. A. and Pinder, G. F. Analysis of alternating-direction methods for parabolic equations, *Numer. Meth. P.D.E.* 1985, **1**, 57-70

11 Lapidus, L. and Pinder, G. F. *Numerical solution of partial differential equations in science and engineering*, New York, 1982

12 Prenter, P. M. *Splines and variational methods*, New York, 1975

13 Percell, P. and Wheeler, M. F., A C^1 finite element collocation method for elliptic problems, *SIAM J. Numer. Anal. 17* 1980, 605–622

14 Dyksen, W. R., Lynch, R. E., Rice, J. R. and Houstis, E. N. The performance of the collocation and Galerkin methods with hermit bicubics, *SIAM J. Numer. Anal.*, 1984, **21**, 675–715

15 Frind, E. O. and Pinder, G. F. A collocation finite element method for potential problems in irregular domains, *Int. J. Numer. Meth. Engg* 1979, **14**, 681–701