

OBJECT-ORIENTED PARALLEL ALGORITHMS FOR COMPUTING  
THREE-DIMENSIONAL ISOPYCNAL FLOW

Paul Concus\*, Gene H. Golub†, and Yong Sun‡

SUMMARY

In this paper, we derive an object-oriented parallel algorithm for three-dimensional isopycnal flow simulations. The matrix formulation is central to the algorithm. It enables us to apply an efficient preconditioned conjugate gradient linear solver for the global system of equations, and leads naturally to an object-oriented data structure design and parallel implementation. We discuss as well, in less detail, a similar algorithm based on the reduced system, suitable also for parallel computation. Favorable performances are observed on test problems.

KEY WORDS: isopycnal flow; preconditioned conjugate gradients; parallel computation;  
object-oriented simulation

SHORT TITLE: 3-D ISOPYCNAL FLOW SIMULATION

1. INTRODUCTION

Effective semi-implicit finite difference methods have been developed for the simulation of shallow water equations in two space dimensions [1] and for stably stratified isopycnal free surface flow in three space dimensions [2]. These methods find application to such problems as computing circulation in lakes, estuaries, and coastal seas. As with implicit methods generally, there is an associated computational challenge—at each time step, a linear algebraic system needs to be solved. For problems arising in practical applications, the linear systems can be very large, millions of equations for millions of unknowns, and

---

\*Lawrence Berkeley National Laboratory and Department of Mathematics, University of California, Berkeley, CA 94720, USA; concus@math.berkeley.edu

†SCCM, Stanford University, Stanford, CA 94305, USA; golub@sccm.stanford.edu

‡SCCM, Stanford University, Stanford, CA 94305, USA; ysun@sccm.stanford.edu

their solution consumes most of the computer time needed for simulating the fluid flow. Accordingly, parallel computing can be attractive. Our approach is built on exploiting the sparsity and structure of the finite difference coefficient matrices. It employs a preconditioned conjugate gradient method, as do approaches of V. Casulli in [1] and [2]. Indeed, it was V. Casulli's remark to us of the need for developing conjugate gradient preconditioners to speed up solution of the linear systems in [2] that led us to the present study.

In Section 2 the three-dimensional isopycnal flow model and its semi-implicit finite difference discretization are described. In Section 3 we give a matrix formulation of the problem, and in Section 4 we propose two solution methods, based on similar underlying principles. The first method is suitable for two-dimensional shallow water flow, or three-dimensional isopycnal flow with a small number of vertical layers; the second method is appropriate for three-dimensional isopycnal flow with a large number of vertical layers. We discuss in greater detail the latter case, as it is computationally the more demanding. In Section 5 we discuss for this case an object-oriented parallel algorithm and in Section 6 present computational results on test problems. Conclusions and remarks are given in Section 7.

## 2. THREE-DIMENSIONAL ISOPYCNAL FREE SURFACE FLOW

### *2.1. Physical model*

The layered isopycnal model, as presented in [2], consists of a finite number of moving layers of ideal fluid, stacked vertically, each layer having uniform density (Figure 1). The layers are assumed to be in hydrostatic equilibrium (less dense layers above more dense ones), and the separation surfaces between layers are assumed to be expressible as single-valued functions of height. Under this model, the governing equations can be expressed in terms of layer density  $\rho$  rather than vertical  $z$ -coordinate, resulting in substantial simplification.

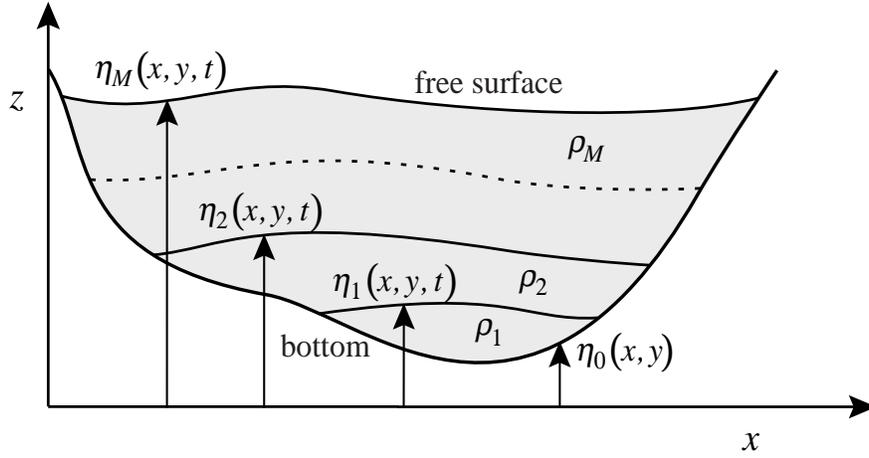


Figure 1. Isopycnal Configuration

For a system of  $M$  layers with densities  $\rho_1 > \rho_2 > \dots > \rho_M > 0$ , denote the separation surface between layers  $k$  and  $k + 1$  by  $z = \eta_k(x, y, t)$ , where  $z$  is the vertical coordinate,  $x, y$  the horizontal coordinates, and  $t$  the time variable. Let  $u_k(x, y, t)$  and  $v_k(x, y, t)$  denote the velocity components in the  $x$  and  $y$  directions, respectively, in layer  $k$  (these velocity components are assumed to be independent of  $z$  within each layer). Then, the equations for layer  $k$  are [2]

$$\frac{\partial u_k}{\partial t} + u_k \frac{\partial u_k}{\partial x} + v_k \frac{\partial u_k}{\partial y} = -g \frac{\partial}{\partial x} \left( \sum_{m=k}^M \frac{\rho_m - \rho_{m+1}}{\rho_0} \eta_m \right) + \nu^h \left( \frac{\partial^2 u_k}{\partial x^2} + \frac{\partial^2 u_k}{\partial y^2} \right) + \frac{\tau_{k+1/2}^x - \tau_{k-1/2}^x}{\eta_k - \eta_{k-1}}, \quad (1)$$

$$\frac{\partial v_k}{\partial t} + u_k \frac{\partial v_k}{\partial x} + v_k \frac{\partial v_k}{\partial y} = -g \frac{\partial}{\partial y} \left( \sum_{m=k}^M \frac{\rho_m - \rho_{m+1}}{\rho_0} \eta_m \right) + \nu^h \left( \frac{\partial^2 v_k}{\partial x^2} + \frac{\partial^2 v_k}{\partial y^2} \right) + \frac{\tau_{k+1/2}^y - \tau_{k-1/2}^y}{\eta_k - \eta_{k-1}}, \quad (2)$$

$$\frac{\partial \eta_k}{\partial t} + \frac{\partial}{\partial x} \left[ \sum_{m=1}^k (\eta_m - \eta_{m-1}) u_m \right] + \frac{\partial}{\partial y} \left[ \sum_{m=1}^k (\eta_m - \eta_{m-1}) v_m \right] = 0. \quad (3)$$

Here  $\tau_{k+1/2}^x$  and  $\tau_{k+1/2}^y$  denote the shear stress components between layers and are taken to be

$$\tau_{k+1/2}^x = 2\nu_{k+1/2}^V \frac{u_{k+1} - u_k}{\eta_{k+1} - \eta_{k-1}}, \quad \tau_{k+1/2}^y = 2\nu_{k+1/2}^V \frac{v_{k+1} - v_k}{\eta_{k+1} - \eta_{k-1}},$$

where  $\nu^V$  is the vertical eddy viscosity coefficient. The parameter  $\nu^h$  denotes the horizon-

tal eddy viscosity coefficient, and  $g$  is the gravitational acceleration. See [2] for details on boundary conditions and on the various parameters.

## 2.2. Semi-implicit finite difference scheme

By means of a careful differencing in space and time and judicious selection of which variables to evaluate implicitly and which explicitly, a discretization of (1)–(3) is derived in [2] that is stable in time and yields linear systems that are suitable for iterative solution by the conjugate gradient method. The physical domain in the  $x$ - $y$  plane is subdivided into  $N_x N_y$  rectangular cells of uniform length  $\Delta x$  and width  $\Delta y$  in each fluid layer. Each cell is numbered at its center correspondingly with indices  $i, j$ . The discrete  $u_k$  velocities are defined at half integer  $i$  and integer  $j$ ;  $v_k$  are defined at integer  $i$  and half integer  $j$ . The  $\eta_k$  are defined at integer  $i$  and  $j$ . The linear system to be solved for updating variables from time step  $n$  to time step  $n + 1$  is then

$$A_{i+1/2,j}^n U_{i+1/2,j}^{n+1} = G_{i+1/2,j}^n - S_{i+1/2,j}^n R (H_{i+1,j}^{n+1} - H_{i,j}^{n+1}), \quad (4)$$

$$A_{i,j+1/2}^n V_{i,j+1/2}^{n+1} = G_{i,j+1/2}^n - \frac{\Delta x}{\Delta y} S_{i,j+1/2}^n R (H_{i,j+1}^{n+1} - H_{i,j}^{n+1}), \quad (5)$$

$$H_{i,j}^{n+1} = \delta_{i,j}^n - \left[ \left( S_{i+1/2,j}^n \right)^\top U_{i+1/2,j}^{n+1} - \left( S_{i-1/2,j}^n \right)^\top U_{i-1/2,j}^{n+1} \right] - \frac{\Delta x}{\Delta y} \left[ \left( S_{i,j+1/2}^n \right)^\top V_{i,j+1/2}^{n+1} - \left( S_{i,j-1/2}^n \right)^\top V_{i,j-1/2}^{n+1} \right]. \quad (6)$$

Here  $U_{i+1/2,j}^{n+1}$  and  $V_{i,j+1/2}^{n+1}$  denote the vector of unknowns for the  $M$  values of  $u_k$  and  $v_k$  at successive fluid layers below the point on the  $x$ - $y$  plane, at time step  $n + 1$ ; similarly  $H_{i,j}^{n+1}$  denotes the vector of the  $M$  values of  $\eta_k$ . Specifically,

$$U_{i+1/2,j}^{n+1} = \begin{bmatrix} u_M^{n+1} \\ u_{M-1}^{n+1} \\ u_{M-2}^{n+1} \\ \vdots \\ u_1^{n+1} \end{bmatrix}_{i+1/2,j}, \quad V_{i,j+1/2}^{n+1} = \begin{bmatrix} v_M^{n+1} \\ v_{M-1}^{n+1} \\ v_{M-2}^{n+1} \\ \vdots \\ v_1^{n+1} \end{bmatrix}_{i,j+1/2}, \quad H_{i,j}^{n+1} = \begin{bmatrix} \eta_M^{n+1} \\ \eta_{M-1}^{n+1} \\ \eta_{M-2}^{n+1} \\ \vdots \\ \eta_1^{n+1} \end{bmatrix}_{i,j}.$$

In general,  $M$  may have different values at different points  $(i, j)$ .

Let  $\Delta\eta_k = \eta_k - \eta_{k-1}$  and  $\Delta\rho_k = \rho_k - \rho_{k+1}$ . Then the matrices  $A$ ,  $S$ , and  $R$  are given by (indices  $i, j$ , and  $n$  are suppressed)

$$A = \begin{bmatrix} \Delta\eta_M + \frac{\nu_{M-1/2}^V \Delta t}{\Delta\eta_{M-1/2}} + \gamma_T \Delta t & -\frac{\nu_{M-1/2}^V \Delta t}{\Delta\eta_{M-1/2}} & & & 0 \\ -\frac{\nu_{M-1/2}^V \Delta t}{\Delta\eta_{M-1/2}} & \Delta\eta_{M-1} + \frac{\nu_{M-1/2}^V \Delta t}{\Delta\eta_{M-1/2}} + \frac{\nu_{M-3/2}^V \Delta t}{\Delta\eta_{M-3/2}} & -\frac{\nu_{M-3/2}^V \Delta t}{\Delta\eta_{M-3/2}} & & \\ & & \ddots & & -\frac{\nu_{3/2}^V \Delta t}{\Delta\eta_{3/2}} \\ 0 & & & -\frac{\nu_{3/2}^V \Delta t}{\Delta\eta_{3/2}} & \Delta\eta_1 + \frac{\nu_{3/2}^V \Delta t}{\Delta\eta_{3/2}} + \gamma_B \Delta t \end{bmatrix},$$

$$S = \frac{\theta \Delta t}{\Delta x} \text{diag}(\Delta\eta_M, \Delta\eta_{M-1}, \dots, \Delta\eta_1) \begin{bmatrix} 1 & & & 0 \\ 1 & 1 & & \\ \cdot & \cdot & \cdot & \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad R = \frac{g}{\rho_0} \text{diag}(\Delta\rho_M, \Delta\rho_{M-1}, \dots, \Delta\rho_1).$$

Here  $\theta$  is an implicitness parameter,  $\rho_0$  a reference density,  $g$  the acceleration due to gravity,  $\Delta t$  the time step, and  $\Delta\eta_{k+1/2} = (\Delta\eta_k + \Delta\eta_{k+1})/2$ ;  $\gamma_T$  and  $\gamma_B$  are non-negative coefficients arising from the boundary conditions.  $A$  is a symmetric, tridiagonal, positive-definite M-matrix,  $S$  is non-negative lower triangular, and  $R$  is diagonal with positive diagonal elements.  $A$  and  $S$  are time and space varying;  $R$  does not vary with respect to either. The vectors  $G_{i+1/2,j}^n$ ,  $G_{i,j+1/2}^n$ , and  $\delta_{i,j}^n$  contain the explicit terms from discretization of (1)–(3).

The staggered tabular points for  $U$ ,  $V$ , and  $H$  in (4)–(6) are illustrated in Figure 2. Associated with a mesh cell  $(i, j)$  are the three unknown vectors at time  $n + 1$ ,  $U_{i+1/2,j}$ ,  $V_{i,j+1/2}$  and  $H_{i,j}$ , each of dimension  $M$ . From (4)–(6) one sees that each  $U$  or  $V$  value (denoted by triangles in the figure) is determined by its two adjacent  $H$  values (denoted by circles) along the  $x$  or  $y$  direction, respectively; each  $H$  value is determined by its four surrounding  $U$  and  $V$  values. All the values are to be updated at each time step.

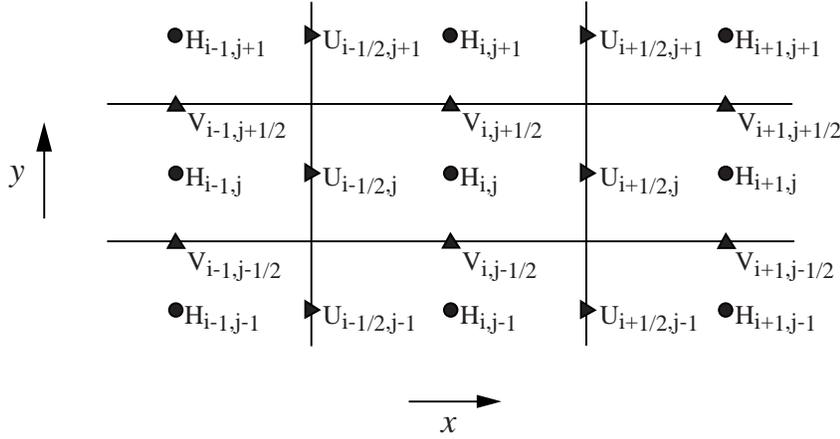


Figure 2. Mesh grid showing staggered tabular points for  $U$ ,  $V$ , and  $H$ .

By eliminating  $U_{i+1/2,j}^{n+1}$  and  $V_{i,j+1/2}^{n+1}$  from (4)–(6) and setting  $E_{i,j}^{n+1} = RH_{i,j}^{n+1}$ , one obtains the reduced equation

$$\begin{aligned}
& \left[ R^{-1} + (S^\top A^{-1} S)_{i-1/2,j}^n + (S^\top A^{-1} S)_{i+1/2,j}^n + \left(\frac{\Delta x}{\Delta y}\right)^2 (S^\top A^{-1} S)_{i,j-1/2}^n \right. \\
& \left. + \left(\frac{\Delta x}{\Delta y}\right)^2 (S^\top A^{-1} S)_{i,j+1/2}^n \right] E_{i,j}^{n+1} - (S^\top A^{-1} S)_{i-1/2,j}^n E_{i-1,j}^{n+1} - (S^\top A^{-1} S)_{i+1/2,j}^n E_{i+1,j}^{n+1} \\
& - (S^\top A^{-1} S)_{i,j-1/2}^n E_{i,j-1}^{n+1} - (S^\top A^{-1} S)_{i,j+1/2}^n E_{i,j+1}^{n+1} = Y_{i,j}^n
\end{aligned} \tag{7}$$

for the (normalized) interface heights  $E_{i,j}^{n+1}$ , where

$$Y_{i,j}^n = \delta_{i,j}^n - (S^\top A^{-1} G)_{i-1/2,j}^n + (S^\top A^{-1} G)_{i+1/2,j}^n - \frac{\Delta x}{\Delta y} (S^\top A^{-1} G)_{i,j-1/2}^n + \frac{\Delta x}{\Delta y} (S^\top A^{-1} G)_{i,j+1/2}^n.$$

In the reduced equation, each  $E = RH$  value is determined by its four neighbors, two in the  $x$ -direction and two in the  $y$ -direction (circular points in Figure 2). This structure is similar to that of the five-point difference scheme for the two-dimensional Poisson equation, except that here the values of  $E$  are vectors rather than scalars.

The reduced equation forms the basis for the conjugate gradient iterative solution method used in [2]. Here we shall consider either the original, unreduced form (4)–(6) or the reduced

form (7) as basis for an iterative method, depending on the number of vertical layers in a problem.

### 3. GLOBAL LINEAR SYSTEM

#### 3.1. Model problem

To simplify the ensuing discussion and notation, we consider a model problem that we shall use as framework for describing the numerical algorithms. In the model problem we take for all vertical layers the same finite difference grid of  $N_x \times N_y$  cells on a rectangular domain and the same value  $M$  for each  $(i, j)$  point, and we suppose that boundary conditions are such that there are exactly a total of  $MN_xN_y$  unknown values for each of  $U_{i+1,j}^{n+1}$ ,  $V_{i,j+1}^{n+1}$ , and  $H_{i,j}^{n+1}$ , with  $1 \leq i \leq N_x$ ,  $1 \leq j \leq N_y$ .

#### 3.2. Matrix formulation

We rewrite (4)–(6) in matrix notation, as the global linear system (8) for the model problem, with dimension  $3MN_xN_y$ . As in (7), we use the normalized surface heights  $E_{i,j}^{n+1} = RH_{i,j}^{n+1}$ :

$$\begin{bmatrix} \hat{\mathbf{A}}_U & \mathbf{0} & -\hat{\mathbf{S}}_U \\ \mathbf{0} & \hat{\mathbf{A}}_V & -\hat{\mathbf{S}}_V \\ \hat{\mathbf{S}}_U^\top & \hat{\mathbf{S}}_V^\top & \hat{\mathbf{R}}^{-1} \end{bmatrix}^{(n)} \begin{bmatrix} \hat{U} \\ \hat{V} \\ \hat{E} \end{bmatrix}^{(n+1)} = \begin{bmatrix} \hat{G}_U \\ \hat{G}_V \\ \hat{\delta} \end{bmatrix}^{(n)}, \quad (8)$$

$$\text{where } \hat{U} = \begin{bmatrix} U_{\frac{3}{2},1} \\ U_{\frac{5}{2},1} \\ \vdots \\ U_{N_x+\frac{1}{2},N_y} \end{bmatrix}, \hat{V} = \begin{bmatrix} V_{1,\frac{3}{2}} \\ V_{2,\frac{3}{2}} \\ \vdots \\ V_{N_x,N_y+\frac{1}{2}} \end{bmatrix}, \hat{E} = \begin{bmatrix} E_{1,1} \\ E_{2,1} \\ \vdots \\ E_{N_x,N_y} \end{bmatrix}, \hat{\mathbf{R}} = \begin{bmatrix} R & & \mathbf{0} \\ & R & \\ & & \ddots \\ \mathbf{0} & & & R \end{bmatrix},$$

$$\hat{\mathbf{A}}_U = \begin{bmatrix} A_{\frac{3}{2},1} & & & \mathbf{0} \\ & A_{\frac{5}{2},1} & & \\ & & \ddots & \\ \mathbf{0} & & & A_{N_x+\frac{1}{2},N_y} \end{bmatrix}, \hat{\mathbf{A}}_V = \begin{bmatrix} A_{1,\frac{3}{2}} & & & \mathbf{0} \\ & A_{2,\frac{3}{2}} & & \\ & & \ddots & \\ \mathbf{0} & & & A_{N_x,N_y+\frac{1}{2}} \end{bmatrix},$$

$$\hat{\mathbf{S}}_U = \begin{bmatrix} S_{\frac{3}{2},1} & -S_{\frac{3}{2},1} & & 0 \\ & S_{\frac{5}{2},1} & & \\ & & \ddots & -S_{N_x-\frac{1}{2},N_y} \\ 0 & & & S_{N_x+\frac{1}{2},N_y} \end{bmatrix}, \quad \hat{\mathbf{S}}_V = \frac{\Delta x}{\Delta y} \begin{bmatrix} S_{1,\frac{3}{2}} & 0 & -S_{1,\frac{3}{2}} & 0 \\ & S_{2,\frac{3}{2}} & \ddots & -S_{N_x,N_y-\frac{1}{2}} \\ & & \ddots & 0 \\ 0 & & & S_{N_x,N_y+\frac{1}{2}} \end{bmatrix}.$$

The right-hand-side quantities are given correspondingly by

$$\hat{G}_U = \begin{bmatrix} G_{\frac{3}{2},1} \\ G_{\frac{5}{2},1} \\ \vdots \\ G_{N_x+\frac{1}{2},N_y} \end{bmatrix}, \quad \hat{G}_V = \begin{bmatrix} G_{1,\frac{3}{2}} \\ G_{2,\frac{3}{2}} \\ \vdots \\ G_{N_x,N_y+\frac{1}{2}} \end{bmatrix}, \quad \hat{\delta} = \begin{bmatrix} \delta_{1,1} \\ \delta_{2,1} \\ \vdots \\ \delta_{N_x,N_y} \end{bmatrix}.$$

The reduced equation (7) can be written in matrix form as well, as a global linear equation (of dimension  $MN_xN_y$ ) for the unknown  $\hat{E}^{n+1}$ . Elimination of  $\hat{U}^{n+1}$  and  $\hat{V}^{n+1}$  from (8) yields

$$\left[ \hat{\mathbf{R}}^{-1} + \hat{\mathbf{S}}_U^\top \hat{\mathbf{A}}_U^{-1} \hat{\mathbf{S}}_U + \hat{\mathbf{S}}_V^\top \hat{\mathbf{A}}_V^{-1} \hat{\mathbf{S}}_V \right]^{(n)} \hat{E}^{n+1} = \hat{Y}^n, \quad (9),$$

where  $\hat{Y}^n = \left[ \hat{\delta} - \hat{\mathbf{S}}_U^\top \hat{\mathbf{A}}_U^{-1} \hat{G}_U - \hat{\mathbf{S}}_V^\top \hat{\mathbf{A}}_V^{-1} \hat{G}_V \right]^{(n)}$ . For the right hand side of (9), the notation  $\hat{Y}^n$  parallels that in (7):  $\hat{Y}^n = [Y_{1,1}^n \ Y_{2,1}^n \ \dots \ Y_{N_x,N_y}^n]^\top$ .

The coefficient matrix of (9) is a symmetric, positive-definite matrix that is  $N_y \times N_y$  block tridiagonal with block size  $N_x \times N_x$ ; the diagonal blocks are themselves block tridiagonal with size  $M \times M$  blocks, and the off-diagonal blocks are block diagonal, also with size  $M \times M$  blocks [2]. As mentioned in Section 2.2, the structure is that of the five-point difference scheme for the two-dimensional Poisson equation (for the particular  $i, j$  ordering used), except that here the values of  $E_{i,j}^{n+1}$  are vectors of length  $M$  rather than scalars, and the coefficients are  $M \times M$  matrices.

We illustrate the structure of (9) for a model problem with just six mesh cells,  $N_x = 3$ ,  $N_y = 2$  and with  $\Delta x = \Delta y$ . This will be helpful in discussing the algorithms in Section 4.

One obtains

$$\begin{bmatrix} T_{1,1} & B_{\frac{3}{2},1} & 0 & B_{1,\frac{3}{2}} & 0 & 0 \\ B_{\frac{3}{2},1} & T_{2,1} & B_{\frac{5}{2},1} & 0 & B_{2,\frac{3}{2}} & 0 \\ 0 & B_{\frac{5}{2},1} & T_{3,1} & 0 & 0 & B_{3,\frac{3}{2}} \\ B_{1,\frac{3}{2}} & 0 & 0 & T_{1,2} & B_{\frac{3}{2},2} & 0 \\ 0 & B_{2,\frac{3}{2}} & 0 & B_{\frac{3}{2},2} & T_{2,2} & B_{\frac{5}{2},2} \\ 0 & 0 & B_{3,\frac{3}{2}} & 0 & B_{\frac{5}{2},2} & T_{3,2} \end{bmatrix}^{(n)} \begin{bmatrix} E_{1,1} \\ E_{2,1} \\ E_{3,1} \\ E_{1,2} \\ E_{2,2} \\ E_{3,2} \end{bmatrix}^{(n+1)} = \begin{bmatrix} Y_{1,1} \\ Y_{2,1} \\ Y_{3,1} \\ Y_{1,2} \\ Y_{2,2} \\ Y_{3,2} \end{bmatrix}^{(n)}, \quad (10)$$

where

$$T_{i,j} = R^{-1} + (S^\top A^{-1} S)_{i-\frac{1}{2},j} + (S^\top A^{-1} S)_{i+\frac{1}{2},j} + (S^\top A^{-1} S)_{i,j-\frac{1}{2}} + (S^\top A^{-1} S)_{i,j+\frac{1}{2}},$$

$$B_{i+\frac{1}{2},j} = -(S^\top A^{-1} S)_{i+\frac{1}{2},j}, \quad B_{i,j+\frac{1}{2}} = -(S^\top A^{-1} S)_{i,j+\frac{1}{2}}.$$

In [2], at each time step the reduced system (9) is formulated from the current solution  $\hat{E}^n$  and solved for  $\hat{E}^{n+1}$  using a conjugate gradient method with diagonal preconditioning. The elements of  $\hat{U}^{n+1}$  and  $\hat{V}^{n+1}$  may be obtained from  $\hat{E}^{n+1}$  using (4) and (5).

## 4. SOLUTION ALGORITHMS

### 4.1. Preconditioned conjugate gradients

The solution algorithms we consider employ the preconditioned conjugate gradient algorithm. However, rather than using the algorithm in its customary form [3, Algorithm 10.3.1], with recursion of two separate vectors—efficient for storage requirements—we use instead the three-term recurrence for the solution vector alone [4]. This latter form is better suited to the matrix structure of the problem for the preconditionings we consider.

**Algorithm 1** (Preconditioned Conjugate Gradient Algorithm)

Let  $\tilde{\mathbf{A}}$  be an  $n \times n$  real symmetric positive-definite matrix, and let  $b$  be a real  $n$ -vector. To solve  $\tilde{\mathbf{A}}x = b$  iteratively with  $n \times n$  preconditioning matrix  $\tilde{\mathbf{P}}$  (also real symmetric positive-definite):

- (i) Let  $x^{(0)}$  be a given vector and define  $x^{(-1)}$  arbitrarily.

(ii) For  $k = 0, 1, \dots$ , until convergence

(a) Solve  $\tilde{\mathbf{P}}z^{(k)} = b - \tilde{\mathbf{A}}x^{(k)}$ .

(b) Compute

$$\alpha_k = \frac{z^{(k)\top} \tilde{\mathbf{P}}z^{(k)}}{z^{(k)\top} \tilde{\mathbf{A}}z^{(k)}},$$

$$\omega_1 = 1, \quad \omega_{k+1} = \left( 1 - \frac{\alpha_k}{\alpha_{k-1}} \cdot \frac{z^{(k)\top} \tilde{\mathbf{P}}z^{(k)}}{z^{(k-1)\top} \tilde{\mathbf{P}}z^{(k-1)}} \cdot \frac{1}{\omega_k} \right)^{-1}, \quad k \geq 1.$$

(c) Compute  $x^{(k+1)} = x^{(k-1)} + \omega_{k+1}(\alpha_k z^{(k)} + x^{(k)} - x^{(k-1)})$ .

We next discuss solution algorithms for our problem for a small and then for a larger number of vertical layers.

#### 4.2. Small number of vertical layers – red-black ordering

If there are only a few vertical layers, then we address solution of the linear systems in terms of the reduced equation. However, instead of employing a conjugate gradient method directly on (9), as in [2], we instead first apply a “red-black ordering” to the mesh values, to take advantage of the two-dimensional-Poisson-like structure. Doing so gives us two benefits: A particular preconditioning for the conjugate gradient method can be applied to the reordered system, which can require only about half of the computational work of the lexicographic ordering in (9); also, this preconditioning is well suited for parallel computation.

Figure 3 illustrates a red-black ordering for the tabular mesh values  $E_{i,j}$ . This figure is essentially Figure 2 with the  $U$  and  $V$  values removed and with the circles denoting the tabular points for  $E$  alternately shaded and open in a checkerboard pattern. As noted in Section 2, from (7) one sees that  $E_{i,j}$  (a “black” point) is determined by the four surrounding (“red”) points,  $E_{i-1,j}$ ,  $E_{i+1,j}$ ,  $E_{i,j+1}$  and  $E_{i,j-1}$ . In a similar way, a red point is determined by its four surrounding black points. Thus one need have the solution only at black points, say; the values at red points can be determined directly from them. In this way, the  $E$  points

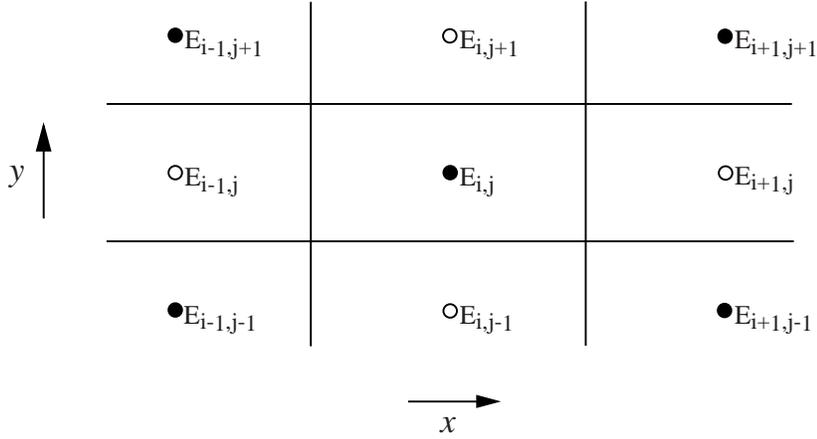


Figure 3. Mesh grid showing red-black designation of tabular points for  $E$ .

are divided into two groups, with the elements of either group independent of others within the group. The unknown vector in the reduced system (9) can be reordered according to the two sets of points such that it consists of two sub-vectors, red and black.

Carrying out the reordering for the six-mesh-cell model problem (10) one obtains

$$\begin{bmatrix} T_{1,1} & 0 & 0 & B_{\frac{3}{2},1} & B_{1,\frac{3}{2}} & 0 \\ 0 & T_{3,1} & 0 & B_{\frac{5}{2},1} & 0 & B_{3,\frac{3}{2}} \\ 0 & 0 & T_{2,2} & B_{2,\frac{3}{2}} & B_{\frac{3}{2},2} & B_{\frac{5}{2},2} \\ B_{\frac{3}{2},1} & B_{\frac{5}{2},1} & B_{2,\frac{3}{2}} & T_{2,1} & 0 & 0 \\ B_{1,\frac{3}{2}} & 0 & B_{\frac{3}{2},2} & 0 & T_{1,2} & 0 \\ 0 & B_{3,\frac{3}{2}} & B_{\frac{5}{2},2} & 0 & 0 & T_{3,2} \end{bmatrix}^{(n)} \begin{bmatrix} E_{1,1} \\ E_{3,1} \\ E_{2,2} \\ E_{2,1} \\ E_{1,2} \\ E_{3,2} \end{bmatrix}^{(n+1)} = \begin{bmatrix} Y_{1,1} \\ Y_{3,1} \\ Y_{2,2} \\ Y_{2,1} \\ Y_{1,2} \\ Y_{3,2} \end{bmatrix}^{(n)}. \quad (11)$$

Eq. (11) has the form

$$\begin{bmatrix} \mathbf{P}_1 & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{P}_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}. \quad (12)$$

$$\text{Here } \mathbf{P}_1 = \begin{bmatrix} T_{1,1} & 0 & 0 \\ 0 & T_{3,1} & 0 \\ 0 & 0 & T_{2,2} \end{bmatrix}, \quad \mathbf{P}_2 = \begin{bmatrix} T_{2,1} & 0 & 0 \\ 0 & T_{1,2} & 0 \\ 0 & 0 & T_{3,2} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} B_{\frac{3}{2},1} & B_{1,\frac{3}{2}} & 0 \\ B_{\frac{5}{2},1} & 0 & B_{3,\frac{3}{2}} \\ B_{2,\frac{3}{2}} & B_{\frac{3}{2},2} & B_{\frac{5}{2},2} \end{bmatrix},$$

$$x_1 = \begin{bmatrix} E_{1,1} \\ E_{3,1} \\ E_{2,2} \end{bmatrix}, \quad x_2 = \begin{bmatrix} E_{2,1} \\ E_{1,2} \\ E_{3,2} \end{bmatrix}, \quad b_1 = \begin{bmatrix} Y_{1,1} \\ Y_{3,1} \\ Y_{2,2} \end{bmatrix}, \quad b_2 = \begin{bmatrix} Y_{2,1} \\ Y_{1,2} \\ Y_{3,2} \end{bmatrix}.$$

A matrix of the form (12) is said to have a block form of Young's *Property A* [5],[6]. Of importance here is that  $P_1$  and  $P_2$  are themselves block diagonal with block size  $M \times M$ ,  $M$  generally being small compared with the dimension  $MN_xN_y$  of the linear system.

It is shown in [4] that particular choices of preconditioner and initial approximation can be advantageous in Algorithm 1 for a coefficient matrix of the form (12); see also [7]. We have:

**Lemma**

In Algorithm 1, let  $\tilde{\mathbf{A}}$ ,  $x$ , and  $b$  have the form indicated in (12), and let the initial vector  $x^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix}$  be such that  $\mathbf{P}_2 x_2^{(0)} = b_2 - \mathbf{F}^\top x_1^{(0)}$ . Then for  $k = 0, 1, \dots$ , there holds  $\alpha_k \equiv 1$ ,  $z_1^{(k)} \equiv 0$  if  $k$  is odd, and  $z_2^{(k)} \equiv 0$  if  $k$  is even, where  $z^{(k)} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \end{bmatrix}$ .

*Proof.* See [4], [7].

The properties, that half of the elements of successive  $z^{(k)}$  are zero and that one of the conjugate gradient parameters  $\alpha_k$  need not be computed, can be exploited to give considerable computational cost savings for Algorithm 1. Based on the Lemma, we give below a special form that Algorithm 1 can take for these matrices [4].

**Algorithm 2** (Preconditioned Conjugate Gradients for Matrices of the Form (12))

For an equation of the form (12) satisfying the hypotheses of Algorithm 1, and with preconditioning matrix  $\tilde{\mathbf{P}} = \begin{bmatrix} \mathbf{P}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_2 \end{bmatrix}$ :

- (i) Let  $x_1^{(0)}$  be a given vector, and solve  $\mathbf{P}_2 x_2^{(0)} = b_2 - \mathbf{F}^\top x_1^{(0)}$  for  $x_2^{(0)}$ . Define  $x^{(-1)} = \begin{bmatrix} x_1^{(-1)} \\ x_2^{(-1)} \end{bmatrix}$  arbitrarily.
- (ii) Take as initial vector  $x^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix}$ , and correspondingly set  $z_2^{(0)} = 0$ .
- (iii) For  $k = 0, 1, \dots$ , until convergence
  - (a) If  $k$  is even

$$\text{Solve } \mathbf{P}_1 z_1^{(k)} = b_1 - \mathbf{P}_1 x_1^{(k)} - \mathbf{F} x_2^{(k)}.$$

Compute

$$\omega_1 = 1, \quad \omega_{k+1} = \left( 1 - \frac{z_1^{(k)\top} \mathbf{P}_1 z_1^{(k)}}{z_2^{(k-1)\top} \mathbf{P}_2 z_2^{(k-1)}} \cdot \frac{1}{\omega_k} \right)^{-1}, \quad k \geq 2.$$

Compute

$$x_1^{(k+1)} = x_1^{(k-1)} + \omega_{k+1}(z_1^{(k)} + x_1^{(k)} - x_1^{(k-1)}),$$

$$x_2^{(k+1)} = x_2^{(k-1)} + \omega_{k+1}(x_2^{(k)} - x_2^{(k-1)}).$$

(b) If  $k$  is odd

$$\text{Solve } \mathbf{P}_2 z_2^{(k)} = b_2 - \mathbf{P}_2 x_2^{(k)} - \mathbf{F}^\top x_1^{(k)}.$$

Compute

$$\omega_{k+1} = \left( 1 - \frac{z_2^{(k)\top} \mathbf{P}_2 z_2^{(k)}}{z_1^{(k-1)\top} \mathbf{P}_1 z_1^{(k-1)}} \cdot \frac{1}{\omega_k} \right)^{-1}.$$

Compute

$$x_1^{(k+1)} = x_1^{(k-1)} + \omega_{k+1}(x_1^{(k)} - x_1^{(k-1)}),$$

$$x_2^{(k+1)} = x_2^{(k-1)} + \omega_{k+1}(z_2^{(k)} + x_2^{(k)} - x_2^{(k-1)}).$$

In each iteration of Algorithm 2, matrices are half the size of those in Algorithm 1. As also there is no need to compute  $\alpha_k$ , Algorithm 2 reduces the computational work by more than half compared with an approach to Algorithm 1 that does not take advantage of the block 2-cyclic matrix structure.

Algorithm 2 not only can significantly reduce the computational work per iteration, but it also has desirable parallel properties. In Algorithm 2, the preconditioning linear systems  $\mathbf{P}_\ell z_\ell^{(k)} = r_\ell^{(k)}$ ,  $\ell = 1, 2$  involve only one of the two colored sets of points, while the values at the other points remain unchanged. Furthermore, as points of the same color are independent of each other and  $P_1, P_2$  are block diagonal, all the corresponding localized linear systems in (7) for our problem can be solved efficiently in parallel. Other computational operations in Algorithm 2, such as matrix-vector multiplication, vector inner product, and vector update, can be carried out in parallel in a similar fashion.

If  $M$  is large, a shortcoming in applying the algorithm to (10) or (11) is that the diagonal blocks  $T_{i,j}$  are dense matrices of size  $M \times M$ . Generating each  $T_{i,j}$  and solving the corresponding linear system requires  $O(M^3)$  operations (and memory requirements  $O(M^2)$ ). Thus the algorithm is attractive primarily if the number of vertical layers in a problem is small; otherwise, alternatives can be more suitable. We present such an alternative next based on the unreduced global linear system (8).

#### 4.3. Large number of vertical layers – unreduced global linear system

We return now to the unreduced linear system (8), on which we base an algorithm for the case of more than just a small number of vertical layers. Although the unreduced system has dimension  $3MN_xN_y$ , as opposed to just  $MN_xN_y$  for the reduced system, it has the computational advantage of all its  $M \times M$  sub-matrices being sparse. Operations that were  $O(M^3)$  for the reduced system can be done for the unreduced system with only linear increase with  $M$  (memory requirements change also from  $O(M^2)$  to  $O(M)$ ).

Specifically, because the diagonal blocks  $A_{i+1/2,j}$  and  $A_{i,j+1/2}$  of  $\hat{\mathbf{A}}_U$  and  $\hat{\mathbf{A}}_V$  in (8) are tridiagonal matrices and  $\mathbf{R}^{-1}$  is a diagonal matrix, the work of generating and solving linear systems involving them increases just linearly with  $M$ . The blocks of the off-diagonal matrices  $\hat{\mathbf{S}}_U$  and  $\hat{\mathbf{S}}_V$  are not sparse, but they are triangular matrices of a special form, the product of a diagonal matrix and a triangular matrix with all nonzero elements unity. Only the diagonal elements of such matrices need be stored, and calculation of the product of the matrices with a vector can be carried out in a manner that grows only linearly with  $M$ : To calculate

$$Sx = \begin{bmatrix} d_1 & & & \mathbf{0} \\ & d_2 & & \\ & & \ddots & \\ \mathbf{0} & & & d_M \end{bmatrix} \begin{bmatrix} 1 & & & \mathbf{0} \\ 1 & 1 & & \\ \cdot & \cdot & \cdot & \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix}$$

set  $t = x_1$ ,  $y_1 = td_1$ , and then for  $i = 2, \dots, M$  calculate  $t = t + x_i$ ,  $y_i = td_i$ .

One sees that the coefficient matrix of the unreduced global linear system (8) is not symmetric, thus Algorithms 1 and 2 can not be applied to it directly. However, the matrix is real positive, i. e., it has positive-definite symmetric part

$$\begin{bmatrix} \hat{\mathbf{A}}_U & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{A}}_V & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \hat{\mathbf{R}}^{-1} \end{bmatrix}.$$

With this symmetric part as preconditioner, a special conjugate gradient algorithm is possible [8]. As the coefficient matrix of (8) has also a block Property A form with sparse diagonal blocks, that feature can be incorporated into the algorithm as well. First, we state from [8]:

**Algorithm 3** (Preconditioned Conjugate Gradient Algorithm for Certain Nonsymmetric Systems)

Let  $\tilde{\mathbf{A}}$  be a nonsymmetric  $n \times n$  real matrix with positive-definite symmetric part  $(\tilde{\mathbf{A}} + \tilde{\mathbf{A}}^\top)/2$ , and  $b$  a real  $n$ -vector. To solve  $\tilde{\mathbf{A}}x = b$  iteratively with preconditioning matrix  $\tilde{\mathbf{P}} = (\tilde{\mathbf{A}} + \tilde{\mathbf{A}}^\top)/2$ :

(i) Let  $x^{(0)}$  be a given vector and define  $x^{(-1)}$  arbitrarily.

(ii) For  $k = 0, 1, \dots$ , until convergence

(a) Solve  $\tilde{\mathbf{P}}z^{(k)} = b - \tilde{\mathbf{A}}x^{(k)}$ .

(b) Compute

$$\omega_1 = 1, \quad \omega_{k+1} = \left( 1 + \frac{z^{(k)\top} \tilde{\mathbf{P}}z^{(k)}}{z^{(k-1)\top} \tilde{\mathbf{P}}z^{(k-1)}} \cdot \frac{1}{\omega_k} \right)^{-1}, \quad k \geq 1.$$

(c) Compute  $x^{(k+1)} = x^{(k-1)} + \omega_{k+1}(z^{(k)} + x^{(k)} - x^{(k-1)})$ .

Note that, as for Algorithm 2, this algorithm requires computation of only one of the conjugate gradient parameters,  $\omega_k$ . The algorithm is attractive when (ii a) is sufficiently simpler to solve than is the original equation, and when also  $\tilde{\mathbf{P}}$  is “near enough” to  $\tilde{\mathbf{A}}$  to be a good preconditioner. If  $\tilde{\mathbf{A}}$  were symmetric, then Algorithm 3 would collapse just to solving the original equation  $\tilde{\mathbf{A}}x = b$  at step  $k = 0$ .

For an equation, such as (8), that has also the structure

$$\begin{bmatrix} \mathbf{P}_1 & -\mathbf{F} \\ \mathbf{F}^\top & \mathbf{P}_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (13)$$

with sparse diagonal blocks, we can take advantage of the structure to obtain the following algorithm, in a manner similar to which Algorithm 2 was obtained from Algorithm 1. For

(8) we have

$$\begin{aligned} \mathbf{P}_1 &= \begin{bmatrix} \hat{\mathbf{A}}_U & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{A}}_V \end{bmatrix}, \quad \mathbf{P}_2 = \mathbf{R}^{-1}, \quad \mathbf{F} = \begin{bmatrix} \hat{\mathbf{S}}_U \\ \hat{\mathbf{S}}_V \end{bmatrix}, \\ x_1 &= \begin{bmatrix} \hat{U} \\ \hat{V} \end{bmatrix}, \quad x_2 = [\hat{E}], \quad b_1 = \begin{bmatrix} \hat{G}_U \\ \hat{G}_V \end{bmatrix}, \quad b_2 = [\hat{\delta}]. \end{aligned}$$

**Algorithm 4** (Preconditioned Conjugate Gradient Algorithm for Certain Nonsymmetric Systems of the Form (13))

For an equation of the form (13) satisfying the hypotheses of Algorithm 3, with  $\mathbf{P}_1$  and  $\mathbf{P}_2$  symmetric and preconditioning matrix  $\tilde{\mathbf{P}} = \begin{bmatrix} \mathbf{P}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_2 \end{bmatrix}$ :

- (i) Let  $x_1^{(0)}$  be a given vector, and solve  $\mathbf{P}_2 x_2^{(0)} = b_2 - \mathbf{F}^\top x_1^{(0)}$  for  $x_2^{(0)}$ . Define  $x^{(-1)} = \begin{bmatrix} x_1^{(-1)} \\ x_2^{(-1)} \end{bmatrix}$  arbitrarily.
- (ii) Take as initial vector  $x^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix}$ , and correspondingly set  $z_2^{(0)} = 0$ .
- (iii) For  $k = 0, 1, \dots$ , until convergence

(a) If  $k$  is even

$$\text{Solve } \mathbf{P}_1 z_1^{(k)} = b_1 - \mathbf{P}_1 x_1^{(k)} + \mathbf{F} x_2^{(k)}.$$

Compute

$$\omega_1 = 1, \quad \omega_{k+1} = \left( 1 + \frac{z_1^{(k)\top} \mathbf{P}_1 z_1^{(k)}}{z_2^{(k-1)\top} \mathbf{P}_2 z_2^{(k-1)}} \cdot \frac{1}{\omega_k} \right)^{-1}, \quad k \geq 2.$$

Compute

$$x_1^{(k+1)} = x_1^{(k-1)} + \omega_{k+1} (z_1^{(k)} + x_1^{(k)} - x_1^{(k-1)}),$$

$$x_2^{(k+1)} = x_2^{(k-1)} + \omega_{k+1} (x_2^{(k)} - x_2^{(k-1)}).$$

(b) If  $k$  is odd

$$\text{Solve } \mathbf{P}_2 z_2^{(k)} = b_2 - \mathbf{P}_2 x_2^{(k)} - \mathbf{F}^\top x_1^{(k)}.$$

Compute

$$\omega_{k+1} = \left( 1 + \frac{z_2^{(k)\top} \mathbf{P}_2 z_2^{(k)}}{z_1^{(k-1)\top} \mathbf{P}_1 z_1^{(k-1)}} \cdot \frac{1}{\omega_k} \right)^{-1}.$$

Compute

$$x_1^{(k+1)} = x_1^{(k-1)} + \omega_{k+1}(x_1^{(k)} - x_1^{(k-1)}),$$

$$x_2^{(k+1)} = x_2^{(k-1)} + \omega_{k+1}(z_2^{(k)} + x_2^{(k)} - x_2^{(k-1)}).$$

For our problem (8), the work required for Algorithm 3 or 4 is linear in terms of  $M$ , as all component operations for the matrix blocks are linear in  $M$ , as discussed above. We focus attention in the following section on an object-oriented parallel implementation of Algorithm 4 for solving (8).

## 5. COMPUTER IMPLEMENTATION

### 5.1. Parallel processing

Algorithm 2 and Algorithm 4 are well suited for parallel processing. We focus here on an implementation for the more complex case of Algorithm 4 and the unreduced system. For the implementation we take vectors  $\hat{U}$  and  $\hat{V}$  in (8) to correspond to “red points” and  $\hat{E}$  to “black points”. As vector components on points of the same color are independent of one another and are determined solely by values at points of the other color, each step of Algorithm 4 can be decomposed into operations on only one set of colored points, while the other set of points remains unchanged. This property is basic to the parallel implementation.

The implementation can be nontrivial, however, for practical problems. The structure of the global system may not be as straightforward as is (8) for the model problem, because, in general, the domain can be irregular with different positions having different numbers of

vertical layers. As a result, matrices  $\hat{\mathbf{S}}_U$  and  $\hat{\mathbf{S}}_V$  would not, for example, have a simple block two-diagonal structure with blocks of the same dimension. Explicit generation of the global linear system could correspondingly be complicated, because of the difficulty in globally tracing vector element values at the various mesh points and in determining matrix element connectivities. A robust procedure would be desirable whereby vectors,  $\hat{U}$ ,  $\hat{V}$ , or  $\hat{E}$ , could communicate locally in a simple manner with neighbors of the opposite color, to generate and to solve independently the corresponding localized linear systems (4), (5), or (6). Object-oriented programming (OOP) provides such a procedure.

### 5.2. Object-oriented programming

In OOP languages like C++, a *class* is a user defined data type that describes a set of *objects* with identical characteristics (*data elements*) and behavior (*functionality*) [9]. Thinking in these terms, we abstract two classes naturally, *VelocityPoint* and *EPoint*, corresponding respectively to the red and black points.

Each object of class *VelocityPoint* represents and solves a localized linear system defined by either (4) or (5). Its data include variables, such as coefficient matrix elements in the localized system, and working variables, such as matrix decompositions for Algorithm 4. Its functionality includes communicating with the two neighboring *EPoints* and generating and solving the localized linear system. Figure 4 shows the major components of the class *VelocityPoint* for Algorithm 4.

In Figure 4,  $e_1$  and  $e_2$  denote the two neighboring *EPoints* for a *VelocityPoint* (a *U*-point or *V*-point).  $P$  denotes the coefficient matrix for the *VelocityPoint*, i.e., the tridiagonal matrix  $A$  in (4) or (5), which is used as preconditioner for solving the localized linear system; bidiagonal matrix  $L$  and diagonal matrix  $D$  are, respectively, the lower triangular and diagonal factors resulting from the Cholesky decomposition of  $P$ . Triangular matrix  $S$  is the

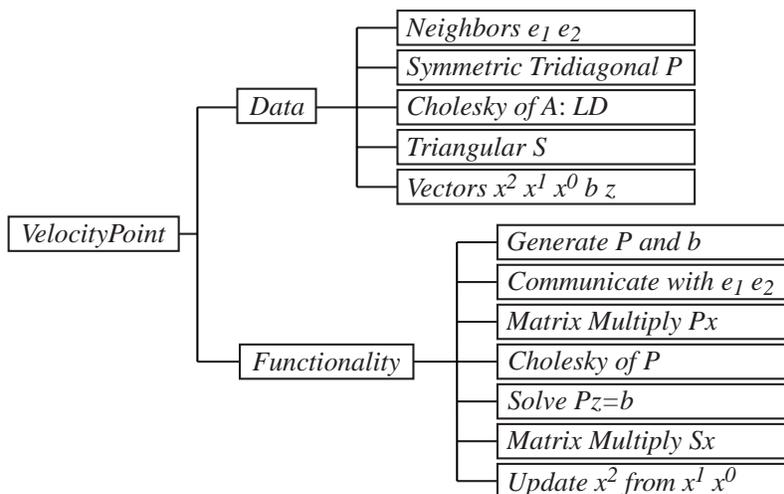


Figure 4. Class VelocityPoint

coefficient matrix in (4) or (5) for the two neighboring EPoints for the current VelocityPoint;  $S$  can be stored as a diagonal matrix, as it is a product of a diagonal matrix and a constant matrix. Vector  $b$  is the right hand side of the localized linear system for the VelocityPoint, i.e., the vector  $G$  in (4) or (5). Vectors  $x^0$ ,  $x^1$ ,  $x^2$  and  $z$  are working vectors, used in the preconditioned conjugate gradient linear solver in Algorithm 4;  $x^0$ ,  $x^1$ ,  $x^2$  arise in the three-term recurrence of the algorithm, and  $z$  is the preconditioned residual vector. Vector  $x$  in the Functionality branch is a generic vector to be multiplied by  $P$  or  $S$ .

In class VelocityPoint, data storage requirements are linear in  $M$ . We apply the Cholesky decomposition at the beginning of the iteration at each time step and use this decomposition for solving the linear systems in iteration steps of Algorithm 4.

Class EPoint represents and solves localized linear systems defined by (6). As shown in Figure 5, EPoint is less complicated than is VelocityPoint, as its localized matrix  $\hat{\mathbf{R}}^{-1}$  is a diagonal constant matrix.

In Figure 5,  $u_1$  and  $u_2$  denote the two neighboring  $U$ -points for an EPoint, and  $v_1$  and

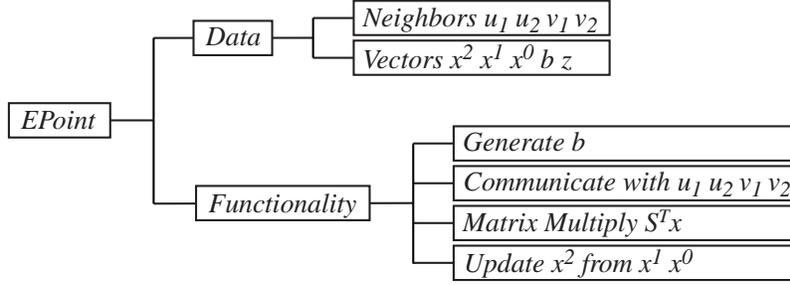


Figure 5. Class EPoint

$v_2$  the two neighboring  $V$ -points. Vector  $b$  is the right hand side of the localized linear system for the EPoint, i.e., the vector  $\delta$  in (6). The other vectors are as described for Figure 4.

In terms of these two classes, VelocityPoint and EPoint, global data structure design and parallel implementation can be readily accomplished. We need only the three object sets  $\hat{U}$ ,  $\hat{V}$ , and  $\hat{E}$  to express the matrices and vectors appearing in (8) and in Algorithm 4. Additionally, we need a matrix array (*Mesh*) for storing information on the geometry and on the mapping between elements of the object sets and points on the mesh. An element in object sets  $\hat{U}$  or  $\hat{V}$  of class VelocityPoint stores the indices of the two neighboring EPoints and communicates with them by directly indexing into array  $\hat{E}$ . Elements in the object set  $\hat{E}$  communicate with their neighboring VelocityPoints in a similar way. Data initialization is a three-step process: first, the Mesh array obtains the geometry information of the problem; second, the mesh points are cycled through to identify their characteristics and to allocate the three vector arrays accordingly, so as to obtain a mapping between mesh points and these arrays; finally, for each element in the three arrays, the neighboring elements of the other class are found.

All global operations, such as matrix-vector multiplication and solution of preconditioned linear systems, can be realized by iterating through elements of the vector arrays and calling their corresponding functionalities. As elements in the same array are independent of one

another, they can be assigned to different processors, which can then work simultaneously in parallel.

### *5.3. Task assignment and load balancing*

In relatively simple problem domains an object takes roughly the same amount of work as others in the same vector array for performing a particular function. A straightforward static-task-assignment scheme for load balancing usually works well in such domains. For our test problem we partition the three vector arrays into essentially equal segments, according to the number of processors, and each processor is then assigned one segment (i.e., the starting and ending indices) from each array.

In more complex domains, the amount of work for a certain function can differ significantly for different objects in the same array, as objects can have different boundary conditions or different numbers of vertical layers. For such domains, each element should be weighted by its computational complexity during the static task assignment, and then some dynamic load balancing scheme employed.

Figure 6 depicts the two types of problem domains. Figure 6a shows an idealized simplification for our numerical test problem of the complex domain in Figure 6b, the surface contour of San Francisco/San Pablo Bay. The domain bottom for the test problem, with depths in the interior varying gradually by about a factor of two or more, is a simplified version of that of the Bay. Correspondingly, there may be differing numbers of vertical layers for different elements.

For our test problem, once all processors are assigned a segment from each of the three arrays, the processors can work on most of the global operations in parallel. Global synchronization (e.g., a barrier or a lock) is required for sequential operations. As an example, Algorithm 5 is a (C++)-like parallel implementation for computing the global quantity

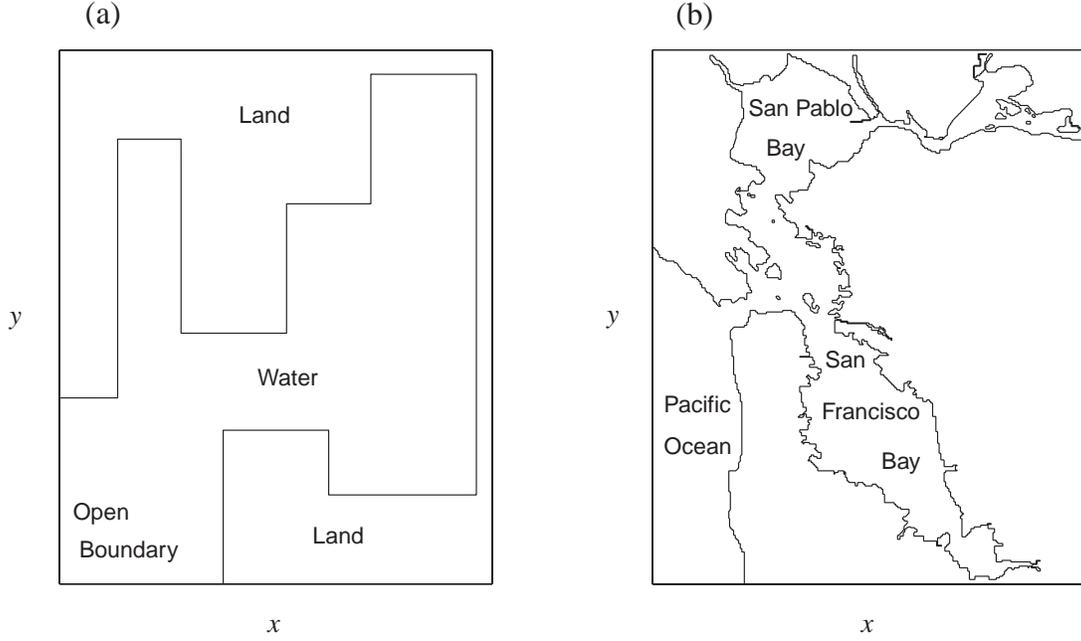


Figure 6. Problem domains: (a) test problem, (b) San Francisco & San Pablo Bays

$z_1^{(k)\top} \mathbf{P}_1 z_1^{(k)}$  in step (iii a) of Algorithm 4. Parallel implementations for other global operations in Algorithm 4 are similar. Algorithms like Algorithm 5 are valid for both shared and distributed memory architectures.

**Algorithm 5** (Parallel Implementation for Computing  $z_1^{(k)\top} \mathbf{P}_1 z_1^{(k)}$ )

```

if my_ID == 0,
    g_zPz = 0;

barrier( );

l_zPz = 0;
for i = l_start_u; i <= l_end_u; i++;
    l_zPz += U[i].zPz( );
for i = l_start_v; i <= l_end_v; i++;
    l_zPz += V[i].zPz( );

lock( );
g_zPz += l_zPz;
unlock( );

```

Note: Global variables start with g-; local variables start with l-.

## 6. COMPUTATIONAL RESULTS

The domain in Figure 6a was used to study the performance of our algorithm. Although a task assignment scheme is easier to implement for the test problem than it is for the actual San Francisco Bay, the geometry is sufficiently complex to justify use of an object-oriented data structure. The (water) domain is discretized on a  $1088 \times 1344$  two-dimensional mesh, and physical parameters are taken to accord with those for San Francisco Bay communicated to us by V. Casulli. Grid size is  $\Delta x = \Delta y = 60$  (meters) and water depth in the interior is generally between about 15 and 40. Boundary conditions are that at the land normal velocities are zero and at the open boundary the layer interfaces are subject to prescribed vertical motion. Generally, initial conditions are that the fluid is quiescent and that the layer interfaces are horizontal and equally spaced; other initial conditions, including random perturbations to these, were tried for investigating behavior of the algorithm. The behavior reported below persisted for these changes without significant deviation.

There are three major goals in the design of our algorithm for the unreduced global system: that it have good convergence rate; that its computational complexity grow linearly with the number of fluid layers (not cubically as in the case of Algorithm 2 for the reduced system (9)); and that it have good parallel performance. As discussed in Section 4.3 and as observed in our numerical experiments, the amount of work for initialization of each localized linear system is proportional to its number of vertical layers. Additionally, generation of the global system requires very little synchronization, so that the total computational cost of initialization at each time step is an order of magnitude less than that of solving the system. Thus in assessing below the behavior of the algorithm, we can focus only on the solving of the linear systems, and neglect initialization considerations.

### 6.1. Linear solver convergence

As we take the diagonal blocks of the coefficient matrix as preconditioner for Algorithm 4, the degree of block diagonal dominance of the matrix can be important in determining the convergence rate of the algorithm. In the numerical experiments for (8) we found that the factor  $\Delta t/\Delta x$  in  $S$  (see the definition of  $S$  following (6)) is the principal quantity on which the linear solver convergence rate depends. The magnitudes of the off-diagonal blocks in the global matrix are proportional to this factor.

Figure 7 shows the convergence rate achieved by the linear solver for typical parameter values. Here the tabular points are the relative residuals  $\|b - Ax\|_2/\|b\|_2$  at even-numbered iterations. The rate is found to be quite satisfactory. For example, for 9 fluid layers (and  $\Delta x = 60$ ) the number of unknowns in the global matrix is approximately  $1.83 \times 10^6$ ; for  $\Delta t = 10$  (secs.) only 36 iterations are required for the solver to decrease the two-norm of the residual by a factor of about  $10^{-9}$ . On a 20-processor 200 MHz Sun Enterprise 10000 workstation with 10 Gbytes of memory, this requires about 62 seconds for a single processor.

The data for Figure 7 were taken after about a dozen time steps and for the previous time step's solution as initial approximation for the conjugate gradient iteration. Essentially the same qualitative convergence results could be observed at other time steps, for different initial approximations for the conjugate gradient iteration, and for the odd-numbered iterates. One phenomenon of interest that we observed in the experiments is that the convergence rate of the linear solver appeared to be insensitive to the number of fluid layers. For convergence, about the same number of iterations were needed over the tested range of three to nine layers. Figure 7 shows the similarity between the three-layer and nine-layer cases. For the problem's particular matrix structure, for the test parameters considered, and for the preconditioner used the convergence rate was found to be insensitive to block size.

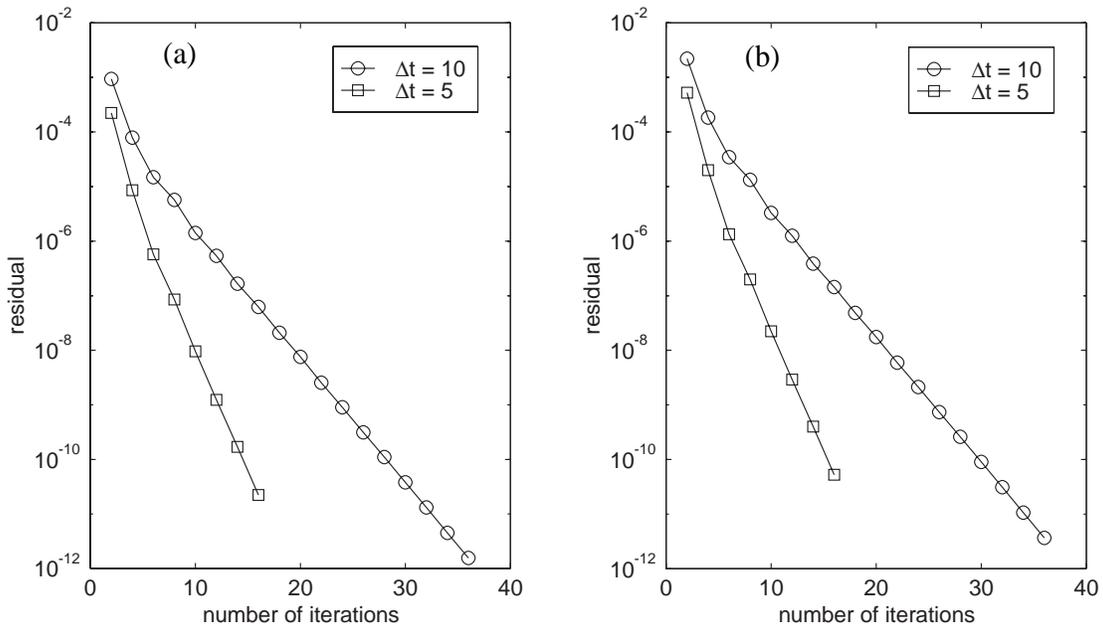


Figure 7. Linear solver convergence: (a) for 3 layers ( $6.11 \times 10^5$  unknowns), (b) for 9 layers ( $1.83 \times 10^6$  unknowns)

### 6.2. Computational cost versus number of vertical layers

To study the dependence of the algorithm’s computational cost on the number of vertical layers, we varied their number (for fixed total fluid depth) and compared the corresponding CPU times consumed by the linear solver. Theoretically the total computational cost of the solver should grow linearly with the average number of layers. This follows from the properties: (i) that the number of unknowns grows linearly with the number of vertical layers; (ii) that, as noted above, the number of iterations required for convergence is observed to be insensitive to the number of layers (i.e., of matrix block size); and (iii) that, as discussed in Section 4.3, the computational cost for each block grows linearly with its block size. Our numerical experiments are in accord with the conclusion of linear growth. Figure 8 shows the relationship between the average number of vertical layers and the CPU time consumed by the linear solver.

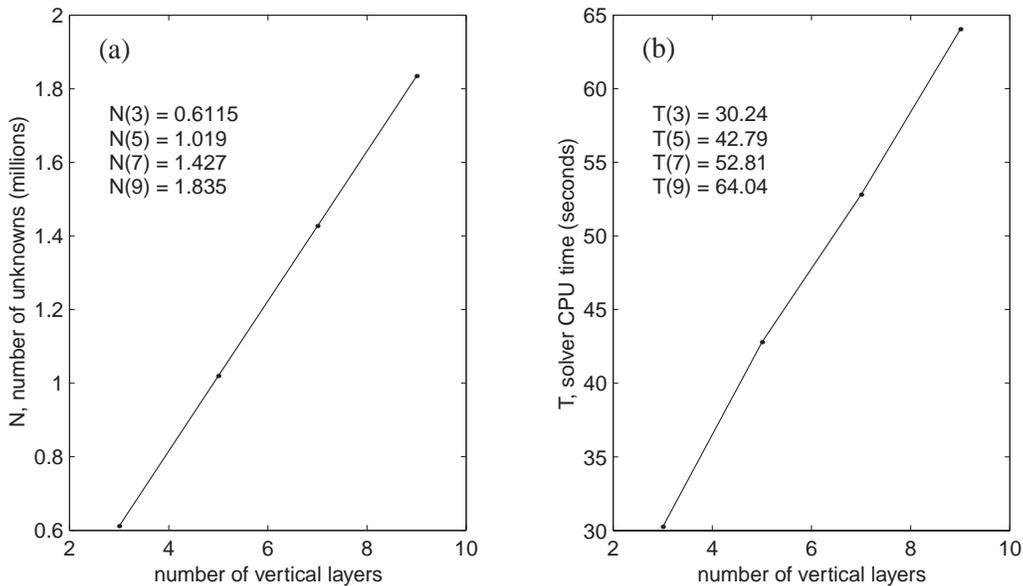


Figure 8. Effect of number of vertical layers: (a) on number of unknowns, (b) on CPU time

### 6.3. Parallel performance

Our parallel implementation is built on multi-threading technology on a shared memory machine. Each processor has its own computational thread for working on the tasks assigned to it by the static-task-assignment scheme, and communication is done implicitly through access to shared data. Synchronization is implemented with barriers and locks.

Figure 9 depicts the parallel performance of our software on the test problem of Figure 6a with nine vertical layers (the resulting number of unknowns is approximately  $1.83 \times 10^6$ ). Figure 9a gives the CPU time used by the linear solver with different numbers of processors. The dotted curve is the theoretical time reduction  $T(n) = T(1)/n$  for ideal parallel codes, where  $T$  is the CPU time and  $n$  is the number of processors. The solid curve is our experimental result. Figure 9b shows the ideal speedup curve (dotted curve,  $S(n) = n$ ) and our experimental one (solid curve). One sees that the code scales well as the number of

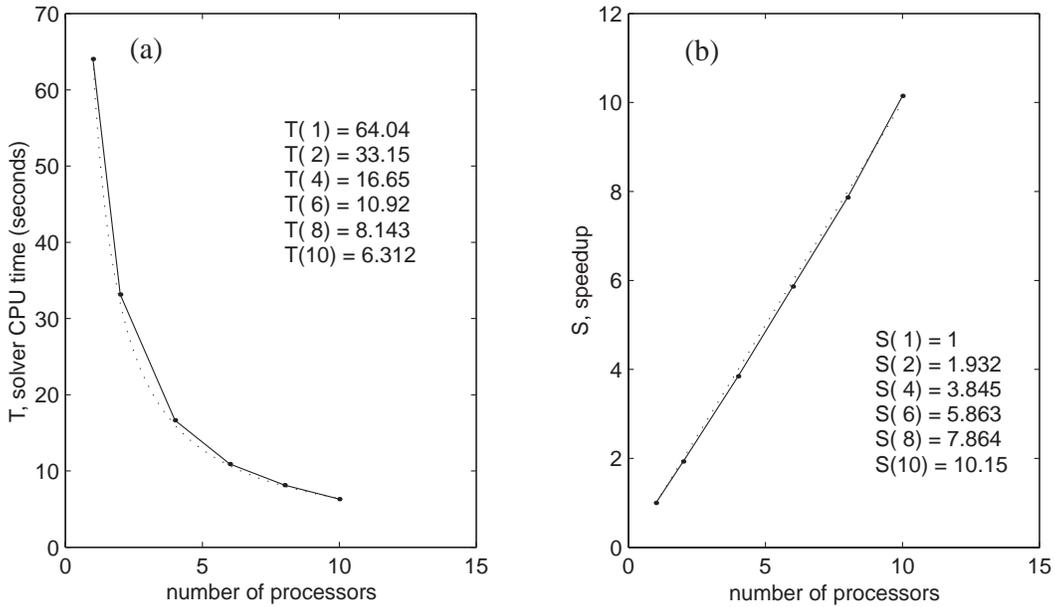


Figure 9. Parallel performance: (a) CPU time reduction, (b) speedup

processors is increased from one to ten.

## 7. CONCLUSIONS AND REMARKS

We have developed algorithms for solving the simulation equations for three-dimensional isopycnal flows with small or large number of vertical layers. The global matrices have special structure, and our linear solvers take advantage of the structure to obtain reduced computational requirements and to achieve beneficial behavior for parallel computation. For the case of a large number of vertical layers, specific object-oriented data structure designs are described. These avoid the use of an explicit matrix framework and lead to a robust parallel implementation. By working with the unreduced global system (8) instead of the reduced one (9), we allow the computational costs to grow only linearly with the number of vertical layers, rather than cubically. Favorable performances are observed for convergence rate, computational cost growth rate, and parallel speedup.

## ACKNOWLEDGMENTS

We wish to thank Vincenzo Casulli for suggesting this problem to us, for his many helpful communications, and for providing us with the use of his computer software. Part of this work was carried out with computing facilities supported by the Office of Science of the U. S. Department of Energy under Contracts DE-AC03-76SF00098 and DE-AC03-76SF00515, which we gratefully acknowledge.

## REFERENCES

1. Casulli V. Semi-implicit finite difference methods for the two-dimensional shallow water equations. *J. Comput. Phys.* 1990; **86**: 56–74.
2. Casulli V. Numerical solution of three-dimensional free surface flow in isopycnal coordinates. *Int. J. Numer. Meth. Fluids* 1997; **25**: 645–658.
3. Golub GH, van Loan CF. *Matrix Computations* (3rd edn). The Johns Hopkins University Press, Baltimore, MD, 1996.
4. Concus P, Golub GH, O’Leary DP. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In *Sparse Matrix Computations*, Bunch JR, Rose DJ (eds). Academic Press: New York, 1976; 309–332. Reprinted in *Studies in Numerical Analysis*, Golub GH (ed). MAA Studies in Mathematics, Vol. 24, Math. Assn. of America, 1984; 178–198.
5. Young DM. Iterative methods for solving partial differential equations of elliptic type. *Trans. Amer. Math. Soc.* 1954; **76**: 92–111.
6. Arms RJ, Gates LD, Zondek B. A method for block iteration. *J. SIAM* 1956; **4**: 220–229.
7. Reid JK, The use of conjugate gradients for systems of linear equations possessing “Property A”. *SIAM J. Numer. Anal.* 1972; **9**: 325–332.
8. Concus P, Golub GH. A generalized conjugate gradient method for nonsymmetric systems of linear equations. In *Computing Methods in Applied Sciences and Engineering*, Glowinski R, Lions RL (eds). Lecture Notes in Economics and Mathematical Systems, Vol. 134, Springer-Verlag: Berlin-Heidelberg-New York, 1976; 56–65.
9. Eckel B. *Thinking in C++*. Prentice Hall: Englewood Cliffs, NJ, 1995.