Mathematical Models and Methods in Applied Sciences
© World Scientific Publishing Company

SPECIAL SOLUTION STRATEGIES INSIDE A SPECTRAL ELEMENT OCEAN MODEL

CRAIG C. DOUGLAS
University of Kentucky, Department of Computer Science,
325 McVey Hall-CCS, Lexington, KY 40506-0045, USA. douglas@ccs.uky.edu.
Also, Yale University, Department of Computer Science,
P.O. Box 208285, New Haven, CT 06520-8285, USA. douglas-craig@cs.yale.edu.

GUNDOLF HAASE
Johannes Kepler University of Linz, Institute for Computational Mathematics,
Altenberger Strasse 69, A–4040 Linz, Austria. gundolf.haase@numa.uni-linz.ac.at.

MOHAMED ISKANDARANI
University of Miami, Rosenstiel School of Marine and Atmospheric Science,
4600 Rickenbacker Causeway, Miami, FL 33149-1098, USA. MIskandarani@rsmas.miami.edu.

STEFAN REITZINGER
Johannes Kepler University of Linz, Spezialforschungsbereich SFB F013,
Freistädter Strasse 313, A–4040 Linz, Austria. reitz@sfb013.uni-linz.ac.at.

We present three ideas how to accelerate the filtering process used in the multilayered Spectral Element Ocean Model (SEOM). We define and analyze a Schur complement preconditioner, a lumping of small entries and an algebraic multigrid (AMG) algorithm. and a algebraic multigrid with patch smoothing algorithm. Finally, we analyze the impact of variations of the Schur complement and AMG methods on memory and computer time.

Keywords: *h-p* finite elements, algebraic multigrid, preconditioning.

1. Introduction

The shallow water equations are good approximations to the equations of fluid motion whenever the fluid's density is homogeneous, and its depth is much smaller than a characteristic horizontal distance.

The shallow water equations can be written in the vector form:

$$\vec{u}_t + g\nabla\zeta = \vec{F} \tag{1.1}$$

$$\zeta_t + \nabla\cdot[(h+\zeta)\vec{u}] = Q \tag{1.2}$$

where $\vec{u} = (u,v)$ is the velocity vector, ζ is the sea surface displacement (which, because of hydrostaticity, also stands for the pressure), g is the gravitational acceleration, h is the resting depth of the fluid, Q is a mass source/sink term. Finally, $\vec{F} = (f^x, f^y)$ is a generalized forcing term for the momentum equations that includes the Coriolis force, non-linear advection, viscous dissipation, and wind forcing. Appropriate boundary conditions must be provided to complete the system. For simplicity, we assume no-slip boundary conditions.

These equations are often used to model the circulation in coastal areas and in shallow bodies of water. Their virtue is that they reduce the complicated set of the 3D equations to 2D, but are still capable of representing a large part of the dynamics. The shallow water equations also arise frequently in the solution of the 3D primitive hydrostatic equations if the top surface of the fluid is free to move. The presence of the free surface allows the propagation of gravity waves at the speed of $\sqrt{gh}$. The gravity wave speed can greatly exceed the advective velocity of the fluid in the deep part of the ocean and results in a very restrictive CFL limit in order to maintain stability in 3D ocean models.[12,13]

In order to mitigate the cost of ocean simulations, modelers often split the dynamics into a barotropic depth-integrated part (external mode), and a baroclinic part (internal modes). The barotropic equations are akin to the shallow water equations and govern the evolution of the gravity waves. The gravity wave speed stability restriction on the baroclinic part is removed and the 3D baroclinic equations can be integrated explicitly using a large time step. The barotropic equations on the other hand are either integrated explicitly using small time steps that respect the CFL condition for the gravity waves, or implicitly using a stable time-differencing scheme[13].

Implicit integration results in a system of equations that has to be solved at each time step. A direct solver can perform adequately only if the number of unknowns is small. For large problems, however, memory limitations preclude the use of direct solvers and iterative solvers are required. The choice of a robust and efficient solver will determine the cost effectiveness of any implicit method.

We use the multilayered isopycnal Spectral Element Ocean Model[12,13] (SEOM). An isopycnal is a constant density surface. In this case the normal to the back of a wave is the isopycnal surface that interests us.

The novel feature of SEOM is the combination of isopycnal coordinates in the vertical and spectral element discretization in the horizontal. The benefits of the spectral element discretization include: geometric flexibility, dual *h*-*p* paths to convergence, low numerical dispersion and dissipation errors, and dense computational kernels leading to extremely good parallel scalability.

Isopycnal models, often referred to as layered models, divide the water column into constant density layers. This division is physically motivated by the fact that most oceanic currents flow along isopycnal surfaces. Mathematically, it amounts to a mapping from the physical vertical coordinate z to a density coordinate system. The rational for a layered model include: ease of development (since it can be achieved by vertically stacking a set of shallow water models), minimization of cross-isopycnal diffusion, elimination of pressure gradient errors, representation of baroclinic processes, and cost savings over a fully three-dimensional formulation. Processes amenable to investigation with the layered model include the wind-driven circulation, eddy generation, and (in part) flow/topography interaction.

In Sec. 2, we define a filtering process that occurs between the layers. In Sec. 3, we define and analyze a Schur complement preconditioner. In Sec. 4, we define and analyze a lumping of small entries and an algebraic multigrid (AMG) algorithm. In Sec. 5, we define and analyze a algebraic multigrid with patch smoothing algorithm. In Sec. 6, we draw some conclusions based on the memory impact of the algorithms and computer times for an example.

2. Filtering

Ocean modeling with spectral element filtering[14] approximates the fully coupled 3D model using N^ℓ (e.g., 5) coupled layers of 2D models in different water depths. Fig. 4 contains a realization of how the layers appear near a coastal region.

Time discretization uses an implicit scheme and the 2D space discretization uses the same mesh τ_h with quadrangle finite elements $\delta^{(r)}$ for each layer. N^v-th (e.g., $7-th$) order test functions are used for the velocity in x-directions (u) and y-direction (v) in each element. ($N^p = N^v - 2$)-th (e.g., $5-th$) order test functions are used for the layer thickness (ξ).

We will concentrate on a specific example using $N^\ell = 5$, $N^v = 7$, and $N^p = 5$. Fig. 1 shows typical reference spectral elements. Fig. 2 contains the surface grid of a particular example for studying wind driven circulation in the North Atlantic Basin (formally known as the NAB08 dataset). The grid is relatively small and has coarse resolution: it has 792 elements, 39610 velocity nodes, and 20372 pressure nodes. Fig. 3 combines features from Figs. 1 and 2. There is an *h*-*p* finite element grid, but inside of each element is the spectral element grid, too.

We follow a particular filtering technique[14]. Starting with u and v, a vortex and divergence calculation takes place that produces a vector $\widetilde{f}$. It turns out that we have to solve the Laplace equation after the filtering twice on each layer for each time step, i.e., for velocity in x-direction:

$$-\Delta\widetilde{u} \;=\; \widetilde{f} \qquad \text{in } \Omega \qquad + \;\text{B.C. on } \Gamma = \partial\Omega. \tag{2.3}$$

The boundary conditions usually consist of a mixture of homogeneous Dirichlet B.C. (rigid wall) and homogeneous Neumann B.C. (free-slip boundary). The equation for the velocity in y-direction is similar with a different right hand side.

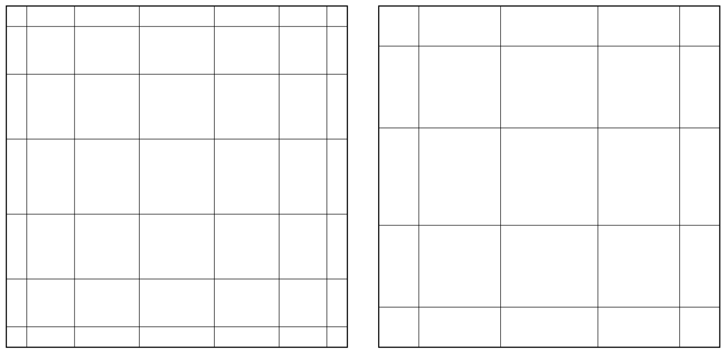
Fig. 1. Gauss-Lobatto points for the velocity (left) and pressure (right) unknowns in a typical spectral element.

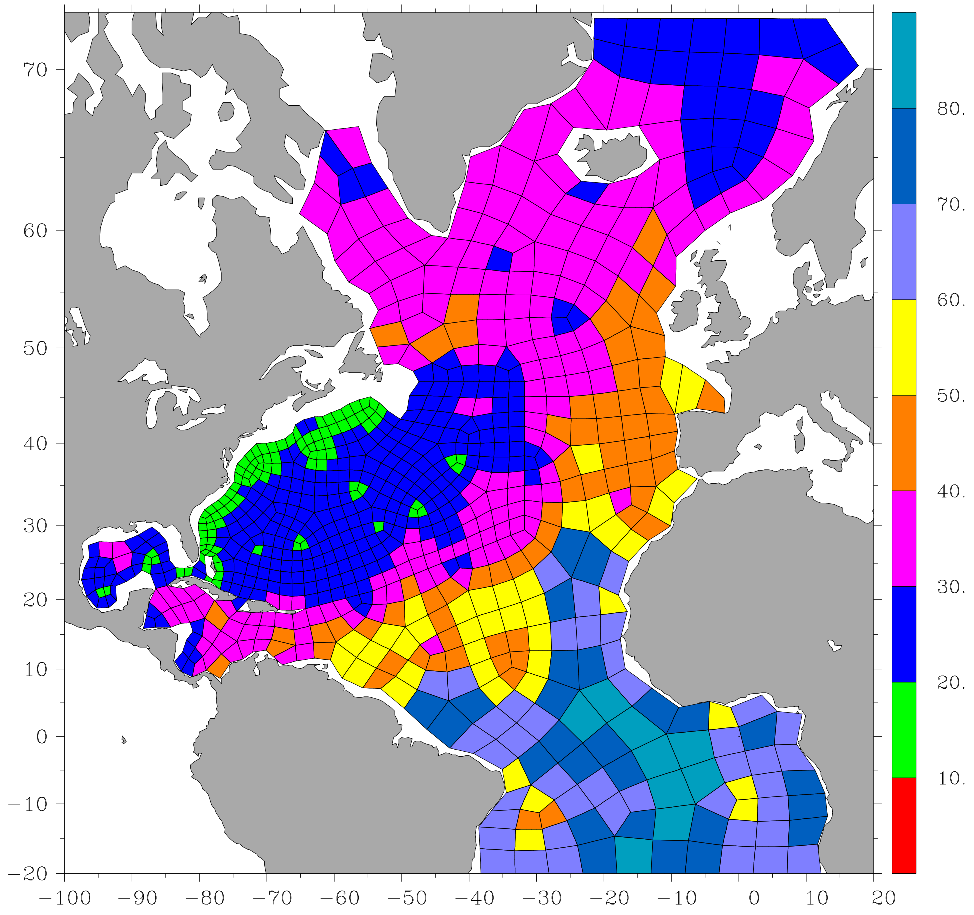

Fig. 2. North Atlantic grid (NAB08) with surface mesh spacing (km) color coded.

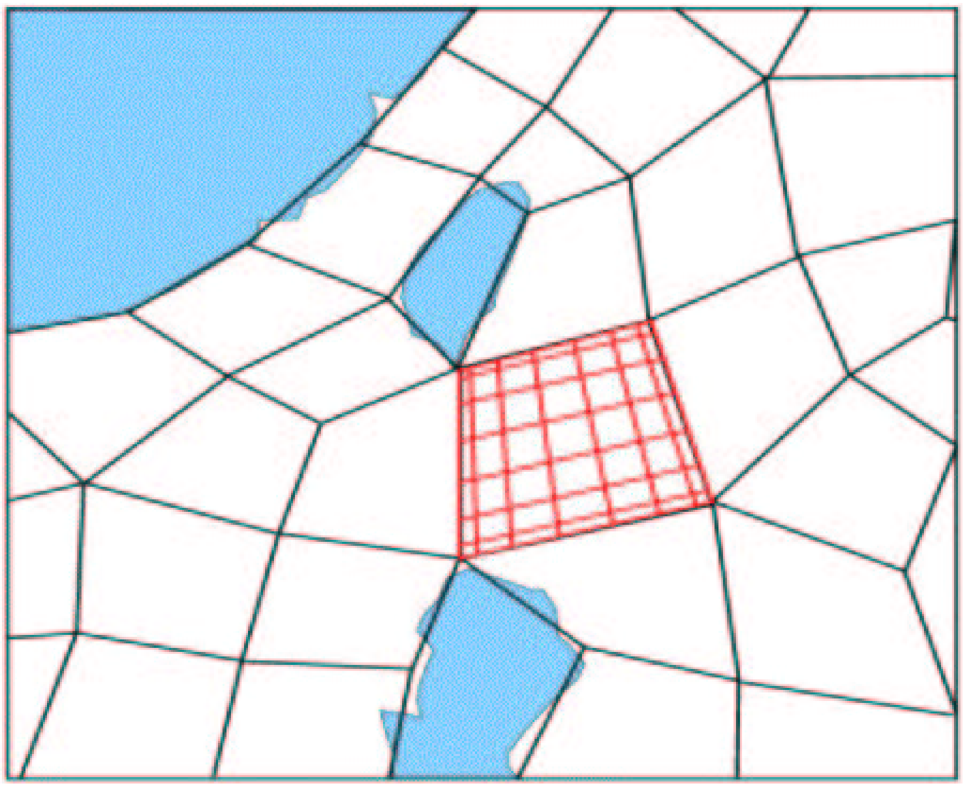

Fig. 3. Actual subgrid inside an element.

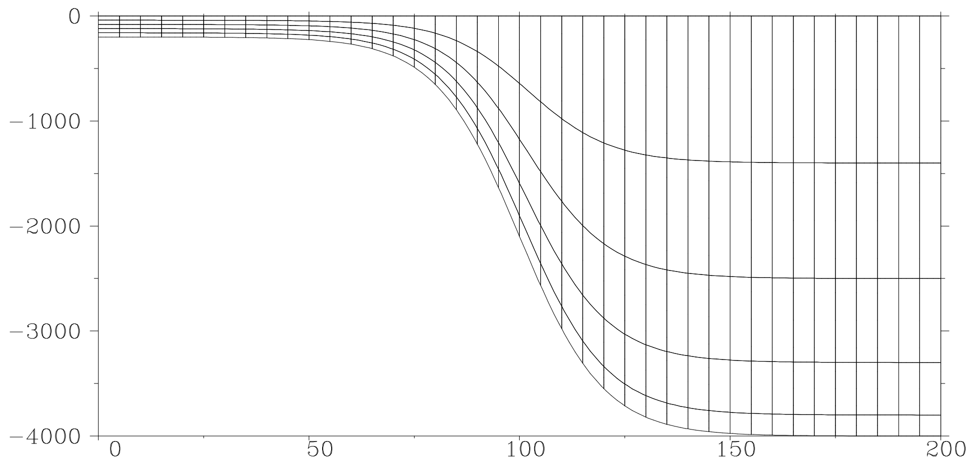

Fig. 4. Five layers.

The finite element functions defined on the finite element discretization of domain Ω span the finite element function space Φ^{FEM}. Using the finite element isomorphism lets us replace solving the scalar PDE (2.3) by instead solving the system of equations

$$K \cdot \underline{u} = \underline{f}. \tag{2.4}$$

The matrix K can be written as sum of the ne element matrices, i.e.,

$$K = \sum_{r=1}^{ne} K^{(r)}.$$

For finite elements consisting of 8×8 nodes and the element matrices $K^{(r)}$ are 64×64 dense matrices. Therefore, the number of non-zero elements $NNZ(i)$ in a row i of matrix K is 64, 120, or 225 for nodes in the interior, on the edges, and in the vertices of the finite element. The matrix K is not stored since it has excessively large storage requirements. In addition, $K^{(r)}\omega^{(r)}$ can be computed in a matrix-free manner very quickly due to the tensor product structure of the elements.

All nodes shared by more than one element will have added a subscript B. The remaining nodes will have added a subscript I. Hence, we can rewrite (2.4) so that the sparse, but dense block structure of K is obvious. Let

$$K_B = \sum_{r=1}^{ne} K_{B,r}.$$

Then

$$\left(\begin{array}{c|cccc} K_B & K_{IB,1} & K_{IB,2} & \cdots & K_{IB,ne} \\ \hline K_{IB,1} & K_{I,1} & 0 & 0 & 0 \\ K_{IB,2} & 0 & K_{I,2} & 0 & 0 \\ \vdots & 0 & 0 & \ddots & 0 \\ K_{IB,2} & 0 & \cdots & 0 & K_{I,ne} \end{array}\right) \cdot \begin{pmatrix} \underline{u}_B \\ \underline{u}_{I,1} \\ \underline{u}_{I,2} \\ \vdots \\ \underline{u}_{I,ne} \end{pmatrix} = \begin{pmatrix} \underline{f}_B \\ \underline{f}_{I,1} \\ \underline{f}_{I,2} \\ \vdots \\ \underline{f}_{I,ne} \end{pmatrix}. \tag{2.5}$$

A well known solution method for (2.5) is the classical element by element method (EBE):

Algorithm 1 Classical EBE-method

```
 1.  for all $r = 1, \dots, ne$
 2.      $S_{B,r} := K_{B,r} - K_{BI,r} K_{I,r}^{-1} K_{IB,r}$
 3.  end for
 4.  $S_B := \sum_{r=1}^{ne} S_{B,r}$
 5.  for all $r = 1, \dots, ne$
 6.      Solve $K_{I,r} \cdot \underline{\widehat{u}}_{I,r} = \underline{f}_{I,r}$
 7.  end for
 8.  $\underline{g}_B := \underline{f}_B - \sum_{r=1}^{ne} K_{BI,r} \underline{\widehat{u}}_{I,r}$
 9.  Solve $S_B \cdot \underline{u}_B = \underline{g}_B$
10.  for all $r = 1, \dots, ne$
11.      Solve $K_{I,r} \cdot \underline{u}_{I,r} = \underline{f}_{I,r} - K_{IB,r} \underline{u}_{B,r}$
12.  end for
```

The Classical EBE method can also be written in terms of a triple factorization of K:

$$\begin{pmatrix} I_B & K_{BI} K_I^{-1} \\ 0 & I_I \end{pmatrix} \begin{pmatrix} S_B & 0 \\ 0 & K_I \end{pmatrix} \begin{pmatrix} I_B & 0 \\ K_I^{-1} K_{IB} & I_I \end{pmatrix} \cdot \begin{pmatrix} \underline{u}_B \\ \underline{u}_I \end{pmatrix} = \begin{pmatrix} \underline{f}_B \\ \underline{f}_I \end{pmatrix} \tag{2.6}$$

with $S_B = K_B - K_{BI} K_I^{-1} K_{IB}$. If we define

$$P_{(EHB)} = \begin{pmatrix} I_B \\ -K_I^{-1} K_{IB} \end{pmatrix},$$

then this triple factorization can also be interpreted as change of the function space

$$\Phi^{FEM} = \Phi_I^{FEM} \cup \Phi_B^{FEM}$$

into a function space

$$\Phi^{EHB} = \Phi_I^{FEM} \cup \Phi_B^{EHB}$$

with

$$\Phi_B^{EHB} = \Phi^{FEM} \cdot P_{(EHB)}.$$

Hence, $P_{(EHB)}$ transforms the finite element space of the boundary node functions into the space of the discrete harmonic extensions of these functions. We call Φ^{EHB} the exact discrete harmonic basis and S_B is the representation of K in the basis Φ_B^{EHB}. In fact, S_B is exactly the matrix from the Galerkin approach

$$\begin{aligned} S_B &= P_{(EHB)}^T \cdot K \cdot P_{(EHB)} \\ &= \begin{pmatrix} I_B & -K_{BI} K_I^{-1} \end{pmatrix} \begin{pmatrix} K_B & K_{BI} \\ K_{IB} & K_I \end{pmatrix} \begin{pmatrix} I_B \\ -K_I^{-1} K_{IB} \end{pmatrix}. \end{aligned} \tag{2.7}$$

The ocean modeling code uses a version of the Classical EBE algorithm. The differences consist of skipping the statement in line 4, in storing the Schur complement element matrices $S_{B,r}$ and using them in a diagonally preconditioned conjugate gradients (CG or PCG) for solving the Schur complement system in line 9. Therefore, this solver is easy to parallelize.

We will discuss in the following sections some alternating approaches for solving (2.4), or (2.5) respectively. We used the NAB08 data set for our experiments and assume that floating point numbers are 8 Bytes long and integer numbers are 4 Bytes long. Table 1 contains a set of variables and their storage estimates.

Table 1. Variables and values for processor 0.

variable	description	value on processor 0
ne	number of elements	99
nn	number of nodes	5146
$ndelbnd$	number of B-nodes	1582
$negdes$	number of element edges	
$ncorners$	number of element vertices	
$npts$	nodes per direction in an element	8
$nint$	I-nodes in an element	$(npts-2)^2 = 36$
$nbnd$	B-nodes in an element	$4(npts-1) = 28$

3. A Schur Complement Preconditioner

The given code solves the Schur complement system

$$S_B \cdot \underline{u}_B = \underline{g}_B$$

(line 9 in the Classical EBE algorithm) using a (parallel) diagonally preconditioned CG algorithm. It stores in each element r the matrices $K_{I,r}^{-1}$, $K_{IB,r}$, $S_{B,r}$ and the diagonal of the Schur complement. Hence, the following amount of storage is required:

$$\begin{aligned} M(CG(S_B)) &= \left[ne \cdot (nint^2 + nint \cdot nbnd + nbnd^2) + ndelbnd\right] \cdot 8\,\text{Bytes} \\ &= 2.35\,\text{MBytes}. \end{aligned}$$

The diagonal preconditioner in the Schur complement CG can be replaced with a better preconditioner. First we have to split the B-nodes into edge and vertex nodes (denoted by the subscripts “E” and “V”) to get the block structure

$$S_B = \begin{pmatrix} S_V & S_{VE} \\ S_{EV} & S_E \end{pmatrix}.$$

Next we factor S_B similarly to the factorization of K in (2.6):

$$\begin{aligned} &\left[\begin{pmatrix} I_V & S_{VE}S_E^{-1} \\ 0 & I_E \end{pmatrix} \begin{pmatrix} S_V - S_{VE}S_E^{-1}S_{EV} & 0 \\ 0 & S_E \end{pmatrix}\right. \\ &\qquad \left.\begin{pmatrix} I_V & 0 \\ S_E^{-1}S_{EV} & I_E \end{pmatrix}\right] \cdot \begin{pmatrix} \underline{w}_V \\ \underline{w}_E \end{pmatrix} = \begin{pmatrix} \underline{r}_V \\ \underline{r}_E \end{pmatrix}. \end{aligned} \qquad (3.8)$$

As in Sec. 2, we could use the exact harmonic transformation

$$\begin{pmatrix} I_V \\ -S_E^{-1}S_{EV} \end{pmatrix},$$

but the matrix S_E has no block diagonal structure. Hence, an exact inversion is simply too costly. We approximate the exact basis transformation $-S_E^{-1}S_{EV}$ using a matrix weighted linear interpolation T_{EV}. This results in the basis transformation operator

$$P_{(HB)} = \begin{pmatrix} I_V \\ T_{EV} \end{pmatrix},$$

which converts

$$\Phi_B^{EHB} = \Phi_E^{EHB} \cup \Phi_V^{EHB}$$

into the hierarchical basis

$$\Phi_B^{HB} = \Phi_E^{EHB} \cup \Phi_V^{HB}$$

with

$$\Phi_V^{HB} = \Phi_B^{EHB} \cdot P_{(HB)}.$$

Finally, the Galerkin method representation of S_B in the basis Φ_V^{HB} is

$$\widehat{S}_V = P_{(HB)}^T \cdot S_B \cdot P_{(HB)} = S_V + T_{EV}^T S_{EV} + S_{VE}T_{EV} + T_{EV}^T S_E T_{EV}, \qquad (3.9)$$

which is easily implemented using an algebraic multigrid (AMG) coarsening routine. As an intermediate step for the preconditioner, we have the matrix

$$\begin{pmatrix} I_V & -T_{EV}^T \\ 0 & I_E \end{pmatrix} \begin{pmatrix} \widehat{S}_V & 0 \\ 0 & S_E \end{pmatrix} \begin{pmatrix} I_V & 0 \\ -T_{EV} & I_E \end{pmatrix}.$$

The inversion of the two block tridiagonal matrices is easy, but we also have to invert the block diagonal matrix in the middle.

The matrix $\widehat{S}_V$ has only $O(ne)$ rows with at most 9 entries per row. We can either invert $\widehat{S}_V$ directly or use an inexpensive iterative scheme.

The arithmetic cost of inverting S_E is nearly the same as inverting S_B. Therefore we approximate S_E by a block diagonal preconditioner

$$C_E = \mathrm{blockdiag}\{\mathrm{C_{E,k}}\}_{\mathrm{k=1}}^{\mathrm{ne}}$$

with submatrices of dimension $(npts-2)\times(npts-2)$. One option in choosing $C_{E,k}$ is to copy all entries from S_E with rows and columns from nodes of edge k and lump the remaining matrix entries into the main diagonal. Alternatively, we could choose $C_{E,k}$ using a method proposed by Dryja[7].

Applying our inverse preconditioner to the correction step $C_B\underline{w}_B = \underline{r}_B$ in the CG looks like

$$\begin{pmatrix} \underline{w}_V \\ \underline{w}_E \end{pmatrix} = \begin{pmatrix} I_V & 0 \\ T_{EV} & I_E \end{pmatrix} \begin{pmatrix} \widehat{S}_V^{-1} & 0 \\ 0 & C_E^{-1} \end{pmatrix} \begin{pmatrix} I_V & T_{EV}^T \\ 0 & I_E \end{pmatrix} \cdot \begin{pmatrix} \underline{r}_V \\ \underline{r}_E \end{pmatrix}. \qquad (3.10)$$

This preconditioner C_B for S_B is analogous to the domain decomposition preconditioner proposed by Bramble, Pasciak, and Schatz.[1,2,3,4]

With $\underline{w} = \left(\underline{w}_1^T, \ldots, \underline{w}_{ne}^T\right)^T$, we can formulate the preconditioning step in the Schur complement CG.

```
**Algorithm 2** Schur complement preconditioner: $C_B \underline{w}_B = \underline{r}_B$
1.  $\widehat{\underline{r}}_V := \underline{r}_V + T_{EV}^T \cdot \underline{r}_E$
2.  Solve $\widehat{S}_V \cdot \underline{w}_V = \widehat{\underline{r}}_V$
3.  **For all** $k = 1, \ldots, ne$
4.      Solve $C_{E,j} \cdot \widehat{\underline{w}}_{E,j} = \underline{r}_{E,j}$
5.  **End for**
6.  $\underline{w}_E := \widehat{\underline{w}}_E + T_{EV} \cdot \underline{w}_V$
```

In a preprocessing step, the matrices $\widehat{S}_V$, $C_{E,j}$, and the weights in the linear interpolation T_{EV} are calculated. The small $(npts-2)\times(npts-2) = 6\times6$ matrices $C_{E,j}$ are easily inverted in a preprocessing step. So the application of $C_{E,j}^{-1}$ only consists of a matrix-vector multiply. Depending on the size of $\widehat{S}_V$, we can solve the system in line 2 of Alg. 2 via a direct (parallel) solver or by some fast (parallel) iterative procedure such as either AMG or AMG with element preconditioning.[11]

Assuming $nedges \approx 2 \cdot ne$, $ncorners \approx ne$, and that $\widehat{C}_V$ is stored in the compact row storage format,[8] the components of C_B require the following amount of storage:

$$\begin{aligned}
M(\widehat{S}_V) &= ne \cdot (9 \cdot 12\,\text{Bytes} + 4\,\text{Bytes}) \;=\; 0.01\,\text{MBytes} \\
M(C_E^{-1}) &= ne \cdot (npts-2)^2 \cdot 8\,\text{Bytes} \;=\; 0.06\,\text{MBytes} \\
M(T_{EV}) &= ne \cdot (npts-2) \cdot 8\,\text{Bytes} \;=\; 0.01\,\text{MBytes}.
\end{aligned}$$

Hence, the the preconditioner and Schur complement CG require

$$\begin{aligned}
M(C_B) &= M(\widehat{C}_V) + M(C_E^{-1}) + M(T_{EV}) \;=\; 0.08\,\text{MBytes} \\
M(PCG(S_B)) &= M(PCG(S_B)) + M(C_B) \;=\; 2.43\,\text{MBytes},
\end{aligned}$$

Note that the Schur complement preconditioner only requires 4% more storage than the Schur complement CG.

Finally, parallelization of Alg. 2 is straightforward to implement.

4. Lumping of Small Entries and AMG

The primary roadblock in applying an alternate algorithm to solve (2.4) consists in how to reduce the huge amount of storage required to store the matrix. The average number of nonzero entries per row is 81 and so K (which is assumed to be stored in the compressed row storage) requires

$$M(K) \;=\; nn \cdot (81 \cdot (12\,\text{Bytes}) + 4\,\text{Bytes}) \;=\; 4.79\,\text{MBytes}.$$

This is a small amount of memory unless there are thousands or millions of elements. If we store the element matrices K_r instead of the accumulated one we have

$$M(K_r) \;=\; ne \cdot (npts^2)^2 \cdot 8\,\text{Bytes} \;=\; 3.10\,\text{MBytes}.$$

An analysis of the matrices K_r shows that many matrix entries are extremely small. So we lumped all entries $K_{r,i,j}$ with an absolute value smaller than 10% of the main diagonal entries $K_{r,i,i}$ and $K_{r,j,j}$ into the main diagonal. We denote the resulting matrices by C_r. The condition number of this element preconditioning is $\kappa(C_r^{-1}K_r)\approx 9$. The advantage of this approach consists in a remarkable decrease of memory required for storing all C_r. In our example, the resulting matrix $C = \sum_{r=1}^{ne} C_r$ possesses $nnz = 34720$ non-zero entries, i.e., less than 7 entries per matrix row.

$$M(C) \;=\; nnz \cdot 12\,\text{Bytes} \;+ \text{nn} \cdot 4\,\text{Bytes} \;=\; 0.42\,\text{MBytes}.$$

The matrix C is still symmetric and positive definite. Unfortunately, neither C nor K are M-matrices, but the positive off-diagonal entries in C are smaller by one order of magnitude than the main diagonal entries. This gives us hope that a parallel AMG algorithm[9,10] will work.

The given code has a routine for the multiplication $K \cdot \underline{v}$ without storing the matrix. An appropriate matrix free CG is also available. One or two AMG iterations with the matrix C is a very good preconditioner for the CG, decreasing the number of CG iterations dramatically. This AMG requires approximately

$$M(AMG(C)) \;\approx\; 3 \cdot M(C) \;=\; 1.26\,\text{MBytes}.$$

The multiplication $K \cdot \underline{v}$ can also be accelerated by storing the element matrices and taking into account symmetry. Therefore we only have to store

$$M(K_r^{symm}) \;=\; ne \cdot \frac{npts^2(npts^2+1)}{2} \cdot 8\,\text{Bytes} \;=\; 1.57\,\text{MBytes}.$$

Hence, the matrix free and stored matrix versions of the preconditioned CG require

$$\begin{aligned} M(PCG, AMG(C)) &\;=\; M(AMG(C)) \;=\; 1.26\,\text{MBytes} \\ M(PCG, K_r^{symm}, AMG(C)) &\;=\; M(AMG(C)) + M(K_r^{symm}) \;=\; 2.83\,\text{MBytes}. \end{aligned}$$

A version of AMG that takes the symmetry of the matrices into account would decrease the storage requirements for the AMG significantly (down to 60%), but this version of AMG requires significant additional programming.

5. Multigrid with Patch Smoothing

Another approach is to construct a preconditioner for the matrix K consisting of applying a two grid technique to reduce the amount of data per element. This two grid technique requires the same algorithmic components as multigrid, namely, an interpolation operator, a coarse grid matrix (and solver), a smoother, and defect correction calculations.

The original stiffness matrices K_r results from 7th degree test functions in both directions on an element $\delta^{(r)}$. We reduce the element information significantly if we assume only linear test functions in both directions. The fine grid basis is

$$\Phi_f = \Phi^{FEM} = \{\varphi_i^{7th}(x,y)\}_{i=1}^{nn}$$

and coarse grid basis consists of

$$\Phi_c = \{\varphi_j^{1th}(x,y)\}_{j=1}^{ncorners}.$$

The interpolation matrix between coarse and fine basis is defined by

$$P_{fc} = \{p_{ij}\}_{\substack{i=1,\dots,nn \\ j=1,\dots,ncorners}} = \left\{\varphi_j^{1th}(x_i,x_i)\right\}_{\substack{i=1,\dots,nn \\ j=1,\dots,ncorners}} .$$

Hence, $\Phi_c = \Phi_f \cdot P_{fc}$ holds. The interpolation matrix P_{fc} can be stored elementwise (the interpolation weights on the edges are coherent). This allows us to apply a theorem from parallel algorithms[9] and to calculate the coarse matrix elementwise:

$$K_c = \sum_{r=1}^{ne} K_{c,r} = \sum_{r=1}^{ne} P_{fc,r}^T \cdot K_r \cdot P_{fc,r}.$$

Accumulation can be performed easily if that should be necessary for the coarse grid solver. One choice for a coarse grid solver is another application of the AMG algorithm. The coarse matrix and the interpolation matrix require

$$\begin{aligned} M(K_c) &= M(\widehat{S}_V) = 0.01\,\text{Mbyte} \\ M(P_{fc}) &= ne \cdot 4 \cdot npts^2 \cdot 8\,\text{Bytes} = 0.20\,\text{MBytes}. \end{aligned}$$

The defect calculation requires $K \cdot \underline{v}$, which is already provided in a matrix free routine.

We use a patch smoother with the elements as patches. Denoting by $\mathcal{K}_r := K|_r$ those entries K_{ij} of the accumulated matrix K with $(x_i, y_i) \in \delta^{(r)}$ and $(x_j, y_j) \in \delta^{(r)}$ then one iteration of the original patch smoother can be formulated as

Algorithm 3 Standard patch smoother: one iteration

```
1.  For all $r = 1, \dots, ne$
2.      $\underline{d}_r := \underline{f} - K \cdot \underline{u}$
3.      $\underline{w}_r := \mathcal{K}_r^{-1} \cdot \underline{d}_r$
4.      $\underline{u} := \underline{u} + \underline{w}_r$
5.  End for
```

This smoother requires at least $M(\mathcal{K}_r^{-1}) = M(K_r^{symm}) = 1.57\,$MBytes plus the same amount of storage if K_r is also stored.

The storage requirements can be reduced significantly if we use the lumped matrix C from Sec. 4 and replace $\mathcal{K}_r$ by

$$\mathcal{B}_r := C|_r .$$

Formally,

Algorithm 4 Modified standard patch smoother: one iteration

```
1.  For all $r = 1, \dots, ne$
2.      $\underline{d}_r := \underline{f} - K \cdot \underline{u}$
3.      $\underline{w}_r := \mathcal{B}_r^{-1} \cdot \underline{d}_r$
4.      $\underline{u} := \underline{u} + \underline{w}_r$
5.  End for
```

The local numbering of nodes in element $\delta^{(r)}$ has to be chosen such that the bandwidth of the sparse matrix $\mathcal{B}_r$ is minimized, the typical bandwidth is $1 + 2npts$ (a skyline storage method is also a possibility). Taking into account the symmetry of $\mathcal{B}_r$ need storage of

$$M(\mathcal{B}_r^{symm}) \;=\; M((\mathcal{B}_r^{symm})^{-1}) \;=\; ne \cdot npts^2 \cdot (1 + npts) \cdot 8\,\text{Bytes} \;=\; 0.44\,\text{MBytes}.$$

Thus, this multigrid algorithm with a matrix free defect calculation requires

$$M(mg, K_c, \mathcal{B}_r^{symm}, P_{fc})) \;=\; M(K_c) + M(\mathcal{B}_r^{symm}) + M(P_{fc}) \;=\; 0.65\,\text{MBytes}.$$

A defect calculation with explicitly stored element matrices K_r requires

$$M(mg, K_c, \mathcal{B}_r^{symm}, P_{fc})) + M(K_r^{symm}) = 2.2\,\text{MBytes},$$

i.e., approximately the same amount of storage as needed for the already implemented Schur complement solver.

Deriving $P_{fc,r}$ on the reference element instead of the real element leads to exactly the same interpolation weights. Hence, it is sufficient to store the interpolation weights only once. This saves $M(P_{fc})$ in storage.

6. Numerical Experiments, Memory Impact, and Conclusions

One of the primary limiting features of ocean modeling is memory. In this paper, we have analyzed three preconditioners for SEOM from a memory viewpoint. What we discovered for a field specific example (the NAB08 dataset) is included in Table 2.

Table 2. Methods and memory requirements.

Method	Section	Memory (MBytes)	Ratio
Schur-CG	2	2.35	1.000
Schur-PCG	3	2.36	1.004
AMG-K_r	4	2.82	1.200
AMG(free)	4	1.26	0.536
AMG(free*)	4	1.39	0.591
2-Grid(free*)	5	0.96	0.408
2-Grid(free,Jacobi)	5	0.20	0.085

We ran four experiments on a single processor in order to see how the algorithms compare. What we discovered for the NAB08 dataset is included in Table 3. The total filtering time (in seconds) includes the preconditioned conjugate gradient time. The $m \times n$ in method description in Table 3 refers to solving m times with n right hand sides. Solving for the correct number of right hand sides simultaneously allows for better cache memory utilization.[5,6]

We note that if we had only included CG iterations to convergence, we would have declared the AMG(free) method the clear winner. However, using CPU time, the Schur complement methods are clearly superior. While the algebraic multigrid

Table 3. Methods and time comparisons.

Method	Filter	CG time	CG iterations
10×1 AMG(free)	30.06	26.10	100
2×5 AMG(free)	29.22	25.38	135
10×1 Schur-PCG	8.89	5.36	278
1×10 Schur-PCG	8.70	4.99	300

approach gave rise to high hopes for running faster than the Schur complement algorithms, in practice, we have not yet been able to accomplish that hope. We consider realizing this hope to be future research.

Acknowledgment

This research was supported in part by the National Science Foundation (grants DMS-9707040, ACR-9721388, CCR-9902022, CCR-9988165) and gifts from the Intel and Hewlett-Packard Corporations.

References

1. J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring I*, Math. Comp. **47** (1986), 103–134.
2. ______, *The construction of preconditioners for elliptic problems by substructuring II*, Math. Comp. **49** (1987), 415–430.
3. ______, *The construction of preconditioners for elliptic problems by substructuring III*, Math. Comp. **53** (1988), 1–24.
4. ______, *The construction of preconditioners for elliptic problems by substructuring IV*, Math. Comp. **55** (1989), 1–24.
5. C. C. Douglas, *Caching in with multigrid algorithms: problems in two dimensions*, Paral. Alg. Appl. **9** (1996), 195–204.
6. C. C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiss, *Cache optimization for structured and unstructured grid multigrid*, Elect. Trans. Numer. Anal. **10** (2000), 21–40.
7. M. Dryja, *A finite element-capacitance matrix method for elliptic problems on regions partitioned into substructures*, Numer. Math. **34** (1984), 153–168.
8. S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *Yale sparse matrix package: I. the symmetric codes*, Tech. Report 112, Department of Computer Science, Yale University, New Haven, 1977.
9. G. Haase, *A parallel amg for overlapping and non-overlapping domain decomposition*, Elect. Trans. Numer. Anal. **10** (2000), 41–55.
10. G. Haase, M. Kuhn, and S. Reitzinger, *Parallel AMG on distributed memory computers*, SFB-Report 00-16, University Linz, SFB F013, June 2000.
11. G. Haase, U. Langer, S. Reitzinger, and J. Schöberl, *Algebraic multigrid methods based on element preconditioning*, Int. J. Comput. Math. **78** (2001), no. 4, 575–598.
12. D. B. Haidvodel and A. Beckmann, *Numerical ocean circulation modeling*, Environmental Science and Management, vol. 2, Imperial College Press, London, 1999.
13. M. Iskandarani, D. B. Haidvogel, and J. P. Boyd, *A staggered spectral element model with application to the oceanic shallow water equations*, Int. J. Numer. Meth. Fluids

20 (1995), 393–414.

14. J. G. Levin, M. Iskandarani, and D. B. Haidenvogel, *A spectral filtering procedure for eddy-resolving simulations with a spectral filtering ocean model*, J. Comp. Phys. **137** (1997), 130–154.