Karlsruhe Institute of Technology

A High-Efficient Scalable Solver for the Global Ocean/Sea-Ice Model MPIOM

Panagiotis Adamidis
Vincent Heuveline
Florian Wilhelm

No. 2011-02

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)

KIT – University of the State of Baden-Wuerttemberg and
National Research Center of the Helmholtz Association

www.emcl.kit.edu

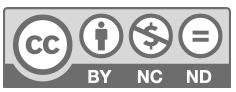
Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)
ISSN 2191–0693
No. 2011-02

Impressum

Karlsruhe Institute of Technology (KIT)
Engineering Mathematics and Computing Lab (EMCL)

Fritz-Erler-Str. 23, building 01.86
76133 Karlsruhe
Germany

KIT – University of the State of Baden Wuerttemberg and
National Laboratory of the Helmholtz Association



A HIGH-EFFICIENT SCALABLE SOLVER FOR THE GLOBAL OCEAN/SEA-ICE MODEL MPIOM

F. WILHELM[1], P. ADAMIDIS[2] AND V. HEUVELINE[1]

[1] *Engineering Mathematics and Computing Lab, Karlsruhe Institute of Technology, Karlsruhe, Germany*
[2] *Scientific Computing, Deutsches Klimarechenzentrum GmbH, Hamburg, Germany*

Abstract. This paper presents the work of the "Scalable-Earth-System-Models for high productivity climate simulations" project in improving the linear solver of the barotropic subsystem in the global ocean/sea-ice model MPIOM developed by the Max-Planck-Institute in Hamburg, Germany. We demonstrate the implementation of the conjugate gradient method and an incomplete Cholesky preconditioner with fill-in p in order to achieve high scalability and performance on an IBM POWER6 based supercomputer. Benchmarks of the new solver compared to the traditionally used Successive-Over-Relaxation-Method are given and analyzed with respect to the number of iterations and runtime.

1. Introduction

Climate change and anthropogenic influence on it, is one of the most important issues mankind has to deal with in this century. It is an largely accepted fact that world climate is changing because of higher CO_2 emissions since the beginning of the second industrial revolution in mid 19^{th} century until today. The first consequences of global warming, like more frequent storms, droughts and glacier melt appear worldwide and raise the question how climate will change in the future due to man-made greenhouse gas emissions. The significance of this field of research is obvious, given the fact that the United Nations Environment Programme (UNEP) established the Intergovernmental Panel on Climate Change (IPCC) in November 1988 to coordinate global research efforts in climate change and to provide an IPCC Assessment Report (IPCC AR) regularly. The crucial part of this report consists of climate projections provided by simulations based on possible future scenarios of CO_2 emissions which are of paramount importance to assess the consequences of economic decisions now to be taken.

Earth-System-Models (ESMs) consisting of coupled components that describe physical and biogeochemical processes in different parts of the Earth System (atmosphere, ocean, land, cryosphere, biosphere etc.) have been developed for many decades to simulate different future scenarios. The recent advancements in parallel computing assume these models to be optimized for scalability on modern hardware in order to enlarge the range of explicitly resolved phenomena by increasing horizontal and vertical model grid resolution and to implement more sophisticated descriptions of unresolved sub-grid processes. This would give more confidence in results of the future climate change projections. The goal of the *Scalable-Earth-System-Models for high productivity climate simulations* (ScalES) project funded by the German Bundesministerium für Bildung und Forschung (BMBF No. 01IH08004E) is to improve the COSMOS ESM [4] in this regard by means of advanced mathematical methods and modern programming techniques on state-of-the-art hardware.

Key words and phrases. hpc, sor, cg method, icc(p), mpiom, climate simulation, mathematical modeling, ocean model, barotropic subsystem.

In this paper we present work done in the ocean/sea-ice model MPIOM developed by the Max-Planck-Institute for Meteorology (MPI-M) to achieve better performance and scalability on the world's largest IBM POWER6 installation in a single InfiniBand cluster at the Deutsche Klimarechenzentrum (DKRZ), the German climate computing center, in Hamburg, Germany.

2. Model Description of MPIOM

MPIOM is an Ocean General Circulation Model (OGCM) based on the ocean primitive equations on a curvilinear C-grid with z-coordinates and free surface [9]. The first version of MPIOM was released in 1997 as a serial program written in FORTRAN 77. In 2000 the code was then parallelized for the NEC SX6 vector parallel supercomputer, accompanied by a switch to FORTRAN 90. Today the code encompasses roughly 40,000 lines of FORTRAN 90/95 code and uses the Message Passing Interface (MPI) library with optional OpenMP for parallelization.

It is the successor of the Hamburg Ocean Primitive Equation (HOPE) model [13]. While the horizontal discretization in HOPE was based on a staggered Arakawa E-grid, MPIOM makes use of a curvilinear C-grid [2]. There are mainly two reasons for this transition. Firstly, the C-grid is computationally more efficient than the staggered E-grid, because a higher horizontal resolution can be achieved with the same number of grid points. Secondly, the E-grid model required additional horizontal numerical diffusion in order to achieve convergence. MPIOM has been applied in numerous scientific studies investigating different aspects of the ocean/sea-ice dynamics and the ocean's role in Earth System dynamics. Simulations with the coupled ESM ECHAM-MPIOM have contributed to the IPCC AR4 and will also provide data for IPCC AR5. In the remaining section the linear system that is solved in MPIOM is shortly derived.

The horizontal conservation of momentum equation for a Boussinesq fluid on a rotating sphere with orthogonal coordinates is

$$\frac{d}{dt}\mathbf{u} + f(\mathbf{z}\times\mathbf{u}) = -\frac{1}{\rho_w}\nabla_H(p+\rho_w g\eta) + \mathbf{F}_H + \mathbf{F}_V, \tag{1}$$

where $\mathbf{u} = (u, v)$ are the horizontal components of velocity $\mathbf{v}$, t is the time, f the Coriolis parameter, $\mathbf{z}$ the vertical unit vector, ρ_w the reference density of sea-water, ∇_H the horizontal gradient operator, p the internal pressure, g the effective gravitational acceleration and η the sea-surface elevation. The total derivative is $\frac{d}{dt} = \partial_t + \mathbf{u}\cdot\nabla_H + w\cdot\partial_z$, where w is the vertical component of $\mathbf{v}$. The terms $\mathbf{F}_H$ and $\mathbf{F}_V$ describe the horizontal and vertical eddy viscosity.

Following [12] the conservation of momentum is now solved by decomposing $\mathbf{u}$ into its barotropic $\bar{\mathbf{u}}$ and baroclinic $\tilde{\mathbf{u}}$ components

$$\bar{\mathbf{u}} = \int_{-H}^{0} \mathbf{u}\, dz, \qquad \tilde{\mathbf{u}} = \mathbf{u} - \frac{\bar{\mathbf{u}}}{H},$$

where H is the local depth of the sea. Applying this to (1) results in a *barotropic subsystem*

$$\partial_t\bar{\mathbf{u}} + \overline{A(\mathbf{u},\mathbf{u})} + f(\mathbf{z}\times\bar{\mathbf{u}}) = -gH\nabla_H\eta - \int_{-H}^{0}\nabla_H\hat{p}\, dz + \bar{\mathbf{F}}_H + \bar{\mathbf{F}}_V,$$

and a *baroclinic subsystem*

$$\partial_t\tilde{\mathbf{u}} + A(\mathbf{u},\mathbf{u}) - \overline{A(\mathbf{u},\mathbf{u})} + f(\mathbf{z}\times\tilde{\mathbf{u}}) = \frac{1}{H}\int_{-H}^{0}\nabla_H\hat{p}\, dz - \nabla_H\hat{p} + \tilde{\mathbf{F}}_H + \tilde{\mathbf{F}}_V,$$

where $\hat{p} = p/\rho_w$ denotes the baroclinic pressure divided by the constant density and the advection term given by

$$A(\mathbf{u},\mathbf{u}) = (\mathbf{u}\cdot\nabla)\mathbf{u} + w\partial_z\mathbf{u}.$$

Partially updating $\tilde{\mathbf{u}}$ and $\bar{\mathbf{u}}$ with respect to the advection terms $A(\mathbf{u},\mathbf{u})$, $\overline{A(\mathbf{u},\mathbf{u})}$ and viscosity terms $\bar{\mathbf{F}}_H$, $\bar{\mathbf{F}}_V$, $\tilde{\mathbf{F}}_H$, $\tilde{\mathbf{F}}_V$ by means of *operator splitting techniques* further reduces the barotropic

and baroclinic subsystems. The baroclinic subsystem, together with an equation for the time evolution of the internal pressure

$$\partial_t \partial_z \hat{p} = \frac{\tilde{w} g}{\rho_w} \partial_z \rho,$$

can be discretized in time by a forward Euler scheme and in space with finite differences with respect to the curvilinear C-grid concluding to an equation system that can be easily solved by a fixed-point iteration scheme, provided the solution of the last time step is used as initial guess.

On the contrary the partially updated barotropic subsystem together with the barotropic continuity equation

$$\partial_t \eta + \partial_x \bar{u} + \partial_y \bar{v} = 0$$

results, after the discretization in time by forward Euler, in the linear equation system

$$\begin{aligned} &\bar{u}^{n+1} - \bar{u}^n - f\Delta t \left(\alpha \bar{v}^{n+1} + (1-\alpha)\bar{v}^n\right) \\ &+gH\Delta t \left(\alpha \eta_x^{n+1} + (1-\alpha)\eta_x^n\right) + \Delta t \int_{-H}^{0} \hat{p}_x^{n+1}\, dz = 0 \end{aligned} \tag{2}$$

$$\begin{aligned} &\bar{v}^{n+1} - \bar{v}^n + f\Delta t \left(\alpha \bar{u}^{n+1} + (1-\alpha)\bar{u}^n\right) \\ &+gH\Delta t \left(\alpha \eta_y^{n+1} + (1-\alpha)\eta_y^n\right) + \Delta t \int_{-H}^{0} \hat{p}_y^{n+1}\, dz = 0 \end{aligned} \tag{3}$$

$$\eta^{n+1} - \eta^n + \Delta t \left(\beta \bar{u}_x^{n+1} + (1-\beta)\bar{u}_x^n + \beta \bar{v}_y^{n+1} + (1-\beta)\bar{v}_y^n\right) = 0, \tag{4}$$

with relaxation parameters α and β, that cannot be solved in the same way. Solving (2) resp. (3) for $\bar{u}$ resp. $\bar{v}$ and substituting into (4) yields an equation system for η^{n+1} and its horizontal derivatives η_x^{n+1} and η_y^{n+1}. Using finite differences to discretize horizontal derivatives on a curvilinear C-grid with m unknowns in zonal, n unknowns in meridional direction and a total of $N = mn$ unknowns results in a Stieltjes matrix $A = (a_{ij}) \in \mathbb{R}^{N \times N}$, that is a real symmetric positive definite matrix with non-positive off-diagonal entries, and a right hand side $b \in \mathbb{R}^N$. A is of block tridiagonal shape

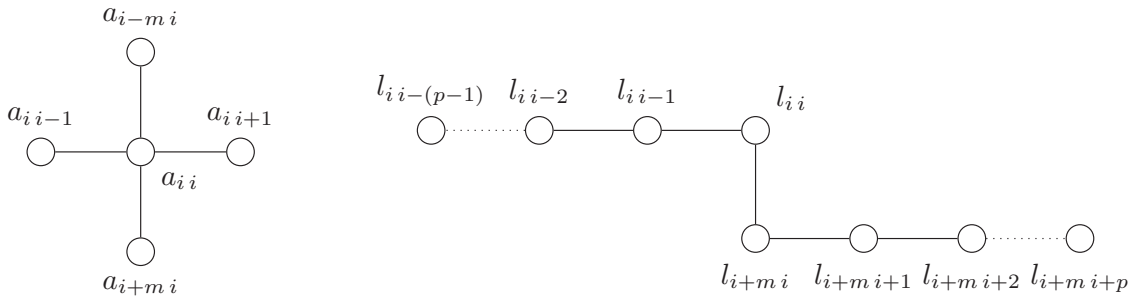
$$A = \begin{pmatrix} D_1 & S_1 & & & \\ S_1 & D_2 & S_2 & & \\ & S_2 & \ddots & \ddots & \\ & & \ddots & \ddots & S_n \\ & & & S_n & D_n \end{pmatrix}, \tag{5}$$

with diagonal matrices $S_i \in \mathbb{R}^{m \times m}$ and tridiagonal matrices $D_i \in \mathbb{R}^{m \times m}$, $i = 1, \ldots, n$. To exploit this sparsity pattern the matrix can be reformulated as a five-point stencil as in Figure 1a. The following sections discuss methods of solving this system of linear equations.

3. Solving the Barotropic Subsystem

In MPIOM the barotropic subsystem is formulated as a five-point stencil (see Figure 1a) on a structured 2D grid realized by means of arrays. The desired solution η^{n+1} is stored in an array `Z1O(i,j)` with $i = 1, \ldots, m$ and $j = 1, \ldots, n$, where i represents a zonal (west–east) and j a meridional (north–south) coordinate. The corresponding barotropic stencil is defined as a set of three 2D arrays `FF` for the central part, `UF` for the zonal arms and `VF` for the meridional arms of the stencil at coordinate (i, j). Keeping the staggered Arakawa C-Grid in mind an application of the stencil to `Z1O(i,j)` translates to

$$\begin{aligned} &\texttt{FF(i,j)} * \texttt{Z1O(i,j)} - \texttt{UF(i,j)} * \texttt{Z1O(i+1,j)} - \texttt{UF(i-1,j)} * \texttt{Z1O(i-1,j)} \\ &\quad - \texttt{VF(i,j)} * \texttt{Z1O(i,j+1)} - \texttt{VF(i,j-1)} * \texttt{Z1O(i,j-1)}. \end{aligned}$$



(A) Five-point stencil of the barotropic subsystem

(B) Stencil of the ICC(p) factorization

FIGURE 1. Stencil structure of the barotropic subsystem and its ICC(p) factorization

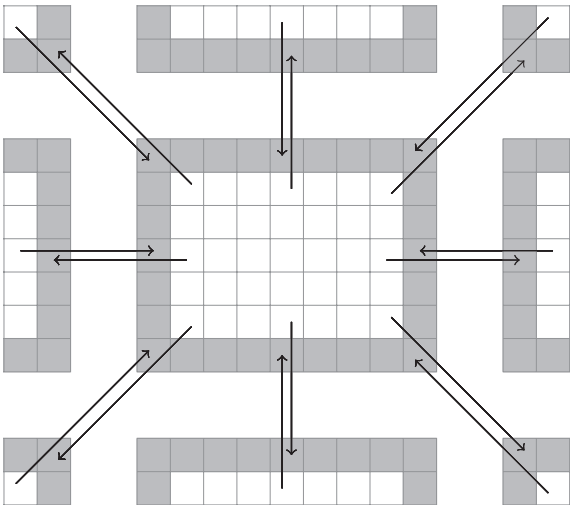


FIGURE 2. Boundary exchange of one partition (white area) with its neighbours. The grey area represents one or more boundary halos [3].

Parallelization is accomplished by a uniform block decomposition of the 2D arrays into $z = x \cdot y$ rectilinear partitions for a total of z processes with x partitions in zonal direction and y partitions in meridional direction. Consequently any partition has four directly and four diagonally neighbouring partitions as illustrated in Figure 2. Communication is performed through additional boundary halos, implemented by appropriately enlarged arrays, that overlap the neighbouring partitions and are updated with the according values each time a boundary exchange function is called. In case of only one boundary halo, communication is only necessary with four neighbouring partitions due to the structure of the stencil. Depending on the application, using more halos reduces the number of necessary boundary exchanges. This results in shorter communication time per data, because of fewer communication operations and therefore fewer latencies.

Traditionally the barotropic subsystem was solved with *Successive-Over-Relaxation* (SOR) in MPIOM which is a splitting method based on $\omega A = (D+\omega L) - ((1-\omega)D - \omega U)$, where D is the diagonal of A, L its strict lower part, $U = L^T$ its strict upper part and ω a relaxation parameter with $\omega \in (0, 2)$. Hence the iteration scheme is

$$(D + \omega L)x_{k+1} = ((1 - \omega)D - \omega L^T)x_k + \omega b,$$

which can be easily translated to a stencil formulation. Parallelization is done by a *red-black ordering* that allows parallel treatment of points with the same colour. The drawback of this ordering is the necessity of two boundary exchanges per iteration if only one boundary halo is used. Therefore two boundary halos are used in MPIOM following the recommendations in [3]. The most important relaxation parameter ω, with regard to the rate of convergence, is estimated by a number of test calculations measuring the rate of convergence followed by manual fine tuning. Still the SOR method proved to be insufficient in terms of a too large number of necessary iterations that impairs the scalability of this method. Furthermore the SOR method provides no implicit way to check for the quality of the current iteration, meaning that additionally the calculation of a residual would be needed to assure the quality of the solution. In the current SOR implementation this is omitted, meaning that one needs to preset a fixed number of iterations. A major task in the ScalES project was to resolve these limitations by replacing SOR with the conjugate gradient method.

3.1. **Properties and Implementation of the Conjugate Gradient Method.** For solving a sparse symmetric positive definite linear system, the conjugate gradient (CG) method is one of the best known iterative techniques [11]. It belongs to the class of projection methods onto Krylov subspaces and was first proposed in 1952 [7]. The CG method consists basically only of two building blocks: matrix-vector multiplication and dot product, whose efficient implementation determines much of the resulting scalability of the algorithm. Besides the operations in each iteration, the overall number of iterations needed to satisfy a given tolerance is of utmost importance. According to the convergence theory of CG, this number depends on the condition number $\kappa(A)$ through the relation

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\Big(\frac{\sqrt{\kappa(A)-1}}{\sqrt{\kappa(A)+1}}\Big)^k,$$

where $e_k = x_k - x$ is the error in the k^{th} iteration and $\|\cdot\|_A$ the energy norm. This inequality justifies the application of a preconditioner M where $\kappa(M^{-1}A) \ll \kappa(A)$ and conclusively fewer iterations are necessary to solve the equivalent system $M^{-1}Ax = M^{-1}b$. The preconditioner M needs to be symmetric and positive definite to sustain those properties for the CG method. Without a preconditioner the number of necessary iterations is too high and the resulting number of network communications impairs the scalability and overall runtime.

In Algorithm 1 the pseudo code of the preconditioned conjugate gradient (PCG) method is illustrated. The implementation in MPIOM was done in a generic way, meaning that the five-point stencil of the barotropic subsystem is provided as a function parameter to the PCG function so that it is independent of the actual stencil implementation. Each process performs the stencil operation on its local partition followed by a boundary exchange to update the boundary halos. In the same way was the preconditioner M^{-1} implemented to allow for different kinds. To calculate the dot product local dot products are summed up using the sum reduction (MPI_Allreduce, MPI_SUM) of the MPI library. It is possible to switch to routines from the Basic Linear Algebra Subprograms (BLAS) library for saxpy operations and dot products with a preprocessor flag. We emphasized a modular implementation to facilitate reuse of the code in other projects.

4. The Incomplete Cholesky Preconditioner

As a preconditioner for the PCG the incomplete Cholesky decomposition (ICC) is one of the most appealing. Depending on the level of fill-in, it is possible to achieve a great reduction in the number of necessary PCG iterations. The symmetric decomposition, as defined in the following subsection, reduces memory requirements in contrast to other decomposition methods like incomplete LU. Parallelization was achieved by performing ICC partition-wise, so that no

Algorithm 1 Preconditioned conjugate gradient [1].

$r_0 = b - Ax_0$
$z_0 = M^{-1} r_0$
$p_0 = z_0$
for $k = 0, 1, \ldots, k_{max}$ **do**
  $\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  $z_{k+1} = M^{-1} r_{k+1}$
  if $r_{k+1}^T z_{k+1} < TOL$ **then**
    exit loop
  end if
  $\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$
  $p_{k+1} = z_{k+1} + \beta_k p_k$
end for

communication is needed when applying the preconditioner. The ICC preconditioner consists of two steps: a setup step where the incomplete decomposition is calculated and its actual application in the PCG algorithm. Those steps are detailed in the following two subsections.

4.1. **Incomplete Cholesky Factorization of A.** Given that A is a symmetric and positive definite matrix, there exists a unique factorization $A = LL^T$, where L is a lower diagonal matrix with positive diagonal entries [5]. By applying the column-wise proceeding Cholesky-Crout algorithm, for $i = 1, \ldots, N$ we have

$$
\begin{aligned}
l_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \\
l_{ji} &= \frac{1}{l_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik} \right), \quad \text{for } j > i.
\end{aligned}
\tag{6}
$$

The choice of a column-wise instead of a row-wise traversal is motivated by the fact that FORTRAN stores arrays column-wise, resulting in a higher cache-hit rate when accessed in the same manner, as we will see later. A complete factorization would lead to a dense matrix L, that must be avoided because of memory limitations and high computational costs, when doing forward and backward substitution. Therefore we introduce for l_{ij} the common definition [11] of the initial level of fill-in

$$
\mathrm{lev}_{ij} = \begin{cases} 0 & if\ a_{ij} \neq 0\ or\ i = j \\ \infty & otherwise \end{cases},
$$

and update the current level lev_{ij} when l_{ij} is updated according to

$$
\mathrm{lev}_{ij} = \min \left(\mathrm{lev}_{ij}, \min_{l=1,\ldots,i-1} (\mathrm{lev}_{jl} + \mathrm{lev}_{il} + 1) \right).
$$

When lev_{ij} exceeds a predefined maximum level of fill-in p we set $l_{ij} = 0$. Let L^p be the factor L due to the incomplete Cholesky decomposition with level of fill-in p. Analysing the resulting sparsity pattern of L^p (see Figure 3) with the help of a naïve implementation in Matlab, we can easily derive a rule to predict the sparsity pattern. Let $\mathrm{diag}_k(L) = (l_{i+k\,i})_{i=1,\ldots n-k}$ be the k^{th} lower diagonal. L^0 has the same sparsity pattern as the lower diagonal matrix of A meaning

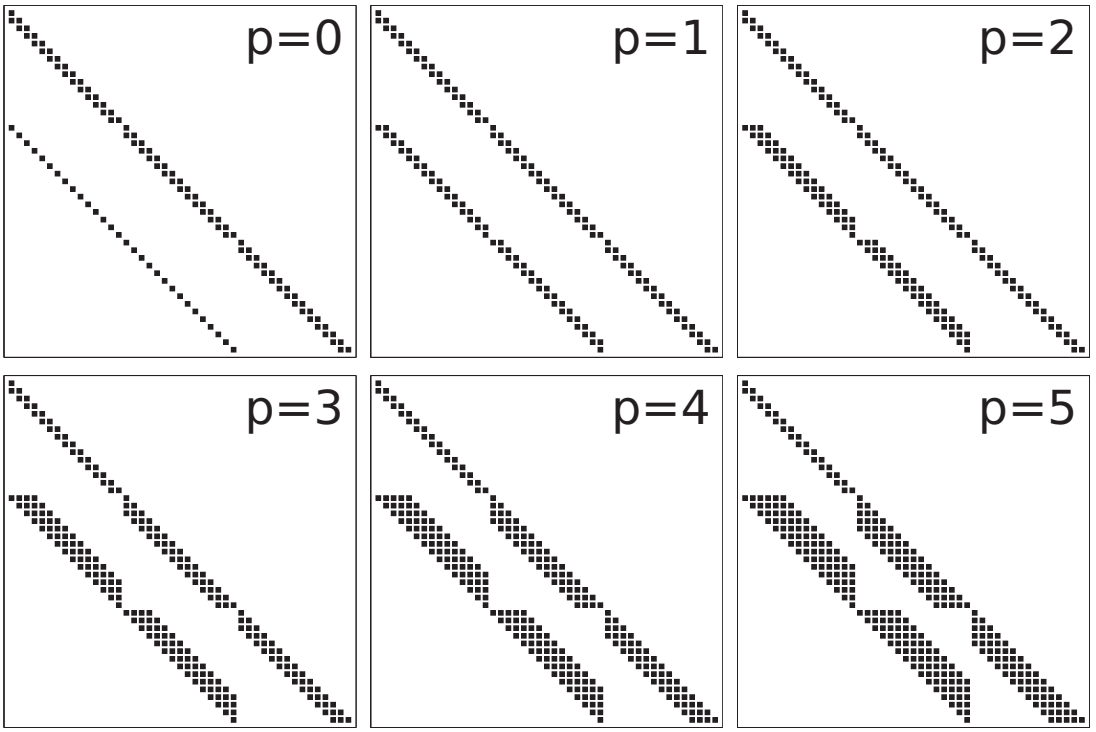


FIGURE 3. Fill-in pattern for ICC(p) with $p = 0, \ldots, 5$.

the non-zero elements are in $\mathrm{diag}_k(L)$ with $k = 0, 1, m$, whereas L^1 gains additional elements in $\mathrm{diag}_{m-1}(L)$. In case of L^p with $p \geq 2$ we have non-zero elements in $\mathrm{diag}_k(L)$ with $k = 0, \ldots, p-1$ and $k = m-p, \ldots, m$. We can extend this rule to include the somewhat exceptional cases L^0 and L^1 to obtain in all

$$l_{ij}^p = 0 \text{ if } l_{ij}^p \notin \mathrm{diag}_k(L) \text{ with } k = 0, \ldots, \max(1, p-1), m-p, \ldots, m. \tag{7}$$

The diagonals $\mathrm{diag}_k(L)$ with $k = 0, \ldots, \max(1, p-1)$ will be denoted with *secondary diagonals* and $\mathrm{diag}_k(L)$ with $k = m-p, \ldots, m$ with *outer diagonals*.

Taking into account the diagonal growth of this pattern for increasing fill-in, the DIAG format [11] is a suitable storage format. The DIAG format stores elements along a diagonal in a two-dimensional array `DIAG(1:n,1:Nd)`, where `Nd` is the number of diagonals with non-zero entries. The offset of each diagonal from the main diagonal is saved in the array `IOFF(1:Nd)`. As in the FORTRAN 95 language the colon notation `A(i:j)` is used to account for the dimension of an array A or a slice consisting of the ordered elements `A(i), A(i+1),..., A(j)`. By adding an increment of `-1` as in `A(i:j:-1)` the order of the elements is reversed.

Considering the fact that we are dealing with a stencil, we modified the DIAG storage scheme into a more stencil compliant form. Transforming the matrix L^p to a stencil representation results in an ICC stencil as pictured in Figure 1b. The western arm $l_{i\,i-1}, \ldots, l_{i\,i-(p-1)}$ of this stencil is stored in the three-dimensional array `ICC_W(1:MAX(1,p-1),1:m,1:n)`, the inverse of the central element l_{ii} in the array `ICC_C(1:m,1:n)` and the southern arm $l_{i+m\,i}, \ldots, l_{i+m\,i+p}$ in `ICC_S(1:p+1,1:m,1:n)`. Again, the indexing of these arrays was chosen so that the element access in the forward and backward substitution step is most cache efficient. By virtue of storing $\frac{1}{l_{ii}}$ in `ICC_C` we can later, when applying the preconditioner, use a multiplication instead of a division which can be computed in 1 cycle compared to 30 cycles on the IBM POWER6.

Algorithm 2 Main loop of the ICC(p) decomposition. A and L as defined in (5) and (6).

```
1  DO i = 1,N
2      ! treat main diagonal
3      tmp = get_value_of_A(i,i)
4      DO k = MAX(1,i-m),i-m+p
5          tmp = tmp - get_value_of_L(i,k)**2
6      ENDDO
7      DO k = MAX(1,i-MAX(1,p-1)),i-1
8          tmp = tmp - get_value_of_L(i,k)**2
9      ENDDO
10     tmp = SQRT(tmp)
11     CALL set_value_of_L(i,i,tmp)
12
13     ! treat secondary diagonals
14     DO j = i+1,i+MAX(1,p-1)
15         IF ( j > n ) EXIT
16         tmp = get_value_of_A(j, i)
17         DO k = MAX(1,i-m),i-m+p ! outer diagonals
18             tmp = tmp - get_value_of_L(j,k) * get_value_of_L(i,k)
19         ENDDO
20         DO k = MAX(1,i-MAX(1,p-1)),i-1 ! secondary diagonals
21             tmp = tmp - get_value_of_L(j,k) * get_value_of_L(i,k)
22         ENDDO
23         tmp = tmp / get_value_of_L(i,i)
24         CALL set_value_of_L(j, i, tmp)
25     ENDDO
26
27     ! treat outer diagonals
28     DO m_j = m_i+m_m-p,m_i+m_m
29         ! same loop body as before
30     ENDDO
31 ENDDO
```

Calculating the ICC(p) decomposition with regard to the arrays of the ICC stencil is then accomplished by the implementation of (6) and a simple transformation that maps an element $e_{i\,j}$ of a matrix to the according stencil at (x, y) and vice versa. To restrict the decomposition to a fill-in of level p and to avoid unnecessary calculations, only the non-zero elements according to (7) are considered as displayed in Algorithm 2.

4.2. **Applying the ICC(p) Preconditioner.** For a given residual $r \in \mathbb{R}^N$, where $r(i, j)$ with $i = 1, \ldots, m$ and $j = 1, \ldots, n$ presents the component at the grid point (i, j), we can now define the forward and backward substitution step in terms of the ICC stencil. In the following, the arms `ICC_W`, `ICC_S` and the inverse of the central element `ICC_C` of the ICC stencil will be abbreviated with W, S and C respectively.

The forward substitution for $Ly = r$, more precisely

$$y_i = \frac{1}{l_{i\,i}} \left(r_i - \sum_{k=1}^{i-1} l_{i\,k} y_k \right),$$

Algorithm 3 Applying the ICC(p) preconditioner

```
1  s_l = p + 1 ! length of southern arm of ICC stencil
2  w_l = MAX(1,p-1) ! length of western arm of ICC stencil
3
4  ! 1.) Forward substitution
5  ! first column
6  r(1,1) = r(1,1)*ICC_C(1,1)
7  DO i=2,m
8      r(i,1) = (r(i,1) - ICC_W(1,i,1)*r(i-1,1))*ICC_C(i,1)
9  ENDDO
10 ! all other columns
11 DO j = 2,n
12     DO i = 1,m
13         r(i,j) = ( r(i,j) - ICC_S(1:s_l,i,j)*r(i:i+s_l-1,j-1) &
14                 - ICC_W(1:w_l,i,j)*r(i-1:i-w_l:-1,j) )*ICC_C(i,j) ! dotprod
15     ENDDO
16 ENDDO
17
18 ! 2.) Backward substitution
19 ! all but first column
20 DO j = n,2,-1
21     DO i = m,1,-1
22         r(i,j) = r(i,j)*ICC_C(i,j)
23         r(i-1:i-w_l:-1,j) = r(i-1:i-w_l:-1,j) - r(i,j)*ICC_W(1:w_l,i,j) ! saxpy
24         r(i:i+p,j-1) = r(i:i+p,j-1) - r(i,j)*ICC_S(1:s_l,i,j) ! saxpy
25     ENDDO
26 ENDDO
27
28 ! first column
29 DO i = m,2,-1
30     r(i,1) = r(i,1)*ICC_C(i,1)
31     r(i-1,1) = r(i-1,1) - r(i,1)*ICC_W(1,i,1)
32 ENDDO
33 r(1,1) = r(1,1)*ICC_C(1,1)
```

for $i = 1, \ldots, N$, can then be transformed into a stencil formulation, that is

$$y(i,j) = C(i,j) \cdot \left(r(i,j) - \sum_{k=1}^{p+1} S(k,i,j) \cdot r(i-1+k, j-1) - \sum_{k=1}^{\beta} W(k,i,j) \cdot r(i-k,j) \right), \tag{8}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, where $\beta = \max(1, p-1)$. We traverse $y(i,j)$ in a cache-friendly way, so that the inner loop iterates over i, to ensure optimal cache utilization. Treating the case $j = 1$ separately without the second term on the right-hand-side of (8) eliminates unnecessary calculations for the first column where no S (resp. no outer diagonals) exist. The actual code is shown in the first part of Algorithm 3. It should be noted that the result $y(i,j)$ is saved in $r(i,j)$ again to avoid storing an additional array.

Backward substitution, $L^T x = y$, is more challenging to accomplish in a cache efficient way, because the first index of W and S now present columns in L^T. To overcome this, the substitution

is reordered to result in a column-wise operation. Instead of

$$x_i = \frac{1}{l_{i\,i}} \left(y_i - \sum_{k=i+1}^{n} l_{k\,i} x_k \right),$$

for $i = N, \ldots, 1$ we perform partial updates of x_i. We start by setting $x_i = y_i$ for all i. At first, the element x_N is updated to its final value by $x_N \leftarrow \frac{x_N}{l_{i\,i}}$. After this, the remaining x_i, $i = N-1, \ldots, 1$ get updated according to the N^{th} column by virtue of $x_i \leftarrow x_i - l_{k\,i} x_N$, which is an efficient saxpy operation. The whole process is now repeated for x_{N-1} until x_1.

This principle can be conveyed to the stencil formulation with some modifications. The saxpy operation can be split into two saxpy operations by S and W elements and is performed such that first $x(i,j)$ is multiplied by $C(i,j)$, then $x(i-1\!:\!i-\max(1,p-1)\!:\!-1,j)$ gets updated due to W followed by an update of $x(i\!:\!i+p,j+1)$ with respect to S. Applying this in the described column-wise fashion results in cache efficient chunk-wise updates of x as seen in line 20 to 26 of the Algorithm 3. Like in the forward substitution the values in the first column $x(i,1)$ need to be updated only with respect to W which is addressed in lines 29 to 33. Furthermore it should be noted that the code avoids conditional statements and dependent iteration variables at the expense of a little extra work to help vectorization. This can be seen for instance in line 14 for $i = 1$ and $j = 2$ the elements $r(0,2),\ r(-1,2),\ldots$ (which wrap to $r(m,1),\ r(m-1,1),\ldots$) are multiplied with 0 only entries in $W(1:\max(1,p-1),1,1)$. Using the Hardware Performance Monitor (HPM) library we could determine that the cache hit rate of our implementation is 99.945%.

5. Numerical Experiments

Projections of future climate changes depend strongly on the accuracy of the simulated ocean circulation, therefore a realistic description of oceanic processes is required. A major factor which influences the accuracy is the resolution of the mesh.

For the simulations in IPCC AR4, that was published in 2007, a grid with a resolution of 256 grid points in longitude by 220 points in latitude was used. A limiting factor for increasing the resolution is the convergence of meridians at the North Pole, which is a source of numerical instabilities. For this reason, in the current calculations for the IPCC AR5, report a new type of grid has been introduced, the *tripolar* grid [10]. Here the North Pole is no longer a single point, but it has been expanded to a line, resulting in two poles on the northern hemisphere and one in the southern hemisphere. The resolution for the IPCC AR5 runs is 0.4 deg (802×404 grid points). Nevertheless, basin-scale ocean models with a resolution of about 0.1 deg or higher are found to produce more realistic simulations. The main reason for such high resolution is the necessity for a proper representation of meso-scale ocean eddies which play a crucial role in determining the mean flow.

For the experiments presented in this work, we used a tripolar model with a resolution of 0.1 deg (3602×2394 grid points) which yields a system of about 8.6 million equations. To measure the actual runtime the function call to the solver of the barotropic system is surrounded by calls to MPI_Wtime. The SOR solver uses a fixed number of 1200 iterations and reaches a relative residual with the order of magnitude 10^{-11}. As start value the null vector was chosen for all tests. The SOR relaxation parameter in all benchmarks was set to 1.934, which was hand tuned by a large number of test calculations. In order to compare the CG method with SOR we set it to reach the same order of magnitude in the relative residual and combined it with several preconditioners. In Table 1 to 3 SOR denotes the traditional red-black ordered SOR method in MPIOM and CG the new implemented conjugate gradient method with no preconditioning. The following entries denote the PCG method combined with a preconditioner. In detail, ILU(0)

32×16 cores			
solver	#iterations	runtime [ms]	speedup
SOR	1200.0	253.3	1.00
CG	1844.9	656.3	0.39
ILU(0)	371.2	371.0	0.68
SSOR	206.4	221.9	1.14
ICC(0)	371.2	429.4	0.59
ICC(1)	240.2	289.5	0.87
ICC(2)	207.3	257.1	0.99
ICC(4)	147.4	212.5	1.19
ICC(6)	133.8	216.1	1.17
ICC(8)	128.6	228.8	1.11

TABLE 1. The number of iterations and runtime of red-black SOR and CG with different preconditioners and fill-in levels needed to reach a relative residual of 10^{-11} with startvalue 0, performed on 16 nodes.

denotes the incomplete LU decomposition with zero fill-in according to [11, p. 303], SSOR the symmetric SOR method and ICC(p) the incomplete Cholesky decomposition with p fill-in. Each solver was used to solve the barotropic subsystem in 30 different time steps to get an average number of necessary iterations and an average runtime in milliseconds (ms). This benchmark was repeated three times and averaged to compensate for effects like non-optimal process distribution in the cluster. All averages were calculated with the arithmetic mean. The setup time for the preconditioners and the SOR parameter was neglected. The partitioning is always given as $x \times y$ meaning x partitions in zonal and y partitions in meridional direction with one core per partition.

All benchmarks were performed on the new DKRZ high-performance supercomputer named Blizzard. The Blizzard cluster is an IBM p575 POWER6 system consisting of 264 nodes with 16 dual core CPUs per node, hence reaching a total of 8448 cores. Each core has a peak performance of 18.8 Gigaflop/s giving a total system peak performance of 158 Teraflop/s. The aggregate bandwidth of the InfiniBand Fat CLOS Tree interconnect is 7.6 Terabyte/s. A detailed description of the IBM POWER6 microarchitecture can be found in [8]. For our benchmarks Symmetric Multithreading (SMT) was disabled because experience has shown that for symmetric memory access patterns, as they are found in many numerical simulations, SMT has at best no benefit. This concludes that on one node a total of 32 cores with 32 MPI processes were exclusively taken.

On the software side the IBM xlf FORTRAN compiler in version 12.1.0.3 running on IBM AIX 6.1.3.0 with IBM's Parallel Envorinment in version PE 5.1.1.5 (which includes POE/MPI) and disabled OpenMP support was used to compile and run MPIOM. The important compiler flags which were used are `O3` and `qhot`. These flags promise to enhance the performance of the code. The drawback is that the optimization is very aggressive which alters the results of MPIOM. For this reason the `qstric`, `qxflag=nvectver` (suppresses vector versioning) and `qxflag=nsmine` (suppresses strip mining) are needed in order to get correct results.

6. Conclusion and Perspectives

From analyzing the presented results we can conclude that CG with an appropriate preconditioner like ICC(p) greatly reduces the number of necessary iterations needed to solve the barotropic subsystem. Since the communication overhead is a major limiting factor, this results in a shorter overall runtime compared to SOR. One should note that one iteration of CG needs three communications: one for the matrix-vector multiplication and two for calculating dot

32×32 cores			
solver	#iterations	runtime [ms]	speedup
SOR	1200.0	222.2	1.00
CG	1844.8	541.2	0.41
ILU(0)	375.0	232.7	0.95
SSOR	219.0	145.4	1.53
ICC(0)	375.0	256.3	0.87
ICC(1)	245.5	184.2	1.21
ICC(2)	213.7	161.3	1.38
ICC(4)	156.3	133.5	1.66
ICC(6)	141.5	129.9	1.71
ICC(8)	136.2	142.3	1.56

TABLE 2. The number of iterations and runtime of red-black SOR and CG with different preconditioners and fill-in levels needed to reach a relative residual of 10^{-11} with startvalue 0, performed on 32 nodes.

64×32 cores			
solver	#iterations	runtime [ms]	speedup
SOR	1200.0	144.2	1.00
CG	1844.8	475.0	0.30
ILU(0)	379.9	163.4	0.88
SSOR	244.2	115.4	1.25
ICC(0)	379.9	181.0	0.80
ICC(1)	251.8	130.7	1.10
ICC(2)	220.9	106.0	1.36
ICC(4)	165.6	092.6	1.56
ICC(6)	151.4	093.1	1.55
ICC(8)	146.1	092.1	1.56

TABLE 3. The number of iterations and runtime of red-black SOR and CG with different preconditioners and fill-in levels needed to reach a relative residual of 10^{-11} with start value 0, performed on 64 nodes.

products. Consequently the number of iterations in CG needs to be significantly lower than 400 iterations to make up for the single communication that SOR needs per iteration. The decisive factor in the communication is the latency for which reason the two-halo-boundary exchange of SOR is almost double as fast as two consecutively performed one-halo-boundary exchanges.

On a smaller setup as shown in Table 1 the maximum speedup of CG compared to SOR is 1.19 in case of ICC(4). When doubling the number of nodes as in Table 2 the maximum speedup increases to 1.71 because CG with ICC(6) scales much better than SOR. The runtime of SOR is decreased by $12\% - 13\%$ while in case of CG with ICC(6) it is decreased by 40%. Doubling the number of nodes again, we can see in Table 3 that the runtime of SOR is decreased by 35% against 28% with ICC(6). In this setup the scalability of CG with ICC(6) saturates because each partition is already as small as 56×75 grid points leaving the communication overhead the highest fraction of the runtime. Additionally the number of iterations increases gradually over all setups, since more and more couplings are dropped due to smaller partitions. Still CG with

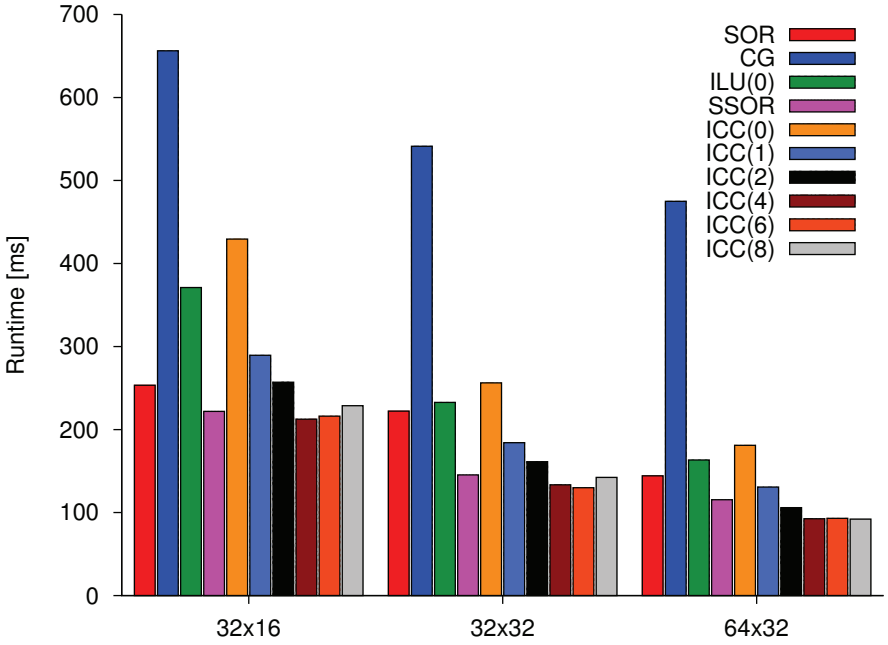


FIGURE 4. Comparison of the runtime of red-black SOR and CG with different preconditioners and fill-in levels for 32×16, 32×32 and 64×32 partitions.

ICC(6) performs about 36% faster than SOR on the largest setup which is an overly promising result. Figure 4 summarizes the runtime of all solvers and preconditioners on the different setups.

Although the actual runtime to solve the barotropic subsystem is already a fraction of a second there is still much need for further improvement. Considering the fact that one simulated scenario for the IPCC AR5 spans a simulation time of 100 years, resulting in millions of time steps where each has a barotropic subsystem to solve, one can easily see the leverage effect we are dealing with. The bright side of solving one linear system with varying right-hand-sides many times is that the setup time of a preconditioner or solver can be neglected.

Currently we are working on the implementation of the *Chebyshev iteration* that requires no inner products and is therefore particularly suited for massively parallel computers with significant communication cost [6]. It has the same theoretical upper bound for the rate of convergence as CG given that a good estimation of the eigenvalues λ_{min} and λ_{max} of the preconditioned system is known. Calculating these, especially λ_{min}, is in general more complex than solving a linear system. With the aforementioned consideration, a considerable amount of work to approximate these eigenvalues pays off later because of the sheer number of time steps.

In the future more complex models with higher accuracies will be needed to better supplement the research in climatology. This development will go hand in hand with a continuously raising demand for better mathematical methods and algorithms on high-performance hardware to make more complex simulations feasible.

ACKNOWLEDGEMENT

The authors greatly thank M. Puetz from IBM for his valuable inputs with respect to the code optimization on the IBM POWER6. Further thanks go to J. Behrens, I. Fast, T. Jahns from DKRZ and H. Haak, E. Maier-Reimer from MPI-M for their support and insightful remarks associated with MPIOM.

References

[1] G. Alefeld, I. Lenhardt, and H. Obermaier, *Parallele numerische Verfahren*, Springer, Berlin, Germany, 2002.

[2] A. Arakawa and V. Lamb, *Computational design of the basic dynamical processes of the ucla general circulation model*, Methods in Computational Physics, 17 (1977), pp. 173–265.

[3] M. I. Beare and D. P. Stevens, *Optimisation of a parallel ocean general circulation model*, Annales Geophysicae, 15 (1997), pp. 1369–1377.

[4] R. Budich, *COSMOS network plan*, February 2008.

[5] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins series in the mathematical sciences, Johns Hopkins Univ. Pr., Baltimore, USA, 3. ed. ed., 1997.

[6] M. H. Gutknecht and S. Röllin, *The chebyshev iteration revisited*, Parallel Comput., 28 (2002), pp. 263–283.

[7] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[8] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, *IBM POWER6 microarchitecture*, IBM J. Res. Dev., 51 (2007), pp. 639–662.

[9] S. Marsland, H. Haak, J. Jungclaus, M. Latif, and F. Röske, *The max-planck-institute global ocean/sea ice model with orthogonal curvilinear coordinates*, Ocean Modelling, 5 (2003), pp. 91–127.

[10] R. J. Murray, *Explicit generation of orthogonal grids for ocean models*, J. Comput. Phys., 126 (1996), pp. 251 – 273.

[11] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2. ed. ed., 2003.

[12] E. Simonnet, T. T. Medjo, and R. Temam, *Barotropic-baroclinic formulation of the primitive equations of the ocean*, Applicable Analysis, 82 (2003), pp. 439 – 456.

[13] J. Wolff, E. Maier-Reimer, and S. Legutke, *The hamburg ocean primitive equation model hope*, tech. rep., DKRZ Hamburg, 1997.

Preprint Series of the Engineering Mathematics and Computing Lab

recent issues

No. 2011-01 Hartwig Anzt, Maribel Castillo, Juan C. Fernández, Vincent Heuveline, Rafael Mayo, Enrique S. Quintana-Ortí, Björn Rocker: Power Consumption of Mixed Precision in the Iterative Solution of Sparse Linear Systems

No. 2010-07 Werner Augustin, Vincent Heuveline, Jan-Philipp Weiss: Convey HC-1 Hybrid Core Computer – The Potential of FPGAs in Numerical Simulation

No. 2010-06 Hartwig Anzt, Werner Augustin, Martin Baumann, Hendryk Bockelmann, Thomas Gengenbach, Tobias Hahn, Vincent Heuveline, Eva Ketelaer, Dimitar Lukarski, Andrea Otzen, Sebastian Ritterbusch, Björn Rocker, Staffan Ronnås, Michael Schick, Chandramowli Subramanian, Jan-Philipp Weiss, Florian Wilhelm: HiFlow^3 – A Flexible and Hardware-Aware Parallel Finite Element Package

No. 2010-05 Martin Baumann, Vincent Heuveline: Evaluation of Different Strategies for Goal Oriented Adaptivity in CFD – Part I: The Stationary Case

No. 2010-04 Hartwig Anzt, Tobias Hahn, Vincent Heuveline, Björn Rocker: GPU Accelerated Scientific Computing: Evaluation of the NVIDIA Fermi Architecture; Elementary Kernels and Linear Solvers

No. 2010-03 Hartwig Anzt, Vincent Heuveline, Björn Rocker: Energy Efficiency of Mixed Precision Iterative Refinement Methods using Hybrid Hardware Platforms: An Evaluation of different Solver and Hardware Configurations

No. 2010-02 Hartwig Anzt, Vincent Heuveline, Björn Rocker: Mixed Precision Error Correction Methods for Linear Systems: Convergence Analysis based on Krylov Subspace Methods

No. 2010-01 Hartwig Anzt, Vincent Heuveline, Björn Rocker: An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations

No. 2009-02 Rainer Buchty, Vincent Heuveline, Wolfgang Karl, Jan-Philipp Weiß: A Survey on Hardware-aware and Heterogeneous Computing on Multicore Processors and Accelerators

No. 2009-01 Vincent Heuveline, Björn Rocker, Staffan Ronnas: Numerical Simulation on the SiCortex Supercomputer Platform: a Preliminary Evaluation