Applied Mathematical Sciences, Vol. 7, 2013, no. 61, 3007 - 3021
HIKARI Ltd, www.m-hikari.com

A Smart GPU Implementation of an Elliptic Kernel for an Ocean Global Circulation Model

R. Farina

Centro Euro-Mediterraneo sui Cambiamenti Climatici
Bologna, Viale Aldo Moro 44, Italy
raffaele.farina@cmcc.it

S. Cuomo, P. De Michele and F. Piccialli

Department of Mathematics and Applications, University of Naples "Federico II"
Via Cinthia, 80126, Napoli, Italy
{salvatore.cuomo, pasquale.demichele}@unina.it francesco.piccialli@gmail.com

Copyright © 2013 R. Farina et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

In this paper, the preconditioning technique of an elliptic Laplace problem in a global circulation ocean model is analyzed. We suggest an inverse preconditioning technique in order to efficiently compute the numerical solution of the elliptic kernel. Moreover, we show how the convergence rate and the performance of the solver are strictly linked to the discretized grid resolution and to the Laplace coefficients of the oceanic model. Finally, we present an easy-to-implement version of the solver on the Graphics Processing Units (GPUs).

Mathematics Subject Classification: 65Y05; 65Y10; 65F08; 65F35; 37N10

Keywords: ocean modelling, preconditioning technique, GPU programming, scientific computing

1 Introduction

The ocean plays a crucial role in regulating the global climate balance of the Earth: it can absorb heat and distribute it to the world through the currents and interactions

with the atmosphere. In recent years, ocean numerical models have became enough realistic as a result of many improved methods, fast computing and global data sets. The rapidly advancing field of numerical ocean circulation modelling is supported by several models and methods as NEMO, Hops, MOM, POP, et al. (see [12] for a nice review). However, several of these numerical models are not yet optimized in terms of ad-hoc preconditioning techniques and smart code implementations. In all these frameworks the numerical kernel is represented by the discretization of the Navier-Stokes equations [13] on a three dimensional grid and by the computation of the evolution time of each variable for each grid point. The high resolution computational grid requires efficient preconditioning techniques for improving the accuracy in the computed solution and parallelization strategies for overcoming to the huge amount of computational demand.

In this paper, for NEMO-OPA ocean model [15], a state of the art modelling framework in the oceanographic research, we propose a preconditioning technique based on inverse formulation [4]. The preconditioning is often a bottleneck in solving linear systems efficiently and it is well established that a suitable preconditioner increases significantly the performance of an application.

The NEMO-OPA elliptic sea-surface equation is originally solved by means of the diagonal Preconditioned Conjugate Gradient (PCG). Here, we prove that, in most cases, it works in a inefficiently and inaccurately way. Moreover, we implement the solver on a Graphics Processor Unit (GPU) and adopt the Scientific Computing libraries, in order to efficiently compute the linear algebra operations. GPUs are massively parallel architectures that require a deep understanding of the underlying computing architecture and programming on these devices involves a massive rethinking of existing CPU based applications [17]. In this paper, we present an easy-to-implement version of the elliptic solver, by using the Compute Unified Device Architecture (CUDA) framework [16]. We develop a code by using CUDA based supported libraries CUBLAS [7] and CUSPARSE [10] for the sparse linear algebra and the graph computations. The library GPU based approach allows a quick code development times and an easy to use GPUs implementations that can fruitfully speed up the expensive numerical kernel of an oceanographic simulator.

The paper is set out as follows. In section 2 we briefly review the mathematical model: elliptic equations that are at the heart of the model. In section 3, the preconditioned conjugate gradient method, used to invert the elliptic equations, are described. In section 4, we outline an implementation strategy for solving the elliptic solver by using standard libraries and, moreover, in section 5 we discuss the mapping of our algorithm onto a massively parallel machine. Finally, the conclusions are drawn.

2 The mathematical model

Building and running ocean models able to simulate the world of global circulation with great realism require many scientific skills. In modelling the general ocean circulation it is necessary to solve problems of elliptic nature. Often, solving this kind of problems becomes difficult [12]. In particular, the elliptic solvers have a low convergence rate due to several problem dependent topics, i.e. complex geometry and topography, space-time scales variability. Moreover, the elliptic solver implementations do not fit properly onto parallel and scientific computing systems [11, 19, 20]. In the NEMO-OPA numerical code the primitive equations are discretized within sea-surface hypothesis [18] and the model is characterized by the three-dimensional distribution of currents, potential temperature, salinity, pressure and density [13]. The numerical method NEMO-OPA is grounded on discretizing of the primitive equation - by using of finite differences on a three dimensional grid - and computing the time evolution to each variable "ocean" at each grid point for the entire globe [1]. A sketch of the NEMO-OPA computational model (Figure 1) shows the complex dynamic processes that mimic the ocean circulation model, composed by steps that are many time simulated.

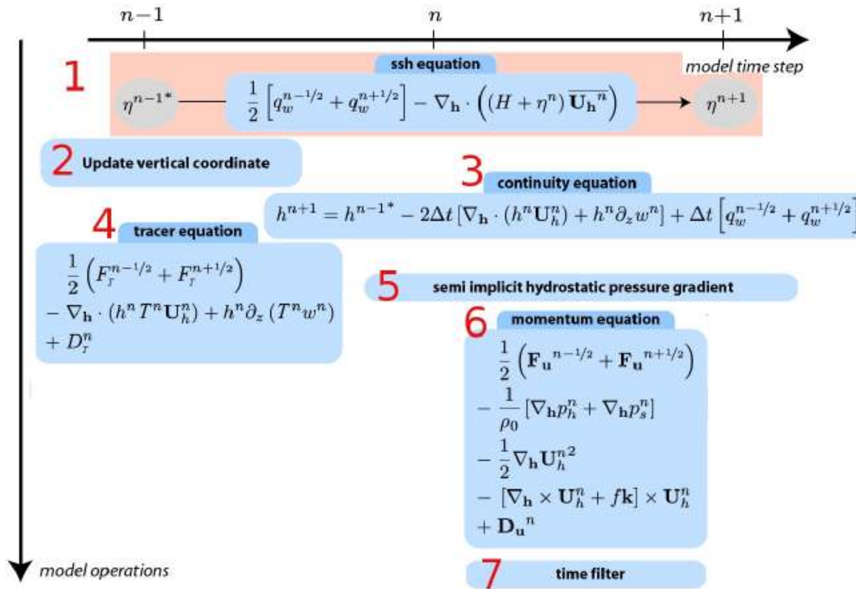

Figure 1: NEMO-OPA model.

The kernel algorithm (highlighted with pink color, step 1) solves the sea-surface height equation η. The elliptic kernel is discretized by the following semi-discrete equations

$$\eta^{n+1} = \eta^{n-1} - 2\Delta t D^n \tag{1}$$

$$2\Delta t g T_c \Delta_h D^{n+1} = D^{n+1} - D^{n-1} + 2\Delta t g \Delta_h \eta^n \tag{2}$$

$$\Delta_h = \nabla\left[(H+\eta^n)\nabla\right] \tag{3}$$

where η^n (with $n \in \mathbb{N}$), which describes the shape of the air-sea interface, is the sea-surface height at the n-th step of the model, D^n is the centered difference approximation of the first time derivative of η, Δt is the time stepping, g is the gravity constant, T_c is a physical parameter, Δ_h is the horizontal Laplacian operator and H is the depth of the ocean bottom [15]. Whereas the domain of the ocean models is the Earth sphere (or part of it), the model uses the geographical coordinates system (λ,ϕ,r). In this system, a position is defined by the latitude ϕ, the longitude λ and the distance from the center of the earth $r = a+z(k)$, where a is the Earth's radius and z the altitude above a reference sea level. The local deformation of the curvilinear geographical coordinate system is given by e_1,e_2 and e_3:

$$e_1 = rcos\phi, \quad e_2 = r, \quad e_3 = 1. \tag{4}$$

The Laplacian Operator in spherical coordinates $\Delta_h D^{n+1}$ in (2) becomes

$$\Delta_h D^{n+1} = \frac{1}{e_1 e_2}\left[\frac{\partial}{\partial i}\left(\alpha(\phi)\frac{\partial D^{n+1}}{\partial i}\right)+\frac{\partial}{\partial j}\left(\beta(\phi)\frac{\partial D^{n+1}}{\partial i}\right)\right] \tag{5}$$

where:

$$\alpha(\phi) = (H+\eta^n)e_2/e_1 \qquad \beta(\phi) = (H+\eta^n)e_1/e_2 \tag{6}$$

For the functions $\alpha(\phi)$ in and $\beta(\phi)$ in (6), we have the following relations:

$$\lim_{\phi \longrightarrow \pm\frac{\pi}{2}} \alpha(\phi) = +\infty \quad \wedge \lim_{\phi \longrightarrow \pm\frac{\pi}{2}} \beta(\phi) = 0 \tag{7}$$

From (7), if we choose $M,\varepsilon \in \mathbb{R}$ with $M >> \varepsilon$, then exists an interval $\left[\frac{\pi}{2}-\delta,\frac{\pi}{2}\right]$ or $\left[-\frac{\pi}{2},\ -\frac{\pi}{2}+\ \delta\right]$, such that the following inequality holds:

$$\alpha(\phi) > M >> \varepsilon > \beta(\phi) \tag{8}$$

In physical terms, in the proximity of the geographical poles, $(\lambda,\pm\phi/2,r)$, there are several orders of magnitude between the functions $\alpha(\phi)$ and $\beta(\phi)$. The result (8) will significantly influence the rate of convergence in the iterative solver.

3 Inverse preconditioning techniques

Let us now consider the elliptic NEMO model [15] defined by the coefficients

$$\begin{aligned} C^{NS}_{i,j} &= 2\Delta t^2 H(i,j)e_1(i,j)/e_2(i,j) \\ C^{EW}_{i,j} &= 2\Delta t^2 H(i,j)e_2(i,j)/e_1(i,j) \\ b_{i,j} &= \delta_i(e_2 M_u) - \delta_j(e_1 M_v) \end{aligned} \tag{9}$$

where δ_i and δ_j are the discrete derivative operators along the axes **i** and **j**. The discretization of the equation (2) by means of a five-point finite difference method gives:

$$\begin{aligned} &C_{i,j}^{NS} D_{i-1,j} + C_{i,j}^{EW} D_{i,j-1} - \big(C_{i+1,j}^{NS} + C_{i,j+1}^{EW} + C_{i,j}^{NS} + \\ &+ C_{i,j}^{EW}\big) D_{i,j} + C_{i,j+1}^{EW} D_{i,j+1} + C_{i+1,j}^{NS} D_{i+1,j} = b_{i,j} \end{aligned} \tag{10}$$

where the equation (10) is a symmetric system of linear equations. All the elements of the sparse matrix **A** vanish except those of five diagonals. With the natural ordering of the grid points (i.e. from west to east and from south to north), the structure of **A** is a block-tridiagonal with tridiagonal or diagonal blocks. The matrix **A** is a positive-definite symmetric matrix with $n = jpi \times jpj$ size, where jpi and jpj are respectively the horizontal dimensions of the grid discretization of the domain.
The Conjugate Gradient Method is a very efficient iterative method for solving the linear system (10) and it provides the exact solution in a number of iterations equal to the size of the matrix. The convergence rate is faster as the matrix is closer to the identity one. By spectral point of view a convergence relation between the solution of the linear system and its approximation x_m is given by

$$\|\mathbf{x} - \mathbf{x}_m\|_A < 2\left(\frac{\sqrt{\mu_2(A)} - 1}{\sqrt{\mu_2(A)} + 1}\right)^{m-1} \|\mathbf{x} - \mathbf{x}_0\|_A. \tag{11}$$

In (11), $\mu_2(A) = \lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are, respectively, the greatest and the lowest eigenvalue of **A**, and $\|\cdot\|_A$ is the A-norm. The preconditioning framework is to introduce a matrix **M**, that is an approximation of **A** easier to invert, and to solve the equivalent linear system

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}. \tag{12}$$

The ocean global model NEMO-OPA uses the diagonal preconditioner, where **M** is chosen to the diagonal of **A**. Let us introduce the cardinal coefficients

$$\alpha_{i,j}^{E} = {C_{i,j+1}}^{EW}/d_{i,j} \qquad \alpha_{i,j}^{W} = {C_{i,j}}^{EW}/d_{i,j} \tag{13}$$

$$\beta_{i,j}^{S} = {C_{i,j}}^{NS}/d_{i,j} \qquad \beta_{i,j}^{N} = {C_{i+1,j}}^{NS}/d_{i,j} \tag{14}$$

where $d_{i,j} = \big(C_{i+1,j}^{NS} + C_{i,j+1}^{EW} + C_{i,j}^{NS} + C_{i,j}^{EW}\big)$ represents the diagonal of the matrix **A**. The (10), using the diagonal preconditioner, can be written as

$$-\beta_{i,j}^{S} D_{i-1,j} - \alpha_{i,j}^{W} D_{i,j-1} + D_{i,j} - \alpha_{i,j}^{E} D_{i,j+1} - \beta_{i,j}^{N} D_{i+1,j} = \bar{b}_{i,j} \tag{15}$$

with $\bar{b}_{i,j} = -b_{i,j}/d_{i,j}$. Starting from the observations (7) and (8) we proof that the diagonal preconditioner does not work very well in some critical physical situations involving curvilinear spherical coordinates.

Proposition 3.1 *In the geographical coordinate, if* $\phi \to +\frac{\pi}{2}^{-}$, $\Delta\lambda \to 0$, $\Delta\phi \to 0$ *then the conditioning number* $\mu(\mathbf{M^{-1}A})$ *goes to* $+\infty$.

Proof. In the geographical coordinate, i.e. when $(i, j) \to (\lambda, \phi)$ and $(e_1, e_2) \to (r\cos\phi, r)$, for $\phi \to +\frac{\pi}{2}^{-}$, $\Delta\lambda \to 0$ and $\Delta\phi \to 0$, the functions α^W and α^E in (13) go to $-1/2$ while β^N and β^S in (14) go to 0. Hence, the eigenvalues of the limit of matrix $\mathbf{M^{-1}A}$ are

$$\lambda_k = 1 + \cos\left(\frac{k\pi}{n+1}\right) \quad k = 1, ..., n \tag{16}$$

then the condition number $\mu_2(M^{-1}A) = \lambda_{max}/\lambda_{min} \approx n^2/2$ (by using the series expansion of $\cos x = 1 - x^2/2 + o(x^2)$). Moreover, for $\Delta\lambda \to 0$ and $\Delta\phi \to 0$, the size n of the matrix $\mathbf{A}$ goes to $+\infty$, hence we obtain the thesis. ■

By the Proposition (3.1), for n large and $\phi \to \pm\pi/2$, it is preferable to adopt more suitable preconditioning techniques or a strategy based on the local change of the coordinates at poles. In this paper we propose an approximate sparse inverse preconditioning techniques [4] for the linear system (10). The problem is how to build a preconditoner that preserves the sparse structure. We introduce a Factored Sparse Approximate Inverse (FSAI) preconditioner $\mathbf{P} = \mathbf{\tilde{Z}\tilde{Z}^t}$ [5, 3], computed by means of a conjugate-orthogonalization procedure. Specifically, we propose an "ad hoc" method for computing an incomplete factorization of the inverse of the matrix $\mathbf{T} \subset \mathbf{A}$, obtained by $\mathbf{A}$ taking only the elements $a_{i,j}$ such that $|j - i| \le 1$. The factorized sparse approximate inverse of $\mathbf{T}$ is used as explicit preconditioner for (10). In the following, we give several remarks for the sparsity pattern selection S of our inverse preconditioner $\mathbf{P}$.

Proposition 3.2 *If* $\mathbf{T}$ *is a tridiagonal, symmetric and diagonally dominant matrix, with diagonal elements all positive* $t_{k,k} > 0$, $k = 1, ..., n$, *then the Cholesky's factor* $\mathbf{U}$ *of the matrix* $\mathbf{T}$ *is again diagonally dominant.*

Proof. Since $\mathbf{T}$ is a tridiagonal matrix, then $\mathbf{U}$ is a bidiagonal matrix. Using the inductive method we proof that $\mathbf{U}$ is diagonally dominant matrix. For $k = 1$ is trivially, indeed by hypothesis we know that $|a_{1,1}| > |a_{1,2}| \Longleftrightarrow |u_{1,1}^2| > |u_{1,1}u_{1,2}|$, then we obtain $|u_{1,1}| > |u_{1,2}|$. Moreover, placed the thesis true for $k-1$ i.e. $|u_{k-1,k-1}| > |u_{k-1,k}|$ then, by the following inequalities,

$$\begin{aligned} |a_{k,k}| > |a_{k,k-1}| + |a_{k,k+1}| \Longleftrightarrow |u_{k-1,k}^2 + u_{k,k}^2| > |u_{k-1,k}u_{k-1,k-1}| + \\ + |u_{k,k}u_{k,k+1}| > u_{k-1,k}^2 + |u_{k,k}u_{k,k+1}| \end{aligned} \tag{17}$$

and subtracting the inequality (17) for $u_{k-1,k}^2$, the thesis also holds for k. ■

This result allows to prove the following proposition:

Proposition 3.3 *The inverse matrix* $\mathbf{Z}$ *of a bidiagonal and diagonally dominant matrix* $\mathbf{U}$ *has column vectors* $\mathbf{z}_k, k = 1, ...n$ *such that, starting from diagonal element* $z_{k,k}$*, they contain a finite sequence* $\{z_{k-i,k}\}_{i=0,...,k-1}$ *strictly decreasing.*

Proof. Applying a backward substitution procedure for solving the system of equations $\mathbf{U}\mathbf{z}_k = \mathbf{e}_k$, we get:

$$z_{k-i,k} = \begin{cases} \frac{1}{u_{k,k}} & if \;\; i = 0 \\ \frac{(-1)^i}{u_{k,k}} \cdot \prod_{r=1}^{i} \left(\frac{u_{k-r,k-r+1}}{u_{k-r,k-r}}\right) & if \;\; 0 < i \leq k-1 \end{cases} \tag{18}$$

By means of the preposition (3.2) we obtain that $z_{k-i,k} > z_{k-i-1,k}$ with $0 < i \leq k-1$ and hence the thesis is proved. ■

The previous propositions (3.2) and (3.3) enable to select a sparsity pattern S by meand of the scheme in Algorithm 1.

Algorithm 1 Sparsity pattern selection.

Step 1. Consider the symmetric, diagonally dominant and triangular matrix $\mathbf{T}$, obtained by $\mathbf{A}$ taking only the elements $a_{i,j}$ such that $|j-i| \leq 1$.

Step 2. Since $\mathbf{T} = \mathbf{U}^T\mathbf{U}$ is a diagonally dominant matrix, its Cholesky factor $\mathbf{U}$ is diagonally dominant (Proposition (3.2)).

Step 3. $\mathbf{U}$ is a bidiagonal and diagonally dominant matrix. $\mathbf{Z} = \mathbf{U}^{-1}$ has columns vector $\mathbf{z}_k$, $k = 1, ..., n$ such that $z_{k-i,k} > z_{k-i-1,k}$ with $0 < i \leq k-1$ (Proposition (3.3)).

Step 4. Fixed an upper bandwidth q, the entries $z_{i,j}$ with $j > i+q$ of $\mathbf{Z}$ are considered negligible.

Step 5. The preconditioner $\mathbf{P} = \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^{\mathbf{t}}$ is built as:

$$\tilde{z}_{i,j} = \begin{cases} z_{i,j} & if \;\; j \leq i+q \\ 0 & if \;\; j > i+q \end{cases} \tag{19}$$

Step 6. The sparse factor $\tilde{\mathbf{Z}}$ is computed by T-orthogonalization procedure posing the sparsity pattern S=$\{(i,j) \;/\, j > i+q\}$.

$\mathbf{T}$ is a diagonally dominant matrix, then the incomplete inverse factorization of $\mathbf{T}$ exists for any choice of the sparsity pattern S on $\mathbf{Z}$ [5]. From a computational point of view, the T-orthogonalization procedure with the sparsity pattern S is based on matrix-vector operations with computational cost of $5(q+1)$ floating point operations. Moreover, for each column vector $\tilde{\mathbf{z}}_k$ of $\tilde{\mathbf{Z}}$ we work only on its $q+1$ components $\tilde{z}_k[k-q], \tilde{z}_k[k-q+1], \ldots, \tilde{z}_k[k]$ with consequently global complexity of $5q(q+1)O(n)$. We observe that in this work we do not justify the choice of the best preconditioning technique among the many possible ones. The inverse

preconditioning techniques are interesting due to the fact that their application involves only sparse matrix-vector products, which can be executed very efficiently on emergent multicore- and GPU-based architectures.

4 Library based GPU implementation

In this section we show that, by means of a GPU CUDA library based approach, it is possible to develop scalable and optimized numerical solvers for free. The matrices $\mathbf{A}$, $\tilde{\mathbf{Z}}$ and $\tilde{\mathbf{Z}^T}$ are stored with the special storage format Compressed Sparse Row (CSR), since this format is compatible with the NEMO-OPA software. The FSAI is performed in serial on the CPU and its building requires a negligible time on total execution of the elliptic solver. We show the implementation of the Algorithm 2 outlines on the GPUs [2, 9, 8]. In details, our solver is implemented by means

Algorithm 2 FSAI-PCG solver

1: $k = 0$; $\mathbf{x}_0 = D^0_{i,j} = 2D^{t-1}_{i,j}$, the initial guess;
2: $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$;
3: $\mathbf{s}_0 = \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^{\mathbf{t}}\mathbf{r}_0$, with $\mathbf{P} = \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^t$ the FSAI preconditioner;
4: $\mathbf{d}_0 = \mathbf{s}_0$;
5: **while** $\left(\|\mathbf{r}_k\|/\|\mathbf{b}\| > \varepsilon \ .and.\ k \leq n\right)$ **do**
6:   $\mathbf{q}_k = \mathbf{A}\mathbf{d}_k$; $\alpha_k = (\mathbf{s}_k, \mathbf{r}_k)/(\mathbf{d}_k, \mathbf{q}_k)$;
7:   $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$; $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{q}_k$; $\mathbf{s}_{k+1} = \tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^{\mathbf{t}}\mathbf{r}_{k+1}$;
8:   $\beta_k = (\mathbf{s}_{k+1}, \mathbf{r}_{k+1})/(\mathbf{s}_k, \mathbf{r}_k)$; $\mathbf{d}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{d}_k$;
9:   $k = k + 1$;
10: **end while**

of the CUDA language with the auxiliary linear algebra libraries CUBLAS, for the "dot product" (`xDOT`), "combined scalar multiplication plus vector addition" the (`xAXPY`), the "euclidean norm" (`xNRM2`) and the "vector by a constant scaling" (`xSCAL`) operations, and CUSPARSE for the sparse matrix-vector operations in the PCG solver. The linear algebra scientific libraries are extremely helpful to easily implement a software on the GPU architecture. In the following, we show how to implement the Algorithm 2 outlines by using library features. In order to use the CUBLAS library it is necessary to inizialize it by means of the `cublasInit()` function and execute the following statements for the initialization, creation and setup of a matrix descriptor:

```
cusparseHandle_t handle=0; cusparseCreate(&handle);
cusparseMatDescr_t descra=0; cusparseCreateMatDescr(&descra);
cusparseSetMatType(descra, CUSPARSE_MATRIX_TYPE_GENERAL);
cusparseSetMatIndexBase(descra, CUSPARSE_INDEX_BASE_ZERO);
```

The libraries utilization avoids to configure the grid of the thread blocks and it allows to write codes in a very fast way. For example, at line 6 of the Algorithm 2, the computation of $\mathbf{q}_k = \mathbf{A}\mathbf{d}_k$ is required, and this operation can be made simply by calling the CUSPARSE routine `cusparseScsrmv()`, that performs the operation $\mathbf{q} = a\mathbf{A}\mathbf{d} + b\mathbf{q}$ as follows:

```
cusparseScsrmv(handle, CUSPARSE_OPERATION_NON_TRANSPOSE, n, n,
                a, descra, A, start, j, d, b, q);
```

In our context, `A`, `j` and `start` represent the symmetric positive-definite matrix **A**, stored in the CSR format. More precisely, the vector `A` denotes the non-zero elements of the matrix **A**, `j` is the vector that stores the column indexes of the non-zero elements, the vector `start` denotes, for each row of the matrix, the address of the first non-zero element and `n` represents the row and columns number of the square matrix **A**. The constants a and b are assigned to 1.0 and 0.0 respectively. Moreover, it happens that at line 6 of the Algorithm 2, the computation of $(\mathbf{s}_k, \mathbf{r}_k)$ is performed by means the CUBLAS routine for the dot product:

```
alfa_num = cublasSdot(n, s, INCREMENT_S, r, INCREMENT_R);
```

The constants `INCREMENT_S` and `INCREMENT_R` are both assigned to 1. Last operation of line 6 in Algorithm 2, is the $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ for updating the solution and it is implemented by calling the CUBLAS routine for the saxpy operation:

```
cublasSaxpy(n, alfa, d, INCREMENT_D, x, INCREMENT_X);
```

The constants `INCREMENT_D` and `INCREMENT_X` are both assigned to 1. In addition, the computation of $\mathbf{s_{k+1}} = \mathbf{\tilde{Z}\tilde{Z}^t r_{k+1}}$ at line 7 is the preconditioning step of the linear system (10) and it is computed by means of two matrix-vector operations:

```
cusparseScsrmv(handle, CUSPARSE_OPERATION_NON_TRANSPOSE, n, n,
                a, descra, Z_t, start_Z_t, j_Z_t, r, b, zt);
cusparseScsrmv(handle, CUSPARSE_OPERATION_NON_TRANSPOSE, n, n,
                a, descra, Z_t, start_Z, j_Z, zt, b, z);
```

In details, in the first call of `cusparseScsrmv()`, $\mathbf{zt} = \mathbf{\tilde{Z}^t r_{k+1}}$ and $\mathbf{s_{k+1}} = \mathbf{\tilde{Z} zt}$ are computed. We have outlined just few of computational operations because the other will be performed in the same way. The parameters `handle`, `descra` and `CUSPARSE_OPERATION_NON_TRANSPOSE` are discussed in the NVIDIA report [10] in more details. In summary, we highlight that using standard libraries, designed for the GPU architecture, it is possible to optimize the computational oceanographic simulation model.

5 Numerical experiments

In this section we focus on the important numerical issues of our elliptic solver implemented with GPU architecture in single precision. The solver is tested on three grid size resolutions of the NEMO-OPA ocean model (Table 1). In the Table 2, we compare the performance in terms of PCG iterations of the proposed inverse bandwidth preconditioner $\mathbf{P}$ respect to $\mathbf{P}^{-1}$, that is the diagonal NEMO-OPA preconditioner. We fix an accuracy of $\varepsilon = 10^{-6}$ on the relative residue $r = ||\mathbf{Ax} - \mathbf{b}||/||\mathbf{b}||$ on the linear system solution. The experiments are carried out in the case of well-conditioned $\mathbf{A}$ matrix, corresponding to the geographical case of $\phi \approx 0$ and in the case of ill-conditioned $\mathbf{A}$ with $\phi \approx \pi/2$. We can observe as in the worst case with n large and $\mathbf{A}$ ill-conditioned the PCG with $\mathbf{P}^{-1}$ has a very slow convergence with a huge number of iterations to reach the fixed accuracy. The experiments, reported in Table 2, highlight the poor performance of the $\mathbf{P}^{-1}$ for solving the Laplace elliptic problem (10) within NEMO-OPA. In the following, we show the performance in terms of PCG iterations of $\mathbf{P}$ respect to the Bridson Class preconditioners, that believe to CUSP library. In details, let us consider the $\mathbf{P}_{B1}$ and $\mathbf{P}_{B2}$ Bridson's preconditioners, obtained by means of the A-orthogonalization method. The first one is given by posing a (fixed) drop tolerance and discarding the elements below the fixed tolerance [6]; in the second one is predetermined the number of non-zeros elements on each its row [14].

Matrix Name	Size	Matrix non-zeros elements
ORCA-2	180×149	133800
ORCA-05	751×510	1837528
ORCA-025	1442×1021	7359366

Table 1: NEMO-OPA grid resolutions.

The required accuracy on the solution is fixed to $\varepsilon = 10^{-6}$ on the relative residue. In Figure 2 (on the left) we report the PCG iterations of $\mathbf{P}$, $\mathbf{P}_{B1}$ and $\mathbf{P}_{B2}$ in the case of the matrix $\mathbf{A}$ well-conditioned ($\phi \approx 0$). In Figure 2 (on the right) we present the case of the ill-conditioned ($\phi \approx \pi/2$) matrix $\mathbf{A}$. The numerical results show how the number of the solver iteration $\mathbf{P}$ is comparable to $\mathbf{P}_{B1}$ and $\mathbf{P}_{B2}$ when the dimensions of the problem are small or middle. Furthermore, it is strongly indicated to use $\mathbf{P}$ with

$\mathbf{A}$ Dimension	$\mathbf{P}$ $(\phi \approx 0)$	$\mathbf{P}^{-1}$ $(\phi \approx 0)$	$\mathbf{P}$ $(\phi \approx \frac{\pi}{2})$	$\mathbf{P}^{-1}$ $(\phi \approx \frac{\pi}{2})$
ORCA-2	271	460	8725	26820
ORCA-05	1128	1593	22447	86280
ORCA-025	2458	3066	28513	139742

Table 2: Comparison between $\mathbf{P}$ and $\mathbf{P}^{-1}$ in terms of Number of Iterations of the PCG in the case $\mathbf{A}$ is well-conditioned ($\phi \approx 0$) and $\mathbf{A}$ ill-conditioned ($\phi \approx \pi/2$), varying the problem dimensions.

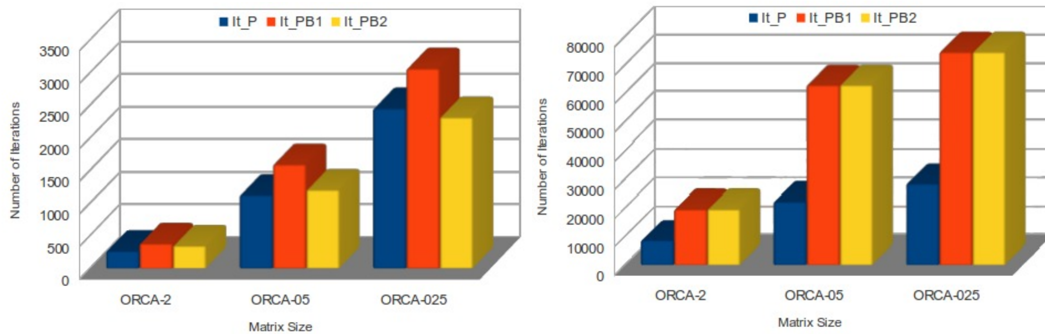

Figure 2: Comparison between **P**, $\mathbf{P}_{B1}$ and $\mathbf{P}_{B2}$ in terms of Number of Iterations of the PCG ($y-$axis) when **A** is well-conditioned (left) and ill-conditioned (right), varying the problem dimensions ($x-$axis).

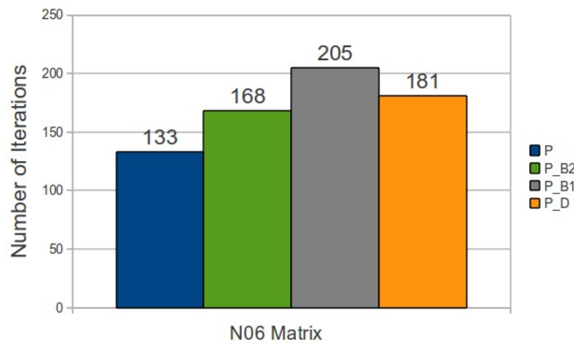

Figure 3: PCG iterations. P is the proposed preconditioner, P_d is $\mathbf{P}^{-1}$, P_B1 P_B2 are $\mathbf{P}_{B1}$ and $\mathbf{P}_{B2}$

a huge problem dimension. We have tested **P**, $\mathbf{P}^{-1}$, $\mathbf{P}_{B1}$ and $\mathbf{P}_{B2}$ on the sparse matrix `NOS6` of the Market Matrix database (`http://math.nist.gov/MatrixMarket`) by setting the required accuracy on the computed solution to $\varepsilon = 10^{-6}$ and the band q of the preconditioner to 4. This sparse matrix is obtained in the Lanczos algorithm with partial re-orthogonalization Finite difference approximation to Poisson's equation in an L-shaped region, mixed boundary conditions. The figure 3 shows as **P** achieves the best performance in terms of iterations. Finally, we have tested the elliptic solver implementation on GPU architecture. The numerical experiments are carried out on an "NVIDIA TESLA S2050" card, based on the "FERMI GPU". The "TESLA S2050" consists of 4 GPGPUs, each of which with 3GB of RAM memory and 448 processing cores working at 1.15 GHz. All runs are given on 1 GPU device. We have adopted CUDA release 3.2, provided by NVIDIA as a GPGPU environment and the numerical code is implemented by using the single precision arithmetic. As described in the previous sections, by using scientific computing library it is not necessary manually setting up the blocks and grid configuration on

the device memory. The number of blocks required to store the elliptic solver input data (in CSR format) do not have to exceed the maximum sizes of each dimension of a GPU grid device. Schematic results of GPU memory utilization for ocean model resolutions are presented in the Table 3. Observe that, in our numerical experiments, we do not fill the memory of the TESLA GPU and the simulations run also on cheaper or older boards, as for example the Quadro 4700FX. Generally, it is possible to grow the grid dimensions of the ocean model according to the memory capacity of the available GPU. The elliptic solver requires a large amount of Sparse-Matrix Vector (`cusparseCsrmv()`) multiplications, vector reductions and other vector operation to be performed. CPU version is implemented in ANSI C executed in serial on a 2.4GHz "Intel Xeon E5620" CPU, with 12MB of cache memory. Serial and GPU versions are in single precision. In Table 3 we report the execution times and speed-up of the CPU and GPU implementation of the solver on several matrix configuration sizes. We observe the well speed-up values incresing the size of the problem from ORCA-2 to ORCA-025 due to utilization of GPU implementation and optimized numerical libraries. Moreover, we tested the

Matrix Name	Non-zeros Elem.	Mem. GPU.	CPU	GPU	Speed-up
ORCA-2	133800	4MB	61.5	37.03	1.6
ORCA-05	1837528	37MB	4521.29	240.11	18.8
ORCA-025	7359366	135MB	57898.59	1090.53	53.1

Table 3: Matrix memory occupancy and execution times (in seconds) on the CPU and GPU and speed-up, varying the size of the problem from ORCA-2 to ORCA-025.

performance of the solver in terms of Floating Point Operations (FLOPs). The performance of the numerical experiments (reported in the Figure 4) are given in the case of A ill-conditioned matrix ($\phi \approx \pi/2$). We count an average of the iterations of solver and the complexity of all linear algebra operations involved in both serial and parallel implementations. The "GPU solver" (blue) and "CPU solver" (orange) curves represent the GFLOPS of the solver, respectively, for the CPU and GPU versions. The main recalled computational kernels in the solver are the Sparse-Matrix Vector. In Figure 4 we highlight the improvement in terms of GFLOPs speed-up by replacing `gemv()` with `cusparseCsrmv()` function. These results prove that, increasing the model grid resolution, it is possible to exploit the computational power of the GPUs. In details, the GPU solver implementation in the ORCA-025 configuration has a peak performance of 87 GFLOPS respect to 1,43 GFLOPS of the CPU version.

6 Conclusions

In the ocean modelling, the elliptic Laplace equations represent critical computational points since the convergence of the numerical solvers to a solution, within a

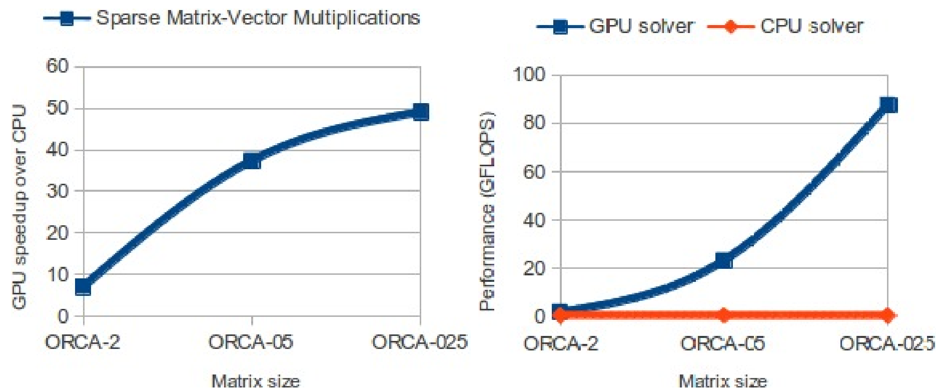

Figure 4: Left: Sparse-Matrix Vector multiplications speed-up. Right: CPU and GPU comparison of the solver in terms of GFLOPS.

reasonable number of iterations, it is not always guaranteed. In our case, this happens to the preconditioning technique of the NEMO-OPA ocean model, for which we prove to be inefficient and inaccurate. In this paper, we propose a new inverse preconditioner that shows better results respect to the NEMO-OPA diagonal one and to others of the Bridson class. Moreover, we show a parallel approach of the elliptic solver on GPU, by means of the scientific computing libraries. The library based implementation of the computing codes allows to optimize oceanic framework reducing the simulation times and to develop computational solvers easy-to-implement.

Acknowledgment. The research leading to these results has received funding from the Italian Ministry of Education, University and Research and the Italian Ministry of Environment, Land and Sea under the GEMINA project.

References

[1] A. Arakawa and F. Mesinger, *Numerical Methods used in Atmospheric Models*, Garp Publication Series NO 17, Paris, France, 1976.

[2] N. Bell and M. Garland, *Efficient sparse matrix-vector multiplication on cuda*, Technical report, NVIDIA Corporation NVR-2008-004, 2008

[3] M. Benzi, *An explicit preconditioner for the conjugate gradient method*, Proceedings of the Cornelius Lanczos International Centenary Conference, Philadelphia USA, 1994.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J Comput Phys **182**: 417–477, 2002.

[5] M. Benzi, C. D. Meyer and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput **17**: 1135–1149, 1996.

[6] R. Bridson and W. Tang, *Refining an approximate inverse*, J. Comput. Appl. Math. **22**, 2000.

[7] CUBLAS Library, Technical report, NVIDIA. `http://developer.download.nvidia.com`, 2007.

[8] S. Cuomo, P. De Michele and R. Farina, *An inverse preconditioner for a free surface ocean circulation model*, AIP Conference Proceedings, pp. 356-362, Vol. 1493, 2012.

[9] S. Cuomo, P. De Michele and R. Farina, *A CUBLAS-CUDA implementation of PCG method of an ocean circulation model*, AIP Conference Proceedings, pp. 1923-1926, Vol. 1389, 2011.

[10] *CUSPARSE Library*, Technical report, NVIDIA. `http://developer.download.nvidia.com`, 2012.

[11] J. K. Dukowicz, *A reformulation and implementation of the bryan cox semtner ocean model on the connection machine*, J Atmos Ocean Tech **10**: 195–208, 1993.

[12] S. M. Griffies, C. Boning, F. O. Bryan, E. P. Chassignet, R. Gerdes, H. Hasumi, A. Hirst, A. Treguier and D. Webb, *Developments in ocean climate modelling*, International Journal of Applied Mathematics and Computer Science **2**: 123–192, 2000.

[13] R. L. Higdon, *Numerical modeling of ocean circulation*, Acta Number **15**: 385–470, 2006.

[14] C. J. Lin and J. More, *Incomplete cholesky factorizations with limited memory*, SIAM J.Sci. Comput. **21**, 2000.

[15] G. Madec, *NEMO-OPA Ocean Engine*, Institute Pierre-Simon Laplace, Paris, France, 2012.

[16] *NVIDIA CUDA programming guide*, Technical report, NVIDIA, 2012.

[17] F. Piccialli, S. Cuomo and P. De Michele, *A regularized MRI image reconstruction based on Hessian penalty term on CPU/GPU systems*, International Conference on Computational Science (ICCS), 2013.

[18] G. Roullet and G. Madec, *Salt conservation, free surface, and varying levels: a new formulation for ocean general circulation models*, J Geophys Res **105**: 23927–23942, 2000.

[19] D. J. Webb, *A multiprocessor ocean general circulation model using message passing*, Comput Geosci **22**: 569–578, 1996.

[20] D. J. Webb, A. C. Coward, B. A. de Cuevas and C. A. Gwilliam, *An ocean model code for array processor computers*, J Atmos Ocean Tech **22**: 175–183,1997.

Received: March 21, 2013