PETSc-based Parallelization of the fully 3D-curvilinear Non-hydrostatic Coastal Ocean Dynamics Model, GCCOM

Manuel Valera, Neelam Patel,
and Jose Castillo

December 1, 2017

Publication Number: CSRCR2017-02

Computational Science & Engineering Faculty and Students Research Articles

Database Powered by the Computational Science Research Center Computing Group & Visualization Lab

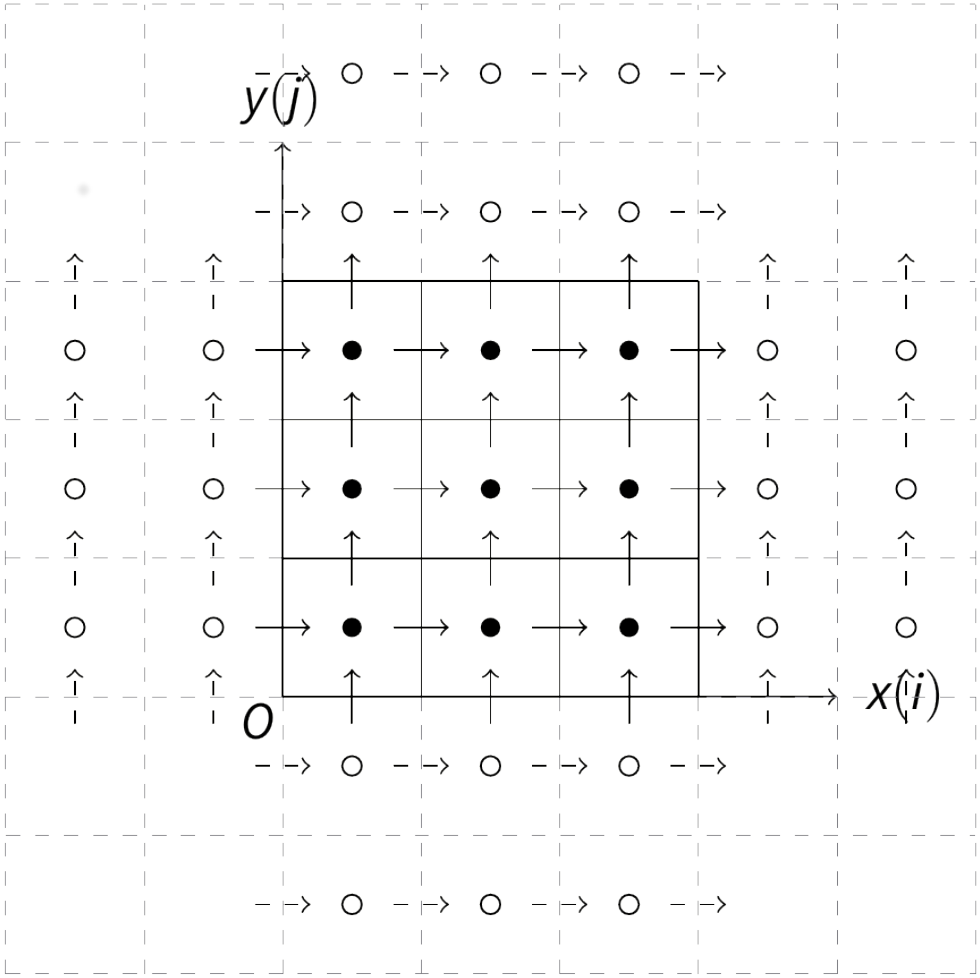
COMPUTATIONAL SCIENCE & ENGINEERING

SAN DIEGO STATE UNIVERSITY

Computational Science Research Center
College of Sciences
5500 Campanile Drive
San Diego, CA 92182-1245
(619) 594-3430

PETSc-based Parallelization of the fully 3D-curvilinear Non-hydrostatic Coastal Ocean Dynamics Model, GCCOM

Valera, M. Patel, N. Castillo, J.



Contents

CHAPTER 1

Introduction

1.1 Motivation

The Coastal Ocean is defined as the stretch of sea closest to the shore, from the continental shelf up to one mile into land [1]. According to [2] 50% of human population lives in coastal areas and 61% of the world GDP comes from exploiting these areas, thus, they are preponderant and highly important to human development in terms of food production, hazard prevention, risk management and so on. With the global-scale ocean models mostly settled [ref], the Coastal Ocean Dynamics (COD) models take the spotlight in the oceanographic community to unravel the secrets of turbulence mixing, multiscale physics and geochemical production of species in these environments. For our computational science research center, the coastal ocean have been a priority since it's creation, and such, have taken effort into developing it's very own COD model, the General Curvilinear Coastal Ocean Model, GCCOM [3–6].

1.2 The GCCOM model

GCCOM stands for General Curvilinear Coastal Ocean Model, but it is also a fully-3D nonhydrostatic, Large-Eddy simulation, thermodynamics and data adquisition model, among other capabilities. It's first version was released in 2006 by C. Torres [3], and since then several other implementations and redesigns have enable us to have a robust and top-of-the-line COD suite with competitive performance.

Specially making use of boundary-fitted coordinates, or curvilinear coordinates in the full 3D domain, makes it different from the bulk of COD models, these kind of coordinates

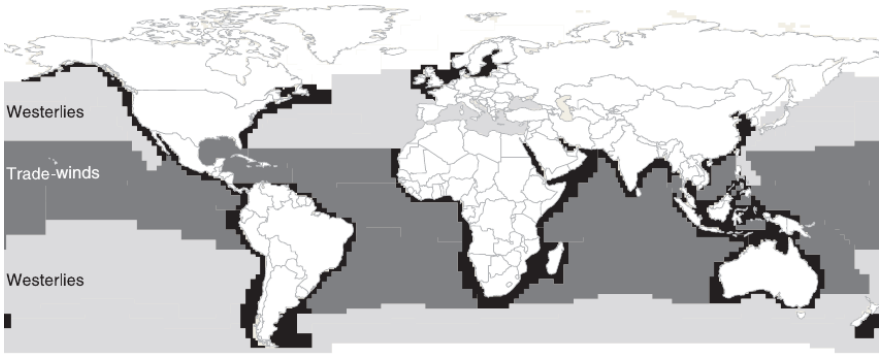


FIGURE 1.1 Coastal areas of the world marked as black [ref].

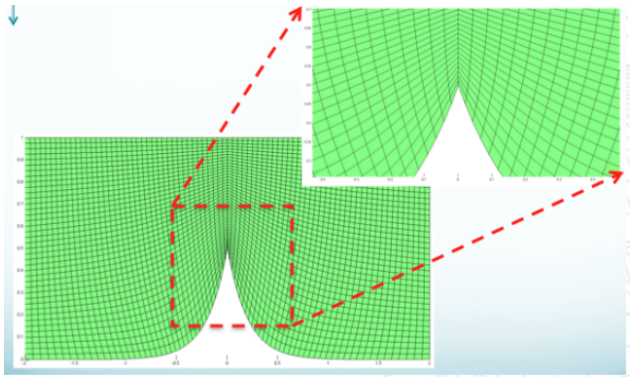


FIGURE 1.2 Curvilinear of boundary-fitted coordinates.

are posed such that they envelop and follow the boundaries shape naturally at every point, instead of the errors and approximations typical of the rectangular coordinates, while at the same time being orthogonal and completely conservative. The drawback of these coordinates lies in the metrics transformation needed to model the physics from the Navier-Stokes equation into the computational domain, given a curvilinear coordinate. This application creates a bottleneck that quickly stales the models performance in a single computer or core, for which high performance computing and MPI as paralellization paradigm becomes a must.

1.2.1 The pressure solver

The GCCOM scheme takes advantage of an Arakawa type-c staggered stencil and a predictor-corrector scheme to solve for spatial coordinates, this implies that solving for the central quantity of the scheme, in our case the pressure, requires the most expensive computation, since the information of the entire domain is solved as a laplacian matrix of size (IMax x JMax x KMax)x(IMax x JMax x KMax) as the first step before predicting and correcting the velocities and updating the rest of the scalar quantities, as temperature, density, and so on.

The pressure laplacian used to take up to 98% of the total computational time [7], several improvements in the design, specially solving the entire laplacian in implicit form made the pressure solver account for just 65% of the total running time, remaining the most computationally intensive element of the model. At the same time, as a fully elliptic problem, Multigrid techniques are known to be the unparalleled best solvers and/or preconditioners for this type of systems. For all of these reasons AGMG library [8] was adopted as the pressure solver, multigrid library of use inside the GCCOM.

Higher complexity problems made the need for a faster, more versatile and widely used library to solve for pressure imperative, in this regard PETSc [9], the Portable, Extensible Toolkit for Scientific Computation was investigated and finally adapted to the GCCOM. The first part of this report resumes the implementation of this new library into the GCCOM as a mean to solve for pressure. Being PETSc natively compatible with MPI, the parallel paradigm was at grasp and thus scalability test were carried for the first time for the pressure solver in the GCCOM. Those tests along with some other PETSc tools made the objective of fully parallelizing the GCCOM a viable reality in the short-term.

1.2.2 Parallelization of GCCOM using PETSc

Having developed, tested and validated a fully parallel and scalable solution for the GCCOM pressure solver, and PETSc providing the needed tools to parallelize the rest of the model, a plan and design of the GCCOM to work in parallel was devised and it's implementation is currently being tested. This work in progress is being currently made with the help of the Computational Science Master's student Neelam Patel, as part of her degree project.

PETSc, besides providing libraries for solver and preconditioners, also have tools in it to distribute arrays, called DMDAs they require special ordering to be stored and operated in. Additionally, the Fortran version of PETSc uses the 0-based ordering native to C. On the other hand, GCCOM is written in Fortran 95 and takes advantage whenever possible of the variable array ordering of the language. All of these design differences need to be reconciled before succesfully implementing PETSc into GCCOM as a fully MPI parallel tool.

In this report a study of the adaptation and implementation of the PETSc library is presented, in Chapter 2 the pressure solver is redesigned to work with PETSc, while scalability tests are carried in two clusters, along with optimization tests done for the Monterey testcase on both the multigrid preconditioner and the MPI directrices given. In Chapter 3, the complete parallelization of the GCCOM using PETSc is devised, it's main differences with the current serial implementation and design, as well as the tools implemented to attain the parallelization are posed and explained, specially the DMDA objects, along with some prototype tests carried at this time by the author. Finally, some closing remarks are given in the hope of updating this report in the near future with the full model able to scale reasonably well in MPI machines.

CHAPTER 2

PETSc Implementation for Linear Solvers

The Portable, Extensible, Toolkit for Scientific Computing, PETSc, is a compendium of tools tailored for the most common modelling needs, is natively written in C and ported to Fortran, Python and other Languages. Some of the most basic tools are data structures as vectors (VEC) and matrices (MAT) PETSc objects. In order to solve a linear system it also provides routines for preconditioners (PC) and Krylov subspace solvers (KSP). Also, all of it is natively compatible with MPI, adding a new layer of functions over the MPI calls and enabling the MPI to interface within it at any level.

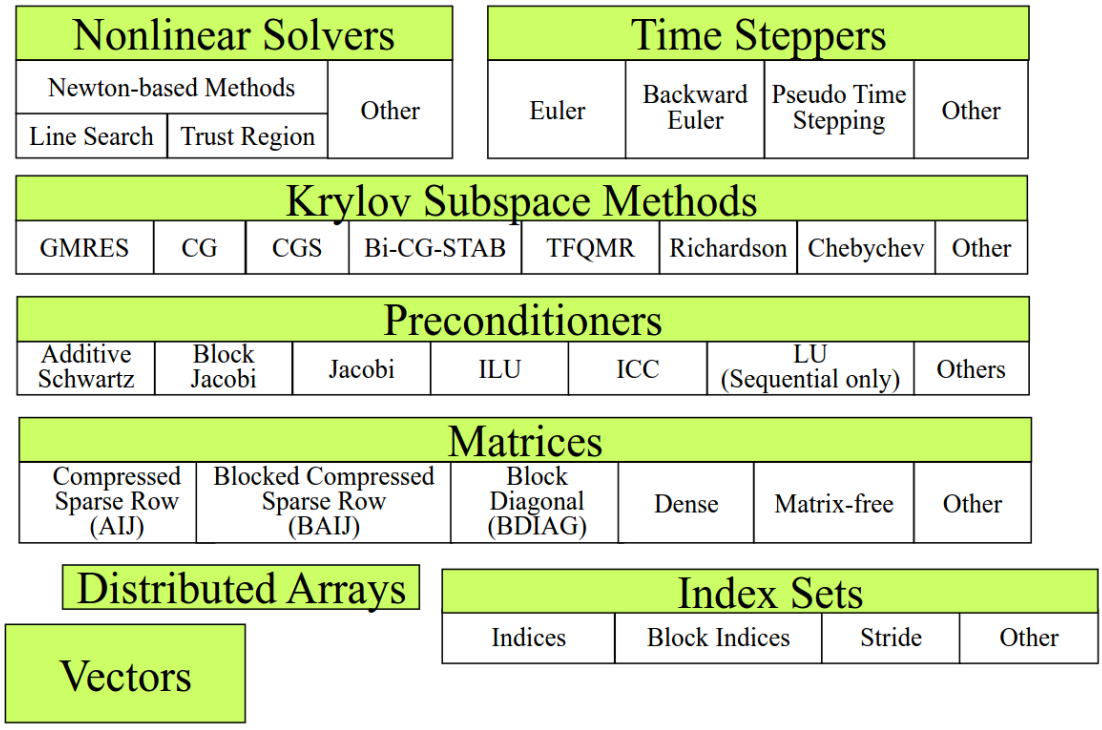


FIGURE 2.1 PETSc library components [ref].

In this chapter we will narrate the steps taken in order to adapt the PETSc objects into the laplacian solver of the GCCOM, including the CSR matrix format, the preconditioners and solvers used, the validation of the implementation, and the optimization steps so far taken to minimize the running time

2.1 PETSc Installation

Assuming we have previously installed GCCOM and it's dependencies (NetCDF, zlib, curl, etc), we will proceed to clone the PETSc repo in it's own folder, i.e.:

```
$ git clone -b master https://bitbucket.org/petsc/petsc petsc
$ git pull
```

And then depending on the compiler to be used and external packages to use, we run the building command as:

```
$ cd petsc
$ ./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran --download-fblaslapack
 --download-mpich --download-hypre --with-debugging=no
```

Notice here that the option `--with-debugging=no` implies an improvement in PETSc performance but turns off most of the debugging options for the toolkit, if we need to further develop and/or debug we would recompile with `--with-debugging=1`

Once the download and configure process is finished, we run `make all test` and follow any instructions that come up at the end of the installation process.

2.1.0.1 Streams Tests

PETSc tests the machine where was installed by several tests which stresses the nodes in the machine so an speedup curve can be gained from it, the results of the latest post-installation streams test can be seen in Table 2.1 As a rule of thumb, what we see here is that no real speedup can be gained from using PETSc on COD just as it is, this is why we will use the options `--map-by core` and `--bind-to core` in order to help to the performance of the machine.

2.1.1 Environment variables

Two variables, `PETSC_DIR` and `PETSC_ARCH` should be set up as environment variables to work with a specific PETSc installation, we may have several installations on own system and switch between them by activating these variables, they are set up in our `.bashrc` as:

```
export PETSC_DIR=/petsc/petsc-3.7.5
export PETSC_ARCH=linux-gnu-c-debug
```

Keep in mind the architecture and PETSc version may be different in your case, but the `PETSC_DIR/PETSC_ARCH` folder structure will be the same. Also, for a sistem wide installation we don't use a `PETSC_ARCH` variable. For more installation options go to `http://www.mcs.anl.gov/petsc/documentation/installation.html`

TABLE 2.1 Streams tests for PETSc on COD after latest installation, np stands for number of processors

np	speedup
1	1.0
2	1.94
3	1.43
4	1.04
5	1.23
6	1.05
7	1.17
8	1.05
9	1.15
10	1.05
11	1.13
12	1.06
13	1.13
14	1.06
15	1.12
16	1.07
17	1.19
18	1.1
19	1.25
20	0.99
21	1.12
22	0.99
23	1.11
24	1.25
25	1.03
26	1.06
27	1.09
28	1.0
29	0.97
30	1.04
31	1.03
32	1.05

2.2 Linear Solver Setup

Once we installed and tested PETSc, and compiled and run our Seamount testcase in GCCOM, we can start using the PETSc tools to solve for the Laplacian linear system. An implicit linear system is as simple an idea as to solve for a vector x, given a matrix A (in this case our laplacian) and a right hand side vector b, i.e:

$$Ax = b \tag{2.1}$$

$$b \backslash A = x \tag{2.2}$$

Where we first wrote our system in the usual mathematical way and later in the most common computational way. In any case this is our main idea, and to solve it we will need the VEC objects and the MAT objects in a way it works along with the stablished GCCOM model.

As we know, vectors x,b and matrix A must share the same sizes, which is related to our problem size, given by IMax, JMax and KMax parameters. These paremeters are read from the files in the `ProbSize.f90` module and then converted to the needed sizes in the `MultigridVars.f90` module:

Code 1 The number of nodes and non-zero elements of linear system.

```
nbnodos = (IMax-1)*(JMax-1)*(KMax-1)

nx = IMax-1; ny = JMax-1; nz = Kmax-1

nnz = nbnodos + 2*(nbnodos-nx*nz) + 2*(nbnodos-ny*nz) +2*(nbnodos-nx*ny) &
+ 4*(nx-1)*(ny-1)*nz + 4*nx*(ny-1)*(nz-1) + 4*(nx-1)*(ny)*(nz-1)
```

Here `nbnodos` is the length of x,b and the size of one of the sides of matrix A. `nnz` is the number of non-zero elements in matrix A, which is be sparse and is stored in Compressed Row Storage (CSR) format.

Also, from now on, we will add a `p` at the end of the variable names to emphasize they are PETSc variables, for which `nbnodos = nbdp` and `nnz = nnzp`

Having our vector size, we start coding our PETSc objects, to create our vectors x,b which we will call xp,bp, we use the following code which is part of the `PetscObjs.f90` module:

This code also includes a buffer vector called work. We can also appreciate we use a `PETSC_COMM_WORLD` MPI communicator object and that we let `PETSC_DECIDE` for the program to establish the local size of our parallel vector from the `nbdp` global size given it is being run in parallel, in this way we already see the MPI native capabilities of PETSc coded in our program. Finally, to build the VEC object information we need to call the assembly routines as seen in the code. Notice we have not yet assembled bp vector.

2.2.1 SolveP_Rhs.f90

The right hand side (RHS) vector, bp is created, as every other PETSc object, in the `PetscObjs.f90` module, but it's values are loaded in the `SolveP_Rhs.f90` subroutine, as that is the place in GCCOM where the boundary conditions are imposed to the RHS and this

Code 2 Create, duplicate and assemble a PETSc vector.

```
call VecCreate(PETSC_COMM_WORLD,xp,ierr); CHKERRQ(ierr)
call VecSetSizes(xp,PETSC_DECIDE,nbdp,ierr); CHKERRQ(ierr)

call VecSetFromOptions(xp,ierr); CHKERRQ(ierr)
!initializing SOLN to zeros
call VecSet(xp,0.0D0,ierr); CHKERRQ(ierr)
call VecDuplicate(xp,bp,ierr); CHKERRQ(ierr)

call VecDuplicate(xp,work,ierr);   CHKERRQ(ierr)

call VecAssemblyBegin(xp,ierr) ; call VecAssemblyEnd(xp,ierr)
call VecAssemblyBegin(work,ierr) ; call VecAssemblyEnd(work,ierr)
```

vector is written in lexicographical ordering to match the laplacian matrix. The Rhs() array is converted into the bp PETSc vector with the following code:

Code 3 Populate the RHS vector in one call.

```
do ii=0,nbdp-1,1
ind(ii+1) = ii
enddo

if (sizex==1)then
call VecSetValues(bp,nbdp,ind,-Rhs,INSERT_VALUES,ierr)
call VecAssemblyBegin(bp,ierr) ; call VecAssemblyEnd(bp,ierr)
else
call VecSetValues(bp0,nbdp,ind,-Rhs,INSERT_VALUES,ierr)
call VecAssemblyBegin(bp0,ierr) ; call VecAssemblyEnd(bp0,ierr)
endif
```

As we see in this call, we make use of the `ind` integer array of size `nbdp`, which is nothing else than an index order for the RHS vector. Notice we are already remapping the ordering to 0-based ordering in this index. Then, we have the same routine in slightly different flavor, when we use `VecSetValues()` followed by `VecAssemblyBegin()`, with the difference being the vector bp0 which is only used when the program is being run in parallel cores, and it's function is to being scattered across the cores by a special PETSc object called `VecScatter`

2.2.2 Laplacian Matrix

The final and more complicated part of our linear system is the matrix, as a laplacian matrix will be a banded, sparse matrix of big size, we need an optimal storage for it, for the GCCOM this is the CSR format.

2.2.2.1 CSR Arrays Setup

The CSR format consists of three arrays of different sizes, which contain the full information of our matrix: `row_ptr` or the row pointer is the position in the row where the nonzero entry is, `Col_id` (column identifier) is the column number where the nonzero entry is, and between these two arrays we have completely mapped our matrix, so we only need a third array with all of the nonzero entries in it, this is `Acsr`. There arrays in GCCOM are located in the `MultigridVars` module.

Code 4 Create, order and fill the CSR arrays into PETSc.

```
        !Load CSR Arrays:
allocate(iapi(nbdp+1), stat=ierr)
allocate(japi(nnzp), stat=ierr)
allocate(app(nnzp), stat=ierr)

!print*, nbdp, size(row_ptr)
!print*, size(iapi)

iapi = row_ptr
japi = Col_id
app = -Acsr

do ii=1,nbdp,1
spa = iapi(ii+1)-iapi(ii)
select case (ii)
case(1)
call PetscSortIntWithScalarArray(spa,japi(ii),app(ii),ierr) !1st row case
case default
call PetscSortIntWithScalarArray(spa,japi(iapi(ii)+1),app(iapi(ii)+1),ierr)
!other rows case
end select
enddo
```

In the previous code we can see we rename the CSR arrays into their PETSc arrays names `iapi`,`japi`,`app`, where the last `i` in the name denotes is an integer array, while the `app` array is a `PetscScalar` array.

We also see the use of `PetscSortIntWithScalarArray()` and the casting of a spacing array `spa`, this loop makes sure we re-arrange the CSR arrays in incremental order, a requirement for the PETSc CSR matrix creation and something GCCOM did not do before.

Next, we create the matrix object in CSR format, called AIJ in PETSc, with the following code:

Here, several steps are taken, first we create and feed the matrix with `MatCreate()` and `MatCreateSeqAIJWithArrays()`, then we regularize the non-invertible fully elliptical laplacian matrix with `MatNullSpaceCreate()` and `MatSetNearNullSpace()`, we assemble it and then we write it as a PETSc binary file with `PetscViewerBinaryOpen()` and `MatView()` so it is loaded and distributed in parallel when run with more than one core. Finally, the PETSc objects used are destroyed.

Code 5 Create and save the laplacian matrix as binary file.

```
if (sizex==1) then
     call MatCreate(PETSC_COMM_WORLD,Ap,ierr);  CHKERRQ(ierr)
     call MatCreateSeqAIJWithArrays(PETSC_COMM_WORLD,nbdp,nbdp,iapi,japi,app,Ap,ierr)
     call MatSetUp(Ap,ierr)
     call MatNullSpaceCreate(PETSC_COMM_WORLD,PETSC_TRUE,0,PETSC_NULL_VEC,nullsp,ierr)
     call MatSetNearNullSpace(Ap,nullsp,ierr)
     call MatNullSpaceDestroy(nullsp,ierr)
     call MatAssemblyBegin(Ap,MAT_FINAL_ASSEMBLY,ierr)
     call MatAssemblyEnd(Ap,MAT_FINAL_ASSEMBLY,ierr)

 call PetscViewerBinaryOpen(PETSC_COMM_WORLD,petsc_filename,FILE_MODE_WRITE,viewer,ierr)
 call MatView(Ap,viewer,ierr)  !prints

    call PetscViewerDestroy(viewer,ierr)
 call MatDestroy(Ap,ierr)

 deallocate(iapi,stat=ierr0)
 IF (ierr0 /= 0) STOP "*** iapi ***"
 deallocate(japi,stat=ierr0)
 IF (ierr0 /= 0) STOP "*** japi ***"

 CHKERRQ(ierr)

 deallocate(app,stat=ierr0)
 IF (ierr0 /= 0) STOP "*** app ***"
 CHKERRQ(ierr)
 print*, "Finished load and saving matrix in PETSc"

endif !end creating matrix binary
```

2.2.3 Preconditioners and Solvers

After we have created our linear system components, we have to design a strategy to solve for it, a solve can be done directly or with preconditioning, using preprocessing algorithms to make the solution converge faster. It is known that the best method to solve for elliptical problems are multigrid methods. In this section we present the code implemented for different solvers and preconditioners.

Code 6 Set up the KSP solver.

```
call KSPCreate(PETSC_COMM_WORLD,ksp,ierr);  CHKERRQ(ierr)
call KSPSetOperators(ksp,A0p,A0p,ierr)
call KSPGetPC(ksp,pc,ierr)
tol = 1.e-14
call KSPSetTolerances(ksp,tol,PETSC_DEFAULT_REAL,PETSC_DEFAULT_REAL,PETSC_DEFAULT_INTEGER,ierr)
    [..]
    call KSPSetType(ksp,KSPGCR,ierr)
call KSPSetCheckNormIteration(ksp,25,ierr)
CHKERRQ(ierr)
call KSPSetFromOptions(ksp,ierr)
CHKERRQ(ierr)
call KSPSetUp(ksp,ierr);
```

In this code we create the Krylov-Subspace (KSP) solver and set it's convergence tolerance, then we set the solver type to GCR, which is a modified GMRES algorithm used also by AGMG, ensuring the solver performance will be the same than that for the last library, finally we take the necessary PETSc steps to set up the KSP object.

The following step is to set up the preconditioner (PC) object, several preconditioners were tried and finally the Block-Jacobi algorithm showed the best performance so far. This is not in agreement with literature, something we assume is because lack of optimization for the multigrid solver(s) more than anything else, this is explored in a latter section. The code to set up the PC object is:

Code 7 Set up the preconditioner PC.

```
call PCGetOperators(pc,A0p,pmat,ierr)
call PCCreate(PETSC_COMM_WORLD,mg,ierr); CHKERRQ(ierr)
call PCSetType(mg,PCJACOBI,ierr)
call PCSetOperators(mg,A0p,pmat,ierr)

call PCSetUp(mg,ierr)
call PCApply(mg,xp,work,ierr)
```

Having all of the elements in place, we use `KSPSolve()` to solve our linear system and get the xp solution vector.

The vectors xp,bp are scattered back and forth in our current parallelization scheme, the rest of the model runs on the root processor and even tho we can solve the pressure up to

44 times faster than before, the communication for this scatter creates a big overhead and the rest of the model needs to be parallelized, we will see this in the following section.

2.2.4 Parallelization Scheme

The linear solver for the laplacian described before, being written in PETSc, makes is somewhat straightforward to parallelize it's workings, the only problem remaining is to ensure the xp,bp vectors have the right information in them, for this, we placed the rest of the GCCOM code to run only on the root processor, this enables us to attain the scheme depicted in Figure 2.2.

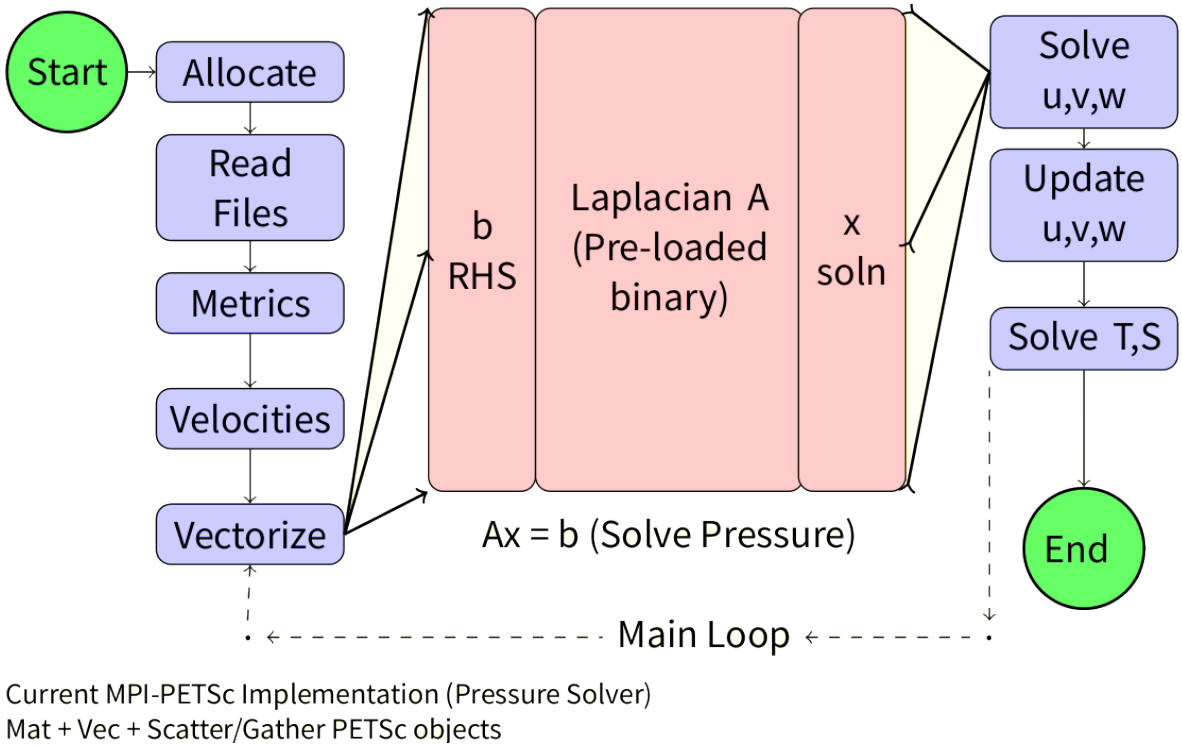


FIGURE 2.2 Parallel scheme in place in GCCOM. Blue runs on serial, red runs in parallel. Arrows show scatter back and forth.

In this scheme we see most of the computation is done in root processor, so is still serial, with the exception of the update of the vectors bp and xp, going in and out of the parallel scheme, respectively.

In PETSc, the usual Scatter/Gather calls from MPI are unified in one Scatter-back or Scatter-forth calling, the following code shows how to setup and distribute a PETSc vector among the used processors:

The rest of the GCCOM code was pipelined in order to run only in the root processors, this meaning the `ucmsMain.f90` module is segmented in parts with MPI commands up to the main loop, where the vectors are scattered, the linear system solved and the solution gathered back to the root processor, this process is repeated for as long as the iterations of the main loop are needed. This hybrid scheme creates an ideal prototyping and timing environment to test the capabilities of PETSc in the GCCOM model, but it is far from ideal in speeding up the code as a whole. A full GCCOM parallelization is posed in chapter ??.

Code 8 Use of VecScatter().

```
VecScatter :: ctx,ctr

if(sizex/=1)then
!Scattering bp0 sequential RHS into bp parallel RHS
!  print*, "scattering RHS into all processors from master..."

call VecScatterBegin(ctr,bp0,bp,INSERT_VALUES,SCATTER_REVERSE,ierr)
call VecScatterEnd(ctr,bp0,bp,INSERT_VALUES,SCATTER_REVERSE,ierr)

call VecAssemblyBegin(bp,ierr) ; call VecAssemblyEnd(bp,ierr) !rhs
!call PetscBarrier(PETSC_NULL_OBJECT,ierr)
endif

  call KSPSolve(ksp,bp,xp,ierr)

!  print*, "Gathering xp into xp0..."
call VecScatterBegin(ctx,xp,xp0,INSERT_VALUES,SCATTER_FORWARD,ierr)
call VecScatterEnd(ctx,xp,xp0,INSERT_VALUES,SCATTER_FORWARD,ierr)

if (rank==0) then
call VecGetArrayF90(xp0,soln,ierr)
sol = soln             !Copying new solution to regular scalar array
call VecRestoreArrayF90(xp0,soln,ierr)
endif
```

2.3 Results and Optimization

In this section we present the results of applying the PETSc toolkit to the laplacian linear system, first compared to the AGMG multigrid library used before, and then the scalability tests carried at the CSRC machine COD, located at SDSU, and the XSEDE machine COMET located at UCSD.

2.3.1 GCCOM Testcases

In order to present our results we first need to present the GCCOM testcases we have currently working, each one of them is different in themselves and we won't present they're differences much deeply here, beyond the ProbSize for each one, which is a good predictor of the amount of complexity in our testcase. A figure depicting the ProbSizes for each testcase is seen in Figure 2.3.

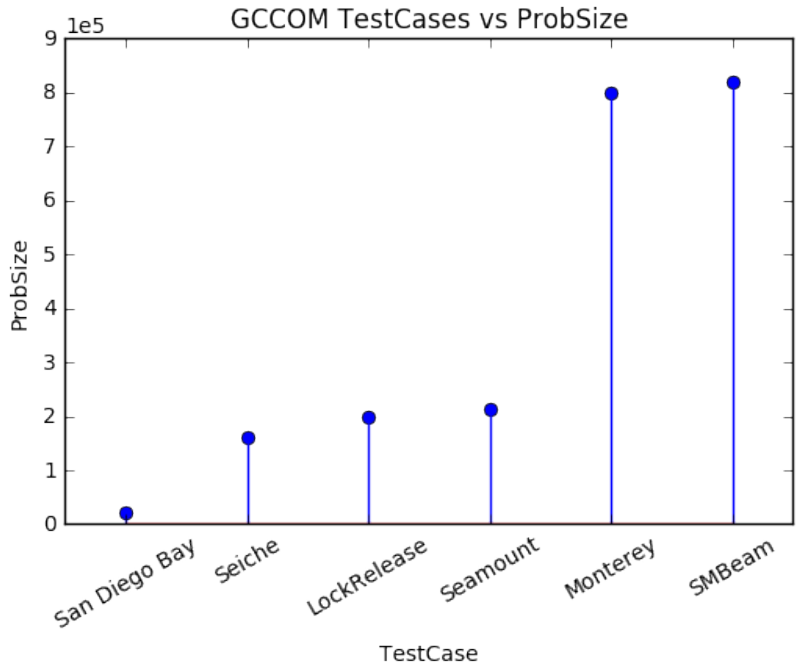


FIGURE 2.3 GCCOM Testcases by ProbSize.

As we can see, San Diego Bay is our smallest testcase so far, while Seiche, LockRelease and Seamount are all around 200k in ProbSize, in the other hand, Monterey and SMBeam are both around 800k computational points in ProbSize. The experience tells us, that the last two TestCases are too big to be run in serial and expect a full run to be completed in a reasonable time.

In Figure 2.4 and example of our motivation is shown, an increase in the number of computational points translates into a higher level of detail in the simulation, giving rise to higher order phenomena.

2.3.1.1 Validation

The validation of our model was made in regards of comparisons with the previous results obtained for the same testcase. Figure 2.5 shows in red the areas where the discrepancies

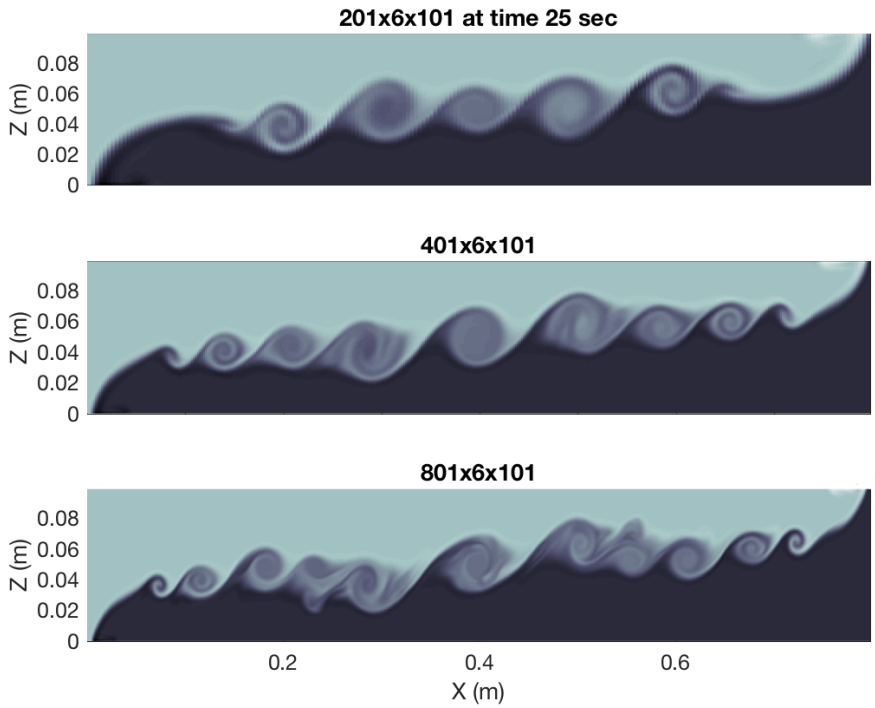


FIGURE 2.4 Increase in detail resolution with double computational points in our mesh for the LockRelease Testcase.

are of the order of 10^{-12} between the output of the PETSc implementation and the previous one after several time steps. It can be appreciated the errors are of this size and the model is accepted to be validated.

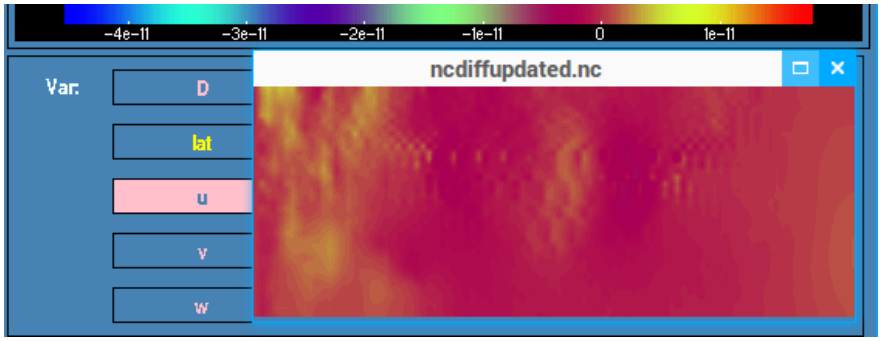


FIGURE 2.5 Validation of the results of solving the linear system by PETSc vs AGMG.

2.3.2 Serial Timings

Disclaimer

This is a work in progress. The timings presented are only a part of the complete code and scalability is still restricted. These results are presented as a proof of concept before the full parallelization. PETSc on this implementation scales very poorly with total runtime due to the overhead created by the Scatter back/forward of the hybrid scheme.

The first comparison we want to draw is the performance difference between our PETSc implementation and the previous library, this was our first comparison point to engage in the paralellization of the code, results can be seen in Table 2.2 and Figure 2.6

TABLE 2.2 Serial timing comparison between previous Pressure Solver Library (AGMG) and PETSc

TestCase	AGMG [s]	PETSc [s]	Speedup
San Diego Bay	0.12	0.042	2.38
Seiche	0.48	0.33	1.45
LockRelease	0.73	0.41	1.78
Seamount	1.25	1.62	0.77
Monterey	5.35	1.79	2.99

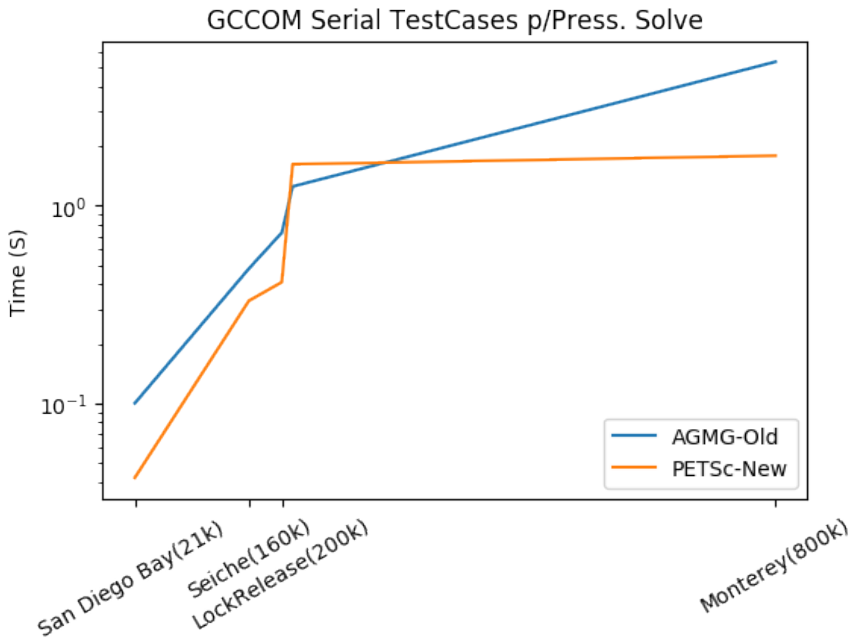


FIGURE 2.6 Serial timings from AGMG and PETSc compared.

From these we can appreciate there is a speedup for every case except for Seamount, this is attributed to the use of complex geometry in the simulation, which needs the fully 3D curvilinear coordinates working at full capacity, in any case results are encouraging for all the other testcases and as we will see next, they are even better when run in parallel.

These serial tests were run in the COD machine from CSRC-SDSU.

2.3.3 Parallel Timings

In the parallel case, we were able to do scalability test in up to 16 processors in COD (CSRC-SDSU) and 24 processors in COMET (XSEDE-UCSD), results are summarized in Table 2.3 and Figure 2.7 below.

TABLE 2.3 PETSc MPI Timings for GCCOM/PETSc runs in COD-16 processors (CSRC-SDSU) and Comet-24 processors (SDSC-XSEDE)

TestCase [s]	AGMG-Old	COD-16	Comet-24	Speedup
San Diego Bay	0.12	0.015	0.01	12
LockRelease	0.73	0.104	0.03	24.3
Seamount	1.25	0.122	0.04	10.24
Monterey	5.35	2.890	0.17	31.47
SM-Beam	7.62	4.951	0.17	44.05

The previous Table 2.3 shows the overall results of our timing, but the machines we used have different performance, to see this Table 2.4 show the timings obtained in COD an Table 2.5 the timings obtained in the Comet machine.

TABLE 2.4 Parallel Pressure solver timings on COD (CSRC-SDSU) by Testcases and relative speedup.

TestCase [s]	np=1	np=4	np=8	np=16	Speedup
SDBay	0.054	0.029	0.023	0.015	3.6
Seamount	1.642	0.878	0.871	0.553	3.0
LockRelease	0.432	0.195	0.175	0.104	4.2
Monterey	1.866	-	3.437	2.905	0.6
SM-Beam	23.99	-	8.769	4.951	4.8

From what we see, most if not all of the testcase show good scaling of up to 24 processors in COMET, whereas it stales at that point in this machine. For the case of COD we see that most testcases scale very well up to 8 cores, but then it stales. In the case of the biggest testcases and specially Monterey, we see a very poor scaling for COD, we attribute this to the architecture differences in both clusters, specially to the memory available in COMET, much larger than that of COD. In any case, several studies are being carried to optimize the performance of the COD cluster and also to better execute the GCCOM.

In any case, for the pressure solver with the current hybrid parallel implementation, we attained speedups from 12 to up to 44 times the original serial speed of the model.

2.3.4 Current Shortcomings

Preliminary timings results show that the current PETSc implementation scales in the XSEDE machine but not so in COD, while the Pressure solver timings shown here are a proof of concept, the complete timings for the model does not show the same trend still. Figure 2.8 shows the overall model timings for several simulation times, it can be seen that PETSc performs better only in when run in COMET.

The overhead created by the communication of the x and b vectors in the linear system is the main reason why we cannot translate the current scheme into computational savings in the total runtime in COD, as they create a still time that even tho it does not grow with

TABLE 2.5 Parallel pressure solver timings on Comet (SDSC-XSEDE) by Testcases and relative speedup.

TestCase [s]	np=1	np=4	np=8	np=12	np=24	Speedup
SDBay	0.04	0.01	0.01	0.00	0.01	4.8
Seamount	0.41	0.14	0.09	0.05	0.04	10.3
LockRelease	0.38	0.11	0.06	0.05	0.03	11.3
Monterey	1.60	0.58	0.27	0.21	0.17	9.3
SM-Beam	1.73	0.44	0.25	0.20	0.17	10.1

more processors allocated, it is still too large and masks all the advancements from solving the linear system in parallel. This behavior can be seen in Figure 2.9

2.3.5 Preconditioner Optimization

In order to optimize our solver and preconditioner, we tried implementing a specific type of multigrid which gives big versatility to try different options and have been subject of different optimization studies [ref], the HYPRE library is interfaced with PETSc to solve our system with multigrid and obtain similar results to that of the PCJACOBI, default solver in PETSc. The options used are listed below as PETSc configuration options:

Code 9 HYPRE Boomeramg optimization options used.

```
call PCSetType(mg,PCHYPRE,ierr)
call PCHYPRESetType(mg,'boomeramg',ierr)

call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_nodal_coarsen','4',ierr)
call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_vec_interp_variant','1',ierr)
call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_max_levels','5',ierr)
call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_smooth_type','Euclid',ierr)

call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_cycle_type','V',ierr)

call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_relax_type_all',&
&'backward-SOR/Jacobi',ierr)
call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_coarsen_type',&
&'modifiedRuge-Stueben',ierr)
call PetscOptionsSetValue(PETSC_NULL_OPTIONS,'-pc_hypre_boomeramg_interp_type','classical',&
ierr)
```

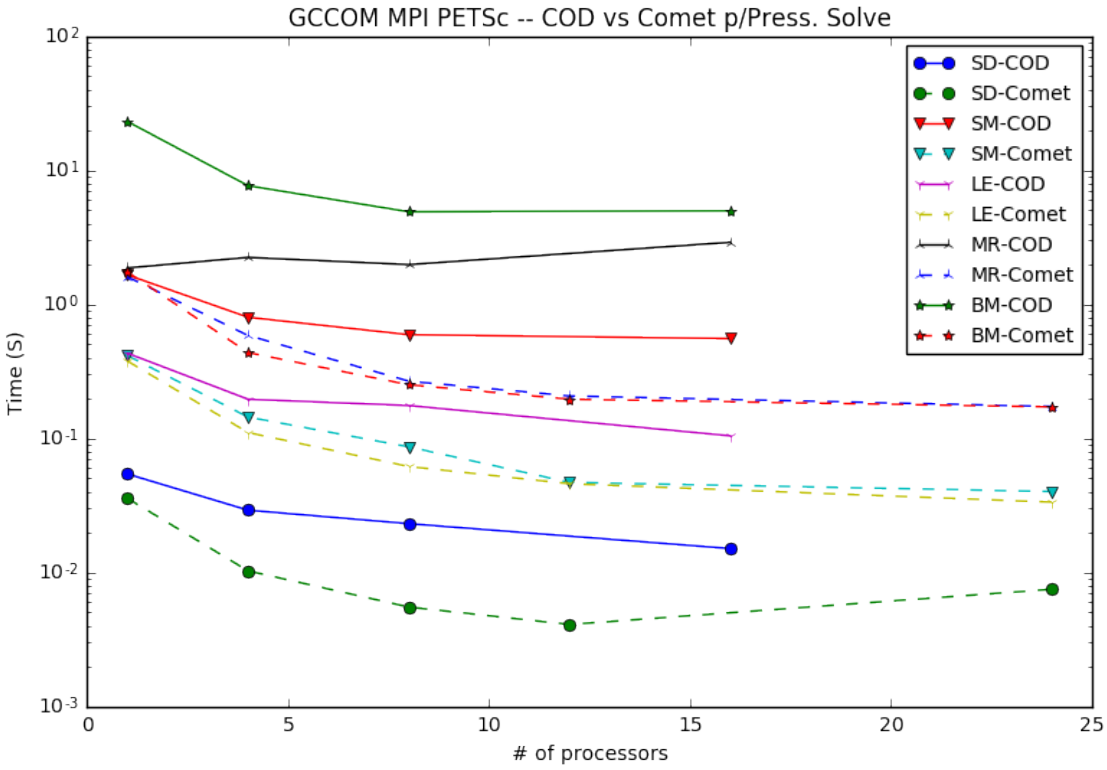


FIGURE 2.7 Pressure solution scaling up to 24 processors in Comet (SDSC-XSEDE) and 8 cores in COD (CSRC-SDSU)

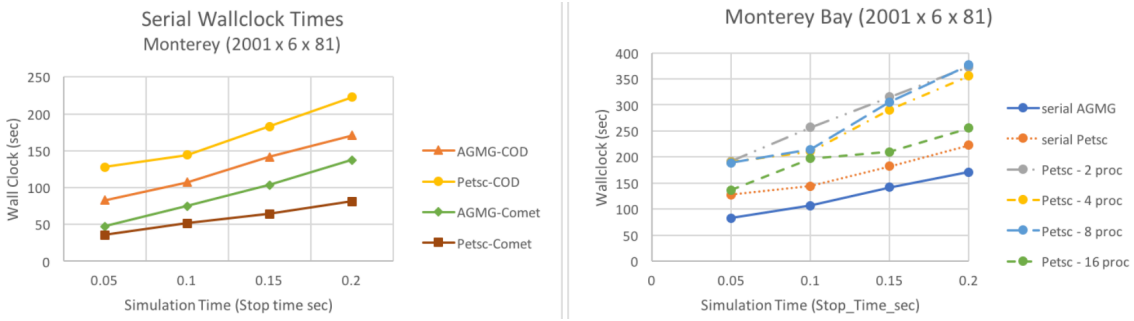


FIGURE 2.8 Shortcomings of the current implementation, the PETSc version performs better in COMET machine (left) but not so in COD (right) not even for parallel use. Timings by Neelam Patel.

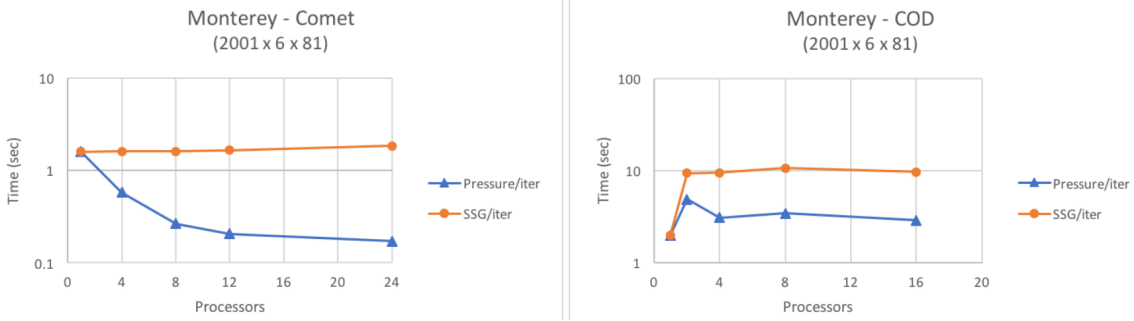


FIGURE 2.9 Overhead created by the scatter of the x, b vectors. Timings by Neelam Patel.

CHAPTER 3
Distributed Arrays - Full Parallelization

Having witnessed the versatility and scalability potential of a PETSc implementation for the pressure solver, the most natural thing to do is to design a way to have similar scaling across the whole GCCOM model, the complete scheme should take advantage of the scatter routines already implemented for all of the arrays that GCCOM uses, and let them be read and created in root processor so we don't have to deal with parallel I/O problems. In this new phase our parallel scheme will look like Figure 3.1.

3.1 Arakawa C-type Stencil

The Arakawa C-type stencil[ref] is the most stable and accurate of the Arakawa grids, and overall one of the best possible choices for computational fluid dynamics models, the stencil places the scalar quantities, or those solved in second derivatives, in the center of the cell, while the velocities are placed on the sides of the cells as seen in Figure 3.2. It can be seen the horizontal u and vertical v velocities are displaced, there is also the third velocity w but is not depicted.

This kind of stencil can be coded in several ways, maybe the more straightforward is with three separated arrays, one for each velocity field and one for the scalar, one of these arrays would describe a position within the stencil and then they would be communicated inside the program. This is the way GCCOM does this, but is not exempt of caveats as we will see in the next section.

Finally, depending on the layout of our array we can have at most two ghost points to each side, as depicted in Figure 3.3.

3.2 3D DMDAs Setup

The missing PETSc piece at this point are the `DMDA` objects, these are objects preconcieved to optimize operations on distributed arrays given an stencil width, a stencil shape and a type of boundary.

`DMDA` have a special ordering to optimize them so they need to be reordered from the row-major ordering of Fortran arrays to the DMDA ordering. Also, remember PETSc objects use 0-base ordering and such we have to reorder our arrays. GCCOM arrays have different layouts reminiscent of the actual physical problem, a collection of all of the layouts used by GCCOM at this moment are seen in Table 3.1.

Remap in Table 3.1 refers to the number needed to add to the callings made to arrays in each layout. There exists 160 different arrays distributed between all of these layouts and

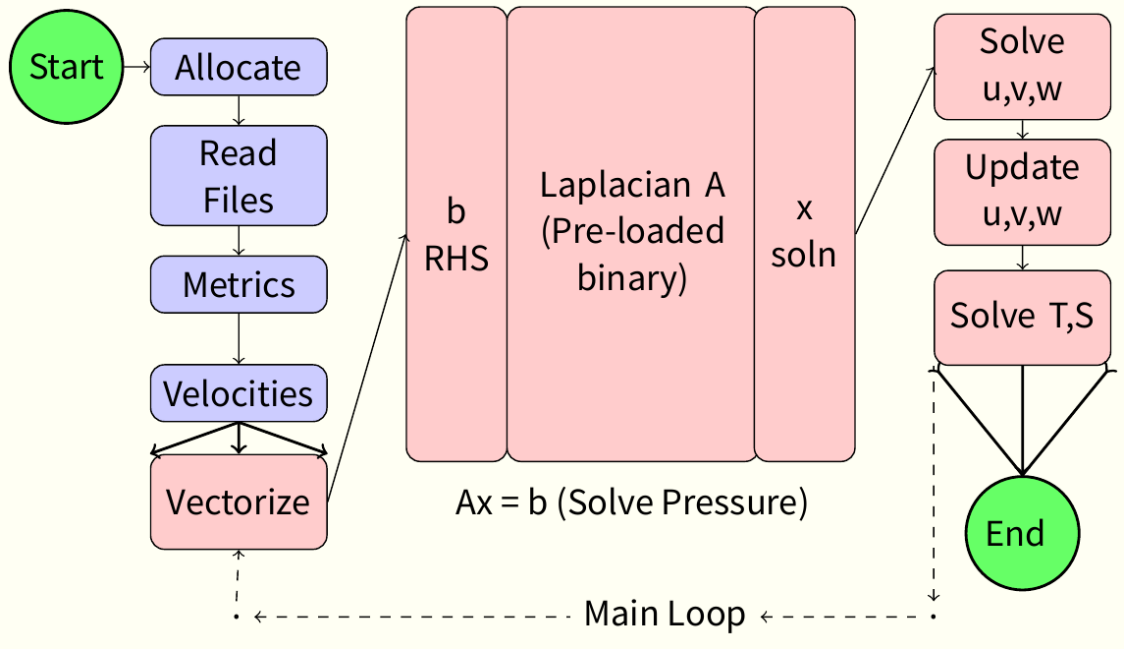


FIGURE 3.1 Parallelization scheme for the GCCOM after distributing arrays with DMDA objects.

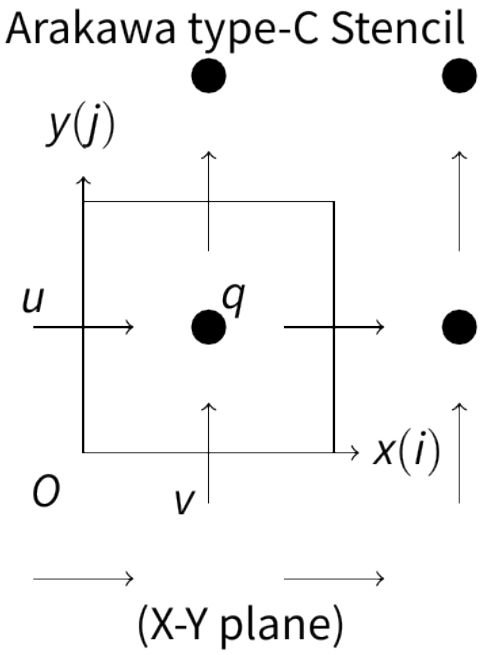


FIGURE 3.2 Minimum cell of the Arakawa C-type stencil, three grids overlapped can be appreciated.

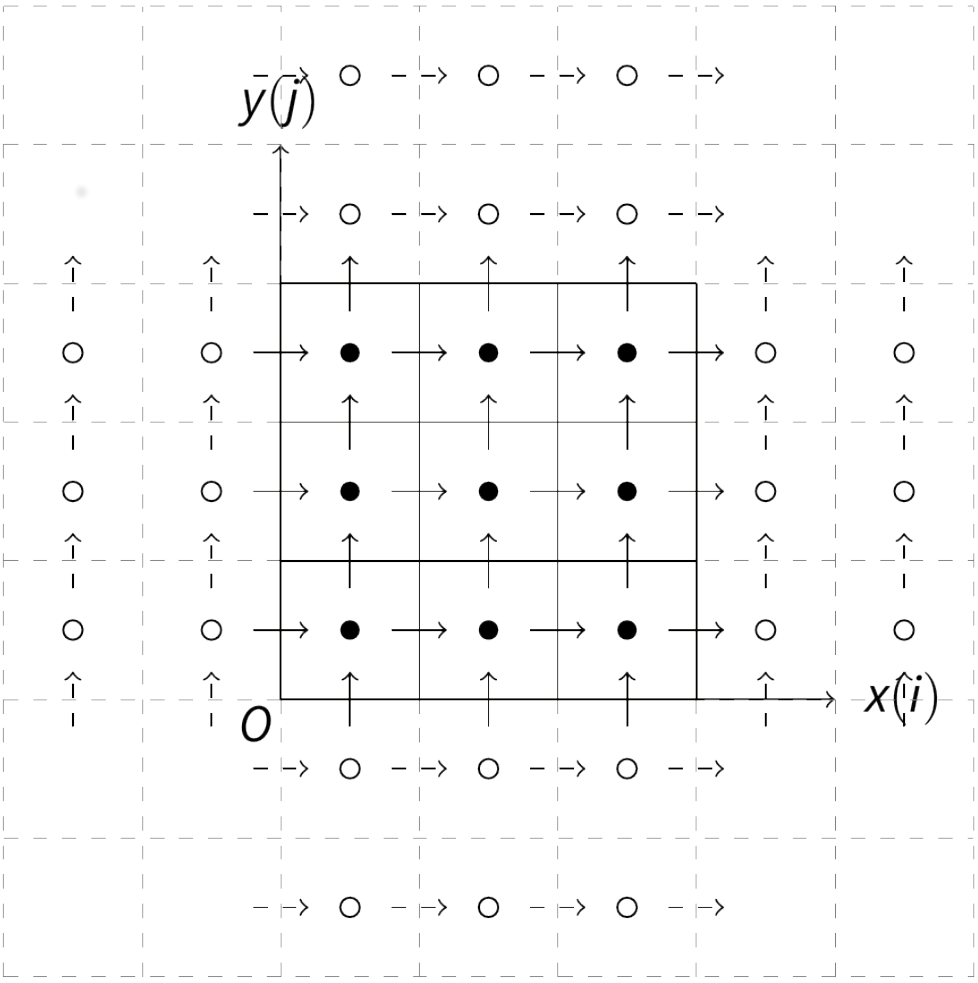


FIGURE 3.3 Arakawa C-type stencil with two rows of ghost boundaries.

TABLE 3.1 Different array layouts in GCCOM. They must all be translated to DA layout.

Serial:	Size	Remap
da	[1:IMax,1:JMax,1:KMax]	-1
dac	[1:IMax-1,1:JMax-1,1:KMax-1]	-1
dau	[1:IMax,1:JMax-1,1:KMax-1]	-1
dav	[1:IMax-1,1:JMax,1:KMax-1]	-1
daw	[1:IMax-1,1:JMax-1,1:KMax]	-1
dasb	[-1:IMax+1,-1:JMax+1,-1:KMax+1]	+1
daub	[-1:IMax+2,-1:JMax+1,-1:KMax+1]	+1
davb	[-1:IMax+1,-1:JMax+2,-1:KMax+1]	+1
dawb	[-1:IMax+1,-1:JMax+1,-1:KMax+2]	+1
DA	[0:IMax+4,0:JMax+4,0:KMax+4]	

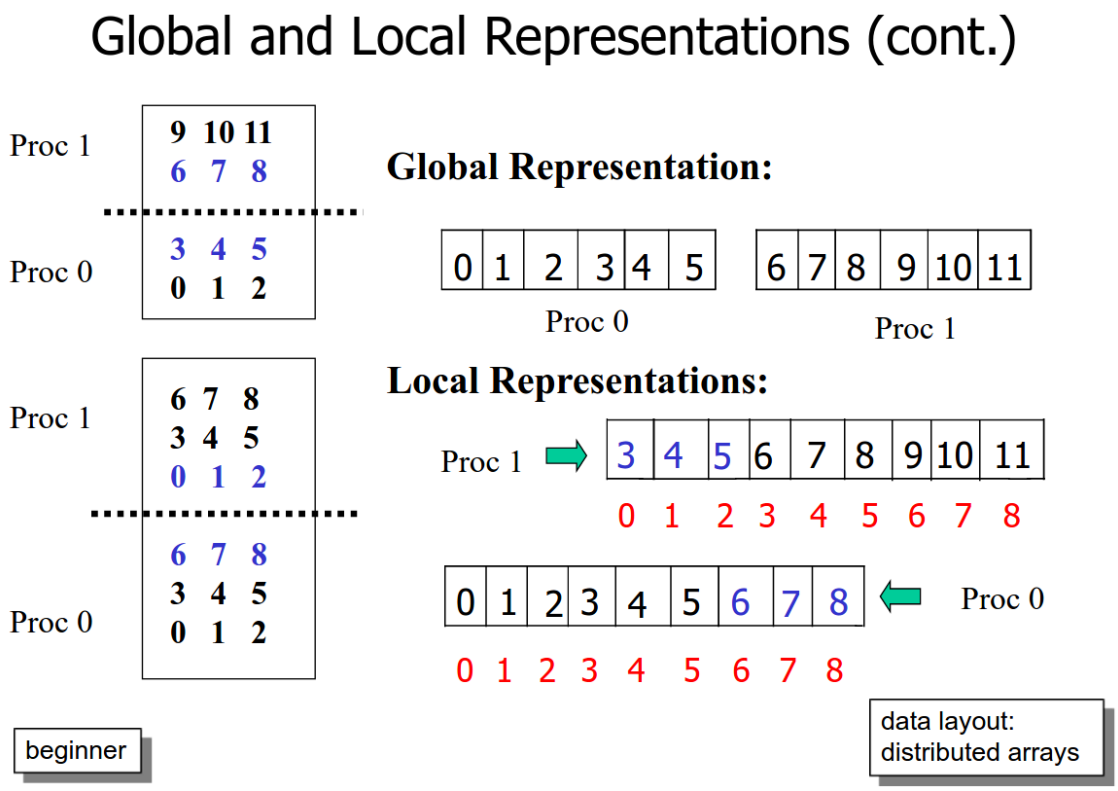


FIGURE 3.4 Global and local representation of data in our PETSc DMDA layout. Most of our system will be developed in local representation.

each grid call must be rewritten by adding the remap factor above. As seen from the table, the layouts have also different sizes, which must also be equalized to DA dimensions so all arrays have the same number of elements and so PETSc divides them at the same point and one common DMDA may be used. Another important factor is that PETSc automatically divides the data boundaries according to the number of nodes available, so a general rule to access the correct division values needs to be created, for this, is important to understand the concept of Global and Local indices in our PETSc system, this can be appreciated in Figure 3.4. We will use local indices whenever possible.

We developed subroutines that helps us attain of all the work described above, it will be introduced in the next section.

Next Steps. From the existing point, every one of the arrays must be created in a PETSc data type using the same DA object, and their calls inside the main loop must be updated according to Table 3.1. From there, tests need to be carried to assess the array ordering is correct and the results are consistent, this includes a way to communicate the arrays taking into account the local ordering division done by PETSc. The list of routines that need updating includes: `SolveU_explicit_RK3()`, `SolveV_explicit_RK3()`, `SolveW_explicit_RK3()`, `SolveP_Rhs()`, `CorrectU4Pre`

3.2.1 Auxiliary routines

The following code uses no specific boundary and creates a DMDA in 1D, which could be used to distribute and operate in a 1D array, for example.

In this call, the dim1d argument refers to the total size of our vector, the next argument

Code 10 Creation of a DMDA in 1D.

```
 SUBROUTINE CDA1(da1d,dim1d)
DM, intent(inout)         :: da1d
PetscInt, intent(in)      :: dim1d
DMBoundaryType            :: bx

bx = DM_BOUNDARY_NONE

call DMDACreate1d(PETSC_COMM_WORLD,DM_BOUNDARY_NONE,dim1d,1,1,PETSC_NULL_INTEGER,da1d,ierr)

call DMSetFromOptions(da1d,ierr)
call DMSetUp(da1d,ierr)

END SUBROUTINE CDA1
```

being 1 refers to the number of variables we want to store/access within this object, it could be more than one if we wanted to operate identically several equal sized 1D arrays. Next we set as 1 our stencil width, which for GCCOM we only access the first neighbors in all directions. Finally, the second to last argument is the local size of the array that will allowed in each processor, this is left for PETSc to decide.

Next routine does the same for a 2D and 3D array, we would use this routine if we needed to distribute a matrix.

Code 11 Creation of a DMDA in 2D.

```
 SUBROUTINE CDA2(da2d,dim2dx,dim2dy)

DM, intent(inout)         :: da2d
PetscInt, intent(in)      :: dim2dx,dim2dy
DMBoundaryType            :: bx,by

bx = DM_BOUNDARY_NONE
by = DM_BOUNDARY_NONE

call DMDACreate2d(PETSC_COMM_WORLD,bx,by,DMDA_STENCIL_BOX,dim2dx,dim2dy,&
PETSC_DECIDE,PETSC_DECIDE,1,1,PETSC_NULL_INTEGER,PETSC_NULL_INTEGER,da2d,ierr)

call DMSetFromOptions(da2d,ierr)
call DMSetUp(da2d,ierr)

END SUBROUTINE CDA2
```

For the 2D and 3D cases we have the same arguments repeated for each dimension, with the addition of a `stencyl_type` argument we have set to `DMDA_STENCIL_BOX` because GCCOM uses the corner points from the stencil as well as the first neighbors in the canonical directions.

Next, as we mentioned before, we need a way to resize our arrays into the final size we will need to operate with them with only one DA layout, that is currently attained with the

Code 12 Creation of a DMDA in 3D.

```
 SUBROUTINE CDA3(da3d,dim3dx,dim3dy,dim3dz)


DM, intent(inout)         :: da3d
PetscInt, intent(in)      :: dim3dx,dim3dy,dim3dz
DMBoundaryType            :: bx,by,bz

!Not sure if we need ghosted boundaries TODO

!GHOSTED generally breaks the DA indices counting TODO

bx = DM_BOUNDARY_NONE
by = DM_BOUNDARY_NONE
bz = DM_BOUNDARY_NONE
width = 2

call DMDACreate3d(PETSC_COMM_WORLD,bx,by,bz,DMDA_STENCIL_BOX,dim3dx,dim3dy,dim3dz,PETSC_DECIDE,

call DMSetFromOptions(da3d,ierr)
call DMSetUp(da3d,ierr)

END SUBROUTINE CDA3
```

following code:

Next code opens a DMDA object into writable mode:

And so we also need an auxiliary routine to close our DMDA object:

Finally, sometimes we need to duplicate a vector inside the local context of a DMDA, which is done with the following routine:

3.2.2 Diagnostics routines

3.2.3 Tests Carried

Basic communication tests were carried for a 3x3x3 array, in which the data distribution was seen working. Further full-scale tests must be implemented and done.

3.2.4 Example Batch Script

The ideal way of executing PETSc+GCCOM code in parallel is with the help of a batch script that takes advantage of the core binding options and the qsub system, such bath script is of the like of:

Code 13 Resize an array for PETSc.

```
 SUBROUTINE DAexp(da,array)

PetscScalar,allocatable,dimension(:,:,:)  :: array
PetscScalar, allocatable, dimension(:,:,:) :: tmp_arr
DM,intent(inout)              :: da

if(sizex/=1)then
if(rank==0)then
allocate(tmp_arr(IMax+4,JMax+4,KMax+4))
tmp_arr( 1:size(array,1),1:size(array,2),1:size(array,3) ) = array
deallocate(array)
allocate(array(IMax+4,JMax+4,KMax+4))
array = tmp_arr
deallocate(tmp_arr)

endif
endif

END SUBROUTINE DAexp
```

Code 14 Open a DMDA object to write.

```
 SUBROUTINE ODA(da,localv,array)

Vec,intent(inout)             :: localv
PetscScalar,pointer,intent(inout)     :: array(:,:,:)
DM,intent(inout)              :: da


call DMDAVecGetArrayF90(da,localv,array,ierr)

END SUBROUTINE ODA
```

Code 15 Close a DMDA object.

```
SUBROUTINE CDA(da,globalv,localv,array)

Vec,intent(inout)             :: globalv,localv
PetscScalar,pointer,intent(inout)     :: array(:,:,:)
DM,intent(inout)              :: da

call DMDAVecRestoreArrayF90(da,localv,array,ierr)
call DMRestoreLocalVector(da,localv,ierr)


END SUBROUTINE CDA
```

Code 16 Duplicate a local vector inside a DMDA object.

```
SUBROUTINE DupVecDA(da,localv,copyv,array)
Vec,intent(inout)             :: localv,copyv
PetscScalar,pointer,intent(inout)     :: array(:,:,:)
DM,intent(inout)              :: da


call VecDuplicate(localv,copyv,ierr)
call DMDAVecGetArrayF90(da,localv,array,ierr)

END SUBROUTINE DupVecDA
```

Code 17 Get the size information for a DMDA object.

```
SUBROUTINE DAinf(da,xind,yind,zind,xwidth,ywidth,zwidth)

DM,intent(inout)              :: da
PetscInt         :: gdimx, gdimy,gdimz, procX,procY,procZ
DMBoundaryType   :: bx, by, bz
PetscScalar      :: val=2
PetscInt         :: xind_gh, yind_gh,zind_gh, xwidth_gh,ywidth_gh,zwidth_gh
PetscInt,intent(inout)    :: xind, yind,zind, xwidth, ywidth,zwidth
PetscInt         :: xstart, xend, ystart, yend,zstart,zend

!Init all to zero just in case:
gdimx=0 ; gdimy=0; gdimz=0; procX=0; procY=0; procZ=0;
xind_gh=0; yind_gh=0; zind_gh=0; xwidth_gh=0; ywidth_gh=0; zwidth_gh=0;
xind=0; yind=0; zind=0; xwidth=0; ywidth=0; zwidth=0;
xstart=0; xend=0; ystart=0; yend=0; zstart=0; zend=0;


call DMDAGetCorners(da,xind,yind,zind,xwidth,ywidth,zwidth,ierr)

call DMDAGetGhostCorners(da,xind_gh,yind_gh,zind_gh,xwidth_gh,ywidth_gh,zwidth_gh,ierr)

call DMDAGetInfo(da,PETSC_NULL_INTEGER,gdimx,gdimy,gdimz, &
& procx,procy,procz,&
& PETSC_NULL_INTEGER,PETSC_NULL_INTEGER,PETSC_NULL_INTEGER, &
& PETSC_NULL_INTEGER,PETSC_NULL_INTEGER,PETSC_NULL_INTEGER,ierr)

END SUBROUTINE DAinf
```

Code 18 Print the information from a DMDA object.

```
 SUBROUTINE DAprint(da)

DM,intent(in)    :: da
PetscInt         :: gdimx, gdimy,gdimz, procX,procY,procZ
DMBoundaryType   :: bx, by, bz
PetscScalar      :: val=2
PetscInt         :: xind_gh, yind_gh,zind_gh,xwidth_gh,ywidth_gh,zwidth_gh
PetscInt         :: xind, yind,zind, xwidth, ywidth,zwidth
PetscInt         :: xstart, xend, ystart, yend,zstart,zend

call DAinf(da,xind,yind,zind,xwidth,ywidth,zwidth)

print*,'****Corners Rank',rank,' ****'
print *,'rank',rank,'xind  =',xind,'yind  =',yind,'zind  =',zind
print *,'rank',rank,'xwidth=',xwidth,'ywidth=',ywidth,'zwidth=',zwidth

print*,'****GhostCorners Rank',rank,' ****'
print *,'rank',rank,'xind_gh  =',xind_gh,'yind_gh  =',yind_gh,'zind_gh  =',zind_gh
print*,'rank',rank,'xwidth_gh=',xwidth_gh,'ywidth_gh=',ywidth_gh,'zwidth_gh=',zwidth_gh

! Print everyone's indices on which they will compute on, and the info from
! DMDAGetInfo()
print *, '****Info Rank',rank,' ****'
print*,'rank',rank,'xstart=',xstart,'xend=',xend
print*,'rank',rank,'ystart=',ystart,'yend=',yend
print*,'rank',rank,'zstart=',zstart,'zend=',zend
print*,'rank',rank,'gdimx=',gdimx,'gdimy=',gdimy,'gdimz=',gdimz
print*,'rank',rank,'procX=',procX,'procY=',procY,'procZ=',procZ

END SUBROUTINE DAprint
```

Code 19 Batch script example for a PETSc+GCCOM testcase.

```
#!/bin/bash
#Batch controller
# run with:
#  qsub batch.PP
#PBS -V
#PBS -l nodes=1:ppn=4:mpi
#PBS -N MRBig
#PBS -j oe
#PBS -r n
#PBS -q batch

cd $PBS_O_WORKDIR
echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo JobID is $PBS_JOBID

mpirun -n 4 --map-by core --bind-to core --machinefile $PBS_NODEFILE ./ucmsMR -log_view

# EOF
```

Appendices

CHAPTER A

Monterey Bay Testcase

The Monterey Bay is the largest underwater canyon of the West Coast [10], and such, is an ideal media for internal mixing to rise in it, such behavior have been documented and insitu measurements have been taken. This is of special interest for our GCCOM model as a mean of validation with real data, with that goal in mind a Monterey Bay stratified water 3D model have been developed and is in testing phase. Results are not conclusive since the high resolution needed and the long simulation times are still a challenge for our model, nonetheless we present here our latest results:

Figure A.1 shows our grid in place, this testcase have been created in two resolutions, one with full length in the x-axis (2km at 10m per point) and other one with a quarter length (0.5km). The boundary conditions are closed and reflecting in the end near the shore and forced externally with the tidal wave in the open ocean side, Figure A.2 shows this outer forcing in place.

We used full thermodynamics capabilities of the GCCOM in the stratified water, such temperature profile and the visualization probes in place can be seen in Figure A.3.

The entire simulation time needed for the interest phenomena to arise takes around six hours, which in GCCOM computational time took about two months in the case of the 0.5km grid. The full-scale grid is still running. There was, however, a instability at the end of the simulation, this can be appreciated in Figures A.4, A.5. This kind of instability seems to be high frequencies being reflected back at the end of the domain and they show more strongly in the vertical direction. This was a bug discovered after this simulation started running and since have been corrected.

As to this date, a new set of Monterey testcases with 2km, 0.5km and 0.25km are running in our cluster with the latest version of our code, which solves these bugs. Also, simulation time have been doubled from 12 to 24 hours total.

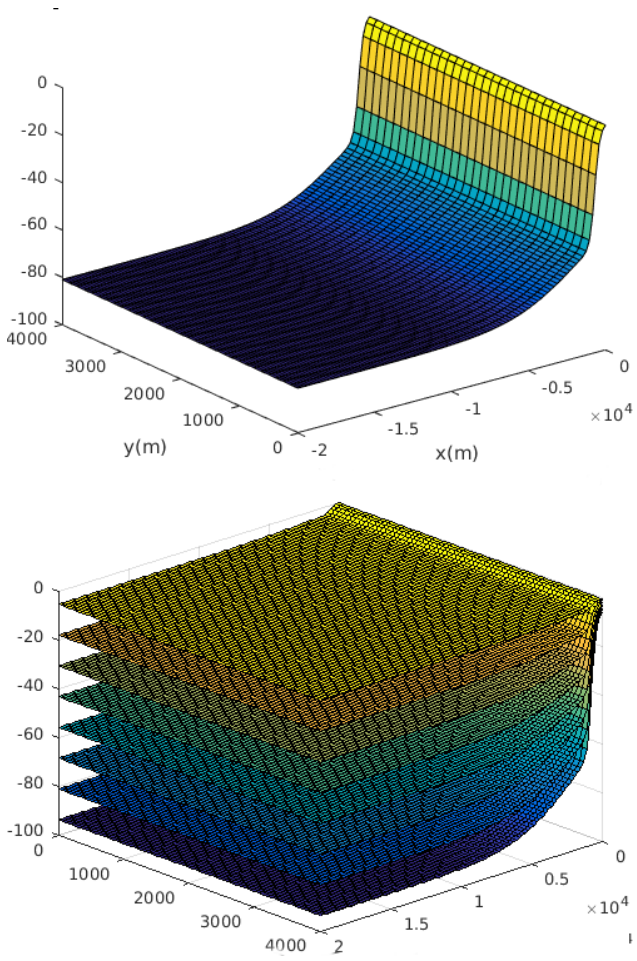


FIGURE A.1 Monterey sigma grid in place for cases 2001x6x81 and 501x6x81.

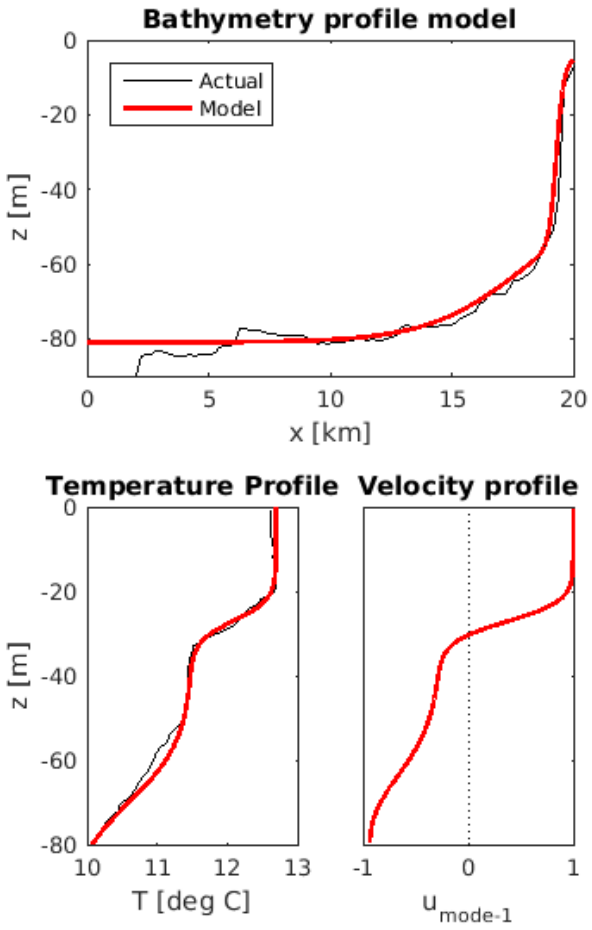


FIGURE A.2 Forcing condition for the open sea boundary of Monterey.

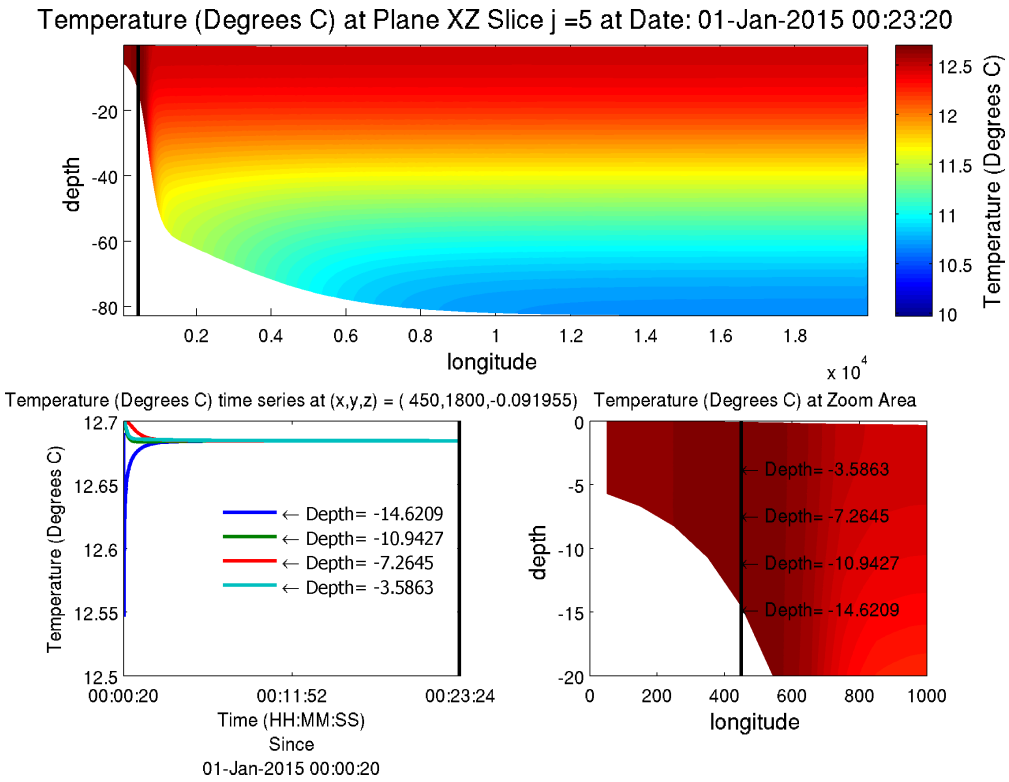


FIGURE A.3 Temperature profile in Monterey Bay Testcase at t=0.

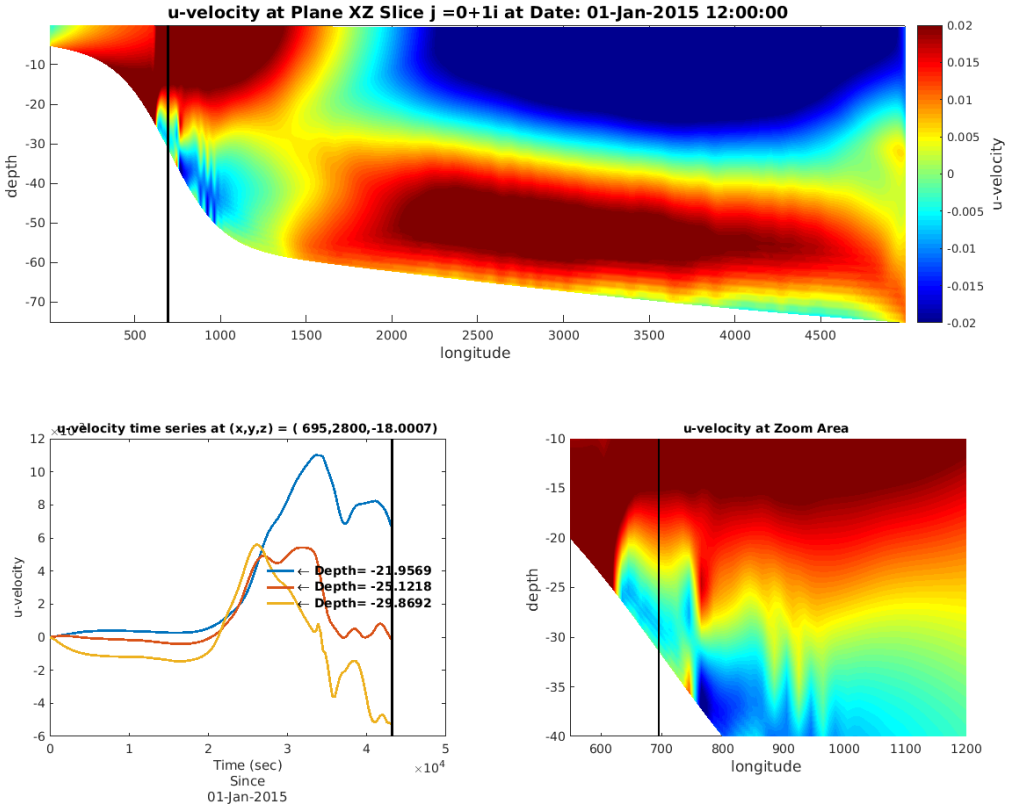


FIGURE A.4 u velocity instabilities in Monterey Bay Testcase after 12hrs of simulation time.

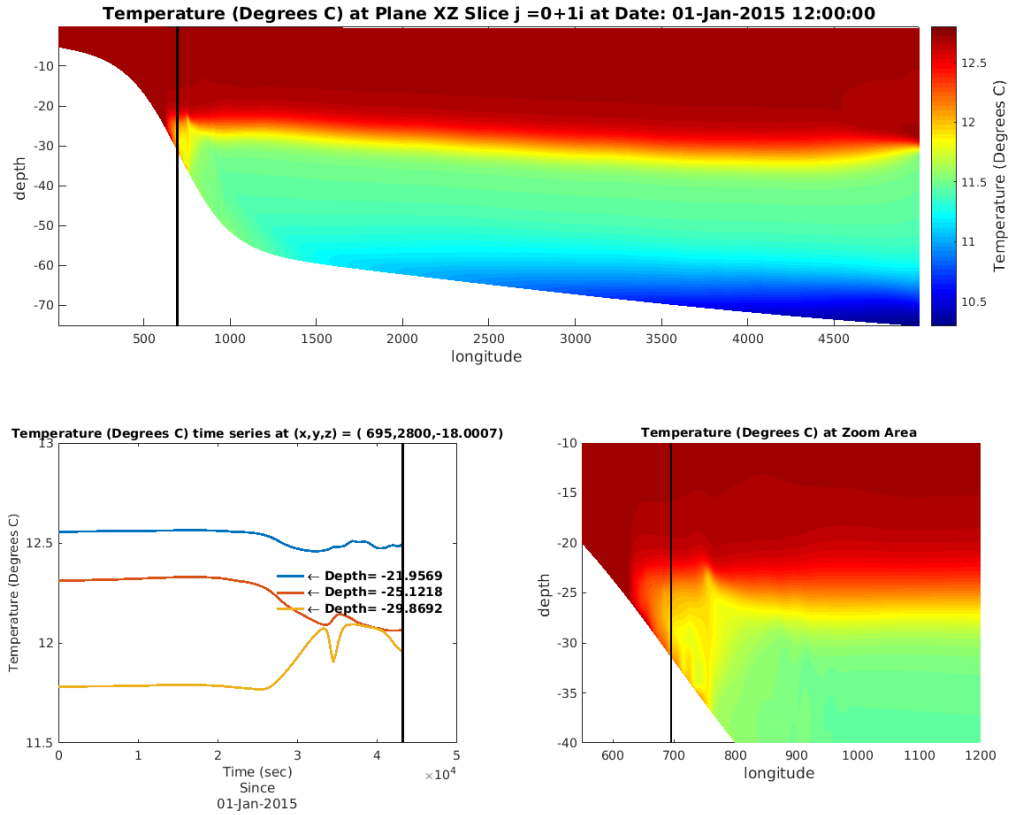


FIGURE A.5 Temperature instabilities in Monterey Bay Testcase after 12hrs of simulation time.

Bibliography

[1] M. E. Assessment, "M ARINE AND COASTAL ECOSYSTEMS AND HUMAN WELL-BEING."

[2] A. D.M., B. Reguera, G. Pitcher, and H. Enevoldsen, "THE OffICIAl MAGAzINE Of THE OCEANOGRAPHY SOCIETY," Oceanography, vol. 24, no. 3, pp. 162–173, 2011. [Online]. Available: http://dx.doi.org/10.5670/oceanog.2011.80.

[3] C. Torres, "Modeling Ocean Circulation in Boundary-fitted Coordinates: The GCOM Project." in III Pan-American Adv. Stud. Inst. Comput. Sci. Eng., 2006, pp. 1–37.

[4] M. Abouali and J. E. Castillo, "General Curvilinear Ocean Model (GCOM) Manual." Tech. Rep., 2010.

[5] ——, "Unified Curvilinear Ocean Atmosphere Model (UCOAM): A vertical velocity case study," Mathematical and Computer Modelling, vol. 57, no. 9-10, pp. 2158–2168, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.mcm.2011.03.023

[6] M. Garcia and J. E. Castillo, "Data Assimilation Unit for the General Curvilinear Environmental Model," Ph.D. dissertation, 2015.

[7] M. P. Thomas and J. E. Castillo, "Parallelization of the 3D Unified Curvilinear Coastal Ocean Model: Initial Results." in Int. Conf. Comput. Sci. Its Appl., 2012, pp. 88–96.

[8] Y. Notay, "Aggregation-based algebraic multigrid for convection-diffusion equations ," vol. 34, no. 4, pp. 2288–2316, 2012.

[9] N. L. Argonne, "PETSc Users Manual," Tech. Rep.

[10] R. K. Walter, C. Brock Woodson, R. S. Arthur, O. B. Fringer, and S. G. Monismith, "Nearshore internal bores and turbulent mixing in southern Monterey Bay," Journal of Geophysical Research: Oceans, vol. 117, no. 7, pp. 1–13, 2012.

[11] M. Valera, M. Garcia, R. K. Walter, P. Choboter, and J. E. Castillo, "Modeling nearshore internal bores and waves in Monterey Bay using the General Curvilinear Coastal Ocean Model (GCCOM)," in ACSESS 2016, no. 7, San Diego, 2016.

[12] M. Valera, N. Patel, and J. E. Castillo, "PETSc-based Parallelization of the fully 3D-curvilinear Non-hydrostatic Coastal Ocean Dynamics Model , GCCOM," in SRS - Student Research Symposium, San Diego State University., San Diego, 2017.

[13] S. K. Venayagamoorthy and O. B. Fringer, "On the formation and propagation of nonlinear internal boluses across a shelf break," Journal of Fluid Mechanics, vol. 577, pp. 137–159, 2007.

[14] G. S. Carter, M. C. Gregg, and R.-c. Lien, "Internal waves , solitary-like waves , and mixing on the Monterey Bay shelf," vol. 25, pp. 1499–1520, 2005.

[15] E. Model, "San Diego State University and Claremont Graduate University," no. December, 2015.

[16] S. O. Centre and U. Kingdom, "The Generation of Alongslope Currents by Breaking Internal Waves," no. 1974, pp. 29–38, 1999.

[17] E. Santilli and A. Scotti, "The Stratified Ocean Model with Adaptive Refinement," Journal of Computational Physics, vol. 291, pp. 60–81, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2015.03.008

[18] R. K. Walter, N. J. Nidzieko, and S. G. Monismith, "Similarity scaling of turbulence spectra and cospectra in a shallow tidal flow," vol. 116, no. March, pp. 1–14, 2011.

[19] A. N. A.-b. Algebraic and M. Method, "An aggregation-based algebraic multigrid method ," vol. 37, pp. 123–146, 2010.

[20] E. Kunze and S. G. L. Smith, "Special Issue Bathymetry from Space The Role of Small-Scale Topography in Turbulent Mixing of the Global Ocean," vol. 17, no. 1, pp. 55–64, 2004.

[21] E. Santilli and C. Evans, "THE STRATIFIED OCEAN MODEL WITH ADAPTIVE REFINEMENT (SOMAR)," Ph.D. dissertation, 2015.

[22] C. B. Woodson, J. A. Barth, O. M. Cheriton, M. A. Mcmanus, J. P. Ryan, L. Washburn, K. N. Carden, B. S. Cheng, J. Fernandes, L. E. Garske, T. C. Gouhier, A. J. Haupt, K. T. Honey, M. F. Hubbard, A. Iles, L. Kara, M. C. Lynch, B. Mahoney, M. Pfaff, M. L. Pinsky, M. J. Robart, J. S. Stewart, S. J. Teck, and A. True, "Observations of internal wave packets propagating along shelf in northern Monterey Bay," vol. 38, no. July 2008, pp. 1–6, 2011.

[23] N. E. T. Al, "Internal Tide Reflection and Turbulent Mixing on the Continental Slope," pp. 1117–1134, 2004.

[24] "Munk_Wunsch_DSR_1998_32129.pdf."

[25] S. S. K. Venayagamoorthy and O. B. O. Fringer, "Internal wave energetics on a shelf break," International Journal of Offshore and Polar Engineering, vol. 17, no. 1, pp. 22–29, 2007. [Online]. Available: http://www.onepetro.org/mslib/servlet/onepetropreview?id=ISOPE-07-17-1-022

[26] J. Pineda, "Internal tidal bores in the nearshore: Warm-water fronts, seaward gravity currents and the onshore transport of neustonic larvae," Journal of Marine Research, vol. 52, no. 3, pp. 427–458, 1994.

[27] A. Napov and Y. Notay, "An algebraic multigrid method with guaranteed convergence rate," 2011.

[28] A. Fallis, "No Title No Title," Journal of Chemical Information and Modeling, vol. 53, no. 9, pp. 1689–1699, 2013.

[29] M. Garcia, "Data Assimilation Unit (DAU). Dissertation Defense." Ph.D. dissertation, 2015.

[30] A. Dissertation, "Investigating Castillo-Grone ' s Mimetic Difference Operators in Development of Geophysical Fluid Dynamics Models Implemented on GPGPUs by Mohammad Abouali," 2014.

[31] S. Venayagamoorthy and O. Fringer, "Examining Breaking Internal Waves on a Shelf Slope Using Numerical Simulations," Oceanography, vol. 25, no. 2, pp. 132–139, 2012. [Online]. Available: http://tos.org/oceanography/archive/25-2{_}venaya.html

[32] M. Abouali and J. E. Castillo, "Stability and performance analysis of the Castillo-Grone Mimetic operators in conjunction with RK3 time discretization in solving advective equations," Procedia Computer Science, vol. 18, pp. 465–472, 2013.

[33] S. B. Pope, "Ten questions concerning the large-eddy simulation of turbulent flows," New Journal of Physics, vol. 6, 2004.

[34] U. Piomelli, "Large-eddy simulation: achievements and challenges," Progress in Aerospace Sciences, vol. 35, no. 4, pp. 335–362, 1999.

[35] L. C. Berselli, M. Cerminara, and T. Iliescu, "Disperse Two-Phase Flows, with Applications to Geophysical Problems," Pure and Applied Geophysics, vol. 172, no. 1, pp. 181–196, 2014.

[36] M. Abouali and J. E. Castillo, "Shallow Water Equations Implemented on GPUs," vol. 1245, no. 619, 2014.

[37] M. Melaku, S. Thompson, and O. D. Company, "Groundwater Hydrology CE 40460 Example 1 – Grid Approach Building," 1970, pp. 1–4.

[38] S. K. Venayagamoorthy and O. B. Fringer, "On the formation and propagation of nonlinear internal boluses across a shelf break," Journal of Fluid Mechanics, vol. 577, p. 137, 2007.

[39] M. Abouali and J. E. Castillo, "Unified Curvilinear Ocean Atmosphere Model (UCOAM): A vertical velocity case study," Mathematical and Computer Modelling, vol. 57, no. 9-10, pp. 2158–2168, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.mcm.2011.03.023