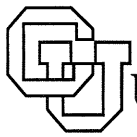
Portable, Efficient, Parallelization of a 3d Quasi-Geostrophic Multi-Grid code *

Clive F. Baillie
James C. McWilliams
Jeffrey B. Weiss
Irad Yavneh

CU-CS-789-95



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

*Paper for Cray Users Group '95, September 25-29, 1995

Portable, Efficient, Parallelization of a 3d Quasi-Geostrophic Multi-Grid code *

Clive F. Baillie
Department of Computer Science
University of Colorado, Boulder, CO 80309

James C. McWilliams
Department of Atmospheric Science
University of California, Los Angeles, CA 90095-1565

Jeffrey B. Weiss
Program in Atmospheric and Oceanic Sciences, APAS
University of Colorado, Boulder, CO 80309

Irad Yavneh
Department of Computer Science
Technion - Israel Institute of Technology, Haifa, Israel

Abstract

We have parallelized our existing auto-tasked vector Cray C-90 3d Quasi-Geostrophic Multi-Grid (QGMG) code in a portable and efficient way for todays MPPs. The QGMG code addresses one of the most important computational problems today: the numerical simulation of high Reynolds number fluid turbulence. In this paper we describe how we did this parallelization, emphasising both portability and efficiency. In addition we discuss how we implemented our own parallel I/O so that the MPP code can read and write original C-90 files. We give performances on the Cray T3D MPP. The bottom line is that on all 16 processors of the C-90 the code achieved 6 Gflops; currently on 512 processors of the T3D it is slightly faster.

*Paper for Cray Users Group '95, September 25-29, 1995

1 Introduction

One of the most important computational problems today is the numerical simulation of high Reynolds number fluid turbulence. As members of the NSF HPCC Grand Challenge Applications Group (GCAG), "Coupled Fields and Geophysical and Astrophysical Fluid Dynamical Turbulence", we are studying incompressible fluid dynamics in several regimes involving environmental rotation and/or stable or unstable density stratification, all of which are motivated by geophysical phenomena. The common theme in this GCAG research is that significant new insights into the dynamics of turbulence can be obtained from high resolution, high Reynolds number computational solutions obtained with efficient algorithms on Massively Parallel Processors (MPPs).

We use the quasi-geostrophic equations to describe the nonlinear dynamics of rotating, stably stratified fluids. The computational methods used to solve these equations are explicit and implicit multigrid (MG) solvers. We have developed an efficient implicit Quasi-Geostrophic Multi-Grid (QGMG) solver and used it to investigate fluids with periodic horizontal boundary conditions and various vertical boundary conditions: periodic, solid-boundary and Ekman drag [1].

This paper is organized as follows. In section 2 we explain the quasi-geostrophic equations. In section 3 we outline the implementation of the QGMG code. The performance of the code on the Cray T3D is given in section 4, and we end with some conclusions.

2 The Quasi-Geostrophic equations

Planetary-scale fluid motions in the Earth's atmosphere and oceans are influenced by strong stable stratification and rapid planetary rotation. The appropriate equations of motion for this asymptotic regime are the Quasi-Geostrophic (QG) equations [2]. The extremely turbulent nature of planetary flows leads us to perform high-resolution numerical simulations of QG turbulence in an effort to better understand the large-scale flows which are so important to the Earth's climate.

Due to stratification and rotation, QG flow is nearly incompressible in horizontal planes, and can be described in terms of horizontal velocities u and v, and an associated streamfunction $\psi(x,y,z)$: $u = -\partial\psi/\partial y$, $v = \partial\psi/\partial x$. The resulting small vertical velocities, together with the thinness of the Earth's atmosphere and oceans, make it appropriate to use a vertical coordinate stretched by N/f. Here, f is the Coriolis frequency, equal to twice the vertical component of the planetary rotation rate, and N is the vertical average of $N(z)$, the Brunt-Väisälä frequency, which is related to gradients in the mean density profile $\rho(z)$ and the acceleration due to gravity g by $N^2(z) = g\partial \ln \rho/\partial z$. On

the Earth, N/f is typically of order 100. In this stretched coordinate system the QG equations of motion are

$$\frac{\partial q}{\partial t} + \frac{\partial \psi}{\partial x}\frac{\partial q}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial q}{\partial x} + \beta\frac{\partial \psi}{\partial x} = -\mathcal{D}, \tag{1}$$

where the potential vorticity q is

$$q = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} + \frac{\partial}{\partial z}\left(\frac{1}{S(z)}\frac{\partial \psi}{\partial z}\right). \tag{2}$$

Vertical inhomogeneity in the stratification is represented by $S(z) = N^2(z)/N^2$, and the so-called β-plane approximation is used to include the effect of the variation in f with latitude, $\beta = \partial f/\partial y$. The dissipation operator $\mathcal{D}$ represents the effects of all scales of motion smaller than those explicitly resolved in the numerical calculation; we typically use hyperviscous diffusion, $\mathcal{D} = \nu\nabla^4 q$, where ν is a small hyperviscosity [3]. Thus there are only two main variables in the QG code: ψ and q.

Over the last few years considerable effort has been invested into adapting and developing multigrid techniques for non-elliptic and singular perturbation problems, such as the flows found at high Reynolds number in stably stratified fluids. Using tools developed by Yavneh [4] we now have multilevel algorithms for time-dependent systems that describe geophysical flows, such as QG. Integration of the QG equations requires solving an elliptic boundary-value problem in three dimensions even if the nonlinear advection equation for the potential vorticity is integrated explicitly. We use a multigrid algorithm, which is one of the best known methods for this problem. Furthermore, we discretize the nonlinear advection equation implicitly in time and solve the entire system simultaneously, employing the so-called Full Approximation Storage (FAS) version of the multigrid algorithm. We use grid-coarsening only in the horizontal directions.

Our implementation of the above equations of motion allow us to study QG turbulence at unprecedented resolutions. Previous computations on the Cray C-90 focused on the maximally symmetric case, $S = 1$, $\beta = 0$ [1, 5, 6], where we found significant discrepancies from a long-standing theoretical prediction of isotropy [7, 8]. Associated with this anisotropy is the self-organization of the potential vorticity field into a large population of roughly spherical coherent vortices, which then align in the vertical. Currently we are performing several new computations on the Cray T3D – investigating the effects of including nonconstant S and nonzero β, in both decaying and equilibrium turbulence. From January to September 1995 we have consumed almost 14 T3D processor years of CPU time.

3 Implementation of QGMG

In order to parallelize a code like QGMG for a MPP, one writes a message passing program with each processor responsible for part of the problem domain. The message passing system used on the T3D is Cray's version of the Parallel Virtual Machine (PVM) software from Oak Ridge National Laboratory [9]. Therefore we parallelized QGMG using PVM which is portable between a great many machines.

The QGMG code employs the typical V-cycle multigrid, restricting all the way down to a 4×4 coarsest grid. As explained above, coarsening is performed only for the two horizontal (out of the three) dimensions so in what follows we ignore the third dimension for the sake of clarity. We discuss each of the multigrid's three parts – relaxation, restriction and interpolation – in turn, first sequentially and then for the parallel version. Here we give only a summary, a more detailed explanation can be found in [10].

3.1 Sequential

Relaxation of both ψ and q is performed using the Gauss-Seidel algorithm. For q four-color ordering is necessary since we use the Arakawa nine-point discretization stencil for the Jacobian. However for ψ red-black ordering is sufficient (due to five-point stencil for Laplacian). We shall discuss only the simpler code for the red-black checkerboard case i.e. all the even points are updated first, then all the odd points. The sequential relaxation code for ψ looks like:

```
      do ij = 0,1

        do j = 1, bny
          do i = 1+mod(j+ij,2), bnx, 2
            psi(i,j) = 0.25 * (psi(i-1,j) + psi(i+1,j) +
     >      psi(i,j-1) + psi(i,j+1) - rhs(i,j))
          end do
        end do

        call update(psi,bnx,bny,lv)

      end do
```

where `ij`=0 is red and `ij`=1 is black. Note the call to `update`, which exchanges the information on the periodic boundaries.

The restriction operation averages each 2×2 block of points on the fine grid into one point of the coarse grid (`X` denotes point which remains, `0` is point which is destroyed):

```
XOXOXOXO   ->   X X X X
00000000
XOXOXOXO        X X X X
00000000
```

First the residual is calculated on the fine grid and then used on the coarse grid to calculate the right-hand-side.

The most complicated part of the multigrid algorithm is the interpolation from the coarse to the fine grid. The picture is the opposite of the restriction operation (now 0 is a point which is created):

```
X X X X   ->   XOXOXOXO
               00000000
X X X X        XOXOXOXO
               00000000
```

There are actually two ways to do this. The most obvious way is to just send the "top-left-corner" point to the other three points: "top-right-corner", "bottom-left-corner" and "bottom-right-corner", where it is then averaged. However this method is only simple for linear interpolation; for cubic and higher-order interpolation schemes it rapidly becomes complicated due to the number of points required to calculate the average for the "bottom-right-corner" point. Therefore we use a two-step implementation: during the first step the points in the i-direction are interpolated, then in the second step the points in the j-direction. This looks like:

```
X X X X   ->   XOXOXOXO   ->   XOXOXOXO
                               00000000
X X X X        XOXOXOXO        XOXOXOXO
                               00000000
```

3.2 Parallel

It is relatively straight-forward to parallelize both of the sequential implementations detailed above assuming a square grid, and square processor mesh, and that the number of points per processor is always at least one. However, in practice **none** of these assumptions are valid! We can of course choose to use a square rather than a rectangular grid, but only if the physical system being simulated is in a horizontally 1:1 ratio box. We cannot choose the processor mesh to be square because it unduly limits the number of processors that can be used. Finally, the multigrid algorithm restricts all the way down to a 4×4 (or sometimes even a 2×2) grid, therefore having at least one point per processor would restrict simulations to a maximum of 16 (or 4) processors.

Thus, we must develop a general parallel multigrid code which works on non-square girds, on non-square processor meshes and where the number of points can be less than the number of processors. It is this last condition which causes the parallel multigrid code to be more complicated than one would initially think.

Let us assume that the finest grid we are using has `nx` $\times$ `ny` points and the processor mesh consists of `n` $\times$ `m` = `np` processors. We also assume `n` divides `nx` and `m` divides `ny`. We arrange the processors, numbered `me` = `0,1,...,np-1`, as in the following example for a 32 processor 8$\times$4 mesh:

```
 0  1  2  3  4  5  6  7
 8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
```

where we have drawn the first index `i` increasing from left to right and the second index `j` increasing downwards. Using domain decomposition, each processor gets a part of the grid of size `(nx/n)` $\times$ `(ny/m)` starting and ending as follows:

```
start(lv,1) = 1 + mei * (nx/n)
end(lv,1)   = (nx/n) + mei * (nx/n)
start(lv,2) = 1 + mej * (ny/m)
end(lv,2)   = (ny/m) + mej * (ny/m)
```

where `mei` and `mej` are the i, j positions of processor `me` in the grid (e.g. processor `me`=17 has `mei` = 1 and `mej` = 2) and `lv` is the multigrid level. Of course, each processor has its own local indices for these global pieces of the grid, these are simply

```
loops(lv,1) = 1
loope(lv,1)   = (nx/n)
loops(lv,2) = 1
loope(lv,2)   = (ny/m)
```

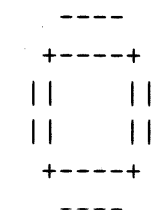
Hence loops in the original code looking like:

```
do bj = 1, bny
  do bi = 1, bnx
```

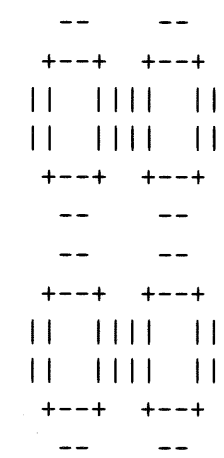
become

```
do bj = loops(lv,2), loope(lv,2)
  do bi = loops(lv,1), loope(lv,1)
```

In addition, this method of domain decomposition takes care of internal boundaries between the processors as follows. In the original code, the 2d grid was surrounded by a layer of "ghost points" which store copies of the points on the opposite sides of the grid in order to take care of the periodic boundary conditions. This looks like:



Therefore when we perform the domain decomposition on, say, 4 processors, each processor will get its own ghost points as follows:



The "outside" layer of ghost points are the original ones for the periodic boundaries. The ones on the "inside", i.e. between the processors, are called "internal processor boundaries" and are an artifact of the domain decomposition. However they are an extremely useful artifact: for in them we shall store copies of the points from the neighboring processors which we shall need while executing the multigrid algorithm. Then we need only exchange the internal boundary data once per call of restrict, interpolate or relax, rather than every time we see an `i+1`, `i-1`, `j+1` or `j-1` index in the code.

As stated above the red-black Gauss-Seidel relaxation is easily parallelized. The only complication is figuring out if a given point `i,j` local to a processor is red or black i.e. even or odd. We do this by having a flag `first_even` telling us whether the first point `i=1`, `j=1` in the processor is even. Thus the parallel relaxation code looks like:

```
do idum = 0,1
```

```
      if (first_even(lv)) then
        ij = idum
      else
        ij = 1-idum
      endif

      do j = loops(lv,2), loope(lv,2)
        do i = loops(lv,1)+mod(j+ij,2), loope(lv,1), 2
          psi(i,j) = 0.25 * (psi(i-1,j) + psi(i+1,j) +
     >      psi(i,j-1) + psi(i,j+1) - rhs(i,j))
        end do
      end do

      call exchange_i(psi,bnx,bny,lv)
      call exchange_j(psi,bnx,bny,lv)

    end do
```

The restriction operation in parallel is straight-forward; it is only the interpolation operation, when the number of points is less than the number of processors, which is a little tricky. To explain this, suppose the grid is 2×2 so the only processors in our 8×4 mesh which have points are

```
 0          4

16         20
```

and when we interpolate to 4×4 we get

```
 0    2    4    6
 8   10   12   14
16   18   20   22
24   26   28   30
```

Therefore the processors no longer communicate with their nearest neighboring processors – as they did when the number of points `bnx` and `bny` is greater than or equal to the number of processors `n` and `m` – now they communicate with processors `skipi` and `skipj` away in the i and j directions respectively, with

```
skipi = n/bnx
skipj = m/bny
```

For both the relaxation and the restriction parts of the multigrid algorithm use of `skipi` and `skipj` in the exchange routines works perfectly well no matter how many points per processor there are. It is only in the interpolate phase that there is any difference. This difference can be reduced to two extra function calls `send_i` and `send_j` invoked only when `skipi` and `skipj` are respectively greater than 1.

3.3 Parallel I/O

Before going onto report the performance of QGMG we briefly discuss the issue of I/O. Despite the fact that MPPs have been around for some time, none of them appear to have any useful vendor provided parallel I/O. For example, on the T3D, one can have all the processors write their part of the global data to separate files or to separate records of one file, which is fine if one always runs on the same fixed number of processors. However as soon as one wants to run on a different number of processors there is no vendor provided software which will read the fragmented files or records back. Therefore we have written our own parallel I/O to get around this problem. After a little reflection the obvious solution is to store the parallel file in the same order as a sequential file would be written by a single processor. Thus when the parallel code is run on N processors, the sequential file is read and split into the appropriate N pieces; then at the end of the run the output file is re-constituted from the N processors. This has the added advantages that the input data files can come from the C-90 and that the output data files can still be read by our analysis programs which run on the C-90.

4 Performance of QGMG

In Table 1 we present times in seconds of a ten multigrid cycle run of QGMG using a grid of size 256^3, on various numbers of processors (and for different processor topologies) for several different versions of the code. Note that this size problem does not fit into the memory of less than 32 T3D processors. When we first implemented QGMG on the T3D, the Cray PVM available was version 3.2 which did not have global reduction operations, that is functions like SUM, MAX, MIN, so we had to write our own. Unfortunately they were very inefficient and did not scale properly, as can be seen from the times in the second column of Table 1. Later, Cray released version 3.3 of PVM which does have global reduction functions and these gave dramatic improvements in performance (column 3). For a further smaller improvement we also tried the so-called shared memory (SHMEM) global reduction functions (column 4). For the PVM 3.3 and SHMEM cases the processor topology

Table 1: Total times in seconds for 10 multigrid cycles of QGMG on various numbers of processors of the T3D.

Processors	PVM 3.2	PVM 3.3	SHMEM	F90
32 (1x32)	321.9	239.8	244.8	-
32 (4x8)	-	209.1	211.7	142.9
64 (1x64)	477.6	144.3	144.6	-
64 (8x8)	-	120.8	116.2	77.1
128 (1x128)	-	99.5	99.9	-
128 (8x16)	-	63.2	60.0	42.7
256 (1x256)	-	95.1	87.2	-
256 (16x16)	-	42.3	31.7	23.4
512 (2x256)	-	43.5	44.0	-
512 (16x32)	-	18.9	18.6	15.0

makes a significant difference: choosing a square configuration (e.g. 16x16) yields more than twice the performance of a linear one (1x256). Recently, it became possible to run in single precision on the T3D by using the Fortran 90 compiler. This effectively doubles the size of the cache, and halves the length of the messages, so QGMG runs significantly faster as shown in the last column of Table 1.

The original QGMG code running on all 16 processors of the C-90 achieved 6.0 Gflops and took 18.9 seconds for this benchmark run. Thus the fastest time we obtained on the T3D in double precision, 18.6 seconds for 512 processors, corresponds to 6.1 Gflops. In single precision, 15.0 seconds is 7.6 Gflops.

5 Conclusions

We have parallelized a 3d quasi-geostrophic multigrid code originally designed for the Cray C-90 in a portable fashion and are running it in production on the Cray T3D. We have done this via domain decomposition and message passing using PVM. On all 16 processors of the C-90 the original code achieved 6 Gflops; currently on 512 processors of the T3D we obtain 6.1 Gflops with the parallel code (in double precision). Currently we are production running large-scale computations of quasi-geostrophic turbulence on the Cray T3D with system sizes of 256^3, to date we have used almost 14 processor years of CPU time.

Acknowledgements

This work is supported by NSF Grand Challenge Applications Group Grant ASC-9217394. CFB is also partially supported by DOE contract DE-FG02-91ER40672 and by NASA HPCC Group Grant NAG5-2218. JBW is also partially supported by NOAA grant DOC-NA-26-GPO-12201. IY and JCM were also partially supported by NSF through the National Center for Atmospheric Research. The Cray T3D runs were performed at the Pittsburgh Supercomputing Center under grant MCA93AS010P through funding from the National Science Foundation. We would like to thank Raghurama Reddy at PSC for his help.

References

[1] I. Yavneh and J.C. McWilliams, "Multigrid solution of stably stratified flows: the quasi-geostrophic equations", submitted to *J. Sci. Comp.* (1995).

[2] P.B. Rhines, *Annu. Rev. Fluid Mech.*, **11**, 401 (1979).

[3] R. Sadourny and C. Basdevant, *C.R. Acad. Sci.*, **39**, 2138 (1981).

[4] I. Yavneh, *SIAM J. Sci. Comput.*, **14**, 1437-1463 (1993).

[5] J.C. McWilliams, J.B. Weiss, and I. Yavneh, *Science*, **264**, 410 (1994).

[6] J.C. McWilliams and J.B. Weiss, *CHAOS*, **4**, 305 (1994).

[7] J.G. Charney, *J. Atmos. Sci.*, **28**, 1087 (1971).

[8] J. Herring, *J. Atmos. Sci.*, **37**, 969 (1980).

[9] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, "PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing" (The MIT Press, Cambridge, MA, 1994).

[10] C.F. Baillie, J.C. McWilliams, J.B. Weiss and I. Yavneh, "Implementation and Performance of a Grand Challenge 3d Quasi-Geostrophic Multi-Grid code on the Cray T3D and IBM SP2", to appear in Proc. Supercomputing '95 (1995).