Optimization of a Parallel Ocean General Circulation Model

Ping Wang
Jet Propulsion Laboratory
California Institute of Technology
MS 168-522, 4800 Oak Grove Drive
Pasadena, CA 91109-8099, U.S.A.
wangp@rockymt.jpl.nasa.gov
http://olympic.jpl.nasa.gov/PERSONNEL/wangp

Daniel S. Katz
Jet Propulsion Laboratory
California Institute of Technology
MS 168-522, 4800 Oak Grove Drive
Pasadena, CA 91109-8099, U.S.A.
Daniel.S.Katz@jpl.nasa.gov
http://nueml.ece.nwu.edu/~dsk/

Yi Chao
Jet Propulsion Laboratory
California Institute of Technology
MS 300-323, 4800 Oak Grove Drive
Pasadena, CA 91109-8099, U.S.A.
yc@comp.jpl.nasa.gov
http://comp.jpl.nasa.gov/~yc/YC/

Abstract:

Global climate modeling is one of the grand challenges of computational science, and ocean modeling plays an important role in both understanding the current climatic conditions and predicting the future climate change. Three-dimensional time-dependent ocean general circulation models (OGCMs) require a large amount of memory and processing time to run realistic simulations. Recent advances in computing hardware have dramatically affected the prospect of studying the global climate. The significant computational resources of massively parallel supercomputers promise to make such studies feasible. In addition to using advanced hardware, designing and implementing a well-optimized parallel ocean code will significantly improve the computational performance and reduce the total research time to complete these studies.

In our present work, we chose the most widely used OGCM code as our base code. This OGCM is based on the Parallel Ocean Program (POP) developed in FORTRAN 90 on the Los Alamos CM-2 Connection Machine by the Los Alamos ocean modeling research group. During the first half of 1994, the code was ported to the Cray T3D by Cray Research using SHMEM-based message passing. Since the code on the T3D was still time-consuming when large problems were encountered, improving the code performance was considered essential.

We have developed several general strategies to optimize the ocean general circulation model on the Cray T3D. These strategies include memory optimization, effective use of arithmetic pipelines, and usage of optimized libraries. The optimized code runs 2 to 2.5 times faster than the original code, which gives significant performance improvements for modeling large scaled ocean flows. Many test runs for both of the original and the optimized code have been carried out on the Cray T3D using various numbers of processors (1-256). Comparisons are made for a variety of real-world problems. A nearly linear scaling performance line is obtained for the optimized code, while the speed up data of the optimized code also shows

excellent improvement over the original code.

In addition to discussing the optimization of the code, we also address the issue of portability. Given the short life cycle of the massively parallel computer, usually on the order of three to five years, we emphasize the portability of the ocean model and the associated optimization routines across several computing platforms. Currently, the ocean modeling code has been ported successfully to the Hewlett Packard (HP)/Convex SPP-2000, and is readily portable to Cray T3E.

This paper reports our efforts to optimize the parallel implementations of the oceanic model. So far, the work has focused on improving the load balancing and single node performance of the code on the Cray T3D. As a result, the atmosphere and ocean model components running side-by-side can achieve a performance level of slightly more than 10 GFLOPS on 512 processors of that machine. We have also developed a user-friendly coupling interface with atmospheric and biogeochemical models, in order to make the global climate modeling more complete and more realistic.

Keywords:
Optimization, Ocean Modeling, Parallel Computations

1. Introduction

One of the principal roles of the ocean in the global heat balance is storage of heat, which moderates seasonal extremes and leads to the contrast between ocean and continent temperatures. In the global warming environment, the ocean could delay the onset of the atmospheric warming by absorbing some of the excess heat trapped in the atmosphere.

Ocean modeling plays an important role in both understanding the current climatic conditions and predicting the future climate change. In situ oceanographic instruments provide only sparse measurements over the world ocean. Although remote-sensed data from satellites cover the globe on the time scale of 2~10 days, it only provides information on the ocean surface. Information below the ocean surface has to be obtained from 3-dimensional ocean general circulation models (OGCMs).

OGCMs are usually used in the following three types of applications: (1) eddy-resolving integrations at ever increasing resolutions, (2) a large number of model sensitivity experiments, and (3) long-period climate integrations over multiple decades or even centuries. Due to the computational cost of running a 3-dimensional OGCM, these applications are far from having been exhaustively run.

The existing OGCMs do not properly resolve synoptic disturbances (or weather) commonly known as meso-scale eddies in the ocean. Much of the ocean energy is concentrated at a small physical length scale known as the radius of deformation, which varies from about 1 degree (on the order of 100 km) near the equator to 1/10 or even 1/20 degree at high latitudes. This is the scale of intense boundary currents as well as transient eddies, and these phenomena are of considerable importance to the larger scale dynamics. It was not until recent years that eddy-permitting (or eddy-resolving) calculations could be carried out on a basin or global scale. Using the vector supercomputers (e.g., Cray Y-MP) at National Center for Atmospheric Research (NCAR), decade-long ocean model integrations have been carried out at 1/4 degree horizontal resolution [1], which was the first OGCM with performance exceeding 1 billion floating-point-operation-per-second (1 GFLOP/s). With the advance of massively parallel computing technology, decade-long integrations at 1/6 degree resolution have been conducted at Los Alamos National Laboratory (LANL) [2] and Jet Propulsion Laboratory (JPL) [3] on the CM-5 and Cray T3D, respectively. Recently, a short two-year integration at 1/12 degree resolution was made on the T3D at the Pittsburgh Supercomputer Center [4]. Despite the recent progress in eddy-resolving ocean modeling, it is not clear whether 1/6 or even 1/12 degree resolution is sufficient to resolve the ocean eddies and their impact on the large-scale circulation. Some run time estimates of global OGCMS on various advanced computers are given on Table 1.

In additional to running OGCMs at ever increasing resolutions, OGCMs also need to be thoroughly tested against the available observations to establish their validity in describing the real world. This often involves conducting a large number of experiments with different model configurations, and seeking the "best fit" between the model and data. Be cause of the limited computing resources available to ocean modeling community, it is often a very challenging task to systematically test various combinations of different model parameters.

When studying the Earth’s climate, OGCMs have to be integrated over long period of time, on the order of 100s or even

1000s of years. This is mainly because of the long memory of the ocean system, which has a large impact on the other components of the Earth system, e.g., atmosphere, land and ice.

Platform/nodes	Sustained Speed (GFLOPS/s)	Grid Size (degree by degree by level)	CPU time (hours/simulated year)
Cray T90/32	15	1/4x1/4x40	10
TMC CM-5/1024	10	1/6x1/6x40	30
Cray T3D/1024	30	1/8x1/8x40	25

Table 1. Run time estimates of global OGCMS on various advanced computers.

Given the limited computing resources available to the ocean modeling community, it is therefore very important to develop an efficient community ocean model, which can be used to conduct the above described applications. It is precisely this objective that motivates the present study. Our goal is to select a commonly used OGCM and improve its computational performance, particularly on advanced parallel computers.

2. Model description

Based upon the above described objective, we chose the most widely used OGCM code as our base code. The OGCM is based on the Parallel Ocean Program (POP) developed at Los Alamos National Laboratory [2]. This ocean model evolved from the Bryan-Cox 3-dimensional primitive equations ocean model [5,6], developed at NOAA Geophysical Fluid Dynamics Laboratory (GFDL), and later known as the Semtner and Chervin model or the Modular Ocean Model (MOM) [7]. Currently, there are hundreds of users within the so-called Bryan-Cox ocean model family, making it the dominant OGCM code in the climate research community.

The OGCM solves the 3-dimensional primitive equations with the finite difference technique. The equations are separated into barotropic (the vertical mean) and baroclinic (departures from the vertical mean) components. The baroclinic component is 3-dimensional, and uses explicit leapfrog time stepping. It parallelizes very well on massively parallel computers. The barotropic component is 2-dimensional, and solved implicitly. It differs from the original Bryan-Cox formulation in that it removes the rigid-lid approximation and treats the sea surface height as a prognostic variable (i.e., free-surface). The free-surface model is superior to the rigid-lid model because it provides more accurate solution to the governing equations. More importantly, the free-surface model tremendously reduces the global communication otherwise required by the rigid-lid model. Building upon the original ocean model developed at LANL, the new JPL ocean model has significantly optimized the original code, and developed a user-friendly coupling interface with the atmospheric or biogeochemical models.

3. General optimization strategies

The original POP code was developed in FORTRAN 90 on the Los Alamos CM-2 Connection Machine [2]. During the first half of 1994, the code was ported to the T3D by Cray Research using SHMEM-based message passing. Since the code on the T3D was still time- consuming when large problems were encountered, improving the code performance was required. In order to significantly reduce wallclock time, the code was optimized using single PE optimization techniques [8] and other strategies. The remainder of this section discusses these strategies and the corresponding improvement in performance of the POP code on the T3D.

3.1 Memory Optimization and Arithmetic Pipelines

The T3D uses the DEC Alpha EV4 processor with a 151 MHz clock and is capable of 151 MFLOP/s of peak performance. The cache is 8 KB, direct mapped, and has 32 byte cache lines. There are several ways to achieve good performance on the T3D. One is through effective use of cache; another is through effective use of pipelined arithmetic. The POP code uses many two dimensional arrays, which can be inefficient when frequent stride-one addressing is encountered. One improvement is to change to explicit one dimensional addressing. For example, the two dimensional array KMU(IMT,JMT) can be replaced by KMU(IMT*JMT). This type of change can increase performance, both by simplifying index calculations and by making the code easier for the compiler to optimize.

Another useful strategy is to unroll loop statements. The DEC EV4 processor has segmented functional units for floating point multiply and addition. Although a multiply or addition can be issued every clock period, the result is not ready for 6 clock periods. (The divide operation needs 61 clock periods as it is not a pipelined function.) Thus, in order to get top performance from FORTRAN code, the user must expose functional unit parallelism to the compiler. Because of these features of the T3D system, the POP code has been optimized by unrolling loop statements if this will expose functional unit parallelism. The number of divide operations also has been minimized by using real variables to store the values resulting from divide operation and moving the divide operation out of loop statements when it is independent of the loop index. After these techniques were applied, the code performed with a significant increase of the total MFLOP/s, as discussed in section 4.

3.2 Eliminating IF and WHERE statements by using mask arrays

In the original POP code, many logical IF and WHERE statements were used in distinguishing ocean points from land points. These statements consumed substantial CPU time and reduced pipelining of operations inside loops. The compiler often was not able to efficiently optimize these loops (especially using automatic loop unrolling), since within the IF and WHERE statements some of the computations were quite complex. These IF and WHERE statements were replaced with land/ocean masking arrays, which store the values 1 for ocean points and 0 for land points, and then use multiplies of these values. For example, the statements:

```
WHERE (KMU.GE.1)
  VUF = SUF
  VVF = SVF
ELSE
  VUF = 0.0
  SUF = 0.0
ENDIF
```

were replaced by:

```
CALL SHAD(IMT*JMT,1.0,UMASK,1,SUF,1,0.0,VUF,1)
CALL SHAD(IMT*JMT,1.0,UMASK,1,SVF,1,0.0,VVF,1)
```

where UMASK, VUF, VVF, SUF, SVF, and KMT are all of size (IMT,JMT), and UMSK was defined after the geometry initialization by:

```
DO I=1,IMT*JMT
  IF (KMU(I,1).GE.1) THEN
    UMASK(I,1) = 1.0
  ELSE
    UMASK(I,1) = 0.0
  ENDIF
ENDDO
```

This can be done because KMU is strictly a function of input geometry, and it results in a speed-up of 3.2 times, for IMT = JMT = 100. We have also applied some mathematical formula to eliminate some IF/WHERE statements. For example, in the subroutine STEP the statements

```
if (mix.eq.0) then
   beta  = alpha
   c2dtt = c2*dtt
   c2dtu = c2*dtu
   c2dtp = c2*dtp
 else
   beta  = theta
   c2dtt = dtt
   c2dtu = dtu
   c2dtp = dtp
 endif
```

are replaced by

```
beta=alpha*(1-mix)+theta*mix
c2dtt=c2*dtt*(1-mix)+dtt*mix
c2dtu=c2*dtu*(1-mix)+dtu*mix
c2dtp =c2*dtp*(1-mix)+dtp*mix
```

which will improve the effective use of pipelined arithmetic.

3.3 Using optimized libraries

There are several optimized libraries are available on the T3D, such as the SHMEM and BLAS libraries. These libraries have been already optimized to give the best possible T3D performance if the user applies them properly. In the POP code, there are some global sums in the conjugate-gradient routine and in the energy diagnostics routine. Those sums can be performed by using SHMEM library rather using PVM calls. This substitution gives a better performance. There are also many matrices and vectors computations in the POP code which consume a fair portion of the total CPU time. We have replaced them by calling BLAS routines. For example,

```
DO I=1,IMAX
  X(I)=ALPHA*(Y(I)*Z(I))
ENDDO
```

was replaced with a call to the extended BLAS routine named SHAD:

```
CALL SHAD(IMAX,ALPHA,Y,1,Z,1,0.0,X,1)
```

This can improve performance by a factor of 5 to 10.

3.4 Equation of state example

As an example, the routine which computes the equation of state for ocean water was optimized by eliminating the IF/WHERE statements, replacing the FORTRAN 90 array syntax with explicit loop structures, converting nested double-loops and triple-loops to single-loops, and performing explicit loop unrolling. Some mathematical formulae were also applied to eliminate some unneeded work. The overall performance improvement in this routine was 450%.

3.5 Portability

Given the short life cycle of the massively parallel computer, usually on the order of three to five years, we want to emphasize the portability of the ocean model and the associated optimization routines across several computing platforms. Thus far, the JPL ocean modeling effort has mainly been conducted on the T3D. We are now in the process of converting the ocean model from T3D to the T3E. We have also started porting the ocean model to the newly available Hewlett Packard (HP)/Convex SPP-2000, using an MPI version of the POP code. Preliminary results (Table 2) show that our ocean model running on the SPP-2000 is about 3 to 4 times faster than on the T3D. As the memory of the SPP-2000 is 4 times larger than that of the T3D, this implies one could also run a problem 4 times larger on the SPP-2000 in the same amount of time.

Number of CPUs	SPP-2000 (grid size per CPU)	T3D (grid size per CPU)
16	1.09(64x32x20)	3.20((64x32x20)
64	3.19(128x64x20)	3.34(64x32x20)
256		3.73(64x32x20)

Table 2. Wallclock time (sec) per time step for T3D, SPP-2000 (MPI codes).

4. Results and discussion

As discussed in the section above, the POP code was ported to the T3D in 1994. The code had previously been rewritten as a message passing code (PVM), and this is the version that was initially ported to the T3D. Those porting this T3D version did not optimize the code beyond trying to choose the fastest code to do identical work as had been done in the previous version. As discussed in the section above, new algorithms and methods to do similar work at a faster rate have now been implemented. The remainder of this section discusses the overall results of this work, and compares them with the initial T3D version of POP.

A test problem was chosen with a local grid size of 37 x 34 x 60 cells. Timings were run for machine sizes from 1 to 256 processors, corresponding to a global grid of up to 592 x 544 x 60. The POP code decompose the grid in blocks in both x and y, and all z data for a given (x,y) is local to one processor. All results shown in this section refer to scaled size problems, where the problem size per processor is fixed and the number of processors is varied.

4.1 Scaling Performance

Figure 1 shows the run time (in wall clock time) per time step vs. the number of processors, for both the original code and the optimized code. The code running on one processor has been improved by 43%, and as the number of processors involved in the calculation increases, so does the improvement due to the optimization, up to 59% on 256 processors.

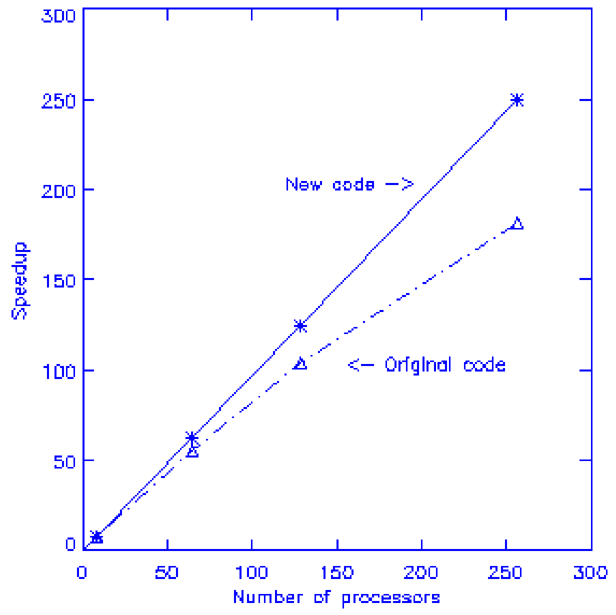

Figure 1. Speed-Up (vs. single processor time) for scaled problems.

4.2 Speed-Up

It is clear from Figure 1 that the code's scaling has also been improved. This improvement is enumerated in Figure 2, showing the speed-up achieved versus the number of processors used in the calculation. For the ideal parallel code, these two numbers would be equal. The optimized code is performing quite well, in running 250 times faster than the single processor code on 256 processors. The original code, however, clearly had scaling problems, as its speed-up on 256 processors is only 182 times.

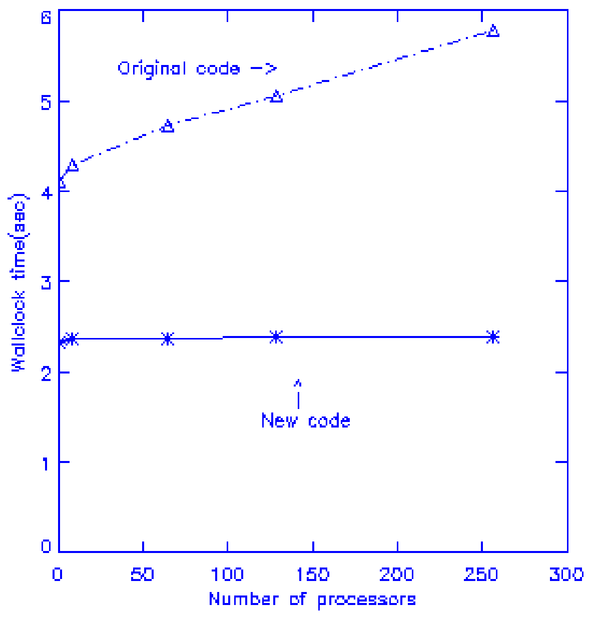

Figure 2. Wallclock time(sec) per time step for scaled problems.

4.3 Flop Rate

Figure 3 shows the actual performance achieved in terms of computation rate. As mentioned above, the T3D processor has a clock period of 6.6 nanoseconds, corresponding to a clock rate of 151.5 MHz. Since the processor can complete one floating point results per clock period, this is equivalent to a maximum floating point performance of 151.5 MFLOP/s. For 256 processors, this maximum performance is 38.8 GFLOP/s.

It can be observed that the new code is attaining only 10% of the maximum possible performance of the various machine sizes examined. There are three reasons why the code performs at this level. One reason is the ratio of computation to communication in these examples. If a larger local grid size was used, this ratio would increase and the overall performance would also increase. Another reason is poor cache reuse, due to the formulation of the code. It is written in terms of vector-vector routines (replaced by BLAS 1 [9] routines), rather than matrix-vector or matrix-matrix routines (which could be replaced by BLAS 2 [10] or BLAS 3 [11] routines) and uses finite difference routines. Neither of these make enough use of the data (each time it is loaded from memory) to achieve very high performance. (Note that while the data structures being used in the finite-difference routines could be changed to achieve better performance than is now being obtained, this still would be a small fraction of peak performance.) The final reason is that a fairly large number of non-floating point calculations being performed is approximately 13% larger for the optimized code that the original code. This is due to the replacement of IF/WHERE statements with floating point work, as discussed in section 3.2. The amount of work being done in the code is the same, but some comparison and branch instructions have been replaced by floating point instructions. This is a good example of why floating point performance is a relatively poor method for calculation performance, but as it is considered an important parameter for comparison of unrelated codes, it is included in this paper.

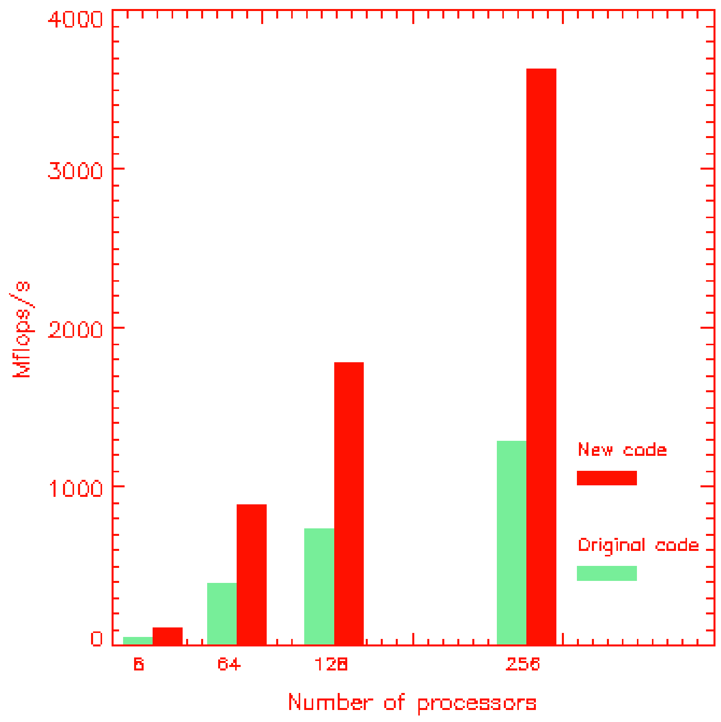

Figure 3. Total floating point performance rates (MFLOP/s) for scaled problems.

4.4 Overall Simulation Results

The test problem size used in the results given above corresponds to either a 1/3 degree global ocean model with 60 depth layers on 512 processors, or a 1/3 degree global model with 30 depth layers on 256 processors. Using the latter model, using the CPU time given as 2.4 sec/ time step, and simulating the circulation of the global ocean for 100 years (or 2.63 million time steps, with each time step being 20 minutes,) would required almost 1800 CPU hours, or about 75 CPU days on the 256 processor T3D. It is possible for this project to use 1.4 CPU days per week of the T3D located at the Jet Propulsion Laboratory, so this large run would take about 53 weeks to complete. This may be contrasted with a similar simulation using the original code, which would take 181 CPU days, or 130 weeks. For large problems such as this, a run which takes 1 year is possible, but 2.5 years is not. On the HP SPP-2000 which has recently been installed at JPL, the run times is forecast to be about 3 months (once all 256 processors are installed.) The T3E installed at NASA Goddard Space Flight Center should be able to run this problem on 384 processors using about 12 CPU days, though it is currently unclear how much time this would take, since this project doesn't have as much access to that machine.

The test problem may also be used to calculate run times for smaller problems, such as a model of the North Atlantic ocean. A model with 1/6 degree resolution, corresponding to a grid size of 640 x 624 x 45 on 256 processors could be run using approximately the same time. Running this model to simulate 1 year (at 10 minutes/time step) would require 36 CPU hours. This can be run on the JPL T3D in just over one week. A series of runs can be done to examine how changes in initial conditions will effect the simulation. A simulation of this size may also be compared with measured data, through a repeated process involving variation of physical parameters, to determine correct values for these parameters. The original code would take about 2 1/2 weeks, making these types of analysis much more difficult to perform. Again, on the 256 processor SPP-2000, three to four runs of this size could be done per week, with the optimized code. And on the 384 processor T3E, each run would take about 6 CPU hours.

This 1/6 degree resolution North Atlantic Ocean model(Figure 4) has been run for 30 years of simulated time on the 256-processor T3D, forced with the climatological monthly air-sea fluxes. In comparison with the previous eddy-resolving ocean model simulations [1,2,12,13], this model shows improved Gulf Stream separation off the coast of Cape Hatteras [3]. As the horizontal resolution increases, increasingly fine scale features and the intensification of the currents are found with many of the larger scale features unchanged. It is quite promising that the physical processes responsible for both water mass and eddy formation can be reasonably simulated in models of this class.

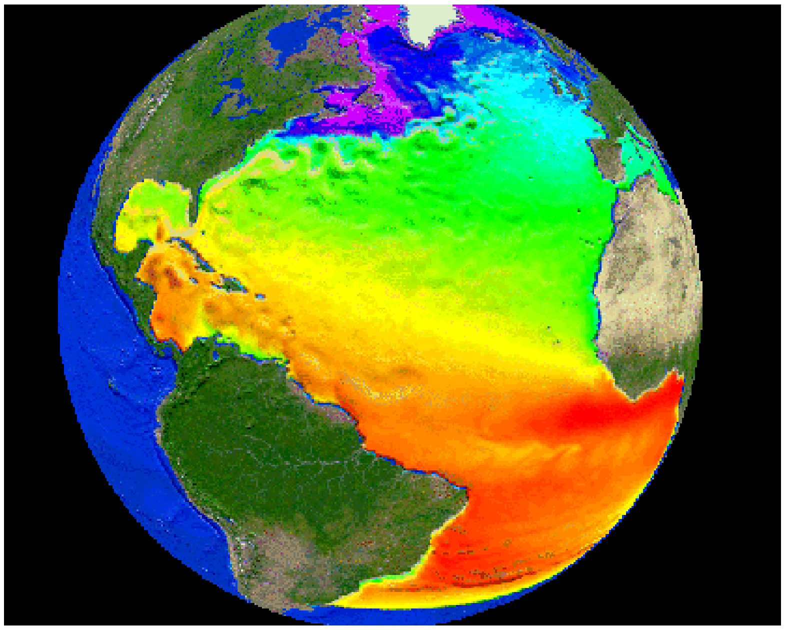

Figure 4. Sea Surface Temperature (in color) and Sea Surface Height Slope (in shade) simulated by a 1/6 degree North Atlantic Ocean model.

5. Conclusions

Optimization methods which were used on the POP code included unrolling loops, both manually and through the compiler, rewriting code to equivalence multi-dimensional arrays as one-dimensional arrays, simplification of algebra, reductions in the number of divide operations, use of mask arrays to trade fast floating point work for slow comparisons and branches, explicitly rewriting statements in array notation as loops, and using optimized libraries. All of these contributed to the large decrease in time required to solve ocean modeling problems.

We now anticipate being able to conduct follow-up eddy-resolving integrations at much higher resolutions. The 1/10 and 1/12 degree North Atlantic Ocean calculations have been carried out on the 512-processor CM-5 at LANL and 512-processor T3D at the MSC, respectively. It is quite feasible that JPL can construct a 1/16 degree calculations on the upcoming 384-processor T3E at Goddard Space Flight Center (GSFC) and 256-processor SPP-2000 at JPL. In additional to the OGCM integrations, we are coupling this OGCM with the atmospheric general circulation models developed at UCLA [14,15] on the T3D/E. A biogeochemical component within the OGCM is also being developed, so that the carbon cycle associated with the increasing CO_2 and global warming can be addressed.

6. Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration. The Cray Supercomputers used in to produce the results in this paper were provided with funding from the NASA offices of Mission to Planet Earth, Aeronautics, and Space Science. The authors wish to acknowledge Rick Smith, Phil Jones, and Bob Malone for providing the original POP code, David Shirley from Cray Research for doing much of the work in the initial port of this code to the T3D, Alan Stagg from the Army CEWES for doing initial timing studies, Hong Ding from NERSC for initial optimization work, and Martin Lewitt from HP for helping to implement the POP code on the SPP-2000.

References

1. A.J. Semtner and R.M. Chervin, **Ocean-General Circulation from a Global Eddy-Resolving Model**, *J. Geophys. Research Oceans*, 97, 5493-5550, 1992.

2. R.D. Smith, J.K. Dukowicz, and R.C. Malone, **Parallel Ocean General Circulation Modeling**, *Physica D*, 60, 38-61, 1992.

3. Y. Chao, A. Gangopadhyay, F.O. Bryan, W.R. Holland, **Modeling the Gulf Stream System: How Far From Reality?**, *Geophys. Res. Letts.*, 23, 3155-3158, 1996.

4. R. Bleck, S. Dean, M. O'Keefe, and A. Sawdey, **A Comparison of Data-Parallel and Message-Passing Versions of the Miami Isopycnic Coordinate Ocean Model (MICOM)**, *Parallel Computing*, 21, 1695-1720, 1995.

5. K. Bryan, **Numerical Method for the Study of the World Ocean Circulation**, *J. Comp. Phy.*, 4, 1687-1712, 1969.

6. M.D. Cox, **Primitive Equation, 3-Dimensional Model of the Ocean**, Group Tech. Report 1, GFDL/NOAA, Princeton, NJ, 1984.

7. R. Pacanowski, R.K. Dixon, and A. Rosati, **Modular Ocean Model User's Guide**, GFDL Ocean Group Tech. Report 2, GFDL/NOAA, Princeton, NJ, 1992.

8. J.P. Brooks, **Single PE Optimization Techniques for the CRAY T3D System**, Benchmarking Group, Cray Research, Inc., 1995.

9. C.L. Lawson, R.J. Hanson, D. Kincaid, and F.T. Krogh, **Basic Linear Algebra Subprograms for FORTRAN Usage**, *ACM Trans. Math. Soft.*, 5, pp. 308-322, 1979.

10. J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, **Extended Set of FORTRAN Basic Linear Algebra Subprograms**, *ACM Trans. Math. Soft.*, 14, pp. 1-17, 1988.

11. J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling, **A Set of Level 3 Basic Linear Algebra Subprograms**, *ACM Trans. Math. Soft.*, 16 pp. 1-17, 1990.

12. A.J. Semtner, **Modeling the Ocean General Circulation**, *Science*, 269, 1379-1383, 1995.

13. J.C. McWilliams, **Modeling the Ocean General Circulation**, *Annual Review of Fluid Mechanics*, 28, 215-248, 1996.

14. C.R. Mechoso, C.C. Ma, J.D. Ferrara, J.A. Spahr, and R.W. Moore, **Parallelization and Distribution of a Coupled Atmosphere-Ocean General Circulation Model,** *Monthly Weather Review*, 121, 2062-2076, 1993.

15. L.A. Drummond, J.D. Farrara, C.R. Mechoso, J.A. Spahr, Y. Chao, D.S. Katz, J.Z. Lou, and P. Wang, **Parallel Optimization of An Earth System Model (100 GIGAFLOPS and Beyond?)**, 1997 Society for Computer Simulation (SCS) International Conference, Atlanta, Georgia, April, 1997.

Author Biography

Ping Wang is a Computational Scientist in the Jet Propulsion laboratory, California Institute of Technology. Her research interests include large-scale scientific computations, parallel computations in fluid dynamics, parallel software design, and numerical methods (finite volume, finite difference, finite element, multigrid) for PDE. She received her PhD in Applied Mathematics from the City University, London, U.K., 1993.

Daniel S. Katz is a Computational Scientist at the Jet Propulsion Laboratory. His research interests include numerical methods applied to parallel computing, and computational methods in both electromagnetic wave propagation and geophysics. He received his PhD in Electrical Engineering from Northwestern University in 1994.

Yi Chao is a Research Scientist in the Earth and Space Science Division of JPL. He has a M.A. and Ph.D. degree in Atmospheric and Oceanic Sciences from Princeton University. Before joining JPL in 1993, he was a post-doctoral fellow in UCLA's Department of Atmospheric Sciences. His research interests are in ocean modeling, satellite remote-sensing of the ocean, air-sea interaction and high-performance computing.