Interdisciplinary Information Sciences Vol. 15, No. 1 (2009) 85–90

ISSN 1340-9050 print/1347-6157 online
DOI 10.4036/iis.2009.85

Improvements for High Resolution Ocean Research with NEMO

Fredrik UNGER[1*] and Dr. Arne BIASTOCH[2]

[1]*NEC Deutschland GmbH*
[2]*Leibniz-Institut für Meereswissenschaften (IFM-GEOMAR)*

Received February 29, 2008; final version accepted December 13, 2008

NEMO is a fluid dynamics code used for oceanographic research. Within the TERAFLOP Workbench in cooperation with the Leibniz-Institut für Meereswissenschaften (IFM-GEOMAR) in Kiel a performance assessment and improvement campaign was carried out, ranging from MPI to memory addressing in solvers. At The High Performance Computing Center Stuttgart (HLRS) tests were made using a large configuration of SX nodes running NEMO at 2.1 Teraflop/s. The improved code is running the test case 29% faster on 512 SX-8 CPUs.

KEYWORDS: NEMO, OPA, Ocean Research, Vector computers, NEC SX-8

1. Introduction

NEMO (formerly OPA) is a coupled ocean — sea-ice model developed at the L'Institut Pierre-Simon Laplace (IPSL) in Paris. It is widely used at major European oceanographic institutes. The Teraflop Workbench at The High Performance Computing Center Stuttgart (HLRS) works together with the Leibniz-Institut für Meereswissenschaften (IFM-GEOMAR) to improve the code used to study the Agulhas system. Figure 1 shows the Agulhas rings which transports warm water from the Indian Ocean to the Atlantic. They are only visible in high resolution simulation.

NEMO has a history of running on vector computers. To further enhance the speed on the NEC SX vector computers the source code has been investigated and improved in the HLRS Teraflop Workbench. NEMO is a structured code, splitting the world in different regions using MPI to enable high resolution research on large machines. The first target was to improve the non-nested high resolution grid. Secondly the new experimental version with grid nesting, using AGRIF, was tested and improved. AGRIF is an adaptive mesh refinement package developed by the Institut d'Informatique et de Mathématiques Appliquées de Grenoble, Laboratoire de Modélisation et Calcul (LMC-IMAG) in Grenoble.

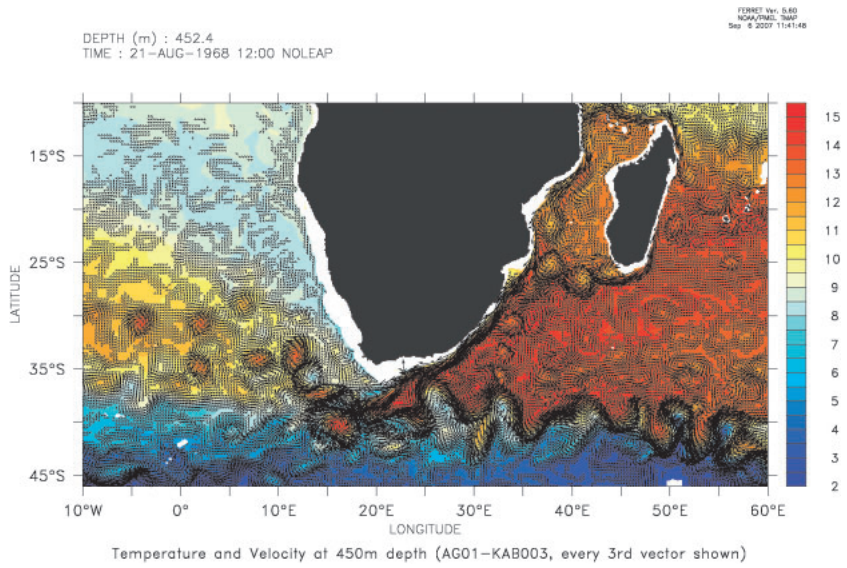

Fig. 1. High resolution image over the Agulhas region showing the Agulhas rings that earlier only had been observed by research ships but now also can be simulated.

* **Contact address:** Heßbrühlstraße 21b, 70565 Stuttgart, Germany.

For the non-nested high resolution grid the internal test case GYRE was used. GYRE is generated at runtime and can be tested in different resolutions and problems sizes without having to construct different test cases.

2. Areas of Improvement

As a first step, the compute intensive routines were investigated in a $1/16^{th}$ degree high resolution setting. Most of the routines work by calculating the three dimensional data using loops accessing the longitudinal data by stride one. The latitudes are treated as the second dimension and the third dimension depth is often in the outer most loop.

The routines are mostly written to use two-dimensional areas for calculation by using nested loops. The first investigation was to classify and find loops that were not properly collapsed, running over the longitude and latitude dimension in one loop. Also loops that for some other reason had poor vector performance were investigated.

As one example the routine calculating ice rheology, one part of the treatment of ice, was rewritten to use less temporary arrays, improving the ability to collapse loops and reducing memory access.

MPI tunings were made in the Teraflop workbench, allowing packaging several arrays into a larger buffer that then is sent using the SX-8 global memory feature. The code for the northern communication was simplified, earlier two different routines were used depending if the code was distributed with MPI or not. The codes could be combined to a subroutine that treats the northern area. Maintenance is easier and the possibility to inline was improved.

Memory optimizations like limiting stride two memory access and using vector registers were introduced. Simplifications of loops and its variables also helped achieving performance.

The tunings below were tested in small environment only using a few NEC SX-8 nodes. The SX ftrace functionality was used to see the effect of the tunings. The ftrace tool instruments the code automatically to measure performance on a subroutine level using the lowlevel performance counters in the SX CPU. It is also possible to instrument to code by hand to measure specific regions.

2.1 Loop Changes

Introduction of local variables simplified some of the complex array addressing. Array values in this structured code are accessed in a box pattern around the current value $X(i, j)$, $X(i + -1, j + -1)$. All combinations brings nine values that are read in different patterns from different arrays. By using local variables the compiler optimizes the access better and the loops run faster. Simplifications in the structure of the loops are also possible, sometimes duplicate parts of the code code are "hidden" in the array notations. When simplified loop indexes can more easily be proven to be the same and loops can be merged or rewritten. This does not only help the SX platform, but any compiler or computer.

This was done in several routines, and was most effective in a routine that calculates the sea ice rheology. This routine was not optimal to begin with due to that some loops did not collapse. At the end several loops could be combined thus no longer posing a performance problem for NEMO. The routine went a performance of 6.9 GFLOP/s per CPU to 10.3 GFLOP/s, reaching 64% efficiency of the CPU.

2.2 MPI Communication

NEMO communicates with MPI in a rather standard pattern for structured codes. Each MPI thread communicates its boarder values to its neighbors East–West, North–South. For the communication around the North Pole there are however some special treatments with different options how to communicate the values. The South Pole does not have the need for special communication in the boarder area, as the South Pole is a landmass. The original North Pole algorithm was implemented twice, one for SMP and one for MPI. By rearranging some buffer addressing one set of algorithms could be used to do the calculations.

All communication in NEMO is done by a link exchange routine taking one 2D or one 3D array as argument. By adding the ability to fill the communication buffers with data, the new version has the ability to send several arrays of any type with one MPI exchange, utilizing the high bandwidth network of the NEC SX-8.

The communication pattern was changed for the North Pole. Instead of doing a MPI_Gather to process 0, calculating and then doing a MPI_Scatter, a MPI_Allgather was made thus each process can continue after the completion of the calculation and there is no need to wait for a central process. Also a second East–West communication could be avoided with special treatment of the corners of the two dimensional communication. The difference in communication is visualized in Fig. 2. The improved version communicates less.

The MPI tunings were kept transparent to the original MPI communication thanks to that Fortran allows different types of arguments. The old MPI communication used a character to select method for the northern communication, and the new uses an integer. One can in the call to the MPI routine decide to use the integer constant or the character constant when picking old or new implementation. This is also good as it provides a framework to run tests on other platforms regarding performance and correctness. A special test environment was also built in that can be enabled with a preprocessor flag. It communicates an array with a predefined pattern with the old method and then with the new method. The results are then compared and if any error occurs it is reported and the program is halted. This procedure serves rather as a support for development than for production. Once the new method is stable on all platform this test code can be removed.

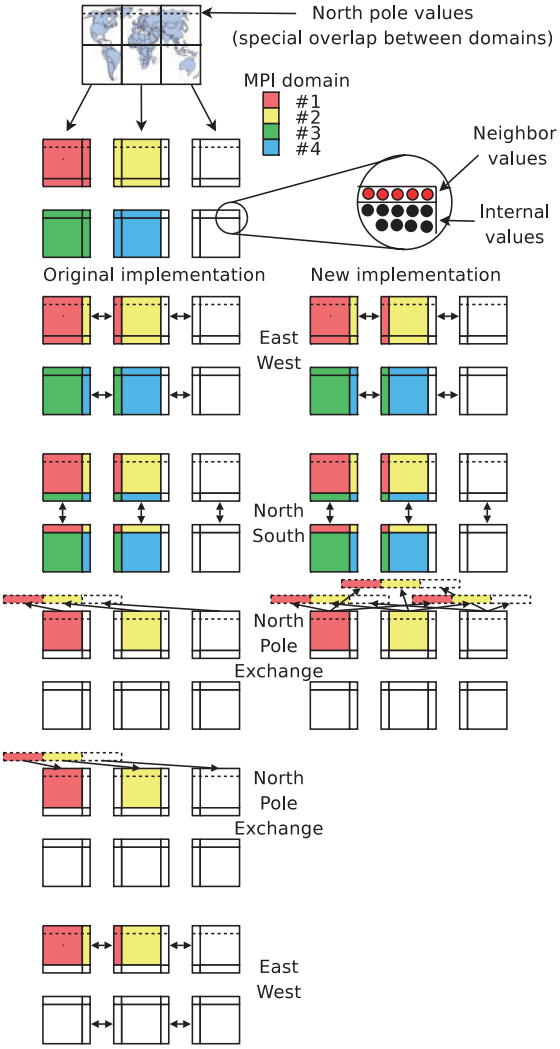

Fig. 2. The new communication contains less synchronization and less communication than the old version. The MPI_Allgather call only blocks the affected MPI Threads once and provides each thread with the data it needs to continue without doing an extra East–West communication. Except the North Pole communication it is an ordinary structured code boarder exchange pattern.

2.3 Solver Memory Access

In order to achieve the optimum memory bandwidth, it is essential to access the memory without any gaps. This is accomplished by a stride one in the FORTRAN do-loops. The SX-8 works best with a one stride access to memory. This is the most common access pattern reading arrays. On other CPUs there could be problems with cache trashing. The SOR solver in NEMO uses a stride two by definition. The algorithm accesses odd and even values in different loops. Analyzing the access pattern one can see that all arrays but gcx do not need to be accessed with stride two. The array gcx is communicated in each solver iteration; for this reason it can not be changed easily. To remove the stride two in the inner loop temporary arrays were introduced that are set up with the original arrays values in a short startup phase of the solver:

```
DO jj = 2, jpjm1
  ishift = MOD( jj-ijmppodd, 2 )
  DO ji = 2+ishift, jpim1, 2
    ztmp =              gcb(ji ,jj  ) &
      & - gcp(ji,jj,1) * gcx(ji ,jj-1) &
      & - gcp(ji,jj,2) * gcx(ji-1,jj ) &
      & - gcp(ji,jj,3) * gcx(ji+1,jj ) &
      & - gcp(ji,jj,4) * gcx(ji ,jj+1)
      ! Estimate of the residual
    zres = ztmp - gcx(ji,jj)
    gcr(ji,jj) = zres * gcdmat(ji,jj) * zres
      ! Guess update
    gcx(ji,jj) = sor * ztmp + (1-sor) * gcx(ji,jj)
```

```
  END DO
END DO
```

Rewriting the code using one stride for the temporary arrays changes the inner loop to have a stride one. It introduces some more index calculation, but they are minor and also used by many arrays. The benefit for the memory access is more important in the top routine:

```
DO jj = 2, jpjm1
  ishift = MOD( jj-ijmppodd, 2 )
  DO ji = 1, (jpim1 - ishift) / 2
    ji2 = ji*2+ishift
    ji1 = ji + 1
    ztmp =               z1gcb(ji1 ,jj  ) &
      & - z1gcp(ji1,jj,1) * gcx(ji2 ,jj-1) &
      & - z1gcp(ji1,jj,2) * gcx(ji2-1,jj ) &
      & - z1gcp(ji1,jj,3) * gcx(ji2+1,jj ) &
      & - z1gcp(ji1,jj,4) * gcx(ji2 ,jj+1)
    ! Estimate of the residual
    zres = ztmp - gcx(ji2,jj)
    z1gcr(ji1,jj) = zres * z1gcdmat(ji1,jj) * zres
    ! Guess update
    gcx(ji2,jj) = sor * ztmp + (1-sor) * gcx(ji2,jj)
  END DO
END DO
```

The gain of this tuning is not so large, but the loops are called many times per iteration making small improvements play a big role. Just this small memory adjustment gained 6.7% in time for the two loops.

2.4 Vector Registers

Some loops that access the same array in this $+-1$ pattern were also rewritten using the VREG directive, placing some variables in vector registers. The loops were strip-mined to the vector length. Below is an example how and ordinary loop with many accesses to the arrays `tb` and `sb` can be rewritten with VREG variables, improving memory access. With the VREG directive the compiler gives more control to the programmer but also more responsibility for correctness:

```
DO jk = 1, jpkm1
  DO jj = 1, jpjm1
    DO ji = 1, fs_jpim1 ! vector opt.
      zt1(ji,jj,jk)=umask(ji,jj,jk)*(tb(ji+1,jj,jk)-tb(ji,jj,jk))
      zs1(ji,jj,jk)=umask(ji,jj,jk)*(sb(ji+1,jj,jk)-sb(ji,jj,jk))
      zt2(ji,jj,jk)=vmask(ji,jj,jk)*(tb(ji,jj+1,jk)-tb(ji,jj,jk))
      zs2(ji,jj,jk)=vmask(ji,jj,jk)*(sb(ji,jj+1,jk)-sb(ji,jj,jk))
    END DO
  END DO
END DO
```

This was rewritten using the vreg variables, defined with the VREG directive. One has to take care that the inner loop checks for the special end vector. Using the SHORTLOOP directive makes the compiler translate the loop to just vector instructions, removing the loop:

```
DO ji_ = 1, fs_jpim1, 256
  DO jk = 1, jpkm1
!CDIR SHORTLOOP
    DO ji = ji_, min(ji_+255,fs_jpim1)
      vr1(ji+1-ji_)=tb(ji,1,jk)
      vr2(ji+1-ji_)=sb(ji,1,jk)
    END DO
    DO jj = 1, jpjm1
!CDIR SHORTLOOP
      DO ji = ji_,min(ji_+255,fs_jpim1) !1, fs_jpim1
        zt1(ji,jj,jk)=umask(ji,jj,jk)*(tb(ji+1,jj,jk)-vr1(ji+1-ji_))
        zs1(ji,jj,jk)=umask(ji,jj,jk)*(sb(ji+1,jj,jk)-vr2(ji+1-ji_))
        zt2(ji,jj,jk)=vmask(ji,jj,jk)*(tb(ji,jj+1,jk)-vr1(ji+1-ji_))
        zs2(ji,jj,jk)=vmask(ji,jj,jk)*(sb(ji,jj+1,jk)-vr2(ji+1-ji_))
        vr1(ji+1-ji_)=tb(ji,jj+1,jk)
        vr2(ji+1-ji_)=sb(ji,jj+1,jk)
```

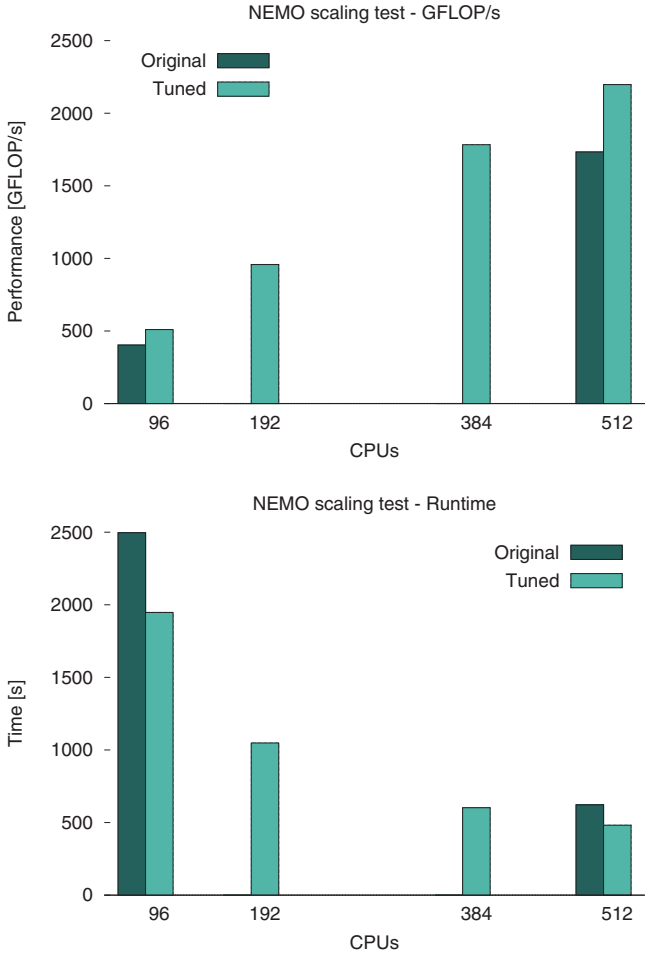

Fig. 3. Scaling results made with NEMO (OPA9) $1/16^{th}$ degree model.

```
      END DO
    END DO
  END DO
END DO
```

A setup loop for the first jj values is needed, and then the combinations of loads for $jj+1$ can be made by the compiler and stored directly in vector registers for the next iteration. This use of vector registers works with a $+-1$ pattern in the second (jj) dimension.

The performance gains are very different per loop and depending on what other loop transformations were made (loop merging, etc.).

3. Results Using High Resolution Model

Measurements were made with the $1/16^{th}$ degree model of the internal test case GYRE. Generated by the application itself, with the dimension $6002 \times 4002 \times 31$. NEMO reached 2.1 TFLOP/s on 64 nodes SX-8.

Time improvements were 29% on 64 nodes SX-8, running the GYRE test case for 1500 iterations. Plots of the scaling in time and FLOP/s can be found in Fig. 3.

4. AGRIF

AGRIF is an adaptive mesh refinement package that is being developed at LMC-IMAG in Grenoble. http://www-lmc.imag.fr/MOISE/AGRIF/ It has been integrated into NEMO and tests were made using AGRIF. Figure 4 shows the setup for the Agulhas region using a $1/10^{th}$ degree model on a $1/2^{th}$ world model. AGRIF is a package that allows a code to be easily adapted to use multiple grid levels. It does so by using an elaborate preprocessor that changes the original code and moves static arrays to a dynamic grid dependant structure.

Initially it had performance problems in the interpolation between the grids due to its object oriented programming model. The data was broken down in small arrays and individually treated creating a lot of subroutine call overhead.

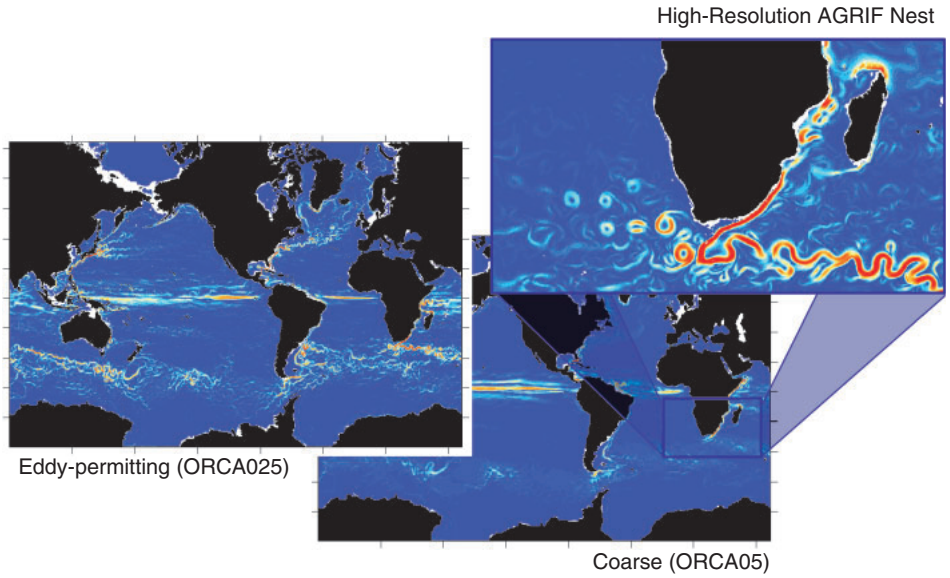

Fig. 4. Example picture of AGRIF multi grid setup over the Agulhas region. The $1/10^{th}$ degree Agulhas model nested into the $1/2^{th}$ degree global model.

Also the final algorithms only worked with small amounts of data per object, not properly utilizing the SX-8 vector capabilities.

At HLRS the MPI communication in AGRIF was investigated. At least for the case using fixed grids all communication of grid limits could be removed. For a short test case using 60 cycles for the coarse and 240 for the fine grid the number of MPI calls went down to 10201 from 145339, i.e. an improvement with the factor of 14! All of these calls were of a collective nature, so the effect should be even more visible if more than the 16 CPUs of the test setup are used. This was implemented as test code deep down in the MPI call structure, by investigating the repetitive pattern of the MPI communication content. Further work is needed to implement that in a higher level of the library, taking advantage of information at that level. The changes have been forwarded to the developers of AGRIF, and are being investigated.

5. Conclusions

Applications for climate research performs well on vector systems, and shows that real applications can achieve TFLOP/s performance on the vector supercomputers. NEMO performs over with over 2 TFLOP/s on the HLRS SX-8 system.

Efforts in minimizing MPI communication, reducing loop complexity and improving memory access do make applications faster thus enabling improved research. Profiling is a powerful tool to find important loops and routines.

The experimental version using AGRIF has shown one approach to do multi-grid calculation using a single grid application as an starting point. The AGRIF was evaluated on the SX-8 platform and some performance problems were recognized. They have been solved and are being implemented into the future version of NEMO.