The NEMO Ocean Modelling Code: A Case Study

Fiona J. L. Reid[1]
[1]*EPCC, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK*

Abstract

We present a case study of a popular ocean modelling code, NEMO, on the Cray XT4 HECToR system. HECToR is the UK's high-end computing resource for academic users. Two different versions of NEMO have been investigated. The performance and scaling of the code has been evaluated and optimised by investigating the choice of grid dimensions, by examining the use of land versus ocean grid cells and also by checking for memory bandwidth problems. Profiling the code identified the time spent carrying out file input/output to be a potential bottleneck. We present a solution to this problem which gives a significant saving in terms of runtime and disk space usage. We also present the results of investigations into the performance of nested models resulting in an optimal processor count being obtained.

1 Introduction

HECToR is the UK's new high-end computing resource available for research funded by the UK Research Councils. This paper presents a case study of the popular ocean modelling code, NEMO [1], on the HECToR system. The main aims of the study were to investigate and where possible to improve the I/O performance and nested model performance of NEMO.

The paper begins with an introduction to the NEMO code along with some motivation for investigating file I/O and nested model performance. Section 3 describes the architecture of the HECToR system. Section 4 presents the baseline performance of the NEMO code and the results of a number of investigations which resulted in improved performance. In section 4.7 the effects of I/O are investigated and a method for improving the I/O performance is presented in section 5. Nested model performance is described in section 6.

2 What is NEMO?

NEMO (Nucleus for European Modelling of the Ocean) is a modelling framework for oceanographic research and operational oceanography. The framework allows several ocean related components e.g. sea-ice, biochemistry, ocean dynamics, tracers etc to work either together or separately. Further information on NEMO and its varied capabilities can be found at, [1].

The NEMO framework currently consists of three components each of which (except for sea-ice) can be run in stand-alone mode. The three components are:

- OPA9 - New version of the OPA ocean model, written in Fortran 90
- LIM2 - Louvain-la-Neuve sea-ice model, also Fortran 90
- TOP1 - Transport component based on the OPA9 advection-diffusion equation (TRP) and a biogeochemistry model which includes the two components LOBSTER and PISCES.

This paper focuses primarily on the OPA9 component and uses a version of the NEMO code which has been modified by the National Oceanography Centre, Southampton (NOCS) researchers. The modified version is essentially the release version of the code with some specific enhancements particular to their science application. Two different versions of NEMO are discussed in this paper, version 2.3 and version 3.0. Our initial investigations were carried out using version 2.3 with version 3.0 being used as soon as it became available.

The code is written in predominantly in Fortran 90 (some Fortran 77 code remains) and has been parallelised using the Message Passing Interface, MPI.

The OPA9 component of the code uses a domain decomposition approach where each processor works on a small part of the ocean grid.

2.1 Motivation for this study

Two main aspects of the code were identified by the research group as potential performance bottlenecks, these were, the I/O performance at larger processor counts and the performance of nested models. The motivation for investigating these two areas is discussed below.

2.1.1 I/O performance

The way in which data is currently input/output to NEMO is not considered ideal for large numbers of processors. Each processor inputs/outputs its own section of the ocean grid to separate files. As researchers move to use increasingly more complex models at higher spatial resolutions larger numbers of processors (and thus files) will be required and this potential I/O bottleneck will therefore need to be addressed. This paper investigates the current scaling and I/O performance of NEMO and identifies methods to improve this via the use of lossless compression algorithms.

2.1.2 Nested model performance

The NEMO code allows nested models to be used which enable different parts of the ocean to be modelled with different resolution within the same global model. E.g. an area of the Pacific Ocean could be model at 1 degree resolution with the remainder of the Earth's Oceans being modelled at 2 degree resolution. This type of modelling can help scientists to gain a better insight into particular ocean features whilst keeping the computational costs reasonable. In the past, setting up such models has been very time consuming and relatively few attempts have been made to run such configuration on high performance computing systems. As such this paper investigates the performance of nested models and attempts to improve their performance, with the main goal being to achieve a stable and optimised nested model with known scalability.

3 Architecture

The HECToR Cray XT5h system began user service in October 2007 and consists of a scalar (XT4) and a vector (X2) component. All the results presented in this paper were obtained on phase 1 of the scalar component. The current configuration of HECToR (phase 2a and phase 2b) is described in [2].

The (phase 1) system comprises of 5664 compute nodes, each with one dual core AMD Opteron 2.8GHz chip, i.e. a total of 11,328 cores. Each core has access to a private 64Kbyte L1 instruction cache, 64Kbyte L1 data cache and 1 Mbyte L2 cache. The two cores within a chip (or node) share 6 Gbytes of main memory. Each node controls a Cray Seastar2 network chip. Each Seastar2 has six links and the network is configured in a 3D toroidal topology.

In addition to the compute nodes there are also dedicated I/O nodes, login nodes and nodes set aside for serial compute jobs. The login nodes can be used for editing, compilation, profiling, de-bugging, job submission etc. When a user connects to HECToR via `ssh` the least loaded login node is selected to ensure that users are evenly loaded across the system and that no single login node ends up with an excessive load.

HECToR (phase 1) has 12 I/O nodes which are directly connected to the toroidal communications network described above. These I/O nodes are also connected to the data storage system (i.e. physical disks). The data storage on HECToR consists of 576 TB of high performance RAID disks. The service uses the Lustre distributed parallel file system to access these disks.

4 NEMO baseline performance

This section presents the results of an investigation into the performance of NEMO considering the compiler choice, optimisation flags and program parameters. To assess the performance a 0.25 degree ocean model is used. This model is chosen to be the same resolution that researchers require for their science but is run over far fewer time steps.

When running the NEMO code it is necessary to re-compile the code for different processor counts as the number of processors and model grid dimensions are set as parameters within the code. The variables `jpni`, `jpnj`, and `jpnij` specify respectively the number of processors in the i direction, the number of processors in the j direction and the total number of processors. E.g. a 16 by 16 processor grid which runs on 256 processors would have `jpni = 16`, `jpnj = 16` and `jpnij = 256`.

4.1 Scaling for equal grid dimensions

We begin by investigating the scaling of the code for grids of equal dimension, i.e where `jpni = jpnj`, for the PGI and PathScale compilers. This restricts us to a relatively small number of processor counts ranging from 64 (8x8) to 1024 (32x32). The results are shown in figure 1.

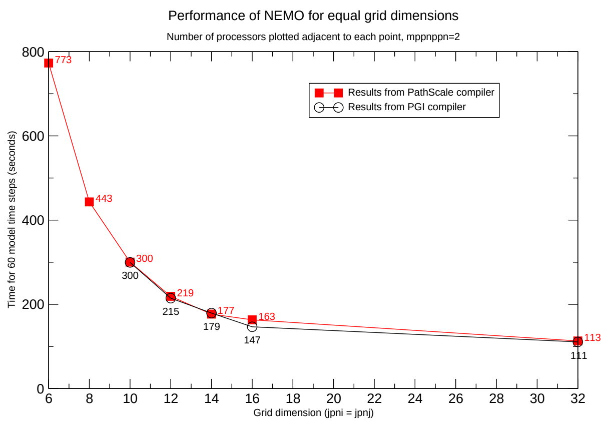

Figure 1: Performance of NEMO when $jpni = jpnj$ for the PGI and PathScale compiler suites.

From figure 1 it is clear that the PGI compiler performs slightly better (a few percent) than the PathScale compiler and thus our remaining experiments will be carried out using the PGI compiler. NEMO scales to 1024 processors but the benefit in using more processors is purely a reduction in runtime and not efficient in terms of the allocation units [1] (AU's) used. Using 128 or 256 processors seems to give the best compromise between AU use and runtime.

4.2 Compiler optimisations

In this section we investigate whether any compiler optimisations can be used to improve the performance of NEMO. Table 1 summarises the results of running a 16 by 16 model grid on 221 processors for a variety of different compiler optimisations.

[1]Calculations run on HECToR use a standard Allocation Unit of CPU time. The AU can be thought of as a unit of computational work, equivalent to a 1 Gflop/s processor running for 1 hour, as assessed by the Linpack benchmark (Rmax). Thus, a 60 Tflop/s computer provides 60,000 AUs per hour.

Compiler flags	Time for 60 steps (s)
`-O0`	163.520
`-O1`	157.123
`-O2`	138.382
`-O3`	139.466
`-O4`	137.642
`-fast`	seg fault
`-O2 -Munroll=c:1 -Mnoframe -Mautoinline -Mscalarsse -Mcache_align -Mflushz -Mlre`	seg fault
`-O2 -Munroll=c:1 -Mnoframe -Mautoinline -Mscalarsse -Mcache_align -Mflushz -Mlre -Mvect=sse`	seg fault
`-O2 -Munroll=c:1 -Mnoframe -Mautoinline -Mscalarsse -Mcache_align -Mflushz`	133.761
`-O4 -Munroll=c:1 -Mnoframe -Mautoinline -Mscalarsse -Mcache_align -Mflushz`	138.965

Table 1: Runtime for 60 time steps for different compiler flags for the PGI compiler suite. Version 7.1.4 of the PGI compiler is used throughout. All tests were run with `jpni=16`, `jpnj=16` and `jpnij=221`.

Increasing the level of optimisation from `-O0` to `-O2` gives an increase in performance. Optimisation levels from `-O2` up to `-O4` gives minimal improvement. The `-fast` flag results in the code failing with a segmentation violation. As this flag invokes a number of different optimisations [2] we tested each of these in turn to ascertain which particular flag(s) cause the problem.

From Table 1 we see that the addition of the flags `-Mlre` or `-Mvect=sse` cause the code to crash at runtime. All other flags invoked by `-fast` do appear to not cause significant issues. The `-Mlre` causes the zonal velocity to become very large suggesting that the loop redundancy elimination may have removed a loop temporary variable that was actually required. The reason for the failure when `-Mvect=sse` is added is unknown. Ultimately, the

[2] The command `pgf90 -help -fast` lists the optimisations invoked by `-fast`, e.g.

```
fionanem@nid15879: > pgf90 -help -fast Reading rcfile
/opt/pgi/7.1.4/linux86-64/7.1-4/bin/.pgf90rc
-fast Common optimizations; includes -O2 -Munroll=c:1
-Mnoframe -Mlre -Mautoinline == -Mvect=sse
-Mscalarsse -Mcache_align -Mflushz
```

inclusion of these extra flags which `-fast` uses dc
not give significant performance improvements a
thus `-O3` will be used in future.

4.3 Single core versus dual core pe formance

To investigate whether NEMO suffers from memc
bandwidth problems on HECToR we compare ru
ning the code in single node (one core per node use
versus running in virtual node mode (both cores p
node used). Table 2 shows the runtimes for 256 a
221 processors using a 16 by 16 grid for both mod

`jpnij`	`jpni`	`jpnj`	**Time for 60 steps (seconds**	
			mppnppn=1	mppnppn=:
256	16	16	119.353	146.60'
221	16	16	112.542	136.180

Table 2: Runtime comparison for 60 time steps for single node (mppnppn=1) and virtual node (mppnppn=2). Runs were performed using the PGI compiler

Examining table 2 we can see that single node mode is up to 18.59% faster than virtual node mode suggesting that NEMO suffers from relatively mild memory bandwidth problems. As the charging structure on HECToR is per core, single node mode will cost significantly more (almost double) AU's than virtual node mode. Thus, running NEMO in single node mode should only be considered if it's critical to obtain a timely solution.

4.4 Performance for different grid dimensions

Using a fixed number of processors we investigate how the shape of the grid affects the performance. We concentrate on 128 and 256 processors with two results from a 512 processor run. All runs are carried out using the PGI compiler suite. The results of this experiment are shown in figure 2.

Figure 2 suggests that for a fixed number of processors the ideal grid dimensions are square i.e. where `jpni=jpnj`. If the number of processors is such that it is not possible to have `jpni=jpnj` (i.e. the number of processors is not a square of an integer) then the results suggest that the values of `jpni` and `jpnj` should be as close to each other as possible with the value of `jpni` chosen such that `jpni < jpnj`.

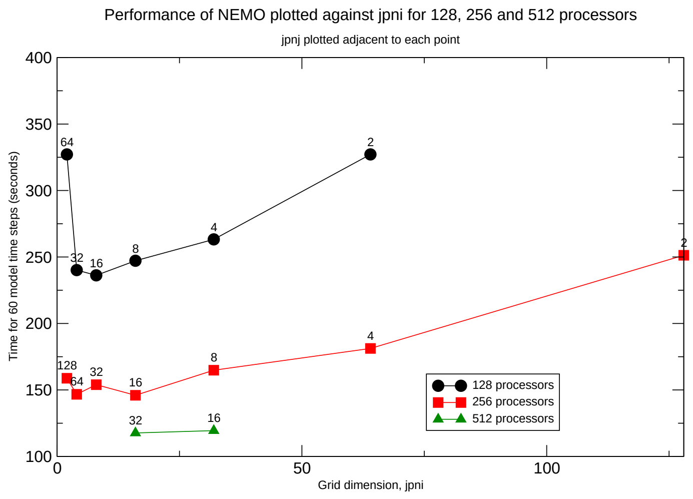

Figure 2: Performance of NEMO on 128, 256 and 512 processors plotted against the number of grid cells in the i direction, $jpni$

4.5 Removing the land only grid cells

So far we have considered decompositions in which all the grid cells are used, i.e. those where the code has `jpnij = jpni x jpnj`. However, many decompositions give rise to grid cells which contain only land. These land only cells are essentially redundant in an ocean model and can be removed. In the code this means that the value of `jpnij` can be reduced such that `jpnij` $\leq$ `jpni x jpnj`. It is anticipated that removing land only cells may improve the performance of the code as branches into land only regions will no longer take place and any I/O associated with the land cells will also be removed. Furthermore, the number of AU's required will be reduced as fewer processors will be required if the land cells are removed.

The NEMO code does not automatically remove the land cells which means the user needs to select their decomposition and then independently determine how many cells contain only land. A tool written by Dr Andrew Coward at the NOCS can be used to determine the number of active (ocean containing) and dead (land only) cells. For example when using a 16 by 16 grid, there are 35 land only cells and thus `jpnij` can be set to 221 instead of 256.

Table 3 gives the number of land only cells for a variety of grid configurations. The reduction in the number of processors required is generally around 10%. For large (>256) processor counts the reduction can be considerably larger and as much as 25%.

jpni	jpnj	Total cells	Land only cells	Percentage saved
9	9	81	6	7.41%
10	10	100	10	10.00%
11	11	121	13	10.74%
12	12	144	14	9.72%
13	13	169	21	12.43%
14	14	196	22	11.22%
15	15	225	29	12.89%
16	16	256	35	13.67%
20	20	400	65	16.25%
30	30	900	193	21.44%
32	32	1024	230	22.46%
40	40	1600	398	24.88%
16	8	128	117	8.59%
32	16	512	92	17.97%

Table 3: Number of land only squares for a variety of processor grids. The percentage saved gives the percentage of cells saved by removing the land only cells and will correspond to the reduction in the number of AU's required for the computation.

We now investigate whether removing the land only cells has any impact on the runtime of the NEMO code. We hope that by avoiding branches into land only regions and the associated I/O involved with the land cells that the runtime should reduce. For this test we have considered only 128, 256, 512 and 1024 processor grids. The results are given by table 4.

jpni	jpnj	jpnij	Time for 60 steps (seconds)
32	32	1024	110.795
32	32	794	100.011
16	32	512	117.642
16	32	420	111.282
16	16	256	146.607
16	16	221	136.180
8	16	128	236.182
8	16	117	240.951

Table 4: Runtime comparison for 60 time steps for models with/without land squares included on 128, 256, 512 and 1024 processor grids.

From table 4 we can see that for 256 processors and above removing the number of land squares reduces the total runtime by up to 10 seconds which corresponds to a reduction of around 7-10%. For the 128 processor run, removal of the land-only cells actually gives a small increase in the total runtime. This difference is within normal repeatability errors and could be a result of heavy load on the system when the test was run. As the runtime does not seem to improve greatly with the removal of the land only cells the main motivation for removing these cells is to reduce the number of AU's used for each calculation. Assuming the runtime is not affected detrimentally then the reduction in AU usage will be as given by table 3.

4.6 Optimal processor count

Ideally, the NOCS researchers aim to obtain an entire model year of simulation data from a 12 hour run on HECToR. This enables them to make optimal use of the machine/queues and also allows them to keep up with the post-processing and data transfer of the results as the run progresses. Initial tests showed the they could only achieve around 300 model days in a 12 hour run using 221 processors.

In this section we investigate whether an optimal processor count which satisfies the desire to complete a model year in a 12 hour time slot can be found. To do this NEMO is executed over a range of processors and the number of model days which can be computed in 12 hours, *ndays*, is obtained from:-

$$ndays = 43200/t_{60} \tag{1}$$

where 43200 is the number of seconds in 12 hours and t_{60} is the time taken to complete a 60 step (i.e. 1 day) run of NEMO. This means we ideally need $t_{60} \leq \frac{43200}{365} = 118.36$ seconds. The processor counts investigated varies from 159 to 430. In all tests, runs have been performed with the land cells removed. Figure 3 shows the results in graphical form with the 365 day threshold marked by the dashed line. All processor counts which plot below the dashed line satisfy the 12 hour requirement. From figure 3 we can see that at least 320 processors are required in order to complete a model year of simulation during a 12 hour job run.

4.7 Impact of file I/O

The previous sections have reported the performance of NEMO based on the time taken to complete 60 time steps of the ocean modelling computation. This does not include initialisation time or file I/O time

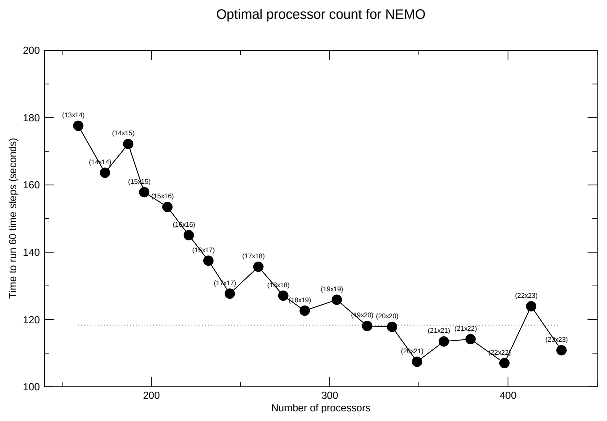

Figure 3: Investigation of optimal processor count for NEMO subject to completing a model year within a 12 hour compute run. The dashed line shows the cut-off point.

which can be a significant fraction of the total runtime. Figure 4 shows the breakdown of the total runtime, for various processor counts.

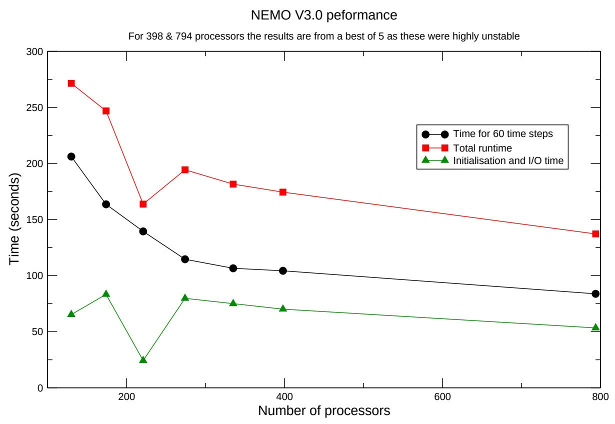

Figure 4: Performance of NEMO V3.0 showing the breakdown of the total runtime for various processor counts. All tests carried out using PGI compiler.

The results displayed in figure 4 were taken from the best of 5 runs and show that as the number of processors increases the relative amount of time spent in initialisation and I/O increases. It should be noted that for 398 and 794 processor runs, the total runtime was found to be highly unstable, varying as much as 400%. The variations in runtime at larger processor counts can be attributed to the variation in the time spent in initialisation (which involves I/O to read in the atmospheric datasets) and writing out the restart file at the end of the run.

The large variation in the initialisation and I/O time occurs because the I/O subsystem on HECToR is a resource shared between other users. Thus, the speed of I/O is governed by the load the system is under at the time when the job runs. If the I/O system is heavily loaded when NEMO attempts to read/write from/to file then the time spent in I/O operations will be increased. As such, any reduction in the amount of time NEMO spends carrying out I/O operations is likely to be highly beneficial to the performance of the code. The approach for improving file I/O in NEMO is described in the next section.

5 Optimising NEMO file I/O

The NEMO input and output files are a mixture of binary and ASCII data. The ASCII input files are typically small and used to define the model parameters or record basic information regarding the status of the run. The binary datafiles are typically much larger (many Mbytes or even Gbytes) and are written in netCDF (network Common Data Form) format (see [3, 4] for details).

NEMO uses a parallel I/O approach for these large binary files where each processor writes out its own data files depending on the part of the model grid that it holds. The current version of the code uses netCDF version 3.6.2. It is expected that significant performance gains may be obtained by converting the code to use netCDF 4.0 which includes lossless data compression and chunking.

As changing NEMO to use netCDF 4.0 is non trivial, the performance of different versions of netCDF was first investigated in order to gain an understanding of the potential performance improvements. Section 5.1 investigates performance of various versions of netCDF on HECToR. Section 5.2 discusses how NEMO was converted to use netCDF 4.0 and the resulting performance gains.

5.1 netCDF performance on HECToR

We investigate the performance of different versions of netCDF on HECToR using the `nocscombine` tool. This is a serial tool which reads in multiple NEMO output files and combines them into a single large file. The tool carries out minimal computation and thus gives a good measure of netCDF and I/O performance.

The results for a number of different versions of netCDF are summarised in table 5.

NetCDF version	`nocscombine` **time (secs)**	**File size (Mbytes)**
3.6.2 classic	343.563	731
4.0 snap unopt	86.078	221
4.0 snap opt	85.188	221
4.0 release	85.188	221
4.0 release*	78.188	221
4.0 Cray	92.203	221
4.0 release classic	323.539	731

Table 5: Comparison of `nocscombine` runtime for various versions of netCDF on HECToR. The * indicates that the system version of zlib was used.

Each run was carried out in batch with the timings reported in table 5 being the best of three runs. The runs were performed consecutively ensuring that the same processing core was used for each. Despite this, considerable variation in runtimes was observed, as much as 100% in some cases. As I/O is a shared resource on the system we have no control over other user activities so these variations are perhaps not surprising.

In table 5 the different versions are defined as follows; 3.6.2 classic denotes the release version of netCDF 3.6.2, 4.0 snap unopt denotes the snapshot release dated 29th April 2008 compiled with default optimisation (i.e. `-O1`), 4.0-opt is the same snapshot release compiled with optimisation set to `-O2`, 4.0 release denotes the final release version compiled with `-O2`, 4.0 Cray denotes the version supplied by Cray and 4.0 release classic is the release version of netCDF 4.0 run in classic (i.e. netCDF 3.6.2 style) mode. The * denotes that the code was compiled using the system version of zlib (version 1.2.1) rather than version 1.2.3.

Examining the results in table 5 we see that netCDF 4.0 clearly outperforms netCDF 3.6.2, both, in terms of runtime performance and in terms of the amount of disk space required. The size of the file output by netCDF 4.0 is 3.31 times smaller than that output by netCDF 3.6.2 and the runtime is roughly 4 times faster. Comparing the results from version 3.6.2 and 4.0 run in classic mode (c.f. 3.6.2 classic with 4.0 release classic) they suggest that the runtime savings do not just result from the reduced file size. It's possible that there are some algorithmic differences between the versions or perhaps the dataset now fits into cache better thus reducing memory latency.

The level of optimisation used to compile the netCDF library appears to have minimal effect on the performance. The system version of zlib (version 1.2.1), outperforms version 1.2.3. However, as netCDF 4.0 clearly states that version 1.2.3 or later is required it is potentially risky to use the older version as functionality required by netCDF 4.0 maybe missing.

In summary, based on the results obtained from the `nocscombine` code, using netCDF 4.0 instead of netCDF 3.6.2 will likely give significant performance improvements for NEMO. The amount of disk space used could be reduced by a factor of 3 and the time taken to write this information to disk could be reduced by a factor of 4. The time taken to compress and uncompress the data at the post-processing stage still needs to be quantified but the early results are promising.

5.2 Converting NEMO to netCDF 4.0

In this section we describe the procedure for adapting NEMO V3.0 to use netCDF 4.0. To compile NEMO with netCDF 4.0 some changes are made to the main Makefiles to ensure that they link to netCDF 4.0, HDF5 zlib and szip. To utilise the chunking and compression features of netCDF 4.0 changes to the NEMO source code are required.

Converting NEMO to use netCDF 4.0 with chunking and compression is best tackled as a two stage process. The first step is to generate netCDF 4.0 format output files without any compression/chunking. The second step is to add chunking and/or compression and thus take advantage of the full functionality of netCDF 4.0.

For step one we need to tell netCDF library that we want to take advantage of the new features. This can be achieved by making some minor modifications to the NEMO code. In particular, all subroutine calls to `NF90_CREATE` need to be modified such that each instance of `NF90_NOCLOBBER` is replaced with `NF90_HDF5`. The two source files which require alteration are:- `histcom.f90` and `restcom.f90`. In addition, the file `fliocom.f90` also requires the variable `m_c` to be set to `NF90_HDF5`.

With these modifications made, the code is then recompiled and tested. The main output files should now be in netCDF 4.0 format which can be verified by attempting to read one of the output files with different versions of `ncdump` [3].

The modifications described above allow netCDF 4.0 format files to be output from NEMO. However, as yet no compression or chunking has been included. To use the chunking/compression features of netCDF 4.0 additional modifications must be made to the code. These modifications are summarised below:-

1. Declare new variables relating to the chunking and compression e.g.

   ```
   integer, dimension(4):: chunksizes
   integer:: chunkalg,shuffle,deflate,deflate_level
   ```

   `chunksizes` is an array containing the chunksize to be used for each dimension of the dataset.

2. Initialise the chunking and compression related variables, e.g.

   ```
   chunksizes(1) = 10
   chunksizes(2) = 10
   chunksizes(3) = 10
   chunksizes(4) = 1
   deflate_level = 1 ! Turn compression on
   deflate = 1       ! Level of compression
   shuffle = 1       ! Allow shuffling
   chunkalg = 0      ! Turn chunking on
   ```

   Here chunksize is chosen such that it is less than the number of data points within that dimension.

3. Following each call to `nf90_def_var`, add new calls to `nf90_def_var_chunking` and `nf90_def_var_deflate` to ensure that chunking and compression is applied to each output variable.

The code was then re-tested and the size of the output files compared with those created without and chunking and compression. The results are summarised in table 6.

The results demonstrate that a significant reduction in disk usage (~4 Gbytes for this example) can be achieved by using the chunking and or compression features of netCDF 4.0 within NEMO. For the test model, no significant improvement in the runtime is observed however this is to be expected as the restart files (not converted to netCDF 4.0) dominate in terms of their size. In our test model we run for 60 time steps, outputting data every 30 time steps with a restart model written output after 60 time steps. However, for a production run of NEMO the model would typically run for in excess of 10,000 time steps with output every 300 steps and a restart file written every 1800 time steps. Therefore an improvement in runtime would be expected due to the reduction in output file size. The actual size of any improvement will depend on the output frequency chosen, which in turn depends on the parameters being modelled and the particular science studied.

[3] `ncdump` is part of the netCDF library and is an executable which can be used to convert a netCDF file into a user readable text file.

File name	**netCDF 3.X disk use (MB)**	**netCDF 4.0 disk use (MB)**	**Reduction factor**
`*grid_T*.nc`	1500	586	2.56
`*grid_U*.nc`	677	335	2.02
`*grid_V*.nc`	677	338	2.00
`*grid_W*.nc`	3300	929	3.55
`*icemod*.nc`	208	145	1.43
`*restart_0*.nc`	9984	9984	1.00
`*restart_ice*.nc`	483	483	1.00

Table 6: Effect of chunking and compression on the size of the NEMO 3.0 output files. For each file name the usage is the sum of all 221 individual processor output files.

6 Nested model performance

In this section we describe how to use nested models within NEMO, investigate their performance and discuss some of the issues associated with getting them to run successfully on HECToR.

In NEMO any number of nested regions can be used and within each model and multiple levels of nesting are also possible. Figure 5 illustrates this with an example showing the main model, A, containing two nested regions B, and C with region C having one level of nesting and region B having two levels of nesting.

Nested models in NEMO use the Adaptive Grid Refinement in Fortran (AGRIF) pre-processor code called, `conv`, to generate the code for the nested regions. By default, the NEMO code uses arrays of dimension `(jpi, jpj)` where `jpi` and `jpj` are set as parameters within the code. The pre-processor completely re-structures the code by inserting interface routines which pass array information from a special dynamically allocated AGRIF data type the size of which is determined based on the desired size of the nested region. Essentially the AGRIF pre-

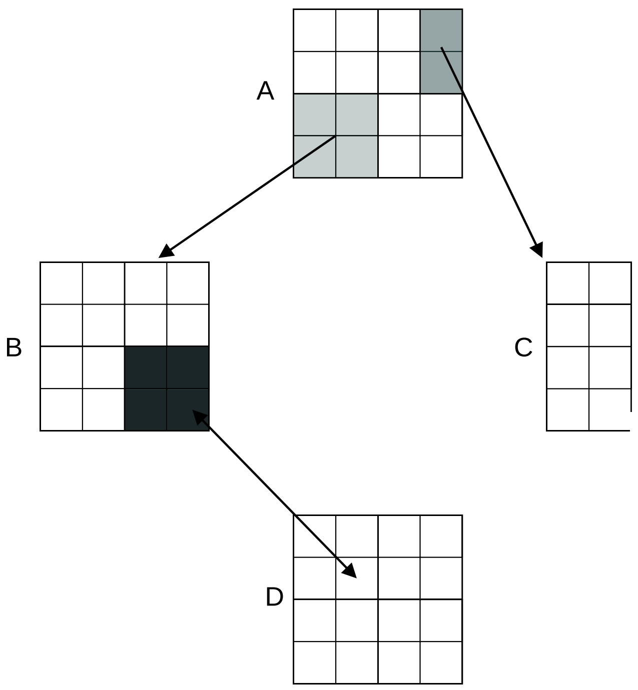

Figure 5: Schematic showing a possible NEMO grid, A, containing two nested regions B and C. Nested region B contains a second level of nesting. Note that the schematic is not drawn to scale, in practice the B, C and D regions would be smaller in physical size and grid spacing than region A. However, their increased resolution may mean they actually contain similar numbers of actual grid points.

processor allows the same ocean model to be run on grids with different resolutions in space and/or time. Further details on running and setting up nested models within NEMO can be found at [5].

Two test models are supplied by the NOCS researchers, BASIC and MERGED. BASIC is a largely unmodified version of the NEMO source code which attempts to use the AGRIF pre-processor to set up a nested model. The BASIC model is a 2 degree model inside which a 1 degree model is run. Only 1 nested region is used in the BASIC model. The MERGED model is an extension of BASIC model including the NOCS specific code changes. Unlike the BASIC model it has two nested regions. It also runs at a higher resolution with the outer model resolution being 1 degree and the nested regions being $\frac{1}{4}$ of a degree. In this paper we will only consider the results obtained from the BASIC model.

The BASIC nested model is compiled and run on HECToR using the same optimisation flags (`-O3`) as for the un-nested version of the code. On running the code an error is reported on time steps 27 and 54 for un-nested and nested parts of the model respectively. By examining the codes output files the error can be attributed to the un-nested part of the model and is found to be due to the zonal velocities increasing too rapidly suggesting a numerical instability in the problem.

Figure 6 shows the zonal velocity plotted against model time for the non-nested model for 2 different model time steps and optimisation levels.

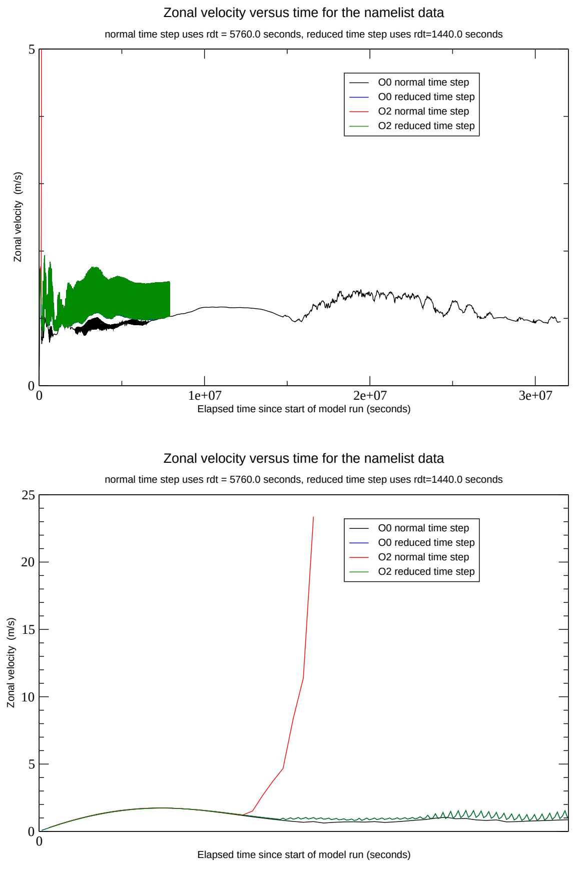

Figure 6: Zonal velocity against model time for the non-nested model. The top figure shows the full plot with the bottom figure zoomed in on the first few time steps.

From figure 6 it is obvious that the problem with the velocity occurs for the non-nested model (see red line on figure 6). The original time steps of 5760.0 and 2880.0 seconds for the non-nested and nested models are simply too large. Reducing the time steps by a factor of 4, to 1440.0 and 720.0 seconds prevents the zonal velocity from blowing up. With the reduced time step the `-O0` and `-O2` versions of the code behave very similarly. The zonal velocities are almost identical (the blue and green lines overlap on figure 6.

It seems there are two choices for running the AGRIF models; run with a code compiled with `-O1` to avoid the numerical instabilities or reduce the time step. The second solution is safer as the model could become unstable even with `-O1` or less. The researchers have reduced the time step as suggested and confirmed that the results appear sensible when compared with those obtained on their SGI cluster.

Using the reduced time step the code was tested from 16 up to 128 processors to examine the scalability of the nested model. The scaling results for the BASIC model are given in table 7. From table

No. of processors	**Time for 5475 steps (seconds)**	**Cost per time step (AU's)**
16	4397	0.0172
32	2164	0.0169
64	1366	0.0208
128	1636	0.0510

Table 7: Total runtime of the BASIC nested model over 5475 time steps for different processor counts. Optimisation levels of `-O3`, `-O2` and `-O2` were applied to the `Makefile`, `AGRIF/Makefile` and `IOIPSL/src/Makefile` respectively.

7 we see that the 64 processor run gives the fastest runtime for this particular model. However, the 32 processor run is actually more efficient in terms of the AU usage per model time step. Providing this longer runtime is acceptable, researchers may wish to run this model on 32 processors in order to minimise the cost (in AU's) per model time step.

7 Conclusions

Two different versions of NEMO (2.3 and 3.0) have been compiled and tested on the HECToR system. The performance of these versions has been investigated and an optimum processor count suggested based on the researchers' requirements for job turnaround. The performance of both the PathScale and PGI compilers has been investigated along with an investigation of how the performance varies with the choice of compiler flags. The NEMO code has been found to scale up to 1024 processors with the best performance in terms of runtime versus AU usage being obtained between 128 and 256 processors. Running NEMO in single core mode is found to be up to 18.59% faster than dual core mode, however, the reduction is not sufficient to warrant the increased AU usage. The choice of grid dimensions has been investigated and is found to be optimal for square grids. Where square grids are not possible choosing the dimensions such that the number of cells in the horizontal direction is less than the number in the vertical direction (i.e. choosing `jpni < jpnj` within the code) gave the best performance. Removal of the land only squares from the computations gave significant reductions to the AU usage, by as much as 25% at larger processor counts. The runtime was also found to decrease, albeit by a lesser extent. Profiling of the code suggests that NEMO spends a considerable amount of time in file I/O and thus any reduction that can be made in this area will be beneficial.

As a means of improving the I/O performance of NEMO the performance of various versions of netCDF was investigated. NetCDF 4.0 is found to give a considerable reduction to both the amount of I/O produced and the time taken in I/O when using the `nocscombine` tool. In addition, the version of netCDF 4.0 installed as part of this work is found to be between 8-20% faster than that installed centrally (via the modules environment) on the system.

The NEMO code has been converted to use netCDF 4.0 for its main output files resulting in a reduction in output file size of up to 3.55 times relative to the original netCDF 3.X code. For the test model no significant runtime improvement is observed. It is expected that a real research type run should benefit more due to the different frequency of output involved.

The BASIC nested model has been compiled and tested and problems with the time step interval identified and rectified. The performance of the BASIC nested model has been investigated with the optimal processor count (in terms of AU usage per time step) found to be 32. The more complex MERGED nested model has not yet run successfully on HECToR.

8 Acknowledgements

This work was funded by a HECToR dCSE (Distributed Computational Science and Engineering) grant administered by NAG Ltd on behalf EPSRC.

The Cray Centre of Excellence staff are gratefully acknowledged for their help with various aspects of this NEMO dCSE project. Chris Armstrong from NAG is acknowledged for his assistance with getting netCDF running on the X2 system.

9 About the Authors

Fiona Reid is an Applications Consultant at EPCC, The University of Edinburgh and can be reached at f.reid@epcc.ed.ac.uk.

References

[1] *NEMO Home Page* `http://www.nemo-ocean.eu/`

[2] *User's Guide to the HECToR Service, Version 2.0*, `http://www.hector.ac.uk/support/documentation/userguide/hectoruser.pdf`

[3] *Network Common Data Form (netCDF) Web Page*, `http://www.unidata.ucar.edu/software/netcdf/`

[4] *NetCDF User's Guide for Fortran 90*, `http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90/`

[5] *NEMO Web Page - AGRIF nesting tool (accessible only to registered users)* `http://www.nemo-ocean.eu/index.php/Using-Nemo/Pre-and-post-processing-packages/Pre-Processing/AGRIF-nesting-tool`