Towards Generic Scalability of NEMO; the oceanographic component of the Unified Model.

Andrew Porter and Mike Ashworth, Advanced Research Computing Group, Computational Science and Engineering Department, STFC Daresbury.

Introduction to NEMO

NEMO (Nucleus for European Modelling of the Ocean, see [Madoc 2008]) has been chosen by the UK Meteorological Office as the replacement for the original ocean-modelling component of the Unified Model (UM). It is therefore important to consider the scaling of NEMO when looking at the scaling of the UM as a whole.

In the past, the development of NEMO has been strongly influenced by the characteristics of vector supercomputers such as the Earth Simulator (ES). The move to the massively-parallel architectures that dominate the current supercomputer landscape has been incorporated in the code comparatively recently and there are a number of consequences of this:

- adherence to static memory (because of early performance implications on the ES in particular) means that the number of processing elements (PEs) to use must be set at compile time;
- every PE writes a number of output files to disk (because the ES does not have a shared file-system) using the serial netCDF library;
- the domain decomposition is regular and calculations are performed at all grid points, irrespective of whether they are over land or ocean, and a mask applied to the results;
- the code does a lot of global communication which will limit scaling to large numbers of processors.

In the form suitable for running on massively-parallel architectures, NEMO is a Fortran90 code that uses the Message Passing Interface (MPI) library for inter-process communication. When running a whole-Earth configuration, the PEs along the northern-most boundary of the domain are collected into an additional MPI communicator. Further computation and communication is performed within this communicator to deal with the boundary conditions imposed by the poles.

The inclusion of sea ice in a configuration increases the computational cost considerably. Unfortunately, since the ice computation uses the same decomposition as the ocean, only a few PEs actually have any ice within their domains and thus this extra cost is not very
evenly shared.

**Model Configurations**

The standard NEMO distribution comes with two configurations; GYRE and ORCA2_LIM. The former is an artificial test case and the latter is a global, ocean model at two-degree resolution coupled with the Louvain-la-Neuve sea-ice model (LIM). Since ORCA2_LIM is a part of the official distribution, this is the configuration that we have concentrated on in the development work carried-out within the project. It must be noted that the low resolution of this model does inherently limit its scalability but because we are working to improve the generic scalability of NEMO, any improvements will carry through to higher-resolution configurations. Although ORCA2_LIM was used for the majority of the work, we did have access to ORCA1 and ORCA025 configurations of NEMO v.2.3 (unfortunately, which model configuration is being run must be built into the NEMO code at compile time), corresponding to resolutions of one and one-quarter degrees, respectively. These higher-resolution configurations were used to assess the scaling of NEMO and identify any bottlenecks.

**Benchmarking Platforms**

The three different machines which were used to assess the scaling performance of NEMO are described in table 1.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPCx</td>
<td>160 IBM POWER5-575 eServer nodes (2,560 P5 processors) delivering 15.36 TeraFlop/s peak, or 12.9 TeraFlops/s sustained (as rated in the Top500 list). The system is equipped with 5.12 TByte of memory (2GB per processor).</td>
</tr>
<tr>
<td>HECToR</td>
<td>60 Cray XT4 cabinets (5,664 AMD Opteron dual-core processors with 3GB of memory per core) delivering 59 Tflops at peak.</td>
</tr>
<tr>
<td>IBM BlueGene/P at STFC Daresbury</td>
<td>One BlueGene/P rack (1024 quad-core PowerPC 450 processors at 850 MHz with 2GB of memory per processor).</td>
</tr>
</tbody>
</table>

Table 1: Machines used for testing the performance of NEMO.

These three machines provide a good test-bed for checking the generic scalability of the new version of NEMO as they all differ in the types of processor and interconnect that
they use.

**Initial Scaling Performance**

The performance of the standard release of NEMO (version 2.3 at the time of the start of this project) was assessed on HPCx for the ORCA1 and ORCA025 configurations. As can be seen from the plot in figure 1, NEMO ceases to scale at 256 processing elements (PEs) for ORCA1 and 512 PEs for the larger ORCA025 configuration.

![Performance of NEMO 2.3 on HPCx for ORCA model resolutions of 1 (blue) and 0.25 (red) degrees. Figures are derived from the wall-clock time taken for a complete run of 720 steps. Lines are guides to the eye.](image)

It is important to note that these results show that increasing the configuration resolution by a factor of four in both the x and the y directions does not significantly improve the scalability of the simulation. This means the "weak scaling" of NEMO is poor which is a significant barrier to the use of the code to study higher-resolution models (required to achieve greater physical fidelity) on large machines.

By default, NEMO uses a pre-conditioned, conjugate-gradient (PCG) iterative solver for the global, surface-pressure distribution (a consequence of the rigid-lid approximation). This solver is not ideal for use on large, parallel machines as it requires global communication amongst all PEs at every iteration. In an attempt to improve on this,
NEMO also includes a version of the Successive Over Relaxation (SOR) solver that uses an additional (configurable) halo in order to reduce the frequency with which PEs must communicate. (The ‘standard’ halo used throughout the rest of the NEMO code is one cell wide.) Figure 2 shows the performance of this solver over a range of halo widths for the ORCA1 configuration running on HPCx.

![Figure 2: Effect of varying the additional width of the SOR solver-specific halo on overall performance (e.g. “SOR+1” means that the SOR solver was used with a halo one cell wider than the standard width). The performance of NEMO when using the PCG solver is shown for comparison (dark-blue curve). Lines are guides to the eye.](image)

Comparing the magenta and dark-blue curves in figure 2, we can see that even without any additional halo, the SOR solver scales considerably better than the PCG solver resulting in an approximate increase of 20% in performance on 256 PEs. Increasing the additional halo width available to the SOR solver improves this until a 40% improvement in performance on 256 PEs is achieved with a halo width of 7. Increasing the width of the additional halo beyond this was found to leave the performance unchanged. Although using this SOR scheme has improved the performance of the model considerably, better scaling is still required if high-resolution models are to be run efficiently.
Optimisations Performed

With the current trend of high-performance computer architectures towards machines with hundreds of thousands of scalar cores, each with a local memory cache, it seems advisable to re-visit the approach taken to dealing with land in NEMO. Achieving optimal performance on a vector machine involves being able to fill pipelines with a long series of identical operations and therefore, rather than continually test to see whether each grid point is over land or not, it makes sense to simply apply a mask to the results of a calculation since the resulting multiplication by a vector containing ones and zeroes can be done very efficiently. However, on cache-based, scalar architectures this approach is inefficient and it is better to avoid doing any unnecessary calculation in the first place. With this approach it is important that the number of ocean points assigned to each PE has minimal variation in order to ensure that the simulation remains load balanced and consequently an irregular domain decomposition is required. Following the group's experience (Ashworth, Holt and Proctor 2004) with the Proudman Oceanographic Laboratory Coastal-Ocean Modelling System, we chose to implement a recursive domain-decomposition scheme in NEMO. Applying such a scheme to the global ORCA2 model running on 16 PEs gives the decomposition shown in figure 3.

Figure 3: an example of the domain decomposition produced by the recursive algorithm for a 16-PE run of a global model.
In order to support such a scheme and the varying domain sizes it produces, NEMO had to be converted to use dynamic rather than static memory. This has the additional advantage of giving the user the freedom to choose the number of PEs to run on at job-submission time rather than at compile time. Looking at a part of the 16-way decomposition in more detail (figure 4) we can see that its irregularity brings additional benefits in that it is now rare to have to explicitly deal with corner- or indeed single-point-communications between domains. As a consequence, the number of MPI messages that must be sent is reduced while their average length is increased (Ashworth, Holt and Proctor 2004). This reduces the amount of overhead that is incurred every time an MPI communication is initiated. Figure 4 also illustrates that the number of neighbours a domain has is no longer guaranteed to be a maximum of eight. As a consequence, the halo-swap section of the code must be extended to cope with situations like that shown in figure 4 where, for example, PE 3 must send a part of its eastern halo to PE 9 and another part to PE 12.

![Figure 4: detail of a part of the 16-PE decomposition of a global model.](image)

**Performance of the new version**

The new version of NEMO was tested on the ORCA2_LIM configuration since that is the only realistic configuration that is distributed as standard with the official NEMO release. It was found initially that the move to dynamic memory caused significant performance degradation on both HPCx and HECToR. The culprit for the majority of the slow down was found to be some large work-space arrays of rank 5. The extent of each of the last three dimensions of these arrays was just two. Re-ordering the ranks of these arrays gave a dramatic improvement in performance. When these arrays were static the compiler
‘knew’ that these last three dimensions were only small and presumably was able to unroll any loops involving them. In changing the code to use dynamic memory, the compiler has much less information making it important that the code be written with performance in mind.

With the version of the code containing the re-ordered arrays, runs were performed on HPcX, HECToR and the BlueGene/P at STFC Daresbury. In order to reduce the variability introduced by doing Input/Output of data, all of the results reported in this section are derived from the time taken for NEMO to go from time-step 10 to time-step 90 of a 100-step run of the ORCA2.LIM configuration (so avoiding start-up and shutdown costs). Results from three separate jobs are averaged to produce each of the data-points plotted in the graphs below.

On HECToR (Phase I), figure 5 shows that the performance of the new ('Dynamic') version of the code is generally very similar to that of the original, especially at the higher PE counts which are the most significant when considering scaling.

![Figure 5: performance comparison of the dynamic-memory and original versions of NEMO on HECToR. Lines are guides to the eye.](image)

The only exception to this are the jobs on 64 PEs where the new version of the code appears to be significantly slower. However, given that the compiler has a lot more information to work with when optimizing the original version of NEMO (because the arrays sizes are known), it is pleasing that there is such close agreement in the
performance of the two versions. (Note that the PGI compiler was used to build both versions of the code.)

Figure 6 shows the results of the same performance comparison carried out on HPCx. The difference in performance between the two versions is far more significant on this machine although the shape of the two scaling curves (and hence the scalability of the two versions) is the same.

As on HECToR, the performance on 64 PEs seems to be better than one would expect from looking at the general shape of the curve. Unlike on HECToR however, this applies to both versions of the code. In general, NEMO scales better on HPCx than on HECToR; maximum performance is obtained on 256 PEs (c.f. 128 PEs on HECToR) although it does subsequently drop-off more quickly.

Finally, on the BlueGene/P the two versions of NEMO again produce very similar scaling curves (figure 7), albeit with the one for the new version shifted down by a roughly constant factor of 15%.
The performance of both versions of the code continues to improve as the number of PEs is increased up to 1024 (on higher PE counts, the simulations fail due to over-decomposition). However, although the scaling performance of both versions is better on this machine, the absolute performance that either achieves is still only 40-50% of that obtained on HPCx.

**Conclusions and Future work**

We have produced a version of NEMO that has considerably more flexibility than the original and that may be compiled with more extensive code optimisations switched on. However, it is not yet significantly faster or more scalable than the original. Key to exploiting the potential of the new version is the way in which NEMO orders its calculations. Currently this is implemented such that iterations are ordered according to depth, latitude and then longitude. *i.e.* the code iterates over the whole length and breadth of a domain at a given depth before moving to the next depth and repeating. This means that any check on whether a grid point is over land or ocean must be repeated for every single loop iteration which severely limits its usefulness. However, if the calculations and associated arrays were to be re-ordered such that they proceed first down a water column...
and then in the latitudinal and longitudinal directions then this would enable whole 'columns' of calculation to be avoided when they are over land. Such re-ordering has been shown to be highly beneficial in the POLCOMS code (Ashworth, Holt and Proctor 2004) and we expect it would markedly improve the computational/single-node performance of NEMO – the area in which the new version compares least well with the original.

Profiling experiments on NEMO on 64 PEs showed that the time spent in MPI communications makes up a significant proportion (35%) of run-time and limits the scaling of the code, even on what is a modest PE count. The new communications scheme that we have introduced gives the application programmer the flexibility of specifying in which directions the halos must be updated. This provides scope for removing any unnecessary communication and thus improving scaling but we have yet to perform such optimization since this requires analysis of the spatial dependencies of the calculations that follow every field update.

Throughout the code there are also points where a series of halo exchanges are performed, one variable at a time. The performance of these exchanges would be improved considerably if they could all be done together. This would require an extension to the new communications scheme in order to allow halo exchanges to be performed on an arbitrary number of fields in a single call. Such an approach will result in fewer, larger MPI messages thus reducing the overhead of setting-up message transfers and making the most of available network bandwidth.

References