



ARCHER2-eCSE04-6 Technical Report

Optimising NEMO-FABM-ERSEM for High Resolution

Andrew Sunderland¹, Dale Partridge², Michael Wathen², Andrew Porter¹

¹ Hartree Centre, STFC Daresbury Laboratory

² Plymouth Marine Laboratory

Abstract.....	2
1 Introduction	2
2 Installation of NEMO-FABM-ERSEM on Archer2.....	4
2.1 Installation Configuration and Datasets	4
2.2 Use of Profiling Tools	5
2.3 Running NEMO-FABM-ERSEM Jobs on Archer2	6
3 Performance Analysis for NEMO-FABM-ERSEM on Archer2.....	6
3.1 Parallel Scalability Summary for AMM7, AMM15 Datasets on Archer 2 ...	6
3.2 Parallel Performance Profiling	9
3.2.1 Identifying Computational Hotspots with Craypat.....	9
3.3 Effect of Passive Tracers at scale	10
3.4 Effect of Active Tracers at Scale	12
3.5 Load-balance variation with node count	16
4 Optimisations of NEMO-FABM-ERSEM on Archer2	17
4.1 Optimal Configurations on Archer2	17
4.1.1 XIOS Server Configuration	17
4.1.2 Ocean Client gap intervals.....	17
4.2 I/O Performance	19
4.2.1 Striping I/O Across Object Storage Targets (OSTs)	20
4.2.2 I/O Environment Settings	20
4.3 FABM-specific settings.....	21
4.4 Optimal Energy Consumption on Archer2.....	21
5 Conclusions	22
6 References	23
Appendices	24
Appendix A	24
Appendix B	26
Appendix C	28

Abstract

NEMO-FABM-ERSEM is a coupled numerical modelling system for simulating the marine environment, used by many research groups and led by Plymouth Marine Laboratory [1], National Oceanographic Centre [2] and the Met Office [3]. NEMO-FABM-ERSEM is geographically versatile and able to address topics from Climate change to resource provision, pollutants etc. A conservative estimate is that 20 individual users of this code addressing 15 separate projects have recently, are or will utilise ARCHER 2.

This is a relatively expensive code system to run, especially as both model complexity and spatial resolution continues to increase. This report describes the performance optimization of the newest version of the code on Archer2 based upon simulations in the Northwest European Shelf, namely the AMM7 and AMM15 grids.

1 Introduction

3D coupled marine models such as NEMO-FABM-ERSEM represent the marine environment with a discrete grid of layered, (usually squared) prisms. A key characteristic of these models is their resolution, expressed as the distance between prisms and/or layers, with higher resolution generally providing more skilful simulations and better addressing issues at the scale of biological processes and stakeholder interaction. However, increasing resolution greatly impacts the computational and storage requirements for a given simulation. This introduces a balance between the desire to use the highest resolution/skill possible and the computational aspects; memory, I/O, and communication. As HPC platforms develop and we move towards ever higher resolution it is vital to ensure that code is optimised to obtain the maximum benefit from the available resources. In the case of NEMO-FABM-ERSEM, simulations can be computationally expensive. By dedicating time to ensure the code is as optimized as possible to run on ARCHER2 significant time and resources can be saved.

NEMO: The Nucleus for European Modelling of the Ocean (NEMO) [4] is a framework for ocean and climate modelling (www.nemo-ocean.eu). The ocean component of NEMO is a primitive equation model employed for a range of idealised, regional, and global ocean circulation studies. It provides a flexible tool for studying the ocean and the wider earth climate system over a wide range of space and time scales [5]. NEMO is coded in Fortran and makes use of the parallel HDF5/NetCDF and MPI libraries.

FABM: The Framework for Aquatic Biogeochemical Models [6] provides a generic, easy to use, high performance coupling layer that connects a hydrodynamic model (1D column to 3D world ocean) with multiple available biogeochemical sub-models. FABM enables complex biogeochemical models to be developed as sets of stand-alone, process-specific modules with a layer of separation from the complexity of the spatial domain simulation. This approach has been adopted to implement several large ecosystem models in FABM, including ERSEM at PML. FABM is coded in Fortran,

but it does not itself handle parallelization. Instead, it transparently works with any parallelization of the hydrodynamic model.

This project will use NEMO v4.0.4 coupled with FABM v1.0. This coupled code is provided by PML and available for download on github [7].

ERSEM: The European Regional Seas Ecosystem Model [8] is a complex marine ecosystem model which has grown in scale and scope in recent years. It addresses biogeochemical and ecological systems in many applications in global regional seas and more recently the global ocean, engaging in a range of problem solving, predictive and impact studies.

Its strength and uniqueness lies in its ability to define relatively complex ecosystems in both pelagic and benthic environments. The model utilises a functional group approach to describe 4 classes of phytoplankton, 3 classes of zooplankton, bacteria, dissolved and labile organic matter (DOM) and 3 size-classes of particulate organic matter (POM). This configuration introduces $O(50)$ additional state variables with associated diagnostics and parameter fields into the system at significant computational cost.

ERSEM is also coded in Fortran and relies on FABM to handle all communication with the hydrodynamic model.

XIOS: XML-IO Server [9] is a library dedicated to the management of netcdf format input and output within models such as NEMO. It parses XML files at runtime for flexibility and allows for temporal and spatial post-processing operations. Additionally XIOS is frequently run asynchronously in server mode on separate core(s) so data writing does not slow down computation. XIOS is coded in C++.

This project used XIOS v2.5, available for download via svn [10]

At the start of ARCHER2 the NEMO community performed some preliminary tests on various core counts and packing strategies for the physics-only set up on a different domain to AMM7, with findings presented in the ARCHER2 documentation [11]. As part of porting the NEMO-FABM-ERSEM code to ARCHER2, some initial timing experiments have also been performed to assess the impact of varying the number of output variables or the number of cores.

The work presented here expands on these initial experiments to fully understand how the code performs on ARCHER2. We will breakdown the code to identify the most expensive routines and analyse the impact various components have on performance. This includes changing the number of cores used, the core placement strategy, increasing/decreasing the model complexity, increasing the resolution of the model and other smaller options. Any avenues for optimisation will be investigated and recommendations provided to ensure future simulations run as efficiently as possible on ARCHER2 and future HPC platforms.

2 Installation of NEMO-FABM-ERSEM on Archer2

2.1 Installation Configuration and Datasets

The benchmarking experiments are performed on two models of the northwest European shelf, known as the Atlantic Margin Model (AMM).

The first, AMM7 is a key simulation used frequently by multiple UK institutions. It has a horizontal resolution of 7 km and uses $O(3 \times 10^6)$ active grid cells. A NEMO-ERSEM configuration of AMM7 will typically use around 1200 cores as this is the limit of parallelisation for this grid.

The next generation version of this domain is AMM15, which increases the resolution to 1.5km with $O(4.5 \times 10^7)$ active cells [12]. The typical core count for this experiment is yet to be determined.

The following repository is available to set up this AMM7 configuration on ARCHER2:
<https://github.com/dalepartridge/AMM7-NEMO4-FABM-setup>

It contains scripts to source the software from the relevant repositories and compile them under the cray programming environment. There are also all configuration files needed to run the experiment, with larger files available from the n01 shared area: /work/n01/shared/dapa/AMM7/.

AMM15 uses the same code base as the AMM7 experiment. Run configuration files and the larger input files are all available from the n01 shared area: /work/n01/shared/dapa/AMM15.

The general physical and computational characteristics of NEMO-FABM-ERSEM with AMM7 and AMM15 datasets are compared in Table 1.

Characteristics	AMM7 Dataset	AMM15 Dataset
Horizontal Grid Points	~110,000	~2.0 Million
Depth Levels	51	51
Typical Number of Compute Nodes	6-24	12-200
Typical Nemo Tasks	100-1200	1000-20000
Typical Representative Benchmark Run Time (from RESTART)	10 minutes	60 minutes
Output Data	~ 10 GBytes	~ 125 GBytes

Table 1 Computational Characteristics for AMM7 & AMM15 Datasets

2.2 Use of Profiling Tools

The NEMO code itself generates detailed performance data based on a multitude of internal timers within the code. Where a wider range of performance data is required, we have exploited the CrayPat and CrayPat-Lite profilers [13] provided as part of the Archer 2 service. CrayPat can add large runtime overheads, including the generation of huge amounts of profiling data, even for a single NEMO-FABM-ERSEM run with restricted timesteps. CrayPat-Lite has therefore been the profiler tool favoured here. The procedure for generating profiling data on Archer2 via CrayPat is described in [13]. One extra step required for CrayPat profiling of NEMO-FABM-ERSEM is to manually rebuild the `lib_fcm_nemo.a` and `lib_fcm_xios_server.a` libraries after the compilation. This can be achieved by locating the directories `obj` containing the object files of the NEMO and XIOS installations and from there, issuing the commands below to rebuild the libraries:

```
ar rs ..../lib/lib_fcm_nemo.a *.o
```

```
ar rs ./lib/lib_fcm_xios_server.a *o
```

2.3 Running NEMO-FABM-ERSEM Jobs on Archer2

NEMO-FABM-ERSEM runs on Archer2 usually take place using the SLURM batch system [13]. Users can choose to run NEMO-FABM-ERSEM using either *attached* or *detached* mode. A description of both modes is provided in [11]. All the findings described in this report derive from jobs run in detached mode, where external XIOS I/O-servers to help manage the large volumes of data and the user must specify the placement of clients and servers on different cores throughout the node using the `-cpu-bind=map_cpu:<cpu map>` `srun` option to define a CPU map or mask. This process was simplified by employing the `mkslurm_hetjob` script developed by Andrew Coward at the UK National Oceanography Centre. The listing of a SLURM batch script generated by `mkslurm_hetjob` for a parallel job involving 504 ocean cores and 6 XIOS servers is provided in Appendix A. For optimal performance, each XIOS server task is located within its own exclusive NUMA region. From December 2022, the default CPU frequency on Archer2 is set to 2.0 GHz, however the benchmarking and profiling runs presented in this report were undertaken prior to this date and therefore the default frequency used throughout is 2.25 GHz. Testing different CPU frequencies and measuring energy consumption is discussed in Section 4.4.

3 Performance Analysis for NEMO-FABM-ERSEM on Archer2

3.1 Parallel Scalability Summary for AMM7, AMM15 Datasets on Archer 2

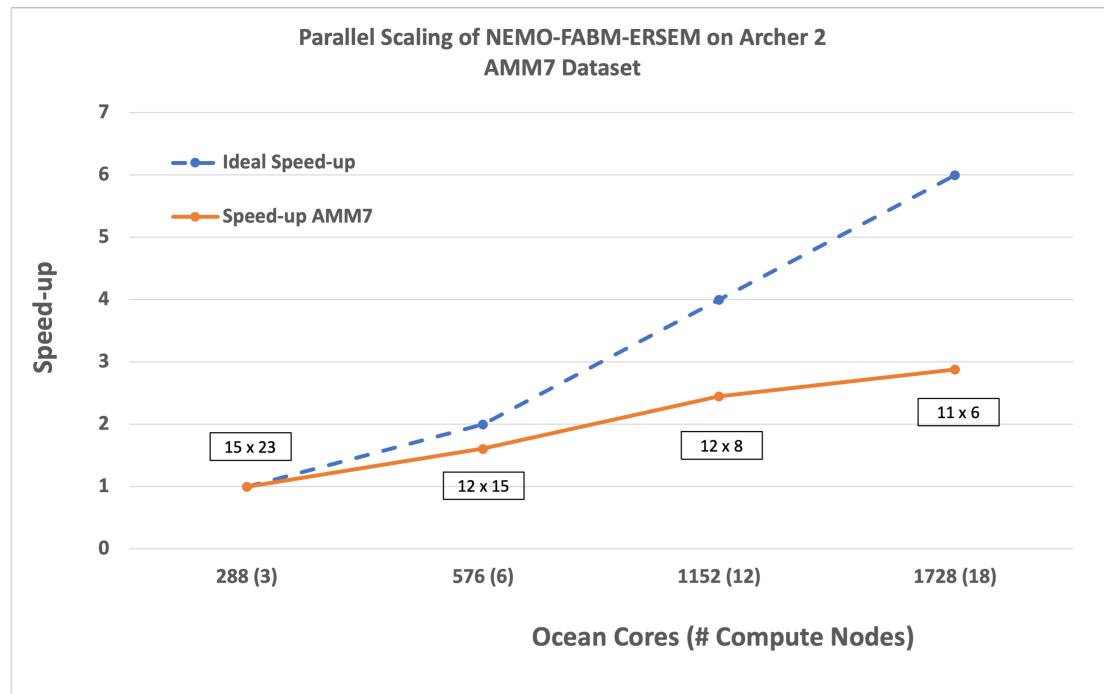


Figure 1 Parallel Scalability of NEMO-FABM-ERSEM on Archer2, AMM7 Dataset (no gaps between Ocean Cores (g0 setting) and 1 XIOS server per Archer2 node). Boxed labels represent the number of points per subdomain.

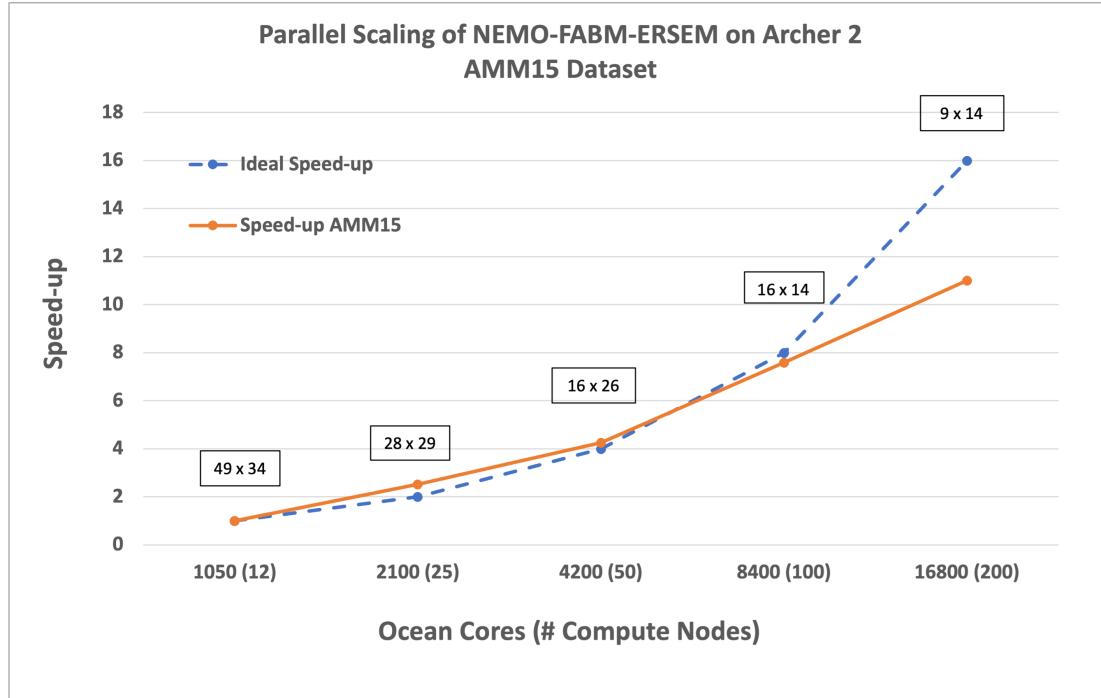


Figure 2 Parallel Scalability of NEMO-FABM-ERSEM on Archer 2, AMM15 Dataset (no gaps between Ocean Cores (g0 setting) and 1 XIOS server per Archer2 node). Boxed labels represent the number of points per subdomain

The parallel performance of NEMO-FABM-ERSEM for the AMM7 and AMM15 datasets on Archer2 is shown in Figure 1 and Figure 2. For both datasets, the NEMO-FABM-ERSEM (or Ocean Cores) tasks are configured to fill compute nodes densely with no spacing intervals (g0) (see Section 4.1.2 for more details). One XIOS I/O server is dedicated to each node and occupies its own 16-way NUMA region on each node. Output frequency for representative datasets is set as daily. For AMM7, parallel scaling is good at lower core counts, but only small relative performance gains are observed at the higher core counts. For AMM15, parallel scaling is almost ideal for up to 100 compute nodes, but the speed-up begins to decrease at 200 compute nodes. An important factor for the parallel scaling is the relative subdomain size per task, shown as labels in the figures. Providing this local compute requirement is sufficiently large, the relative costs of data exchange between tasks, such as halo-exchange operations, remains low. As the number of compute nodes becomes relatively large, halo exchange communication overheads begin to impact performance. The halo exchange functions in NEMO have been the focus of past optimisation efforts, and this is reflected in the generally impressive parallel strong-scaling of the target dataset (AMM15) for Archer2.

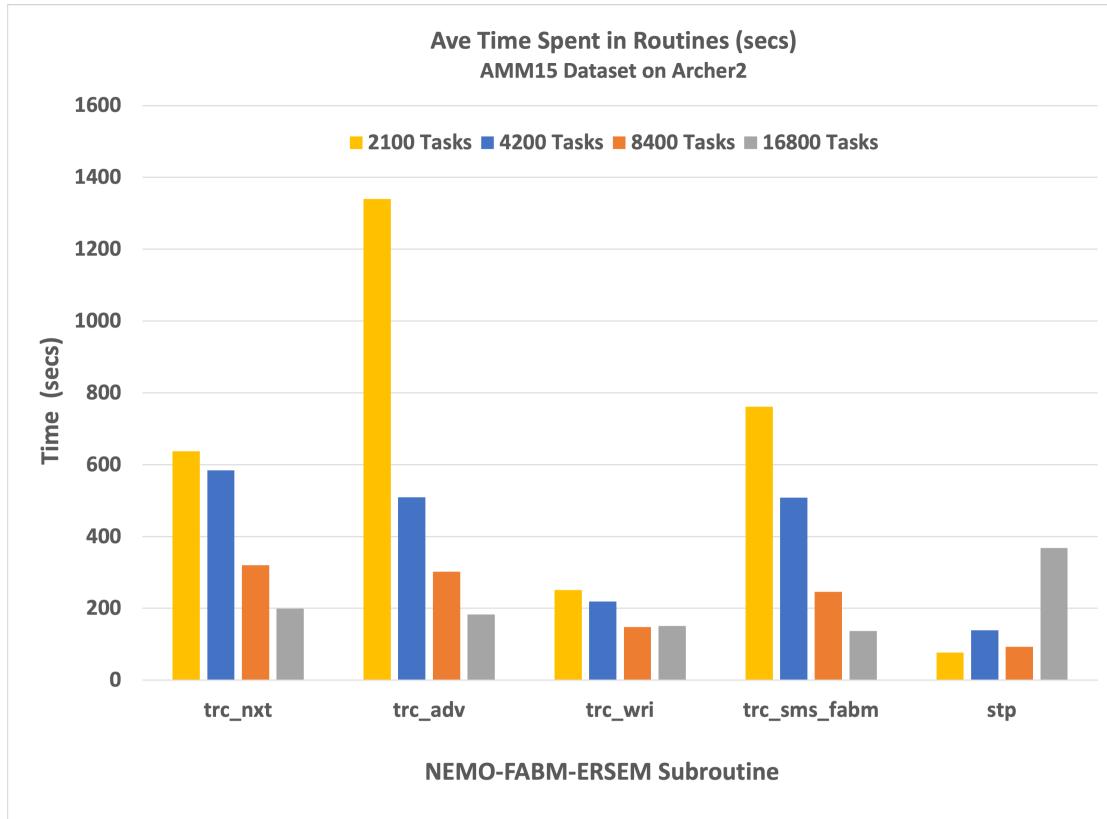


Figure 3 Timing Breakdown by Routine (Average time (s) across MPI processors)

NEMO-FABM-ERSEM can be configured to produce a range of quite detailed performance data at the end of each run. A defined timing framework is available to developers within the code to facilitate these measurements where required. Figure 3 shows the breakdown of timings in seconds from within the top five time-consuming NEMO-FABM-ERSEM subroutines for a range of parallel runs on Archer 2. Figure 4 shows the breakdown of timings for the same subroutines as percentage of overall runtime and also includes data from AMM7 benchmarks. Developers control the scope of the timers with explicit on/off switches and are generally constructed as inclusive timers in that they include time from ‘children’ routines called from within, unless those routines have their own timer switched on.

Details of the subroutines listed are given in Appendix B. Good parallel scaling behaviour per subroutine is represented by descending clusters of columns in Figure 3 and a flat profile of columns within a cluster in Figure 4. It is evident in the two figures that the parallel scaling properties of the main NEMO routines `trc_nxt`, `trc_adv` and the main FABM routine `trc_sms_fabm` is very good for this range of benchmark runs. These subroutines undertake much of the calculation and data halo-exchanges in the calculations and appear to be well optimised. The subroutine `trc_wri` undertakes most of the data outputs to disk, utilising the associated XIOS servers. The parallel scaling for this operation is good out to 8400 tasks, but flatlines at 16800, indicating that the output is no longer scaling with cores used and is now becoming a significant overhead (11% of overall time for 16800 tasks). The subroutine `stp` manages the time stepping for the ocean, tracing and ice. Although it

does not involve high levels of computation or message passing itself, the performance of this routine is a measure of the overall load-balancing within the code. It is shown in Figure 4 that there is a significant increase in the percentage of runtime spent in `stp` from 6.4% for 8400 tasks to 26.9% for 16800 tasks, indicating that the load-balancing of the computation across tasks is deteriorating.

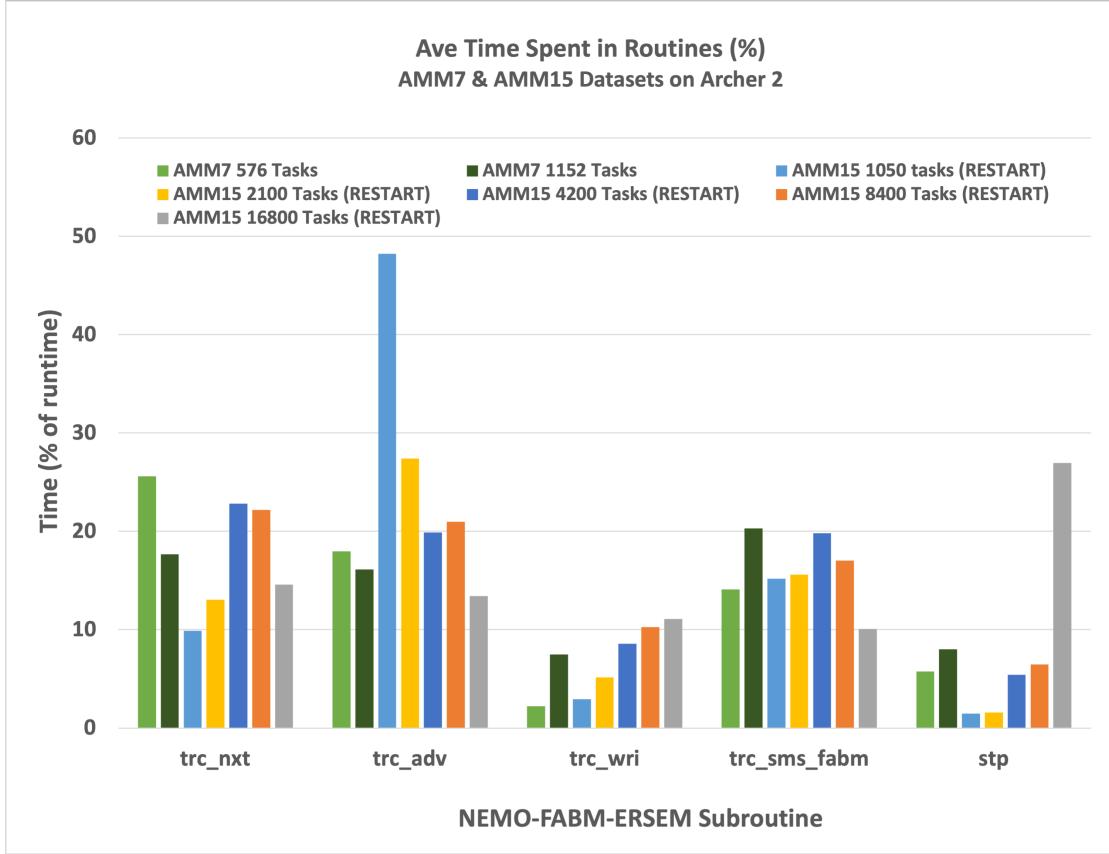


Figure 4 Timing Breakdown by Routine (Average time (%)) across MPI processors)

3.2 Parallel Performance Profiling

3.2.1 Identifying Computational Hotspots with Craypat

A more detailed timing profiling, involving all routines and all source code lines can be obtained by using the Craypat profiler on Archer2. The profiling was undertaken following the guidance in [13] and the profiling report with source code line information was generated via the command

```
$> pat_report -O src+ca <profiling_data_directory>
```

Profiling was only undertaken on the smaller dataset AMM7 due to the large amount of tracing and sampling data generated, even for runs involving reduced numbers of timesteps. Figure 4 shows that the timing distribution is roughly equivalent in both

AMM7 and AMM15 cases. An extract from a typical Craypat profiling report that traces and samples an AMM7 calculation involving two Archer2 compute nodes is provided in Appendix C. Unlike the internal timers in NEMO, timing reports here are *exclusive* – representing only time spent in the subroutine or function itself thereby excluding time spent in any called functions or subroutines.

The profiling identifies the subroutine `tra_adv_fct` (called from `trc_adv`) as the most computationally expensive, consuming 8.6% of overall run time. This is a routine that computes the now trend due to total advection of tracers and adds it to the general trend of tracer equations using a 2nd or 4th order Flux-correction scheme. The source code line identified within subroutine `tra_adv_fct` as a computational hotspot is within the loop:

```

DO jk = 1, jpkm1      !* trend and after field with monotonic scheme
    DO jj = 2, jpjm1
        DO ji = 1, jpi ! vector opt.
            !
            ! total intermediate advective trends
            ztra = - ( zwx(ji,jj,jk) - zwx(ji-1,jj ,jk )  &
            &           + zwy(ji,jj,jk) - zwy(ji ,jj-1,jk )  &
            &           + zwz(ji,jj,jk) - zwz(ji ,jj ,jk+1) ) * r1_e1e2t(ji,jj)
            !
            ! update and guess with monotonic sheme
            pta(ji,jj,jk,jn) = pta(ji,jj,jk,jn) + ztra / e3t_n(ji,jj,jk) * tmask(ji,jj,jk)
            zwi(ji,jj,jk)    = ( e3t_b(ji,jj,jk) * ptb(ji,jj,jk,jn) + p2dt * ztra ) &
            / e3t_a(ji,jj,jk) * tmask(ji,jj,jk)
        END DO
    END DO
END DO

```



The subroutine `tra_adv_fct` involves several such loops involving large multi-dimensional array updates, which already have vectorization optimisations applied via CCP macros to, for example, to enable loop unrolling. The only FABM routine to consume significant compute time is `fabm_work.f90`, which consumed 6.3% of overall runtime.

Upon further analysis, `tra_adv_fct` and several other computationally expensive routines highlighted by the CrayPat profiler (such as `lbclnk.f90`, the routine that manages nearest-neighbour halo exchanges) had evidently been written and tuned with good performance in mind. Any further improvements would have involved work beyond the scope of this project.

3.3 Effect of Passive Tracers at scale

Typically, ERSEM runs with ~52 tracers, with complex interactions between them. Whilst this is the current configuration, in the future more/less complexity could be used which would have a large impact on the runtime, especially for high-resolution models.

One way to analyse the impact of changing the quantity of variables is to convert all the tracers to be independent, or ‘passive’. This is not reflective of what would be done in practice as these variables no longer have any interaction between them, however it enables quickly running the model with any number of variables to see how the runtime scales. Certain parts of the code will now not be performed, but key functions such as I/O still occurs. Task arrangements (see Section 4.1) for the tracer

timing experiments presented in this section use a configuration of one I/O server per Archer2 node, occupying its own NUMA region, and fully packed ocean tasks (g0) across the remaining cores.

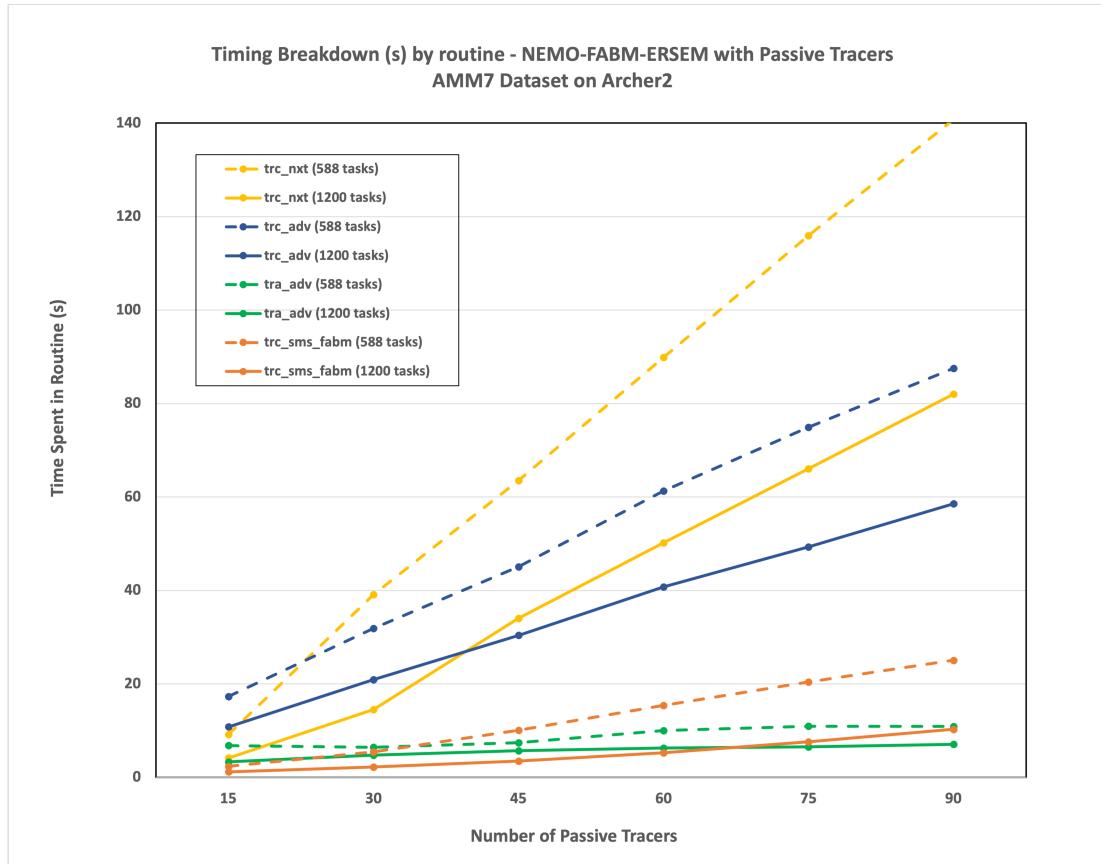


Figure 5 Timing breakdown (secs) by subroutine for 588 & 1200 MPI task NEMO-FABM-ERSEM runs involving varying numbers of passive tracers (AMM7 Dataset)

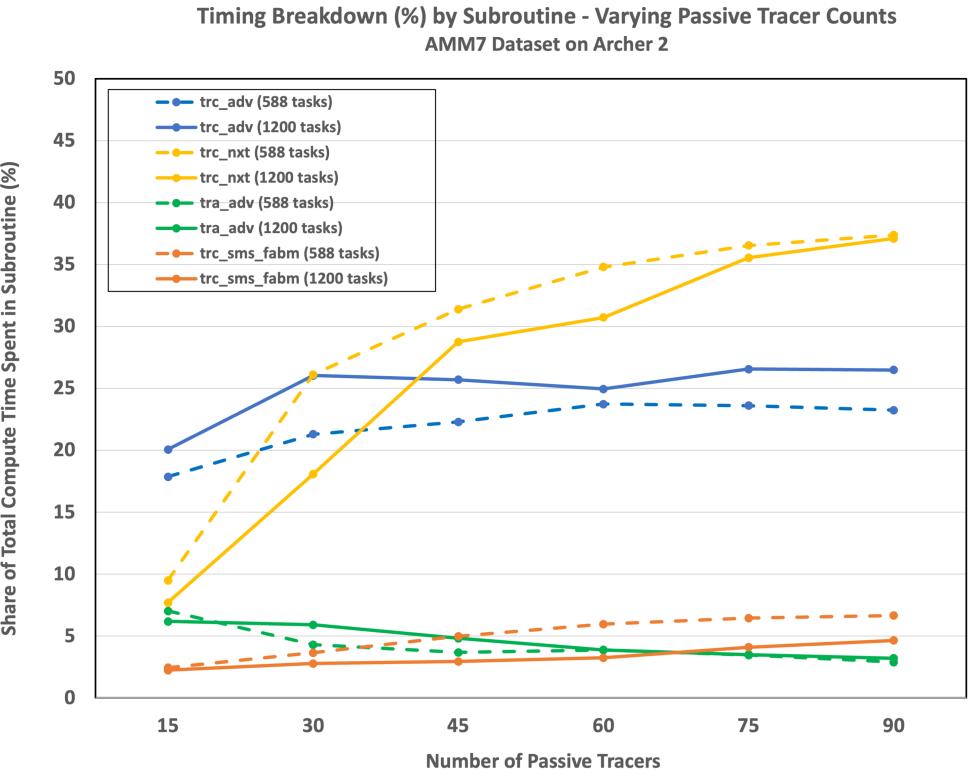


Figure 6 Timing breakdown (%) by subroutine for 588 & 1200 MPI task NEMO-FABM-ERSEM runs involving varying numbers of passive tracers (AMM7 Dataset)

Figure 5 and Figure 6 show the change in the computational load for the top five time-consuming subroutines in NEMO-FABM-ERSEM as the number of **passive** tracers varies between 15 and 90 for AMM7. For details of these subroutines see Appendix B. The timing profile of the matching colours (dashed lines and solid lines) in the figures show that similar timing patterns are obtained with both sizes of parallel MPI task arrays. As expected, the most affected subroutine is `trc_nxt`, which computes passive tracer fields at the next time-step and applies lateral boundary conditions. Time spent in this routine increases from around 8-10% (15 passive tracers) to around 38% (90 passive tracers). Time spent in subroutine `tra_adv`, computing the ocean tracer advection trend, increases the least as the number of passive tracers increases. The percentage of compute time spent in the other subroutines, including the main FABM routine `trc_sms_fabm`, remains relatively constant as the number of passive tracers varies. The percentage of total compute time spent in routine `trc_wri`, which is the most sensitive to I/O overheads, increased from 2.5% (15 passive tracers) to 6.1% (90 passive tracers) on 1200 tasks.

3.4 Effect of Active Tracers at Scale

ERSEMs default set up includes 4 phytoplankton and 3 zooplankton groups, a so called 4P3Z model. The model can be simplified by reducing the number of groups, for example to 3P2Z or 2P1Z which reduces the number of variables from 52 to 45 and 38 respectively. This is a more realistic model simplification than the passive

tracer approach described above. Task arrangements (see Section 4.1) for the tracer timing experiments presented in this section use a configuration of one I/O server per Archer2 node, occupying its own NUMA region, and fully packed ocean tasks (g0) across the remaining cores.

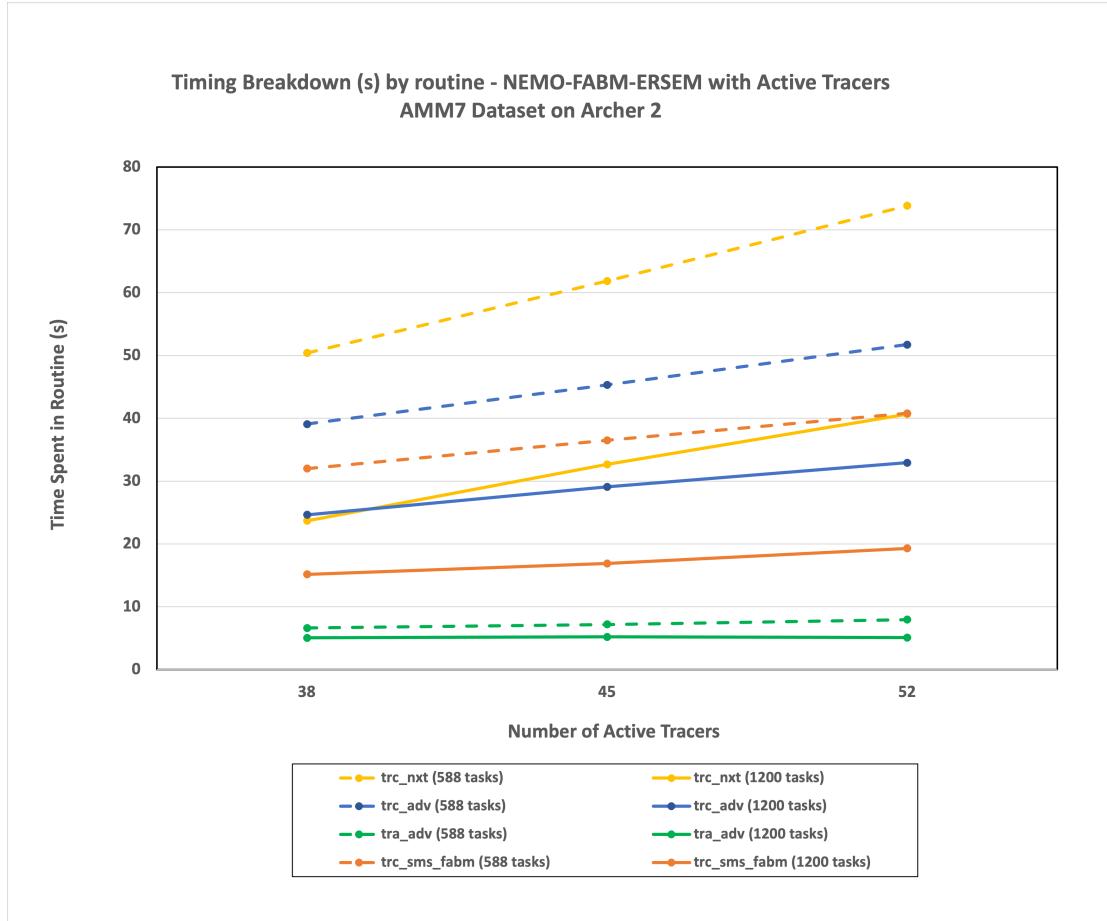


Figure 7 Timing breakdown (secs) by subroutine for 588 & 1200 MPI task NEMO-FABM-ERSEM runs involving varying numbers of active tracers (AMM7 Dataset)

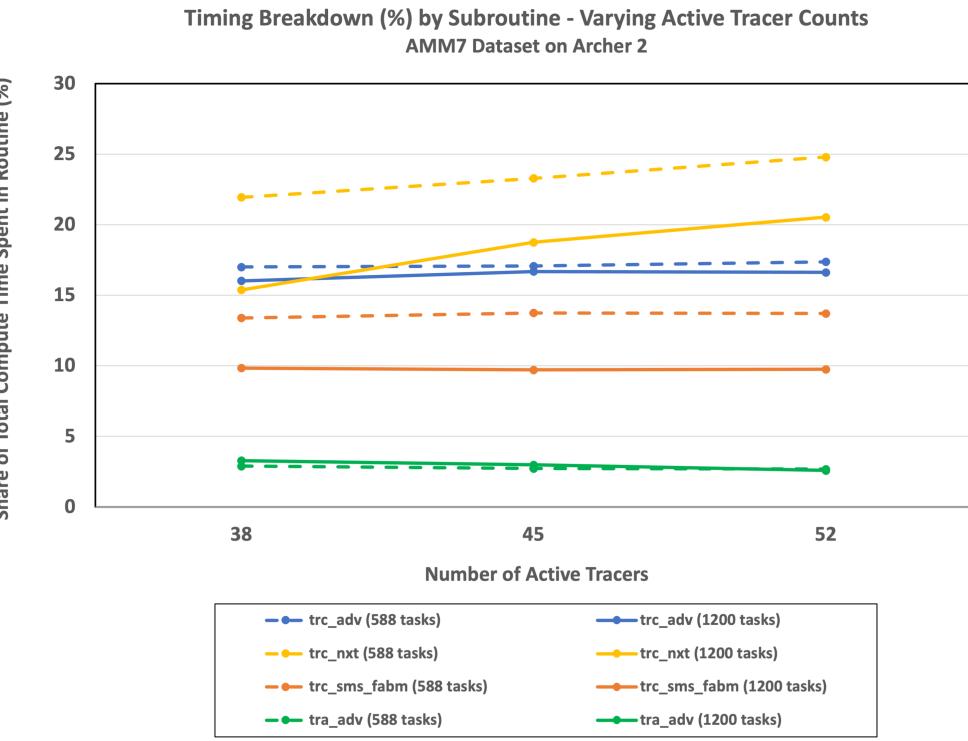


Figure 8 Timing breakdown (%) by subroutine for 588 & 1200 MPI task NEMO-FABM-ERSEM runs involving varying numbers of active tracers (AMM7 Dataset)

Figure 7 and Figure 8 show the change in the computational load for the top five time-consuming subroutines in NEMO-FABM-ERSEM as the number of **active** tracers varies between 38 and 52 (default) for AMM7. The relative timing breakdown between subroutines for both sets of parallel runs is similar to that obtained from the passive tracer experiments in the previous section, though the time spent in the FABM main routine `trc_sms_fabm` now increases to around 10-14%. The percentage of total runtime spent in routine `trc_wri`, which is the most sensitive to I/O overheads, remained roughly constant at around 6% for all the AMM7 active tracer runs involving 1200 MPI tasks.

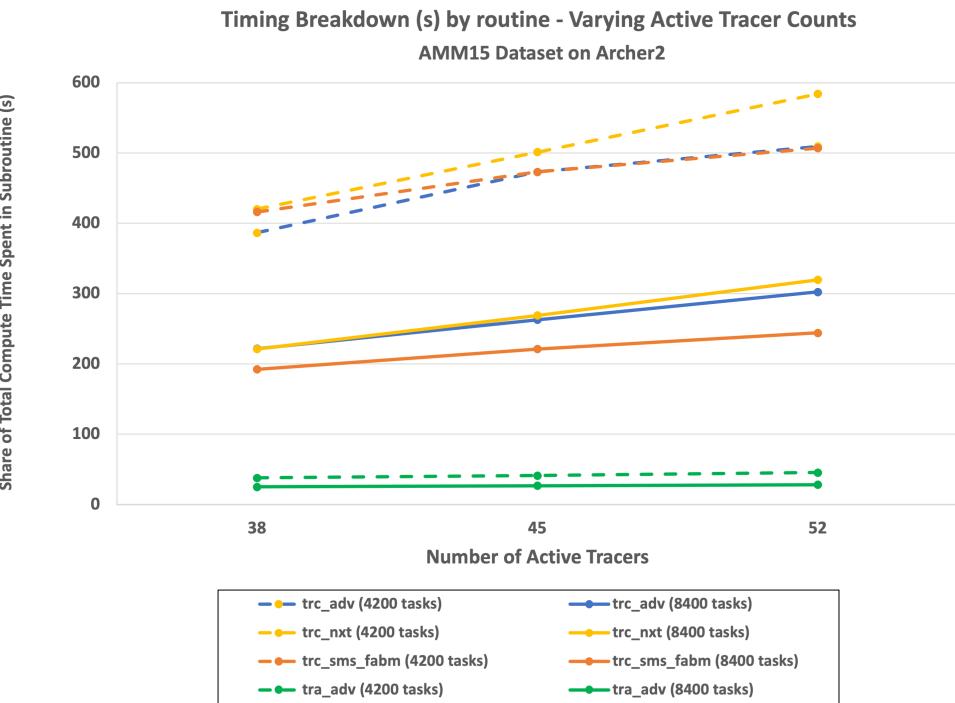


Figure 9 Timing breakdown (secs) by subroutine for 4200 & 8400 MPI task NEMO-FABM-ERSEM runs involving varying numbers of active tracers (AMM15 Dataset)

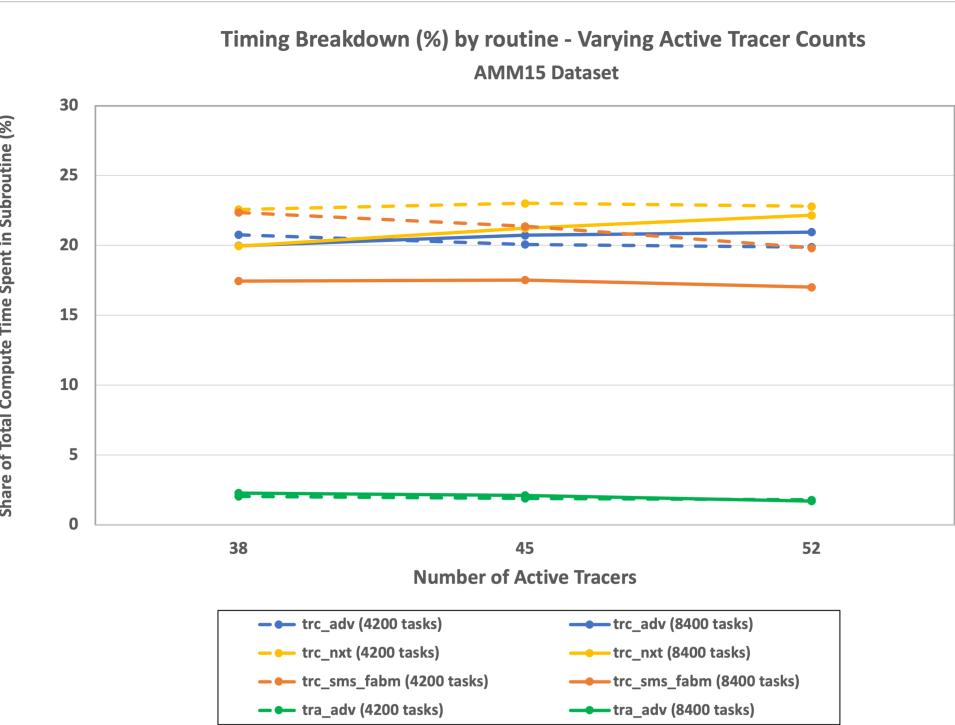


Figure 10 Timing breakdown (%) by subroutine for 4200 & 8400 MPI task NEMO-FABM-ERSEM runs involving varying numbers of active tracers (AMM15 Dataset)

The change in timing breakdowns in NEMO-FABM-ERSEM when varying the number of **active** tracers for the significantly larger dataset AMM15 is shown in Figure 9 and Figure 10. A flatter profile of timing variance than with AMM7 is observed here for both the parallel job sizes, with percentage time spent in each subroutine remaining roughly the same regardless of the number of active tracers. This is likely due to AMM15 having better load balance at the core counts used, as shown in Figure 2. The percentage time spent in the FABM main routine `trc_sms_fabm` increases to around 17-23% for AMM7, but this relative load slightly decreases as the number of active tracers increases to 52 (default). The percentage of total runtime spent in routine `trc_wri`, which is the most sensitive to I/O overheads, increased from 8.9% (38 active tracers) to 10.3% (52 active tracers).

3.5 Load-balance variation with node count

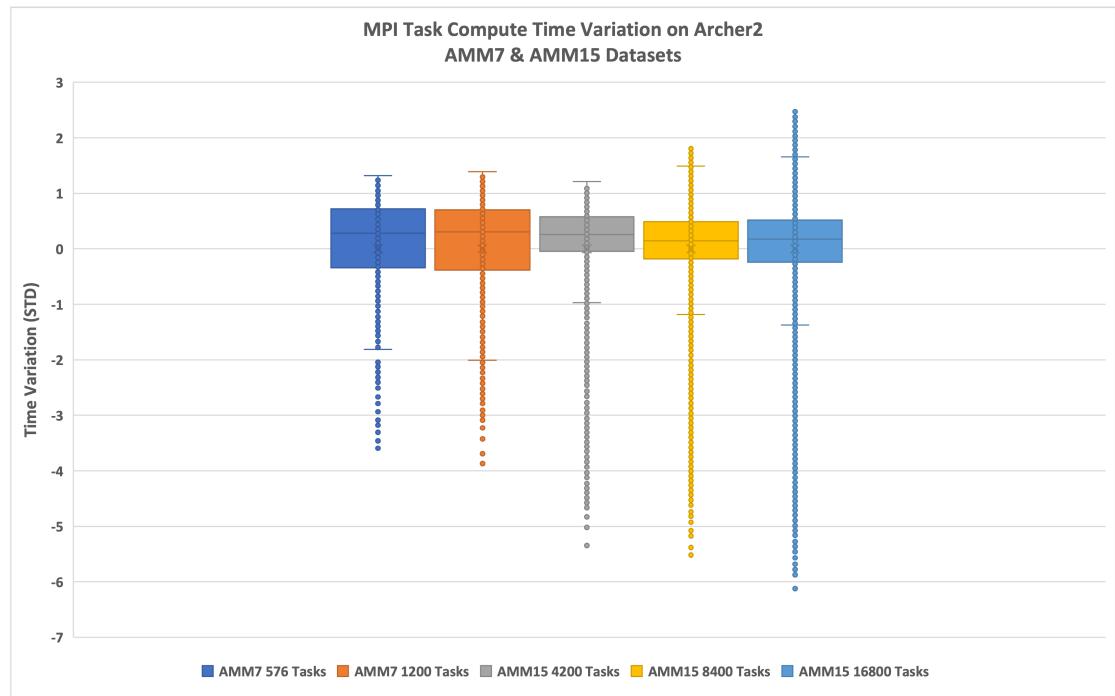


Figure 11 Variation of NEMO-FABM-ERSEM MPI task compute times (AMM7 & AMM15).

The parallel scaling performance of NEMO-FABM-ERSEM on Archer2 is reported and discussed in Section 3.1. For the larger AMM15 dataset, the parallel speed-up starts to decline at runs involving 16800 cores on 200 compute nodes. Box plots summarising the variance of individual MPI task compute times within each parallel run are shown in Figure 11. Multiple lower outlier points (less than 3/2 of lower quartile) are characteristic of all the box plots, reflecting a greater number of land-based points within a sub-domain. The most extreme of these outliers represent a small set of generally idle MPI tasks which have been allocated entirely land-based sub-domains,

as all available ocean sub-domains have been allocated to other tasks. Compute time variance increases with increasing core counts and is greatest in the AMM15 run on 16800 tasks. A combination of this greater load-imbalance across the processor array and an increased communication vs computation ratio for smaller sub-domain sizes, discussed in Section 3.1, is responsible for the decreased rate of speed-up observed between 8400 and 16800 cores for AMM15 (Figure 2).

4 Optimisations of NEMO-FABM-ERSEM on Archer2

4.1 Optimal Configurations on Archer2

4.1.1 XIOS Server Configuration

Typical NEMO production runs perform significant I/O management to handle the very large volumes of data associated with ocean modelling. To facilitate this, NEMO ocean clients are interfaced with XIOS I/O servers. XIOS is a library which manages NetCDF outputs for climate models. NEMO uses XIOS to simplify the I/O management and allow the configuration of dedicated processors to manage the large volumes of data. NEMO-FABM-ERSEM large-scale parallel runs analysed in this project are undertaken in *detached* mode, where ocean clients and external XIOS I/O server processors are separately defined within the job submission script. The analysis of I/O overheads reported in the NEMO-FABM-ERSEM runs for both the AMM7 and AMM15 datasets confirmed that the general rule-of-thumb of 1 dedicated XIOS server per Archer2 node, exclusively occupying a dedicated 16 core NUMA region was the optimal I/O configuration.

4.1.2 Ocean Client gap intervals

NEMO-FABM-ERSEM users can also define an *ocean client server gap* (known here as g) to increase the compute resources available to an individual MPI task on a compute node, particularly memory capacity and memory bandwidth. Three levels of under-populating are analysed – g0 (no gaps), g2 (gap every 2 cores) and g4 (gap every 4 cores). As expected, shown in Figure 12, the highest level of underpopulation, g2, leads to significantly faster performance per core and the full occupation g0 the slowest.

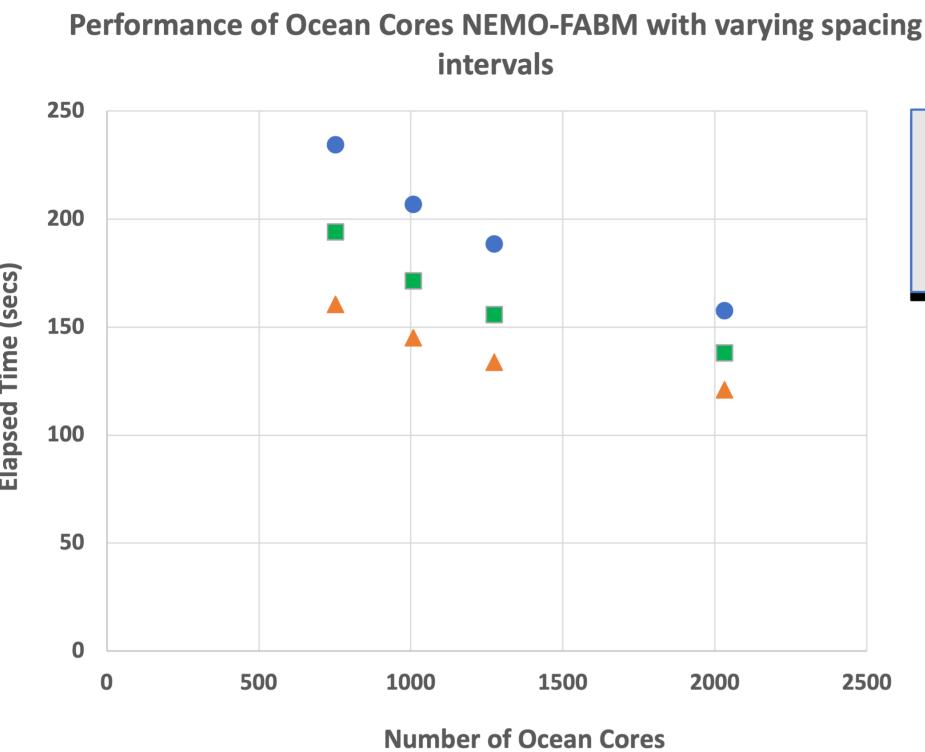


Figure 12 Timing experiments with different ocean client spacings on Archer 2 (AMM7 dataset)

However, if overall resource cost, rather than speed is the primary concern, then the fully occupied g0 ocean client configuration is preferable, as shown in Figure 13, particularly on larger numbers of compute nodes.

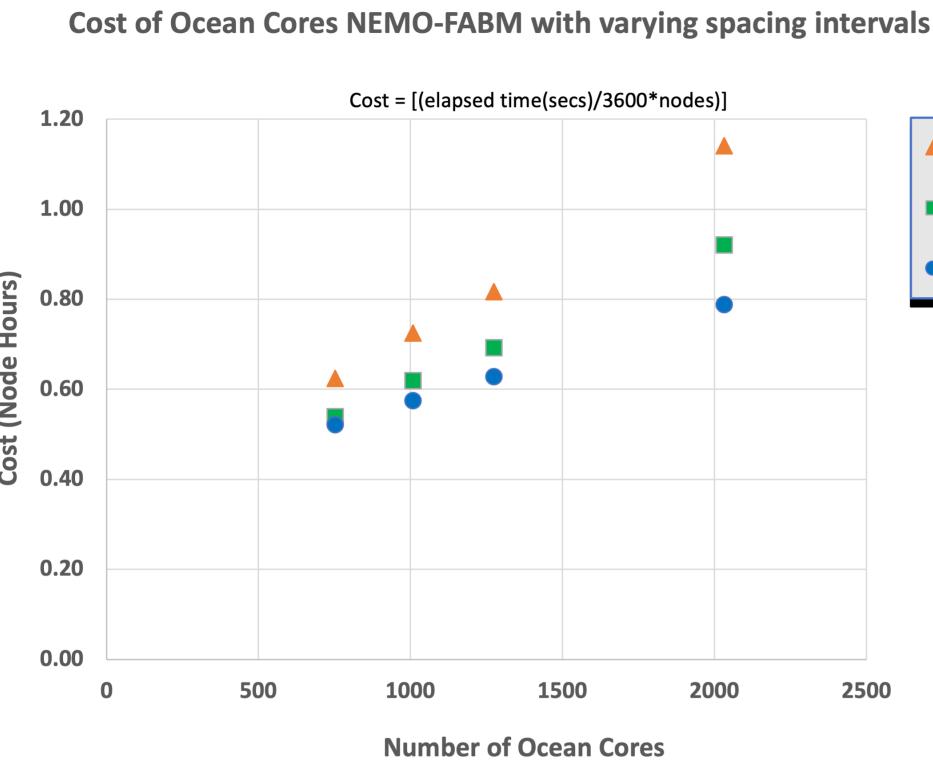


Figure 13 Cost analysis with different ocean client spacings (g) on Archer 2 (AMM7 dataset)

Another consideration when configuring runs is the overall memory requirements of a particular dataset. For example, the AMM7 dataset required a minimum of two half-populated Archer 2 nodes to provide enough memory capacity for the calculation. AMM7 runs involving one node fully populated with the same overall number of ocean clients failed due to memory limitations. Each standard memory Archer node contains 256 GiB of RAM.

4.2 I/O Performance

NEMO-FABM-ERSEM calculations typically require tens of gigabytes of input data and can produce hundreds of gigabytes of output data. Individual file sizes of around 60GBytes are output by the code during benchmark runs for AMM15. Dealing with this large volume of data efficiently during a large-scale parallel run is the main impetus for configuring dedicated XIOS servers on each node. On Archer2 there are several settings affecting the behaviour of the Lustre filesystem that can be applied to runs which can potentially improve I/O performance.

4.2.1 Striping I/O Across Object Storage Targets (OSTs)

On Archer2, large datafiles must be *striped* across multiple OSTs to benefit from the parallel nature of Lustre. To be generally efficient for a wide range of job sizes and characteristics, the default settings on Archer2 do not maximise the use of all available use OSTs. However, the large-scale, data intense nature of NEMO-FABM-ERSEM calculations necessitate the use of all available OSTs where possible. This can be achieved by changing the properties of the directories where I/O takes place on Lustre via the setting

```
$> lfs setstripe -c -1 <directory name>
```

A detailed description of striping can be found in the Archer2 documentation [23].

4.2.2 I/O Environment Settings

It is also recommended to instruct the running application to use collective calls when undertaking parallel I/O to a single shared file (as is typically the case with these runs). This can be achieved via the environment setting

```
$> export FI_OFI_RXM_SAR_LIMIT=64K
```

With both the environmental and striping I/O settings applied there is a noticeable improvement in the overall performance of NEMO-FABM-ERSEM. Table 2 compares the timing results from default I/O settings and optimised I/O for AMM15 on Archer 2. The relative parallel I/O performance gains become more notable as the number of cores used increases.

Number of Compute Nodes	Number of Ocean Cores	Runtime (Default I/O settings) (s)	Runtime (Optimised I/O settings) (s)	% Performance Improvement from I/O settings
12	1008	9887	9844	0.44
25	2100	3929	3877	1.34
50	4200	2327	2271	2.47
100	8400	1302	1253	3.91
200	16800	898	868	3.46

Table 2 Relative performance of AMM15 calculations with I/O settings on/off.

4.3 FABM-specific settings

Users can specify a range of FABM variables to be applied during the preprocessing stage [14]. The most relevant of these settings for performance optimization are `_FABM_MASK_TYPE_` and `_FABM_CONTIGUOUS_`.

Specifying `_FABM_MASK_TYPE_` explicitly sets the data type of the FABM spatial mask. For optimal performance this should match that used internally by the physical host to specify the mask, thereby allowing the mask to be transferred via simple pointer operations rather than data copies. `_FABM_MASK_TYPE_` is set to different data types in the miscellaneous FABM source code drivers.

The `_FABM_CONTIGUOUS_` variable specifies that all arrays passed to FABM will be contiguous in memory. Specifying this allows the compiler to make certain assumptions that can be used to optimize array operations.

4.4 Optimal Energy Consumption on Archer2

Users on Archer2 can control the CPU frequencies of the compute nodes assigned to their batch jobs by setting the environment variable `SLURM_CPU_FREQ_REQ`. For example, to set the CPU frequency to 2.25GHz the following line would be included in the batch script

```
export SLURM_CPU_FREQ_REQ = 2250000
```

Only three different CPU frequencies are permitted – 2.25GHz, 2.0GHz (the default since Dec 2022) and 1.5 GHz. The impact of varying the CPU frequency on performance and energy consumption for NEMO-FABM-ERSEM with the AMM15 dataset on 50 compute nodes (with g0 ocean core spacing) is reported in Table 3.

CPU Frequency (GHz)	Elapsed Time (s)	Consumed Energy (MJ)
1.5	3330	52.29
2.0	2781	58.11
2.25	2326	61.77

Table 3 Timing and energy consumption measurements for AMM15 dataset on 50 compute nodes.

The data from Table 3 shows that the fastest run is, as expected, associated with the CPU clock frequency of 2.5GHz. However, if energy efficiency is prioritised over speed, the lowest CPU frequency of 1.5 GHz is the optimal setting.

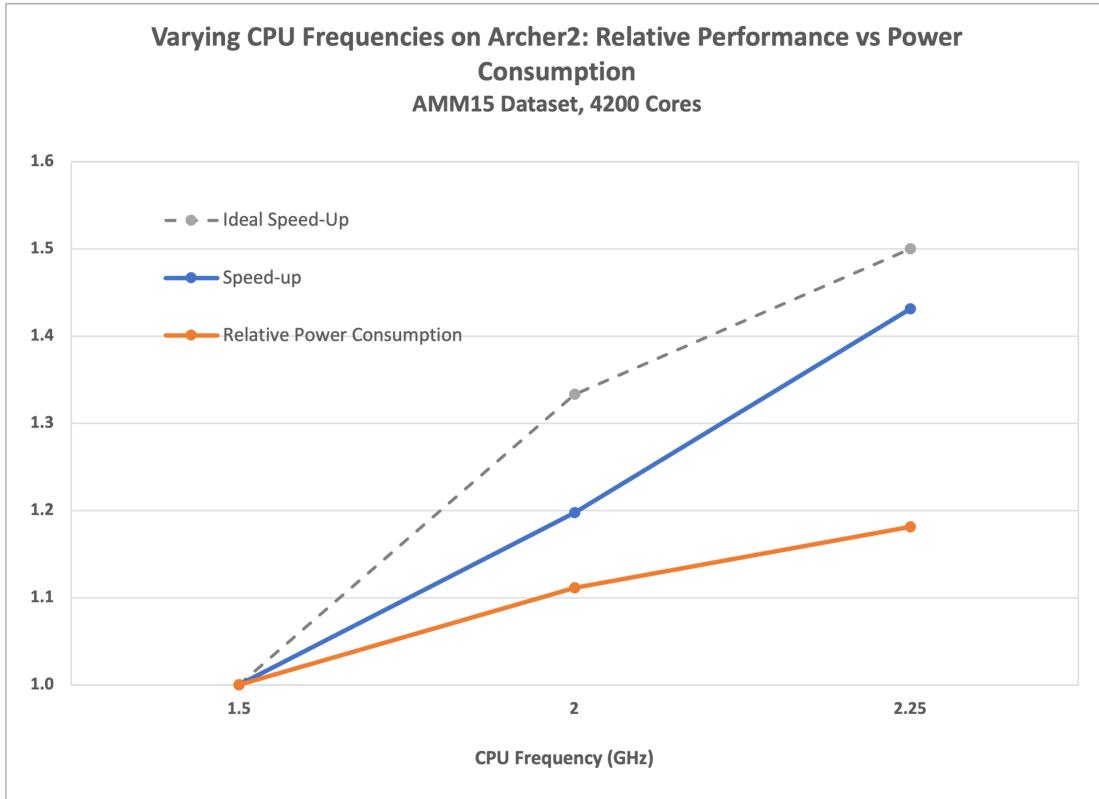


Figure 14 Summary of NEMO-FABM-ERSEM performance and energy consumption with varying CPU frequencies

5 Conclusions

This project has successfully installed the latest version of NEMO-FABM-ERSEM on Archer2 to run the next-generation Atlantic Margin Model AMM15 with an increased grid resolution of 1.5km. The parallel performance of the code, particularly with this larger dataset, is generally very good and large-scale runs using tens of thousands of cores allow simulations to be undertaken relatively quickly, whilst handling the large volume of output data efficiently. Large-scale profiling analysis of the code performance on Archer2 using Craypat has been undertaken and computational hotspots in the source code have been identified and investigated. Detailed internal timing breakdowns are presented that demonstrate how the computational load varies within the code with core count for both AMM7 and AMM15, including varying the numbers of passive and active tracers. Alternative task-to-core placement configurations within a compute node have been investigated for optimising both performance and resource costs on Archer2 and recommended settings have been determined. Good parallel I/O performance is highly important for dealing with datasets of this size. To achieve this, optimal settings for striping across processors and other I/O environment settings are provided. Finally, an analysis of overall energy consumption of the code is provided to help users prioritise either optimal performance or optimal energy efficiency.

6 Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (<http://www.archer2.ac.uk>).

7 References

1. *Plymouth Marine Laboratory*, <https://pml.ac.uk>.
2. *National Oceanography Centre*, <https://noc.ac.uk>.
3. *Met Office*, <https://www.metoffice.gov.uk>.
4. *NEMO Community Ocean Model*, <https://www.nemo-ocean.eu>.
5. *NEMO Ocean Engine*, Madec et al., 2019, doi: 10.5281/zenodo.3878122.
6. *A general framework for aquatic biogeochemical models*, Bruggeman & Bolding 2014. doi:10.1016/j.envsoft.2014.04.002.
7. <https://github.com/pmlmodelling/NEMO4.0-FABM>.
8. *ERSEM 15.06: a generic model for marine biogeochemistry and the ecosystem dynamics of the lower trophic levels*, Butenschön et al, 2016. doi: 10.5194/gmd-9-1293-2016.
9. XIOS Library : XML-IO-SERVER,
<https://mars3d.ifremer.fr/docs/doc.coupling.xios.html>.
10. <http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branches/xios-2.5/>.
11. *NEMO Archer2 User Documentation*, <https://docs.archer2.ac.uk/research-software/nemo/>.
12. *AMM15: a new high-resolution NEMO configuration for operational simulation of the European north-west shelf*, Graham et al., 2017, doi: 10.5194/gmd-11-681-2018.
13. Archer2 User Documentation, <https://docs.archer2.ac.uk/user-guide>.
14. *Using FABM from a physical model*, <https://github.com/fabm-model/fabm/wiki/Using-FABM-from-a-physical-model>.

Appendices

Appendix A

Archer2 SLURM job submission script example, created by the script `mkslurm_hetjob`.

Appendix B

The six main routines consuming compute time in NEMO-FABM-ERSEM.

Subroutine	trc_adv
Location	TOP/TRP/trcadv.f90
Origin	NEMO Source Code
Purpose	Compute the ocean tracer advection trend
Method	Update after tracers (tra) with the advection term following nadv

Subroutine	trc_nxt
Location	TOP/TRP/trcnxt.f90
Origin	NEMO Source code
Purpose	Compute the passive tracers fields at the next time-step from their temporal trends and swap the fields.
Method	Apply lateral boundary conditions on (ua,va) through a call to lbc_lnk routine

Subroutine	trc_wri
Location	TOP/TRP/trcwri.f90
Origin	NEMO Source Code
Purpose	Output passive tracers fields and dynamical trends

Subroutine	trc_sms_fabm
Location	TOP/FABM/trcsms_fabm.f90
Origin	FABM Source Code
Purpose	FABM Main Routine

Subroutine	stp
Location	OCE/step.f90

Origin	NEMO Source Code
Purpose:	Manager of the ocean, tracer and ice time stepping

Subroutine	tra_adv
Location	OCE/TRA/traadv.f90
Origin	NEMO Source Code
Purpose:	Compute the ocean tracer advection trend
Method:	Update (ua,va) with the advection term following nadv

Appendix C

Craypat profiling example for NEMO-FABM-ERSEM.

Table 1: Profile by Group, Function, and Line

Samp%	Samp	Imb.	Imb.	Group
		Samp	Samp%	Function
				Source
				Line
				PE=HIDE
100.0%	15,973.3	--	--	Total
-				
65.0%	10,377.7	--	--	USER
-				
8.6%	1,374.1	--	--	tra_adv_fct\$traadv_fct_
3				BLD/ppsrc/nemo/traadv_fct.f90
4 1.5%	246.6	81.4	25.1%	line.212
8.3%	1,318.5	--	--	nonosc\$traadv_fct_
3				BLD/ppsrc/nemo/traadv_fct.f90
-				
4 1.1%	176.0	44.0	20.2%	line.533
4 1.2%	190.0	41.0	17.9%	line.545
=				
6.3%	1,004.4	--	--	begin_interior_task\$fabm_work_
3				code/fabm/src/fabm_work.F90
-				
4 3.7%	598.7	298.3	33.6%	line.289
4 1.8%	285.9	112.1	28.5%	line.295
=				
5.5%	875.3	--	--	mpp_lnk_4d_ptr\$lbclnk_
3				BLD/ppsrc/nemo/lbclnk.f90
4.1%	662.7	--	--	tra_nxt_vvl\$tranxt_
3				BLD/ppsrc/nemo/tranxt.f90
4 1.3%	209.4	31.6	13.3%	line.362
2.7%	424.6	--	--	tra_zdf_imp\$trazdf_
3				BLD/ppsrc/nemo/trazdf.f90
4 1.3%	201.3	24.7	11.1%	line.269
2.5%	395.9	--	--	mpp_lnk_3d_ptr\$lbclnk_
3				BLD/ppsrc/nemo/lbclnk.f90
1.7%	277.4	--	--	do\$ersem_microzooplankton_
3				code/ersem/src/microzooplankton.F90
1.6%	251.8	--	--	trc_rad_sms\$trcrad_
3				BLD/ppsrc/nemo/trcrad.f90
1.2%	196.6	--	--	do\$ersem_carbonate_
3				code/ersem/src/carbonate.F90
4 1.1%	180.0	90.0	33.7%	line.187
1.1%	172.4	--	--	std::__cxx11::basic_string<>::size
3				include/g++/bits/stl_algobase.h
4				line.200
1.1%	170.4	--	--	mpp_lnk_4d\$lbclnk_
3				BLD/ppsrc/nemo/lbclnk.f90
1.0%	160.6	--	--	blitz::Array<>::operator()
=				
15.3%	2,446.7	--	--	MPI
-				
12.7%	2,024.0	4,161.0	68.0%	MPI_RECV
1.2%	194.5	2,000.5	92.1%	MPI_WAIT
=				
12.0%	1,919.0	--	--	ETC
-				
3.0%	485.2	41.8	8.0%	__memcmp_sse4_1
2.4%	386.9	97.1	20.3%	__cray_memset_ROME
1.4%	224.8	--	--	__cray_RTOR_V_01
3				craylibs/fast_mv/AVX2/__cray_RTOR_V_01.S
=				
6.2%	985.4	--	--	HDF5
-				