eCSE Technical Report on work for ARCHER2-eCSE02-6

eCSE ID:	ARCHER2-eCSE02-6
eCSE Title:	Optimising MITgcm on ARCHER2: efficient numerical simulation and data assimilation tools for studying the ocean, atmosphere, and cryosphere.
PI Name:	Dr Emma JD Boland
Author of this document:	Dr Emma JD Boland, Dr Mike Mineter
Name of technical staff on the project:	Dr Mike Mineter
List of names to acknowledge with the project (by default we will use the PI, Co-Is and Technical staff members):	Dr Emma JD Boland, Dr Dan(i) Jones, Dr Kaitlin A Naughten, Dr Daniel N Goldberg, Dr Mike Mineter

Abstract

MITgcm is a freely available, community-driven, open-source general circulation model designed for the study of the atmosphere, ocean, cryosphere, and climate. It provides the core software upon which specific models are built to run in forward or adjoint modes. Three were ported to Archer2, validated and optimised. Significant speedup and reductions in total node-hours were achieved by fully utilising the cores on each node to the extent that the need to divide the total grid into sub-grids permitted. The AMUNDICE adjoint case was run with a containerised OpenAd, developed in response to this project, and achieved 3-fold speedup by use of the PETSc library. These three exemplars of MITgcm are documented on the Archer2 website, supplemented by a project GitHub repository. Updates have been incorporated into code repositories for MITgcm and the models.

1. Introduction

MITgcm [1] is a freely available, community-driven, open-source general circulation model designed for the study of the atmosphere, ocean, cryosphere, and climate. It provides the core software for hydrodynamics upon which specific models are built. Of these 3 were ported to Archer2, validated and optimised. PAS (Projections of the Amundsen Sea [2] is a regional, high-resolution configuration of the Amundsen Sea region of Antarctica, including ice shelf thermodynamics and sea ice. AmundIce adds to MITgcm a dynamic land-ice package, Streamice [3] to form a dynamic flow model of the Amundsen region. ECCOv4 [4] is a global ocean state estimate on a 5-sided lat-lon-cube grid. It is considered one of the benchmark data assimilation products in oceanography.

These exemplar models are built from the MITgcm Fortran code with additional functions and packages specific to each model. For AmundIce, the PETSc package [5] was added. For adjoint modelling an automatic differentiation tool is used: TAF [6] or OpenAD [7]. MITgcm provides scripts to enable the building of the model. These were modified by the project to add support for a containerised OpenAd.

Each build uses an “opt file” to specify the compilation flags used. The opt file has options for ieee compliance, and for faster maths. The opt file is used by the MITgcm script genmake2 which generates and runs a Makefile. The Cray optfile is the one being formally supported from this work by integration with the MITgcm distribution. GNU Fortran cases performed similarly well for most cases. The GNU compiler is used with AmundIce as explained below.

The porting to Archer2 benefitted from the 2016 eCSE project that had ported MITgcm to Archer [8]. The report from that project, a more extended format than is required for current eCSE projects, describes the software in more detail. We would also like to acknowledge the work of William Lucas, who carried out the initial porting of MITgcm onto ARCHER2.

2. Methodology

The different models use different checkpoints of the MITgcm software, as shown in table 1.

Table 1 MITgcm releases that were used in the porting

	MITgcm checkpoint
ECCOv4	66g
PAS	67s
AmundIce	https://github.com/dngoldberg/MITgcm/tree/streamic_petsc_3_8_update Dan Goldberg's repository, submitted for merging, branched from 68i

First, the standard MITgcm verification tests were run to test the port itself (see section 4). Then the same approach was taken for each of the three specific setups:

1. Build, run and validate an instance of the model to replicate a trusted model for which we had outputs from Archer. We used the same number of cores as had been used on Archer with non-optimised (ieee) compilation flags.
2. Estimate the optimal numbers of nodes and tasks/node, taking account of the 128 cores/node and the costs being per node, not per core so that the total node-hours is also a criterion to be used. This entails checking the memory needed in the initial benchmark (using SLURM sstat and sacct commands). In the chosen cases memory was more than sufficient for 120 processes/node, but caching complicates the picture.
3. Trial alternative compilation flags with this estimated number of nodes/tasks, for a shorter run.
4. Check alternative plausible numbers of nodes, also testing alternatives for the srun distribution flags. These determine how srun and SLURM allocate consecutive MPI processes across nodes (1st option) then across NUMA regions of each node (link to Archer2 documentation). This confirmed that the initial choice of "block:block" is preferred usually – but not always (e.g. a case in ECCO below).
5. Rerun the optimised benchmark case as a final validation and measure of performance
6. In the performance tests at least 2 runs of each viable configuration were made and the quickest times used.

6.1.Additional Tests

Tests were run to explore the alternative less-used MPI layer UCX with both PAS and ECCOv4. This gave a tiny advantage in some cases, but slight deterioration in general and at time of testing UCX was closer to the bleeding edge. Multithreading tests with PAS gave rise to obscure errors and this was not pursued.

6.2.Test Phases

The work was done in two phases. The first phase installed, did runs using aggressive optimisation, checked results, and then made the software available on the 4-cabinet system. We then paused until the full system was available to do further runs to explore alternative compilation flags and allocation of processes to nodes. We had also hoped to have the flash filesystem available but this is still awaited by Archer2. The phasing also ensured the goal that the full Archer2 system can readily be used but disrupted momentum and required some duplication of effort.

Table 2 System software used on 4-cabinet and full systems

	full system craype/2.7.6 cray-mpich/8.1.4 libfabric/1.11.0.4.71 xpmem/ 2.2.40-7.0.1.0_2.7__g1d7a24d.shasta cray-libsci/21.04.1.1	**4 cabinet** craype/2.7.2 cray-mpich/8.0.16 libfabric/1.11.0.0.233 cray-dsmml/0.1.2 xpmem/2.2.35-7.0.1.0_1.9__gd50fabf.shasta cray-libsci/20.10.1.2
GNU	gcc/10.2	gcc/10.1.0
Cray	Cray Fortran : Version 11.0.4	Cray Fortran : Version 10.0.4

Runtimes with the gnu compiler were comparable to those with Cray's Fortran. We note that namelist terminators need to be "&" with Cray and "/" with gnu.

7. Documentation

The build and use of MITgcm and the specific models are documented:

- Where appropriate in the MITgcm documentation [9].
- On the Archer2 research software site [10].
- With additional information on the project's GitHub site. (https://github.com/eCSE-MITgcm-ARCHER2). This holds code, further user documentation, input data. This will also be a resource for further development and enhancement.

8. Verification tests

Standard MITgcm verification tests [11] were run to test the MITgcm port itself using the checkpoint 66g, achieving results deemed satisfactory both with single processor and the MPI runs with the default 2 cores, see table 3. The test of the port had already reproduced the expected benchmark results with PAS and ECCOv4, so these were also taken as indicative of a successful basis port of the MPI-enabled models. The standard verification tests were run using the Cray compiler with the MITgcm *testreport* script.

Table 3 Summary of validation test results with Cray Fortran

	Number of passes	Number of fails
Serial, optimised (4c system)	73	17
MPI (2 processes), ieee	80	5

In the MPI case, 4 of the 5 failures were due to the precision with which cg2d agreed with test data, of these only one case having fewer than 7 significant figures of agreement (lab_sea.hb87). The difference in total numbers of tests is due to some tests not supporting MPI.

A summary of the tests and associated the associated SLURM scripts are held at [12].

9. PAS (Projections of the Amundsen Sea)

Routine changes were made to PAS scripts to replace Archer's PBS with SLURM, including to enable chained jobs, where on completion of one job the next is queued to achieve long runs. (This is currently achieved with a timeout that interrupts a job 3 minutes before the job's time expires. It

was noted that Archer2/SLURM defaults to allowing 30 seconds to handle traps, so there might be some scope for simplifying this script – that was untested)

Failures at runtime led to two simple changes in code being applied to the PAS code. Minor amendments were made to scripts. The serial job for generating NetCDF requires –mem=4G in order that a chained PAS job is successful.

Runs of the PAS benchmark produced 12 months of data, see table 4. Monthly domain-averaged ocean temperatures, matching to 6 decimal places, were taken as the indicator of correctness.

Table 4 Benchmarking runs of PAS_053 (the "ieee" runs are unoptimised. Cce demotes the Cray compiler was used)

	Archer ieee	Archer2, cce ieee	Archer2 cce ieee	Archer2 cce optimised	Archer2 cce optimised	Archer2 gnu	Archer2-4c gnu	Archer2 gnu
Number of cores	192	192	240	192	240	192	240	240
Number of nodes	8	2	2	2	2	2	2	2
Tasks/node	16	96	120	96	120	96	120	120
Flags	-O0 -hfp0	-O0 -hfp0	-O0 -hfp0	-O3 -hfp3 -Oipa5	-O3 -hfp3 -Oipa5	-O3 -funroll-loops	-O3 -funroll-loops	"-O3 -funroll-loops"
Runtime sec.	9102	13391	10768	7332	5288	6493	5624	5352
Node-hours	-	7.4	6.0	4.1	2.9	3.6	3.1	3.0

Runs on two nodes with 240 cores were about 10% faster than those with 192 cores, using optimised code: here the advantage from parallel computation outweighed the cost of caching refreshes and MPI. The minimum node hours indicates the cheapest run, with budgets and energy in mind. Short runs were carried out to compare performance with different compilation flags, numbers of nodes, cores, see table 5. The speedup is shown for the fastest runtime for 120, 240 and 360 cores, with orange highlighting the differences between the columns.

Additional tests found the srun distribution block:block was found to be preferable.

A trial of multithreading with 3 nodes, 30 processes on each and 4 threads per process with each process using 4 cores was tried but without success, the job failing in ways noted in (PAS git). The rationale for hybrid (multimode with some OpenMP) is that the overhead of MPI and perhaps sequential parts of the code, might be reduced and benefit cases where scalability of MPI in this case is seen to reduce. However a concern is that additional validation beyond time available, would be needed to ensure all outputs are correct so this was not prioritised.

The code and scripts with these changes applied are in [13] and are now maintained in [14] by K Naughten.

Table 5 PAS: Effect of compilation flags on shorter runs

	cce	cce	cce	cce	cce	cce	gnu
Number of cores	120	240	240	240	240	360	240
Number of nodes	1	2	2	2	3	3	2
Tasks/node	120	120	120	120	80	120	120
Flags	-O3 -hfp3 -Oipa5	-O3 -hfp3 -Oipa5	-O3 -hfp3 –Oipa4	-O2 -hfp3 –Oipa5	-O3 -hfp3 -Oipa5	-O3 -hfp3 -Oipa5	-Ofast -funroll-loops
Runtime sec.	2985	1723	1909	1910	1775	1564	1744
Speed-up compared to 120 cores/1node	1	1.73				1.91	
Node hours	0.8	0.96				1.3	

10.AmundIce

MITgcm and its STREAMICE package [3] were integrated with OpenAD and PETSc to create the AmundIce dynamic flow model of the Amundsen region.
On the 4-cabinet system,

- PETSc was deployed following EPCC/Archer2 guidance
- MITgcm was integrated with PETSc by amendment to the STREAMICE interfaces https://github.com/MITgcm/MITgcm/pull/630
- OpenAD was deployed in a container, following [15].
- The amendments to the MITgcm genmake2 script to use this OpenAd were added to the MITgcm branch at git@github.com:dngoldberg/MITgcm.git, for which a pull request is issued to merge with the MITgcm checkpoint.

The above developments were deployed on the full Archer2 system, The GNU compiler was used, a decision that predated the containerisation of OpenAD.

Tests were as follows including to assess the advantage of using PETSc (found on Archer to give 3-fold speedup):

- Data from a run of 240 timesteps (each is one month) with 90 cores on 1 node was validated by eye against Fig 2, first panel, of [16].
- A short run (3 timesteps) with the same configuration was used, its values of STREAMICE_FP_ADJ_ERROR and cf then being used to validate further short runs,
- Short runs were used to seek optimal flags, srun arguments, and node numbers for the cases both with and without PETSc being used. The optimisation flags were set in the opt file.
- Using the apparent optimal configurations long runs were repeated with and without PETSc.

The outcomes were:

- Without PETSc, setting the compilation flag to -O2 was faster by 8% as compared to -O3, however the long runs crashed, with both 90 and 120 cores in ways being investigated. Further progress will be reported in the repository (see below)
- With PETSc there was
    - a 3 -fold speedup (as on Archer1) in the short runs
    - Insensitivity in run times as to whether -O2, -O3 or "-O3 –funroll-loops" was used in the opt file.

- A further ~14% speedup was achieved with 120 processes on 1 node as compared to the initial 90 processes.
- Spreading 90 processes across 3 nodes made little difference to runtime but importantly for confidence in the porting, gave the same results.
- The 240 timestep case with 120 cores, -O3 and with PETSc used, ran in just under 3 hours.

The above is documented with the associated input data and code in the repository [17] .

11.ECCOv4

11.1.Cray compiler

We installed and tested a benchmark as run on Archer, with 96 and then 360 cores for the same ECCOv4 case, see table 7. Changing the number of cores entails using different SIZE.h at compile time and corresponding file data.exch2 at run time. The runs were checked against Archer results for correctness using a script that extracted data from STDOUT.0000.

Table 7 ECCOv4 Benchmark using cray compiler, 96 cores and on Archer 2 block:block srun distribution unless stated

	Archer ieee	Archer2, ieee	Archer2, ieee	Archer2 cce ieee	Archer2 cce ieee	Archer2 cce optimised fastest	Archer2 cce optimised; cheapest
Number of nodes	8	8	4	2	1	4	1
Tasks/ node	12	12	24	48	96	24 cyclic:cyclic	96 block:cyclic
Flags	-O0 -hfp0	-O0 -hfp0	-O0 -hfp0	-O0 -hfp0	-O0 -hfp0	-O3 -hfp3 -Oipa5	-O3 -hfp3 -Oipa5
Runtime sec.	10590	12813	12638	13181	13925	1786	2669
Node hours		28.5	14.0	7.3	3.9	2.0	0.7

The above optimised columns highlight in orange the best results in terms of fastest run and cheapest run. The previous columns are of unoptimised code, with 96 cores distributed over different numbers of nodes. Note that the usual block:block distribution was not optimal: with 4 nodes each with 24 tasks, cyclic:cyclic was 30% faster than block:block.

Further runs with 192 processes with 2 nodes, and with 360 processes, with 2,3 and 4 nodes, improved on neither run time nor node-hours, as compared to the optimised 96 core runtimes in the table. In deciding how many processes and cores to use in future research on Archer2, the prioritised criterion would normally be to minimise node hours. Consequently we would expect future runs derived from this case to use 96 cores and 1 node with block:cyclic distribution.

11.2.GNU compiler

Using GNU in place of the Cray compiler, text written to stdout was perplexing in that each / was replaced by #. This was traced to the archive builder “as” having an alias in the MITgcm tools directory, it being included in the $PATH. This had apparently been needed in a previous installation with a different compiler, pgf77. It still exists in recent checkpoints. Simply explicitly scripting the

path to genmake, rather than using $PATH avoids this.

For the 96 cores case there was little difference in performance when comparing GNU-compilation with "-O3 –funroll-loops" with performance from the Cray.

Runs with the 360 processes failed with the GNU compiler – this is being investigated, aware that a similar circumstance with PAS led to identification of two coding corrections related to uninitialized data which could have such effects.

11.3.Adjoint model using TAF

We note that although the use of the commercial tool TAF [6] is outside the scope of the eCSE project, in related work we have also demonstrated this capability on ARCHER2.

All code and scripts related to the above changes are held at [18].

12.Outcomes

With the PAS and ECCOv4 exemplars explored above, the most cost-effective runs were when nodes were fully used, and with the optimisation flags in the Cray opt file. ECCOv4 was optimal with cyclic assignment of processes to NUMA nodes. The AmundIce case was successfully tested with developments that integrated StreamIce with PETSC and with the containerised OpenAD. This was used in research published in December 2021 [16]. PAS has been now been successfully used on Archer2 for multi-century simulations, the walltime per year on these long production runs is about 50% faster than on Archer, and that combined with the faster queuing is speeding up the research to the point where a publication should be submitted by the end of the 2022.

A workshop for all UK MITgcm users on ARCHER2 is planned for summer 2022, which will extend the benefit of the work carried out here. As well as providing a basic outline of installing and running MITgcm, the code developed here can be used as the basis for optimisation of other MITgcm models.

It was recognised in the proposal that these runs were less intensive than most on Archer2, but expect the lessons in tuning MITgcm to be transferable to future more intensive runs. We also highlight the need for ensembles of jobs to be run: in these situations, we would follow the "job array" guidance of [19] and the optimisation of these cases would be even more important.

13.References

1. http://mitgcm.org/
2. Naughten, K. A., Holland, P. R., Dutrieux, P., Kimura, S., Bett, D. T., & Jenkins, A. (2022). Simulated twentieth-century ocean warming in the Amundsen Sea, West Antarctica. *Geophysical Research Letters*, 49, e2021GL094566. https://doi.org/10.1029/2021GL094566
3. https://mitgcm.readthedocs.io/en/latest/phys_pkgs/streamice.html
4. https://www.ecco-group.org/products-ECCO-V4r4.htm
5. https://petsc.org/release/
6. http://fastopt.de/products/taf/taf.shtml
7. https://www.mcs.anl.gov/OpenAD/openad.pdf
8. https://nora.nerc.ac.uk/cgi/stats/report/eprint/516314
9. https://mitgcm.readthedocs.io/
10. https://docs.archer2.ac.uk/research-software/mitgcm/mitgcm/
11. https://mitgcm.readthedocs.io/en/latest/contributing/contributing.html?highlight=test-report#using-testreport-to-check-your-new-code
12. v1.0.0 eCSE-MITgcm-ARCHER2/eCSE-archer2-verification https://doi.org/10.5281/zenodo.6623438
13. v1.0.0 eCSE-MITgcm-ARCHER2/eCSE-archer2-PAS https://doi.org/10.5281/zenodo.6616578
14. https://github.com/knaughten/UaMITgcm/tree/archer2/example/PAS_999/mitgcm_run/scripts

15. https://mitgcm.readthedocs.io/en/latest/autodiff/autodiff.html#building-the-mitgcm-adjoint-using-an-openad-singularity-container
16. Morlighem M, Goldberg D, Dias dos Santos T, Lee J, Sagebaum M. Mapping the sensitivity of the Amundsen Sea Embayment to changes in external forcings using Automatic Differentiation. Geophysical Research Letters. 2021 Dec 16;48(23):e2021GL095440. https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2021GL095440
17. v1.0.0 eCSE-MITgcm-ARCHER2/eCSE-archer2-AmundIce https://doi.org/10.5281/zenodo.6616564
18. v1.0.0 eCSE-MITgcm-ARCHER2/eCSE-archer2-eccov4r4 https://doi.org/10.5281/zenodo.6637671
19. https://docs.archer2.ac.uk/user-guide/scheduler/,

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