

# NEMO on HECToR

---

Dr Fiona J L Reid  
Applications Consultant, EPCC  
[f.reid@epcc.ed.ac.uk](mailto:f.reid@epcc.ed.ac.uk)  
+44 131 651 3394

- Cray Centre of Excellence team for help with X2 porting and netCDF
- Chris Armstrong, NAG for help with netCDF on the X2

- NEMO on HECToR
  - Overview of NEMO dCSE project
  - What is NEMO?
- System introductions – XT and X2
- XT results
- X2 porting/performance comparison
- Conclusions

- The NEMO Distributed Computational Science and Engineering (dCSE) Project is a collaboration between EPCC and the Ocean Modelling and Forecasting (OMF) group based at the National Oceanography Centre, Southampton (NOCS).
- The project is funded by a HECToR dCSE grant administered by NAG Ltd on behalf EPSRC
- The NEMO dCSE Project concentrates on the following areas:-
  - I/O performance on intermediate and large numbers of processors
  - Nested model performance
- In addition, a separate project has investigated porting NEMO to the HECToR vector system, the X2

- NEMO (Nucleus for European Modelling of the Ocean) is a modelling framework for oceanographic research
- Allows ocean related components, e.g. sea-ice, biochemistry, ocean dynamics, tracers, etc to work either together or separately
- European code with the main developers based in France
- Major partners include: CNRS, Mercator-Ocean, UKMO and NERC
- Fortran 90, parallelised using MPI – versions 2.3 and 3.0
- Code has previously run on both scalar and vector machines
- This project uses the official releases (OPA9) with some NOCS specific enhancements



- NEMO input & output files are a mixture of binary and ASCII data
  - Several small **input** ASCII files which set key parameters for the run
  - Several small **output** ASCII files; time step, solver data, run progress
  - Binary **input** files for atmospheric data, ice data, restart files etc
  - Binary **output** file for model results, restart files etc
- All binary data files are in netCDF format
  - netCDF = network Common Data Format
  - Portable data format for storing/sharing scientific data
- NEMO uses parallel I/O, i.e. each processor writes out its own data files depending on which part of the model grid it holds
  - Should be efficient but may suffer at larger processor counts...

- HECToR (Phase 1): Cray XT4
  - MPP, 5664 nodes, 2 AMD Opteron 2.8 GHz cores per node
  - 6 GB of RAM per node (3 GB per core)
  - Cray Seastar2 torus network



- HECToR (Vector): Cray X2
  - Vector machine, 28 nodes, with 4 Cray X2 processors per node
  - 32 GB of RAM per node (8 GB per Cray X2 processor)
  - Cray YARC network



	<b>HECToR – XT</b>	<b>HECToR – X2</b>
Chip	AMD Opteron (dual core)	Cray X2 processor (4 per node)
Clock	2.8 GHz	1.6 GHz vector, 800 MHz scalar
FPU's	1 M, 1 A	Vector + scalar with multiple processing units. 8 vector pipes giving 2 results per cycle
Peak Perf/core	5.6 GFlop/s	25.6 GFlop/s
Cores	11328	112
Linpack	54.6 TFLOP/s	~80-85% peak TFLOP/s
Theor Peak Perf	63.4 TFLOP/s	2.87 TFLOP/s
Cost per proc hour	10 AU	19.2 AU

- X2 can only be accessed in blocks of 4 processors so the actual charged cost is 76.8 AU per hour
- Ideally want your X2 code to be at least 8 times faster than on the XT
- But, the X2 can give a faster time to solution



**XT results**

- Various compiler flags were tested for both PGI and PathScale
  - 256 processors used (16 x 16 grid)
  - `-r8` ensures all reals are treated as double precision

Compiler flags	Time for 60 time steps (seconds)
<code>-O0 -r8</code>	173.105
<code>-O1 -r8</code>	169.694
<code>-O2 -r8</code>	151.047
<code>-O3 -r8</code>	141.529
<code>-O4 -r8</code>	144.604
<code>-fast -r8</code>	Fails on step 6
<code>-O3 -r8 -Mcache_align</code>	155.933

## PGI

Compiler flags	Time for 60 time steps (seconds)
<code>-O0 -r8</code>	325.994
<code>-O1 -r8</code>	203.611
<code>-O2 -r8</code>	154.394
<code>-O3 -r8</code>	152.971
<code>-O3 -r8 -OPT:Ofast</code>	162.148

## PathScale

- `-O3` found to be best for both compilers
- Have also tried latest PGI/PathScale – minimal improvement

- Various compiler flags were tested for PGI version 7.1.4 (7.2.3 also tested)

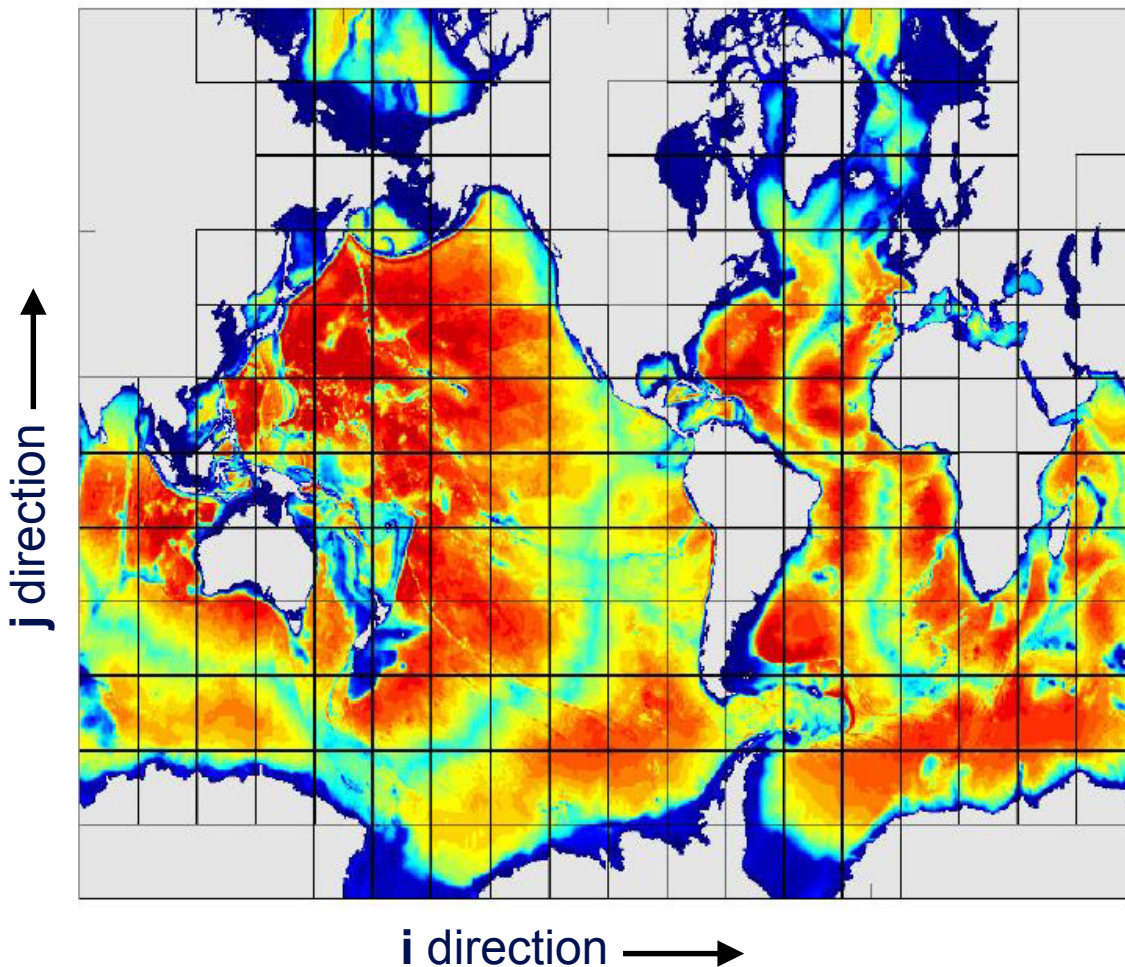
Compiler flags	Time for 60 time steps (seconds)
<code>-O0 -r8</code>	163.520
<code>-O1 -r8</code>	157.123
<code>-O2 -r8</code>	138.382
<code>-O3 -r8</code>	139.466
<code>-O4 -r8</code>	137.642
<code>-fast -r8</code>	FAILS with segmentation violation
<code>-O2 -Munroll=c:1 -Mnoframe -Mautoinline -Mscalarsse -Mcache_align -Mflushz</code>	133.761 with 138.965 for <code>-O4</code>

- `-O4` best, but minimal difference from `-O2` to `-O4`
- `-fast` invokes a number of options; independent testing of each flag shows the problem flags to be:
  - `-Mlre` Loop redundancy elimination – this shouldn't cause a crash!
  - `-Mvect=sse` Allows vector pipelining to be used with SSE instructions
- PathScale compiler – no improvement

- HECToR can be run in single core (SN) or virtual node (VN) mode
- SN mode uses one core per node, VN mode uses both cores
- If your application suffers from memory bandwidth problems SN mode may help

Number of processors	Time for 60 steps (seconds)	
	SN mode	VN mode
256	119.353	146.607
221	112.542	136.180

- Runtime reduces when running NEMO in SN mode
- NEMO doesn't benefit sufficiently to justify the increased AU usage



Grid used for ORCA025 model

$jpni$  = number of cells in the horizontal direction

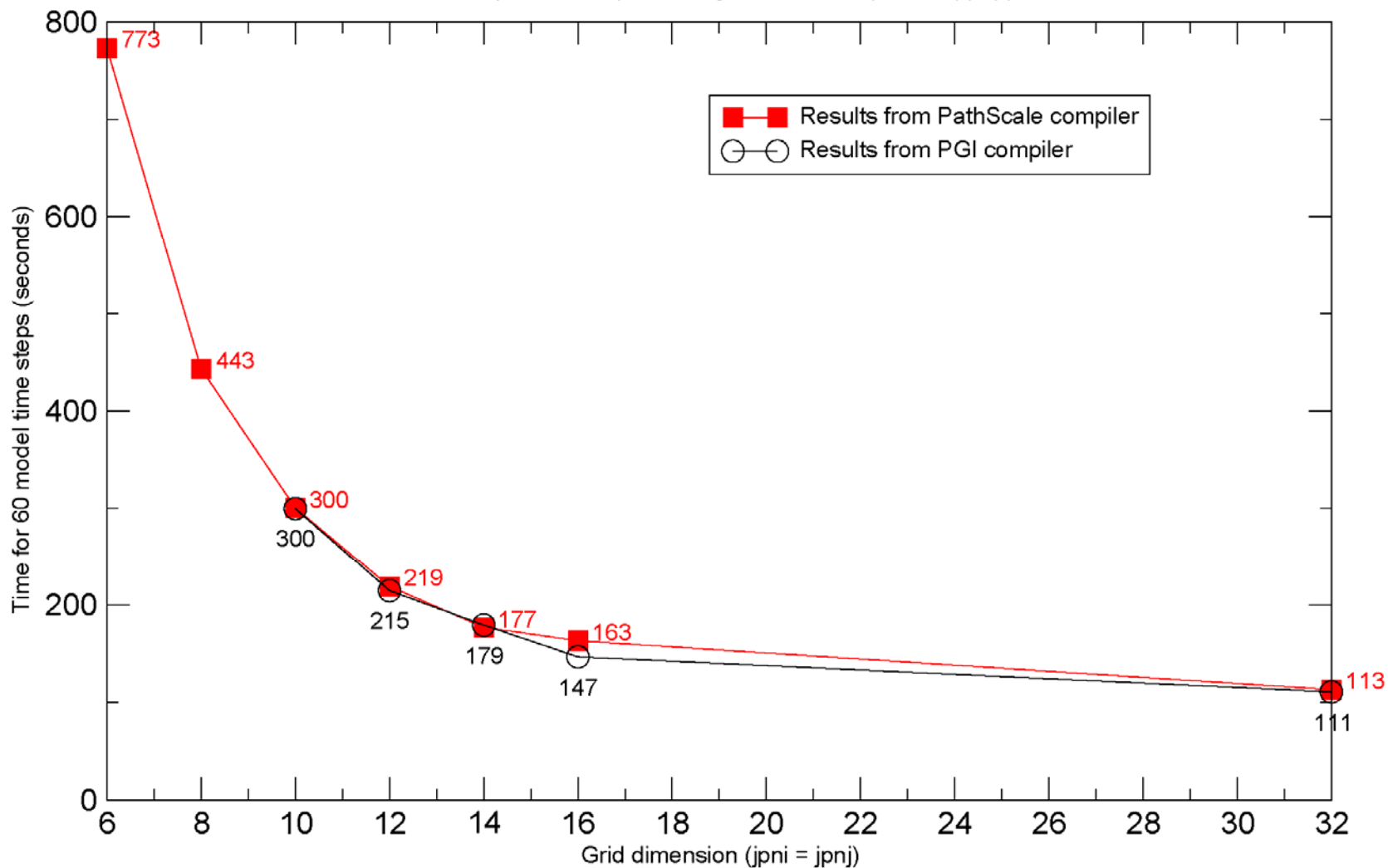
$jpnj$  = number of cells in the vertical direction

Here,  $jpni = 18$ ,  $jpnj = 12$

Image provided courtesy of Dr Andrew Coward, NOCS

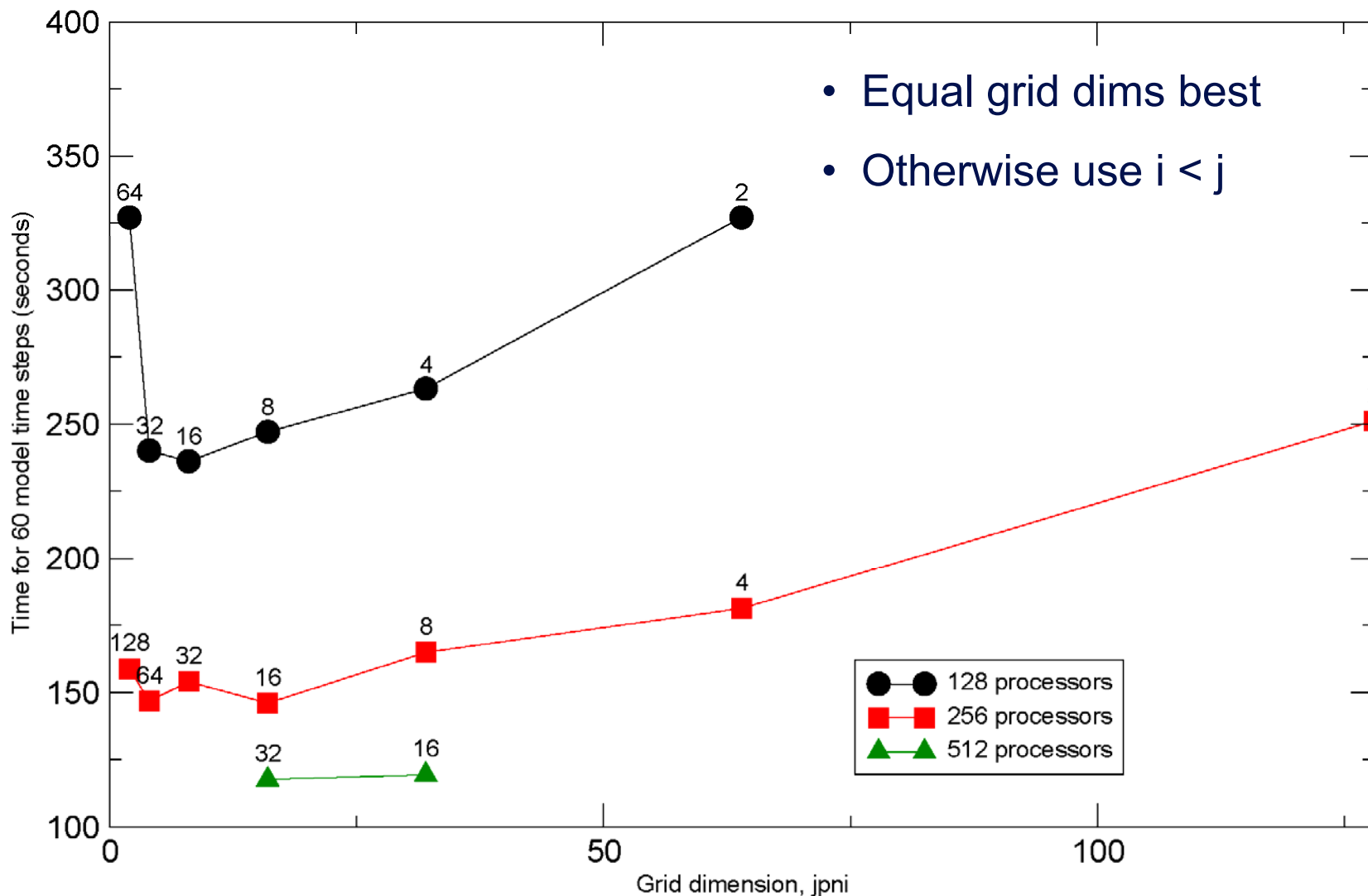
## Performance of NEMO for equal grid dimensions

Number of processors plotted adjacent to each point, mppnppn=2



Performance of NEMO plotted against jpnj for 128, 256 and 512 processors

jpnj plotted adjacent to each point



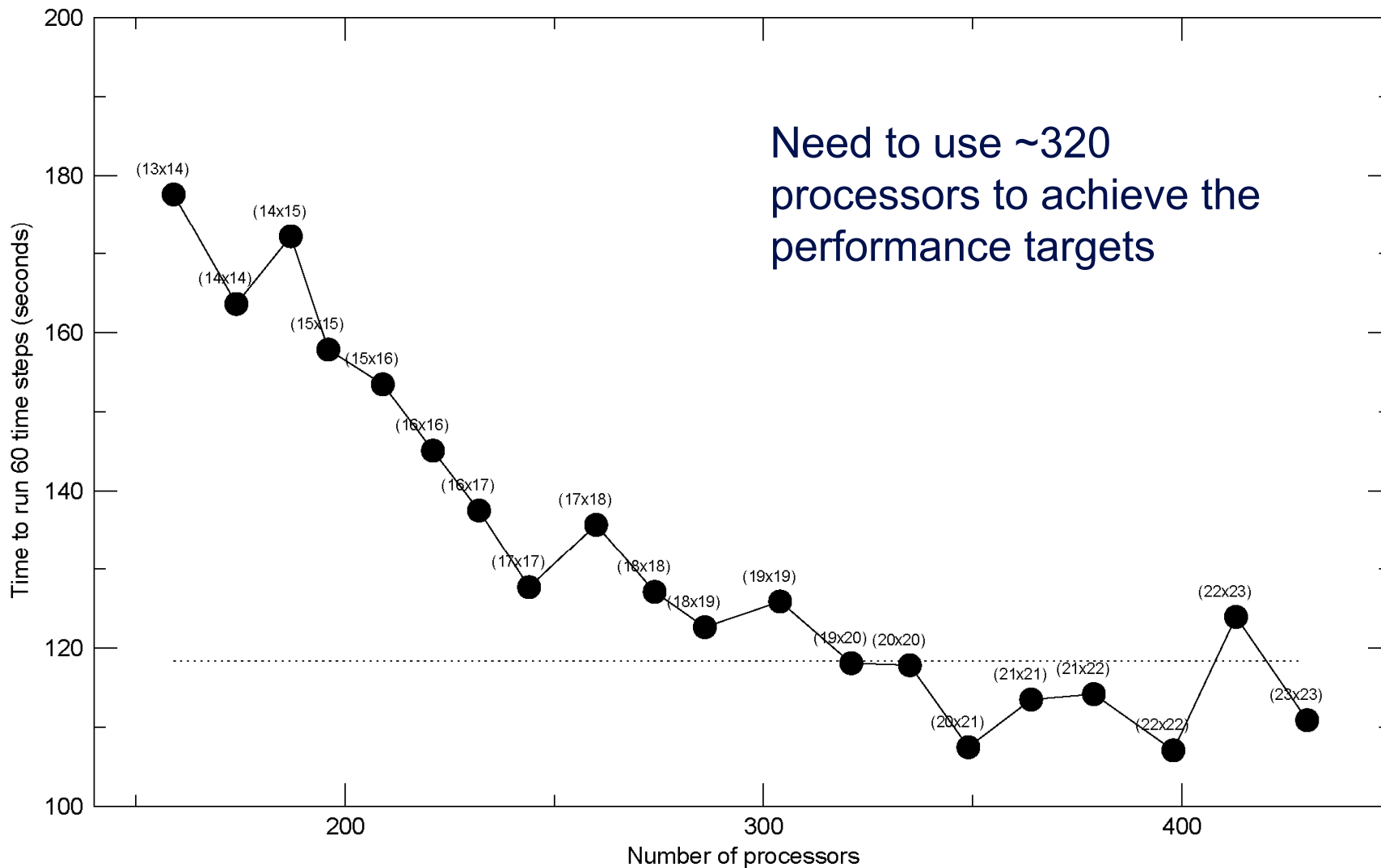
- Ocean models only model the ocean
- Depending on the grid, some cells may contain just land
  - Land only cells do not have any computation associated with them
  - However, they do have I/O
  - A zero filled netCDF file is output for each land cell
- The land only cells can be removed prior to running NEMO
  - Work out how many land only cells there are via the bathymetry file
  - Set the value of jpnij equal to the number of cells containing ocean
  - E.g. for a 16 x 16 grid there are 35 pure land cells so  $jpnij = 221$

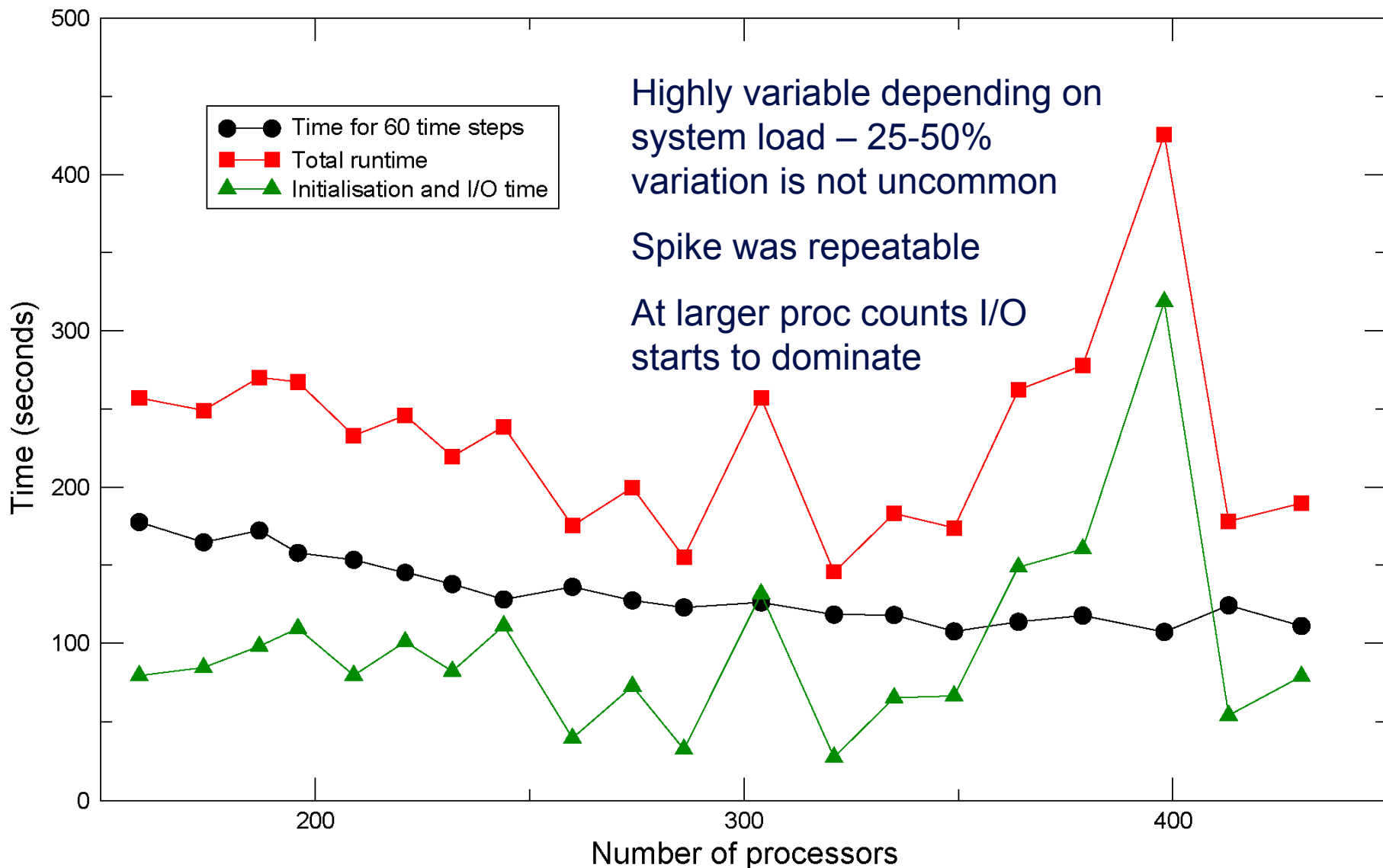
jpni	jpnj	jpni j	Reduction in AU's used	Time for 60 steps (seconds)
8	16	128		236.182
8	16	117	8.59%	240.951
16	16	256		146.607
16	16	221	13.67%	136.180
16	32	512		117.642
16	32	420	17.97%	111.282
32	32	1024		110.795
32	32	794	22.46%	100.011

- Removal of land cells reduces the runtime and the amount of file I/O
  - No unnecessary output for land regions
- In addition the number of AU's required is greatly reduced
  - Up to 25% reduction for a 1600 processor run

- NOCS researchers want to be able to run a single model year (i.e. 365 days) during a 12 hour run
  - Aids the collation and post-processing of results
  - Current runs on 221 processors provide around 300 model days
- Investigated the “optimal” processor count as follows
  - Remove land cells
  - Keep grid dimensions as close to square as possible
  - Compute the number of model days computed in 12 hours from:
$$\text{ndays} = 43000/t_{60}$$
  - Ideally want  $t_{60}$  to be  $\leq 118$  seconds
  - Investigated processor counts from 159 - 430

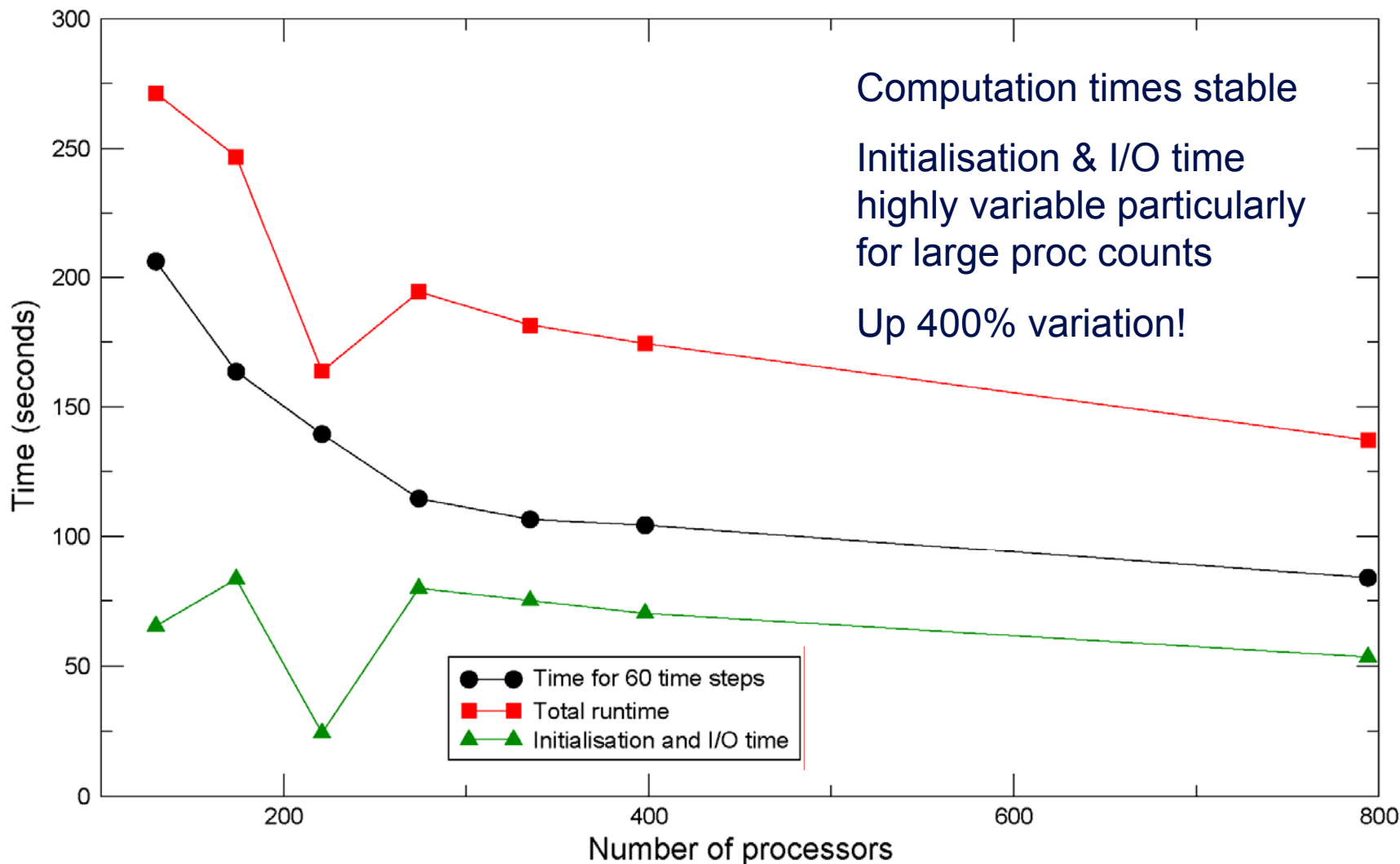
## Optimal processor count for NEMO





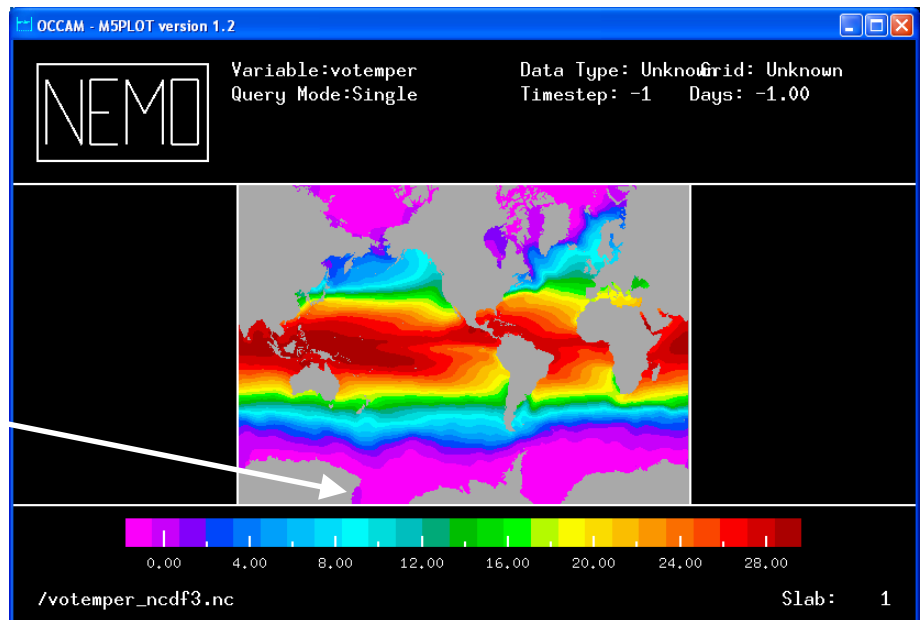
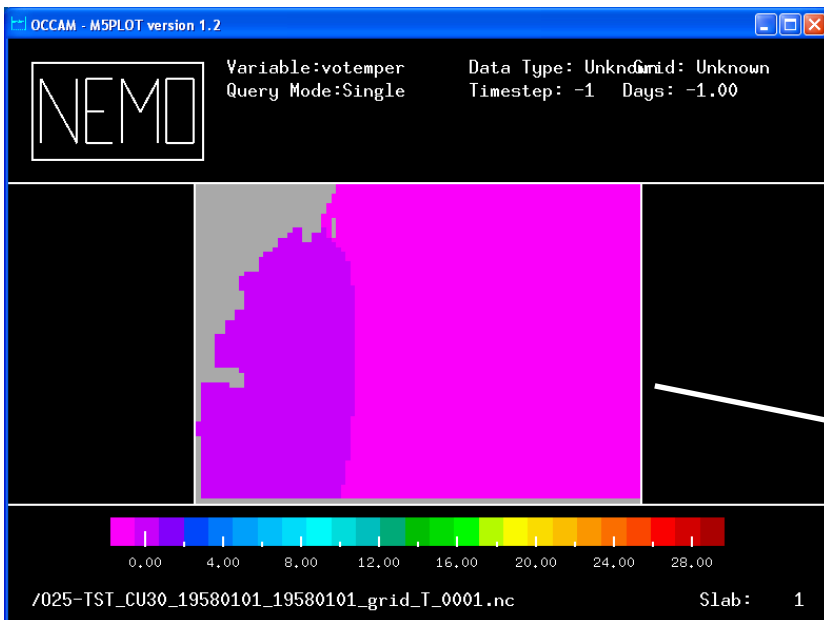
## NEMO V3.0 performance

For 398 & 794 processors the results are from a best of 5 as these were highly unstable



- netCDF 4.0 will be used to improve the I/O performance of NEMO
- Key features
  - Lossless data compression and chunking
    - areas with the same numeric value require far less storage space
  - Backward compatibility with earlier versions
- Requires:-
  - HDF 5.0 1.8.1 or later
  - Zlib 1.2.3
  - Szip (optional)
- All codes tested with supplied test suites – all tests pass
  - Cross compiling caused a few hiccups

- Performance evaluated using the NOCSCOMBINE tool
- NOCSCOMBINE is a serial tool written by the NOCS researchers which reads in multiple NEMO output files and combines them into a single file
  - The entire file can be combined or
  - Single components e.g. temperature can be extracted



- NOCSCOMBINE compiled with various versions of netCDF
- A single parameter (temperature) is extracted across 221 input files
  - Minimal computation, gives a measure of netCDF & I/O performance
  - Time measured and the best (fastest) of 3 runs reported
  - netCDF 3.6.2 and 4.0 output compared using CDFTOOLS\* to ensure results are accurate

\*CDFTOOLS – set of tools for extracting information from NEMO netCDF files

netCDF version	NOCSCOMBINE time (seconds)	File size (Mb)
3.6.2 classic	344.563	731
4.0 snapshot-unopt	86.078	221
4.0 snapshot-opt	85.188	221
4.0 snapshot-opt*	76.422	221
4.0 release	85.188	221
4.0 release*	78.188	221
4.0 release classic	323.539	731

\*system Zlib 1.2.1 used

- Compiler optimisation doesn't help
- System zlib 1.2.1 faster than version 1.2.3
  - Use with care, netCDF 4.0 specifies zlib 1.2.3 or later
- File size is 3.31 times smaller
- Performance of netCDF 4.0 is 4.05 times faster
  - Not just the reduced file size, may be algorithmic changes, c.f. classic

- NEMO is expected to benefit from netCDF 4.0
  - The amount of I/O and thus time spent in I/O should be significantly reduced by using netCDF 4.0
  - NOCSCOMBINE results indicate
    - 3 fold reduction in file size
    - 4 fold reduction in I/O time
  - In addition we need to find optimal chunking/compression parameters so that the data is stored in a sensible way – ongoing work



X2 results

- Ensure X2 environment is loaded
- Compiling/running NEMO
  - Additional flags to ensure source pre-processing is carried out, e.g.

```
FFLAGS = -N 132 -p $(NCDF_INC) -em -F -s real64 -dp -I \  
        $(MODDIR) -I$(MODDIR)/oce -I$(NCDF_INC)
```

```
LINKFLAGS = -N 132 -p $(NCDF_INC) -J $(NCDF_INC) -em -F \  
            -s real64 -dp
```

- **-em** use .mod files for module info (default is .o)
- **-F** enables full macro expansion throughout the source
- **-s real64** adjusts the data size of real data types
- **-dp** disables double precision arithmetic
- **-p, -J** ensures correct path is searched for .mod files

- NEMO requires netCDF library so this must be compiled first
  - Numerous issues encountered with the Fortran Interface and datatypes
  - Cray supplied an X2 port of netCDF 3.5.1
  - NEMO initially crashed with a bus error when the attempting to write out a **fourbytereal** to a netCDF file
  - netCDF tester code replicates this problem, 64 bit version of library fails due to a type mismatch, however, 32 bit version works
  - NOCSCOMBINE and tester code work in 32-bit, NEMO does not
  - New version of netCDF 3.6.2 compiled by Chris Armstrong resolves this problem but NEMO still crashes

- The X2 version of NEMO has not yet run successfully
  - Version 3.0 is now available
  - Unfortunately this also crashes
- NOCSCOMBINE has been tested
- Runtimes
  - 344.563 seconds for the XT
  - 606.535 for the X2
- X2 takes almost twice as long
  - Unsurprising, NOCSCOMBINE is a serial code and was not written with a vector processor in mind
  - The code involves a small amount of computation and is mostly I/O
- We hope that any slow down in from the I/O will be more than offset by the benefits of running on the vector cpus

- NEMO scales to a reasonable processor count on HECToR
  - Square grids give the best performance
  - SN mode helps but not enough to justify increased AU usage
  - Significant time spent in initialisation & I/O
  - Crucial to reduce this time
- Installing netCDF 4.0 is complicated
  - Requires several libraries (HDF5, zlib, szip) before compiling
  - Potential performance improvements are large
  - Other codes may benefit from netCDF 4.0
- X2 porting
  - Not too painful, differences with compiler flags and **aprun** options
  - Issues with netCDF largely resolved
  - NOCSCOMBINE runs successfully, sadly NEMO doesn't