
# Developing NEMO for Large Multi-core Scalar Systems

Stephen Pickles

Advanced Research Computing Group,

Computational Science and Engineering Department

STFC Daresbury Laboratory


# Outline

1. Background
   - NEMO, POLCOMS, prior work
2. Project details and work plan
3. Array index re-ordering
   - ftrans (and gotchas)
   - Early performance results
4. Outlook and conclusions

# NEMO

- Nucleus for European Modelling of the Ocean
- Originally designed for deep ocean and vector architectures
  - Increasingly applied to coastal and shelf seas (more land points)
- Finite-difference, parallel, Fortran code
- MPI everywhere
  - But some promising experiments with OpenMP+MPI hybridisation have started
- Domains defined on regular longitude-latitude grids
  - De-composed geographically in horizontal dimensions, but not over levels
  - Different number of active levels at each grid point
  - Equal-sized sub-domains assigned to each MPI process
- Multiplies by 3-d mask arrays, computing zeroes on land
  - (Apparently) good load balance, but computation on land is redundant
  - Can eliminate processes whose sub-domain is pure land
- Model variables are 3d arrays with level index last
- Prior to 3.3.1, dimensions of all arrays were fixed at compile time

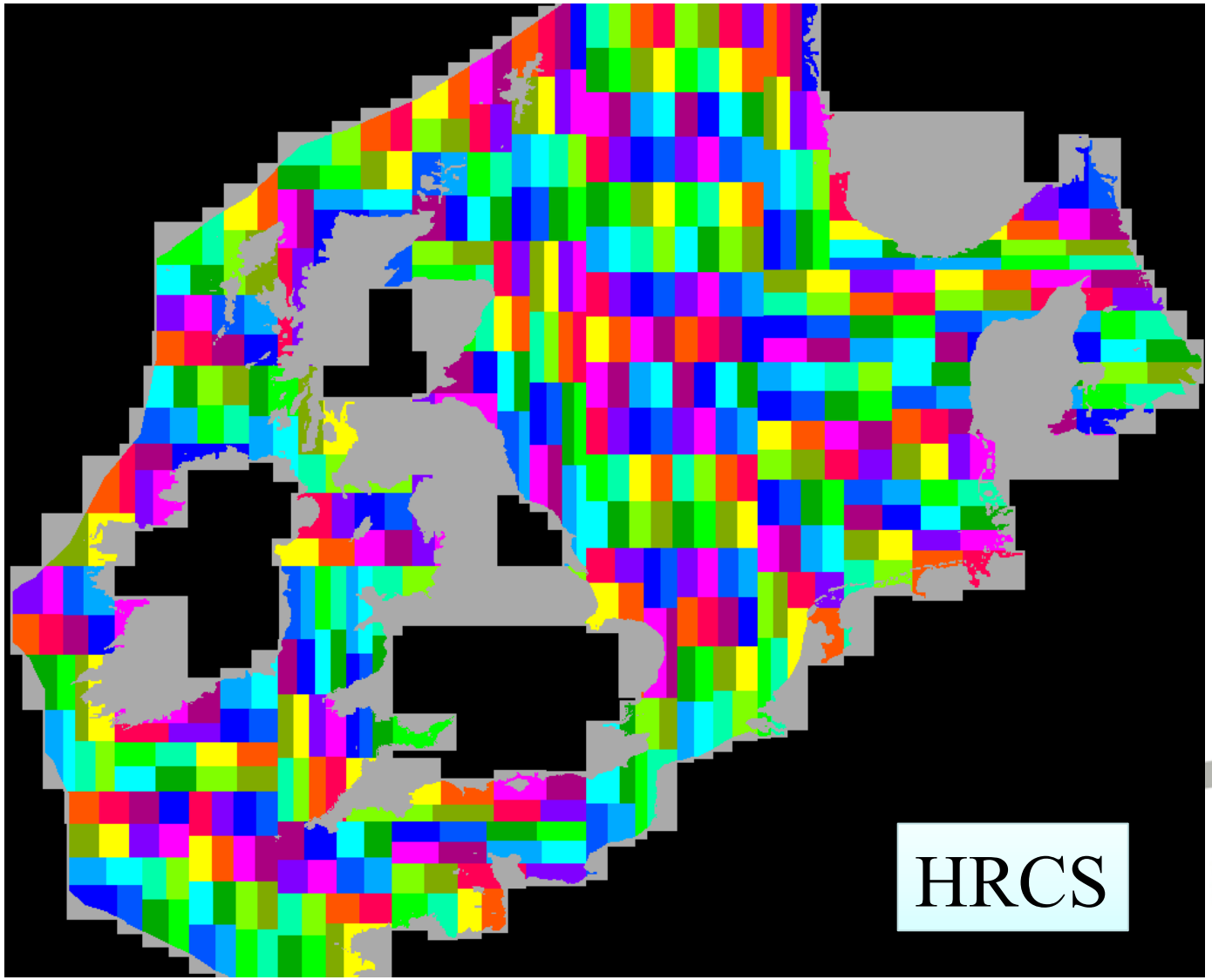
# POLCOMS

- Proudman Oceanographic Laboratory Coastal Ocean Modelling System
- Models coastal and shelf seas
- Adapted to scalar architectures during 1990s
- Finite-difference, parallel, Fortran code
- MPI everywhere
- Domains defined on regular longitude-latitude grids
  - De-composed geographically in 2 dimensions, but not over levels
  - Same number of levels at every grid point
  - Each sub-domain is assigned to one MPI process, not equally sized
  - Using a recursive k-section partitioning algorithm for balancing work load (on basis of number of sea points)
- Uses 2-d wet/dry masks to avoid redundant computation on land points
- Model variables are 3d arrays with level index first

# Comparison

- Anecdotal claims that NEMO is 4-5 times slower than POLCOMS or OCCAM (another deep ocean code) for similar science on scalar architectures
- Possible explanations:
  - Different choices for array index ordering
    - Level index fast (POLCOMS, OCCAM)
    - Level index slow (NEMO)
  - Redundant computation on land in NEMO
    - But not an issue for direct comparisons between NEMO and OCCAM
  - Something else increasing NEMO's affinity for vector architectures
- I suspect that all of these factors (and science reasons too) contribute to the perceived performance gap

# Example partition from POLCOMS




512 processors.

Black points are outside model.

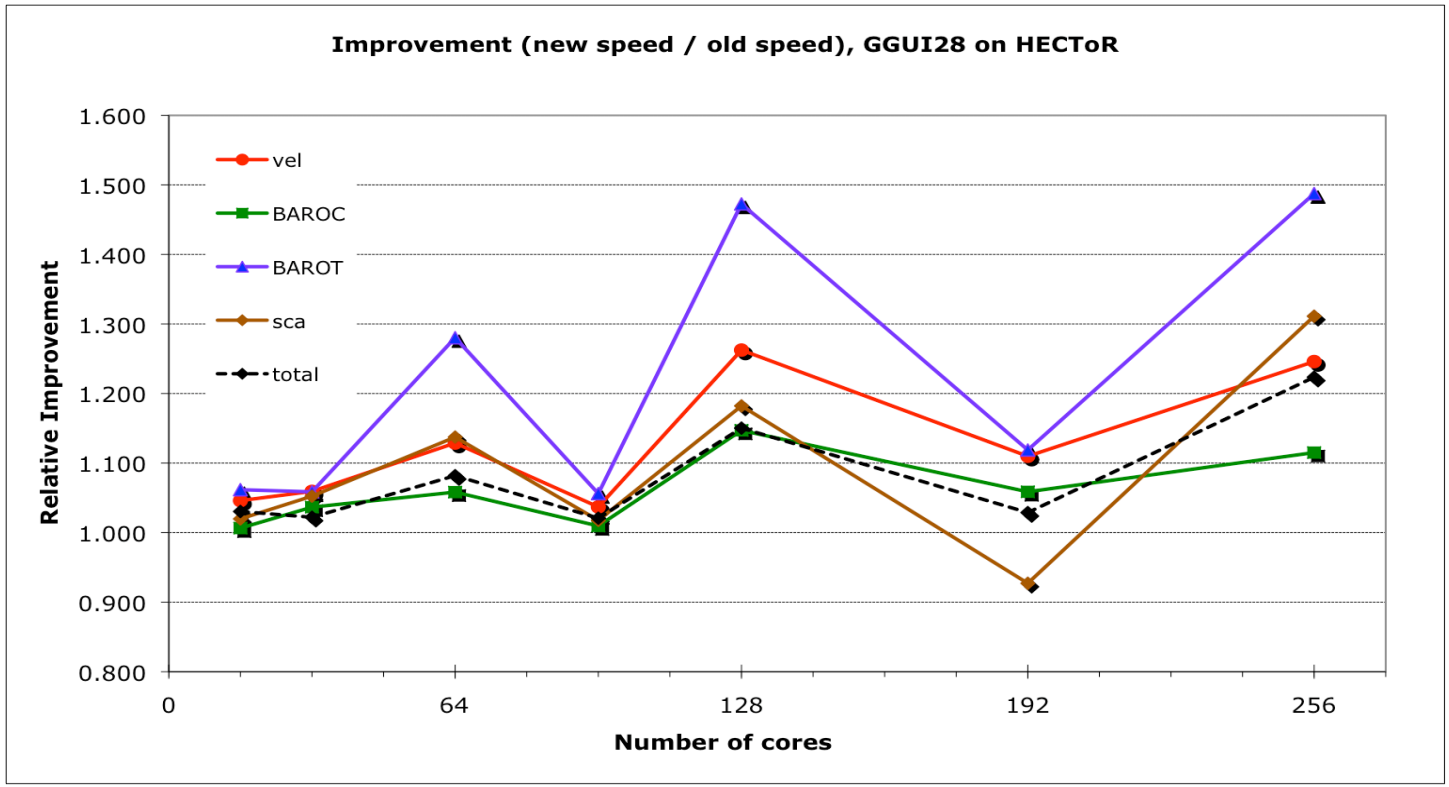
Grey points are dry, but inside model.

Sub-domains have similar numbers of wet points: good computation load balance

Haloes can contain dry points: possible communications load-imbalance

Science & Technology Facilities Council

# Halo exchange optimisations (POLCOMS)

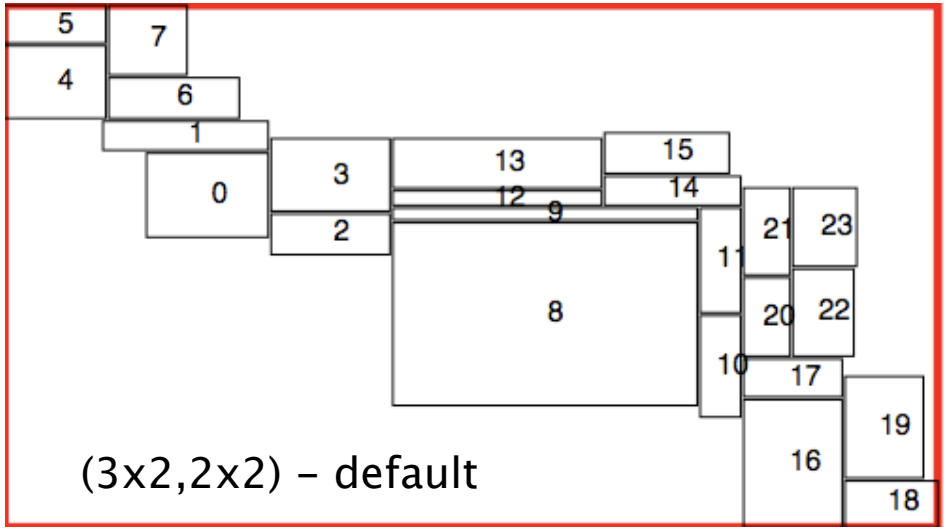


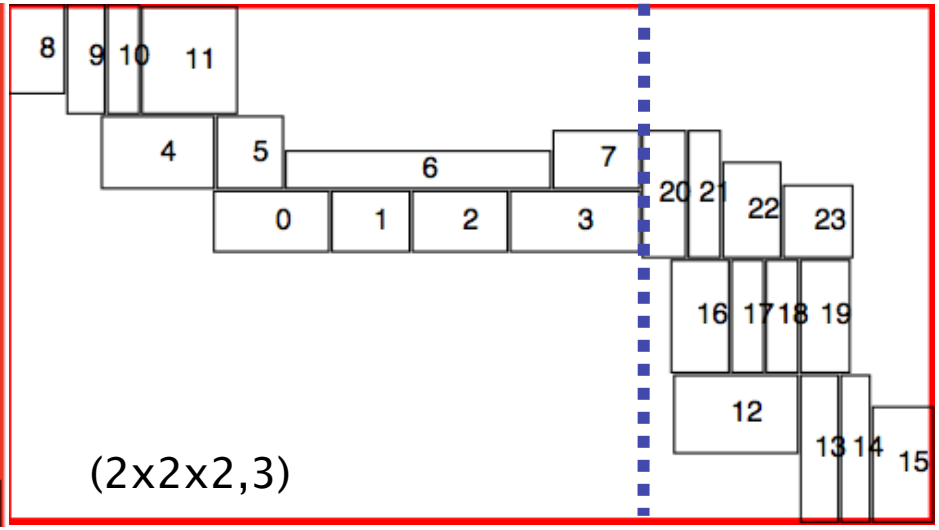

Performance improvement (relative to original) on key physics routines

Optimisations: message combination, dry-point elimination

Science & Technology Facilities Council

# Multicore-aware partitioning



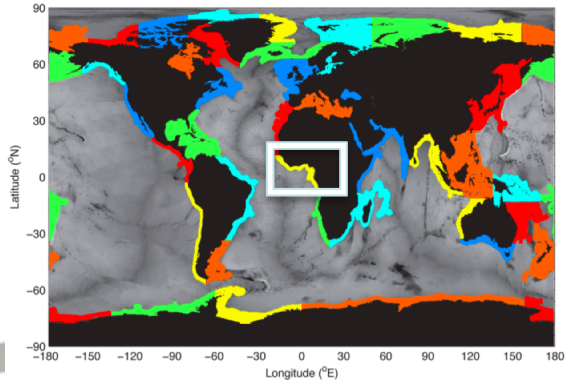




Small domain (Gulf of Guinea) on 24 processors

Different factorizations of processor grid lead to different partitions. Order of cuts changes partition.

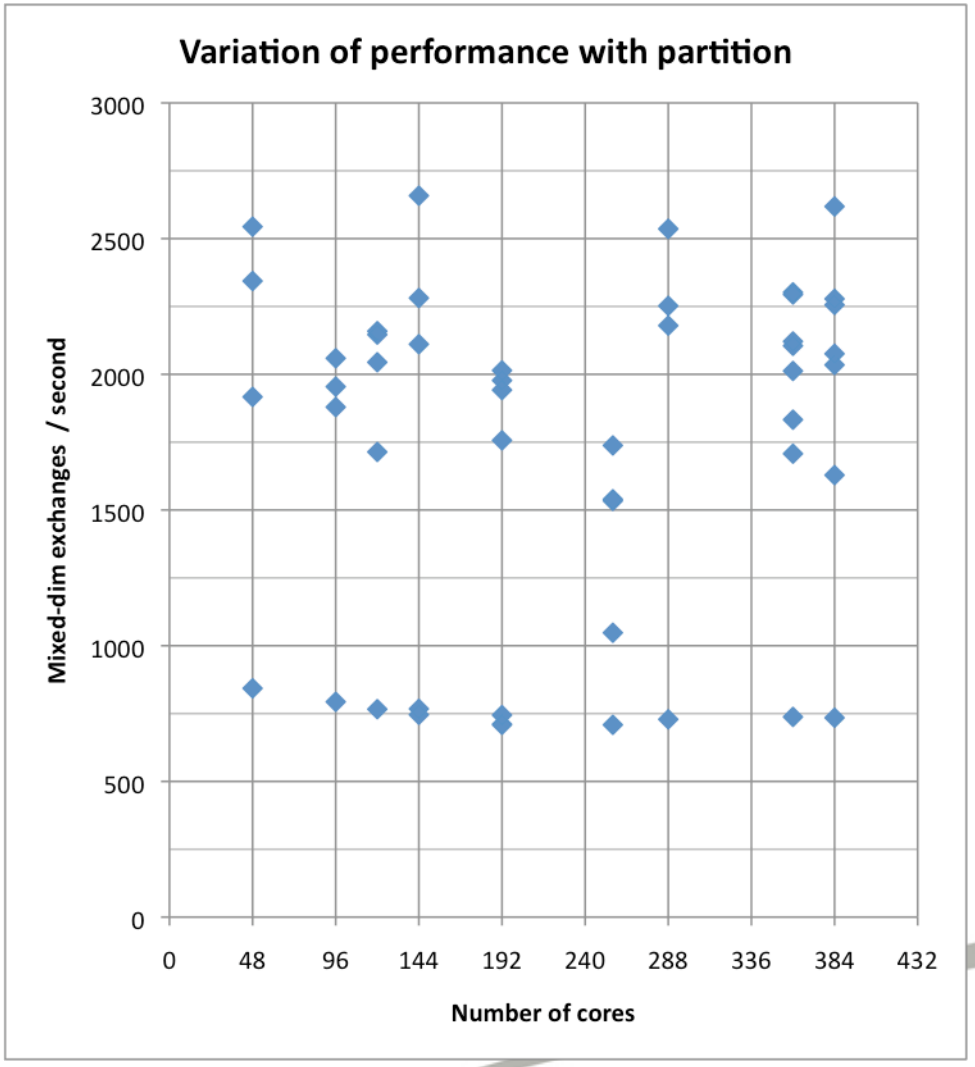
Original default factorization is good for quad-core nodes, but not 6- or 12-core

Choose the “best” from all possible factorizations, in parallel, at run-time!




Science & Technology Facilities Council

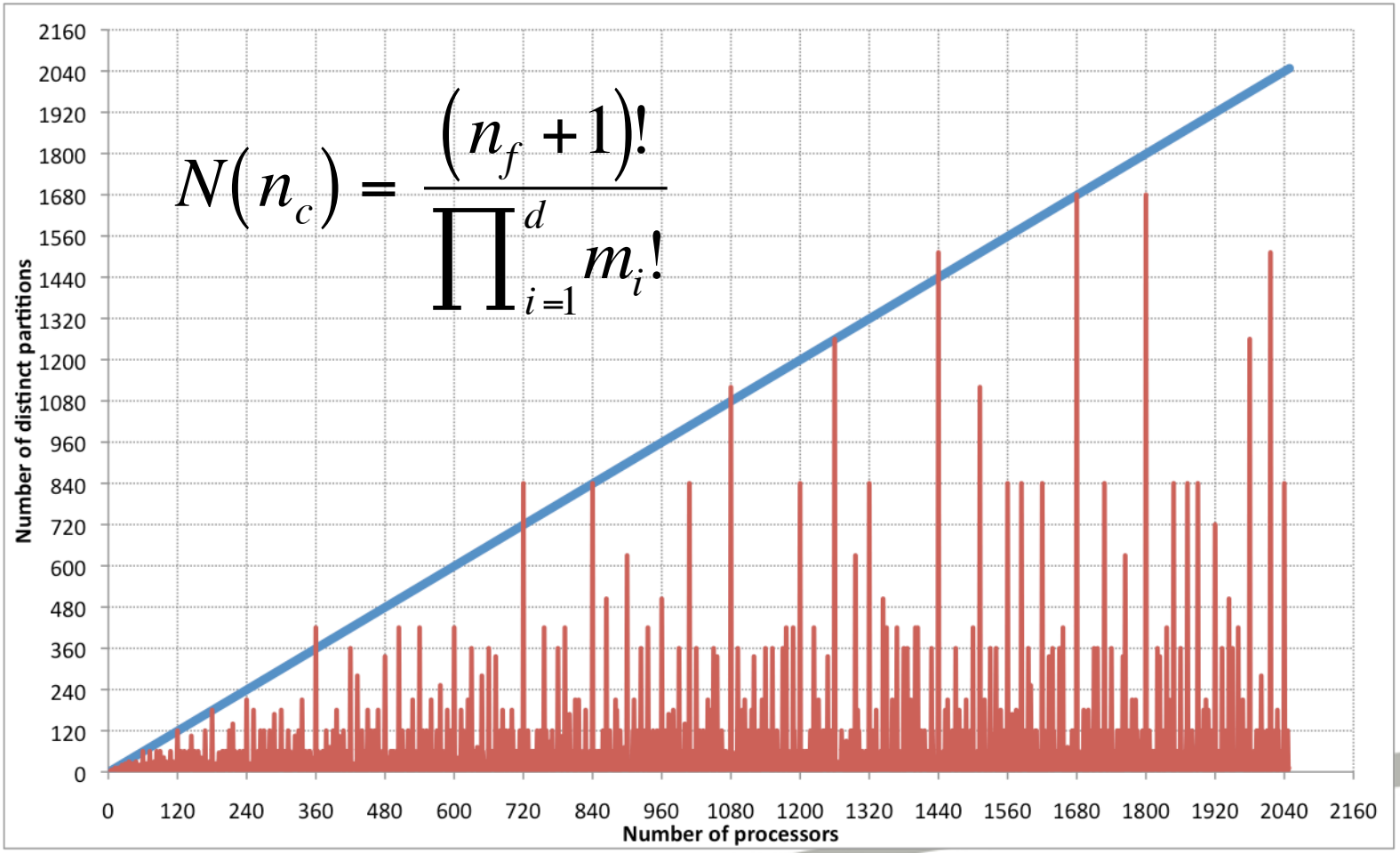
# Performance varies with partition




- Halo exchange performance for different partitions at various core counts
  - Results on rosa (Cray XT5, 2x6-core Istanbul chips/node) using larger HRCS domain
- Some perform much better than others
- Factors of 3 in processor grid give greater opportunities for performance improvement

Science & Technology Facilities Council

# How many distinct partitions?




Science & Technology Facilities Council

# Evaluating partitions in parallel

```
do n=rank, N-1, size
    determine the factors of the n^th distinct permutation
    compute the corresponding partition
    evaluate a cost function for this partition
end do
select the permutation with the best cost function
re-compute the partition for this permutation
```

- Negligible overhead
- Selecting the “best” needs only one call to MPI_All_Reduce
- Visiting the $n^{th}$ distinct permutation was the tricky part
  - My method uses variable radix bases (see CUG2010 paper)
- Challenges in designing a good cost function
  - On XT6, needed to model contention

Science & Technology Facilities Council


# Outline

1. Background
   - NEMO, POLCOMS, prior work
2. Project details and work plan
3. Array index re-ordering
   - Ftrans, gotchas
   - Early performance results
4. Outlook and conclusions

# This Project

- Funded through NAG dCSE programme
- Investigators: Stephen Pickles (STFC), Jason Holt, Hedong Liu, Andrew Coward (NOC)
- Letters of support from John Siddorn (Met Office) and Rachid Benshila (NEMO systems team)
- 12 months effort, 18 months duration
  - Started 1 Jan 2011
  - Stephen Pickles, 50% for 12 months
  - then Andrew Porter, 100% for 6 months
- Builds on earlier work
  - by Stephen Pickles and Mike Ashworth on POLCOMS, and
  - by Andrew Porter on NEMO

Science & Technology Facilities Council

# WP1: Ensemble Capability

- 2 person months
- Use case 1
  - Same domain, different parameters/ forcings
- Use case 2
  - Different domains
  - Lower priority than use case 1
  - Requires dynamic memory,…

Science & Technology Facilities Council

# WP2: Multicore-aware partitioning

- 5 person months
- Decompose into sub-domains using recursive k-section partitioning
  - Borrowing code from POLCOMS
  - Requires dynamic memory allocation
  - Ideas already proven in POLCOMS on HECToR
  - Requires level-index fastest for best results

# WP3: Array index re-ordering

- 5 person months
- Level index in NEMO is slowest varying
  - Plan to produce version with level index varying fastest
- Use Stephen Booth's ftrans utility to help
  - Mark up arrays using ftrans directives
  - Preprocessor handles permutation of indices in all array references
  - Can generate vector and scalar optimised versions from same source, but messier code and build system
- Still need to deal with loop nest order manually
- Cost of testing a mask for land can be amortised over a whole column

# Performance Targets

- 10% (or better) reduction in wall-clock time on deep-ocean domain (no land) from array index re-ordering
- 40% (or better) reduction in wall-clock time on coastal domain (50% land) from multicore-aware partitioning and land/sea masking
- Measured at current limit of strong scaling
- Compared to base version with dynamic memory, I/O disabled.

# Pre-project Concerns

- Raised these concerns with NEMO systems team, Oct 2010
  - Some of the planned optimisations are far-reaching and will touch most of NEMO source.
  - Would like to start from baseline version with dynamic memory allocation
  - We want to maximise likelihood that these optimisations will be accepted into NEMO source tree and incorporated into future releases
  - What is the best way to do version control to assist future merge?
- Agreed:
  - Write access to NEMO SVN granted
  - Dynamic Memory Allocation to be added (by Andrew Porter) immediately following the release of NEMO 3.3 as NEMO 3.3.1, which will form the baseline for all NEMO developments in 2011
- Done:
  - Andrew Porter added dynamic memory allocation, January 2011

# Actual order of implementation

1. Array index re-ordering (WP3)
2. Multicore-aware partitioning (WP2)
3. Ensemble capability (WP1)

Different from original proposal.

Reflects dependencies better: it is not efficient to test land/sea masks at the innermost loop nest.

Revised order agreed in discussions with NEMO systems team, October 2010.


# Outline

1. Background
   - NEMO, POLCOMS, prior work
2. Project details and work plan
3. Array index re-ordering
   - Ftrans, gotchas
   - Early performance results
4. Outlook and conclusions

# ftrans

- Fortran pre-processor
  - Written by Stephen Booth, EPCC, during the early 1990's
  - We use the array index re-ordering feature, after C-preprocessing
- Advantages:
  - Reduces accidental bugs, saves time
  - Can generate code for both orderings from a single source tree
  - Should help facilitate subsequent merge(s)
- Disadvantages
  - Source code is less readable
  - More complex build process, which now depends on ftrans (and lex, yacc)

# Ftrans example (before)

```
!FTRANS a :I :I :z
!FTRANS b :I :I :z :
REAL(wp), ALLOCATABLE :: a(:,:,:), b(:,:,:,:)
…
ALLOCATE(a(jpi,jpj,jpk), b(jpi,jpj,jpk,2))
…
DO jk = 1, jpk
   DO jj = 1, jpj
      DO ji = 1, jpi
         a(ji,jj,jk) = b(ji,jj,jk,1)*c(ji,jj) + …
```

Science & Technology Facilities Council

# Ftrans example (after)

Becomes after pre-processing with

```
cpp <args> | ftrans -f -9 -I -w -s :z :I -
```

```
!FTRANS a :z :I :I
!FTRANS b :z :I :I :
REAL(wp), ALLOCATABLE :: a(:,:,:), b(:,:,:,:)
ALLOCATE(a(jpk,jpi,jpj), b(jpk,jpi,jpj,2))
DO jk = 1, jpk
   DO jj = 1, jpj
      DO ji = 1, jpi
         a(jk,ji,jj) = b(jk,ji,jj,1)*c(ji,jj) + …
```

# Loop nests

Current loop nest orderings are strongly adapted to existing data layout

- Need tweaking for performance
- We use cpp keys to deal with loop nests

```
#if defined key_z_first
DO jj = 1, jpj
   DO ji = 1, jpi
      DO jk = 1, jpk
#else
DO jk = 1, jpk
   DO jj = 1, jpj
      DO ji = 1, jpi
#endif
         a(ji,jj,jk) = …
```

# Gotchas & work-arounds

```
!FTRANS a b :I :I :z
REAL, DIMENSION(jpi,jpj,jpk) :: a, b
```

Ftrans doesn't permute indices inside the DIMENSION attribute. To work around this, manually transform second line to:

```
REAL :: a(jpi,jpj,jpk), b(jpi,jpj,jpk)
```

Unfortunately, this breaks NEMO style rules

# FTRANS limitations & work-arounds

- Variables declared in modules not visible to ftrans in source files that use them
  - Work around: for every module that has public arrays with permuted indices, introduce a corresponding header file <module>_ftrans.h90 containing ftrans directives, and #include it whenever the module is used.
- Re-named module variables (common idiom in NEMO)
  - USE <module>, ONLY :: zwrk => ua
- Fortran scoping of variable declarations different to ftrans
  - Sometimes need !FTRANS CLEAR at the end of a subroutine, then re-include all the _ftrans.h90 header files
    - ftrans generates lots of warnings about unreferenced variables
- Some lines in NEMO source are longer than 132 characters after CPP substitutions
  - Most compilers don't object, but ftrans does

# More gotchas

- Beware the RESHAPE intrinsic!
- a(:,:,1:jpl)=b(:,:,1:jpl)
  - Are a and b really conformable? Check pre-processed sources!
- Using array sections as workspace

```
REAL(wp), POINTER, DIMENSION(:,:) ::   z_qlw
z_qlw  => wrk_3d_4(:,:,1)
```

- Sub-programs where dimensions are passed by argument:

```
INTEGER, INTENT(IN) :: kpi, kpj, kpk
REAL(wp), INTENT(INOUT) :: a(kpi,kpj,kpk)
```

  - Need to manually check all calls, and insert defensive assertions
    - Sometimes kpi==jpi, kpj==jpj, but kpk/=jpj

# Levels vs columns

- NEMO is strongly adapted to level-last ordering
- Common idioms in NEMO
  - Many level by level operations

```
DO jk=1,kpk
   a(:,:,jk) = b(:,:,jk)*c(:,:,jk) …
```

  - References to values at sea surface

```
a(ji,jj,jk)=b(ji,jj,jk)*tmask(ji,jj,1)
```

- Performance issue (poor cache re-use) after index permutation
  - Need to transform loops manually
  - Introduce copies of masks defined only at sea surface?

```
tmask1(:,:)=tmask(:,:,1)
```

# Performance, general

- Operations up and down a water column should go faster after array-index permutation (and sensible transformations of loop nests)
  - Find this to be true, but less instances than expected
  - Sometimes heavily disguised by previous tuning for vector machines
    - Then, transforming loops for level-first order can lead to much simplification

# Performance, general (ctd)

- Some loops slightly favour level first
  - After transformation

```
DO jj=1, jpj
   DO ji=1, jpi
      DO jk=1, jpk
         a(jk,ji,jj)=b(jk,ji,jj)*d(ji,jj)…
```

  - More re-use of d(ji,jj)
- Many operations *should* perform similarly on either ordering

# Performance, MPI

- Halo exchanges should go faster under level-first ordering, as whole water columns at the halo are packed into messages
  - Indeed, halo exchanges are in the top few routines in the level-last profile, but don't show up in the level-first profile
- Some routines (open boundary conditions) use 2-d exchanges on each level
    - Should be much faster in level-first ordering, but needs re-organisation to send/receive whole columns (TODO)
- Will scale better to higher core-counts

# Performance, I/O

- Need to preserve (at least for now) representation of data on disk
- Currently, most data is stored level by level, one file per process
- This means must transform 3-d arrays back to level-last ordering near the I/O layer
  - Inevitable performance hit, hopefully small and temporary
- Much care needed, as third index isn't always the level index, and the same I/O routines are used for both cases!
  - Need to check every call!
  - Restart files done; diagnostics, observations and tracer I/O are work-in-progress

# Status

- Roughly equal overall performance of level first and level last orderings on (small) deep ocean test case (GYRE)
  - Still have more optimisations to apply, including more use of alternative masks defined at sea-surface only, more loop transformations
  - Our target of 10% improvement on deep ocean looks achievable
- Restart files correct (but different, due to changes in order of summation)
- Have reported several bugs in NEMO (and ftrans), and provided a fix for auto-partitioning code in NEMO 3.3.1 choosing poor decompositions


# Outline

1. Background
   - NEMO, POLCOMS, prior work
2. Project details and work plan
3. Array index re-ordering
   - Ftrans, gotchas
   - Early performance results
4. Outlook and conclusions

# TODO

- Address known issues
  - Performance on process 0 markedly worse than other processes (more so than in reference version)
    - Shows up in MPI_SYNC time in craypat profile
    - Needs attention! Pathology in formatted I/O?
  - correctness of some I/O
  - some untreated gotchas in code not covered by CPP keys of test case
- Work on more complex test case
  - hi-res Irish Sea from NOCL, due Nov 2011, based on NEMO 3.3.1, many grid points on land
  - Covers more CPP keys, more functionality
- Can then begin to test masks for land (dry) points, eliminating redundant computation
- Then adapt POLCOMS multi-core aware partitioning to NEMO
- Danger that ensemble capability might slip

# On schedule?

- Not quite
- Array index ordering should be essentially complete by now, but still have work to do on some parts of the code (including ice models, I/O), testing, and more performance tuning
- I underestimated
  - the scale of the task
  - the pervasiveness of NEMO's level-oriented idiom
  - the work necessary to bring ftrans up to date

# Thanks to …

Andrew Porter, Mike Ashworth (STFC), Stephen Booth (EPCC), Jason Holt (NOCL), Andrew Coward (NOCS), John Siddorn, Dave Storkey (Met Office), Rachid Benshila, Gurvan Madec, Claire Levy (NEMO), Italo Epicoco (CMCC)

Science & Technology Facilities Council

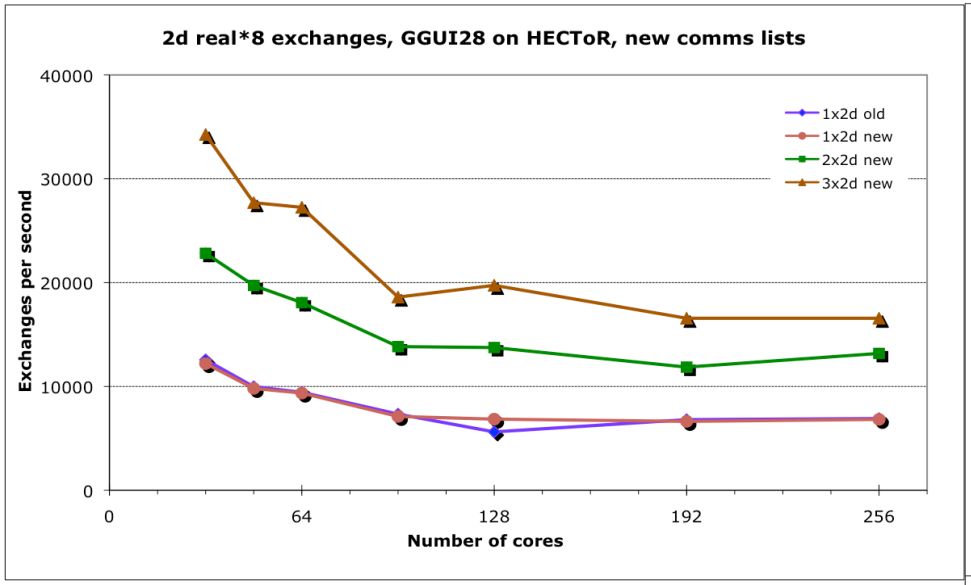

# The end

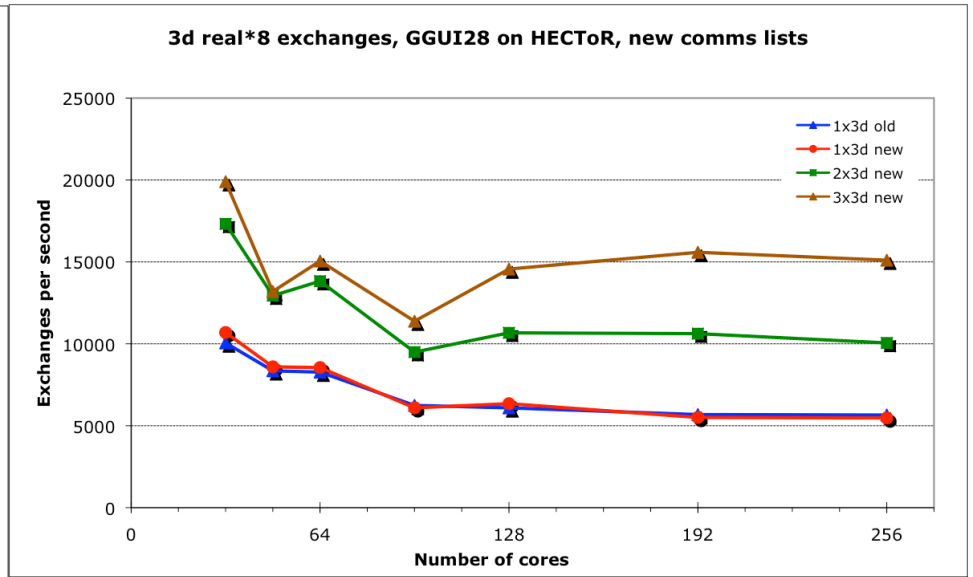
# Spares

# How big is NEMO?

- Excluding tools, external I/O support
- OPA_SRC contains the core of NEMO
  - about 103k lines in 297 source files (about 1 per module)
- About 50k additional lines of science code in other directories
- Lots of CPP keys

# Results, small domain, XT4



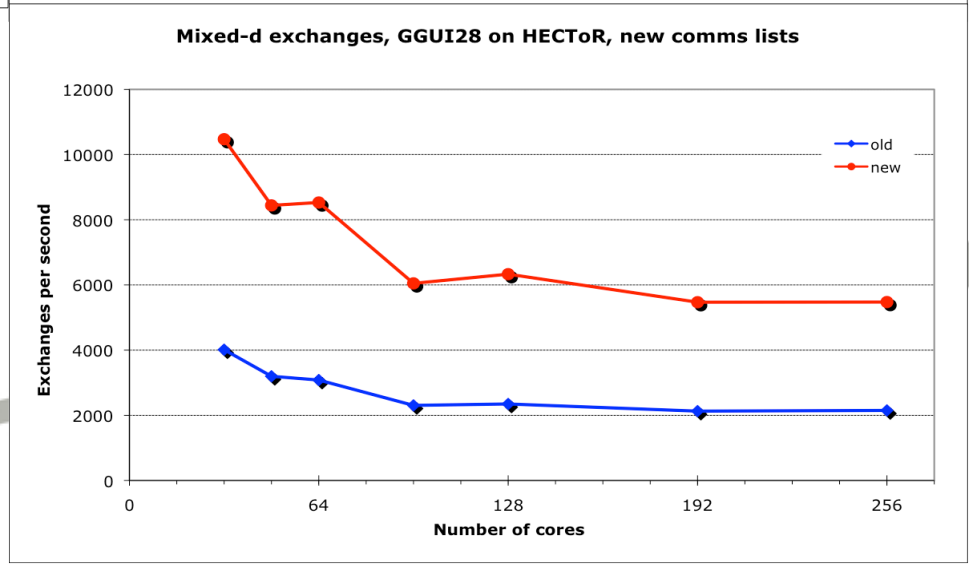




Halo exchange performance, small domain, on HECToR, using message combination and wet patches

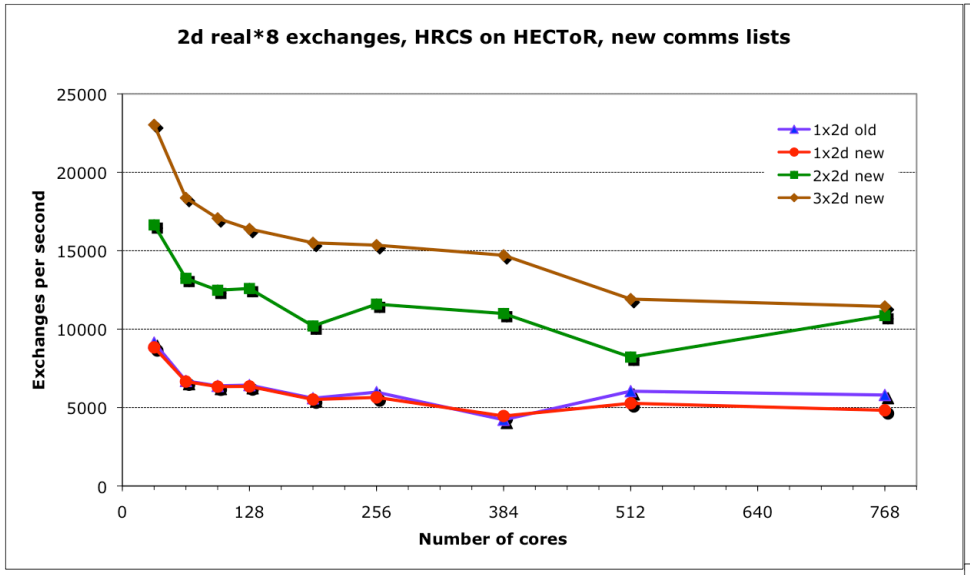
Speeds based on >1000 consecutive exchanges

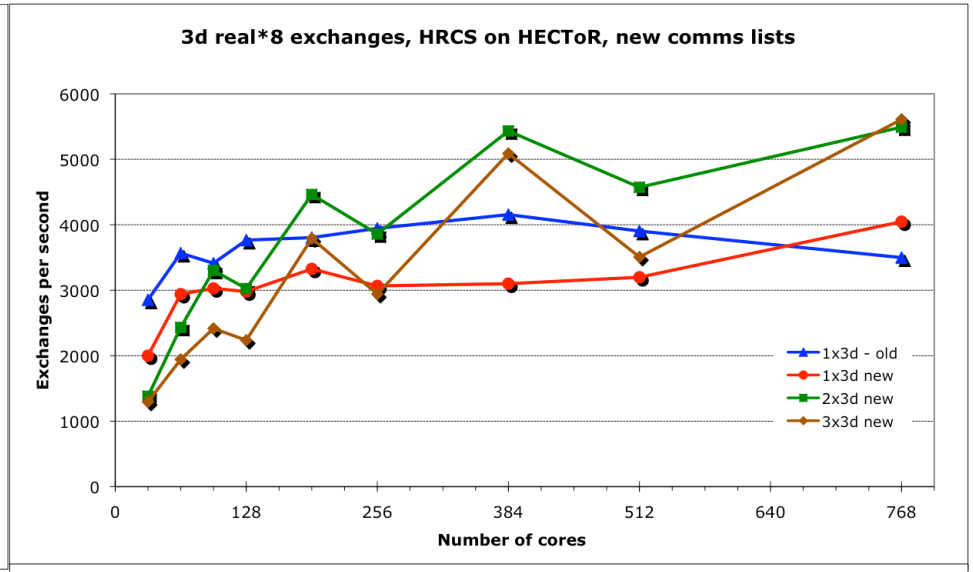
Reference uses old API with clipping

3d exchanges involve a whole water column at each grid point



# Results, larger domain, XT4




Halo exchange performance, larger HRCS domain, on HECToR, using message combination and wet patches