Science & Technology Facilities Council

# Multi-Core Aware Performance Optimization of Halo Exchanges in Ocean Simulations

Stephen Pickles

STFC Daresbury Laboratory

# Abstract

*The advent of multi-core brings new opportunities for performance optimization in MPI codes. For example, the cost of performing a halo exchange in a finite-difference simulation can be reduced by choosing a partition into sub-domains that takes advantage of the faster shared-memory mechanisms available for communication between MPI tasks on the same node. I have implemented these ideas in the Proudman Oceanographic Laboratory Coastal-Ocean Modelling System, and find that multi-core aware optimizations can offer significant performance benefit, especially on systems built from hex-core chips. I also review several multi-core agnostic techniques for improving halo exchange performance.*

Science & Technology Facilities Council

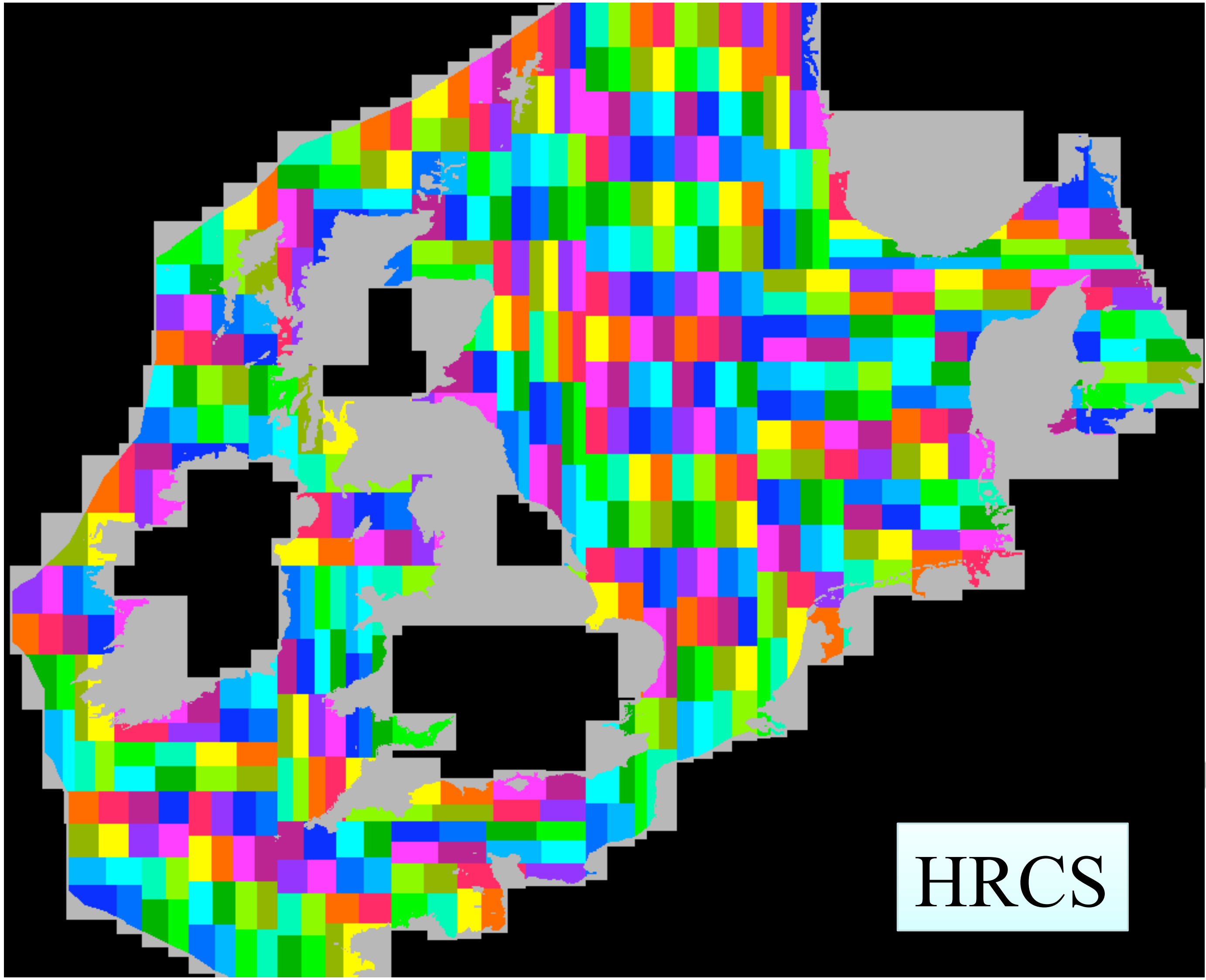
# Outline

1. POLCOMS
2. Various halo exchange optimizations
   – Multi-core agnostic
3. Evaluating distinct partitions in parallel
   – Multi-core aware
4. Conclusions

# POLCOMS

- Proudman Oceanographic Laboratory Coastal Ocean Modelling System
- Models coastal and shelf seas
- Finite-difference, parallel, Fortran code
- Domains defined on regular longitude-latitude grids
  - De-composed geographically in 2 dimensions
  - Using a recursive k-section partitioning algorithm
  - Each sub-domain is assigned to one MPI process
- Uses wet/dry masks to avoid redundant computation on land points

# A sub-domain partition




512 processors.

Black points are outside model.

Grey points are dry, but inside model.

Sub-domains have similar numbers of wet points.

Haloes can contain dry points.

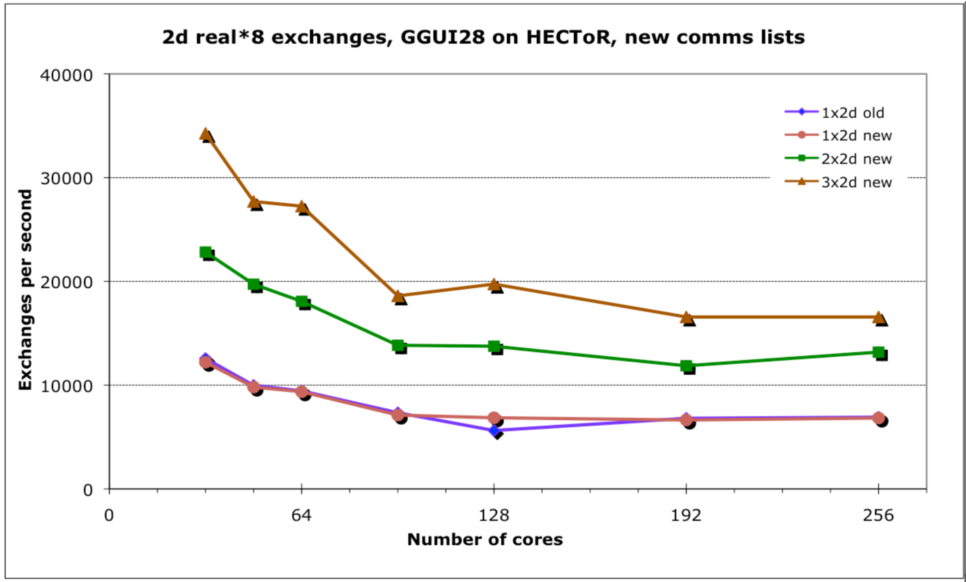
Possible communications load-imbalance.

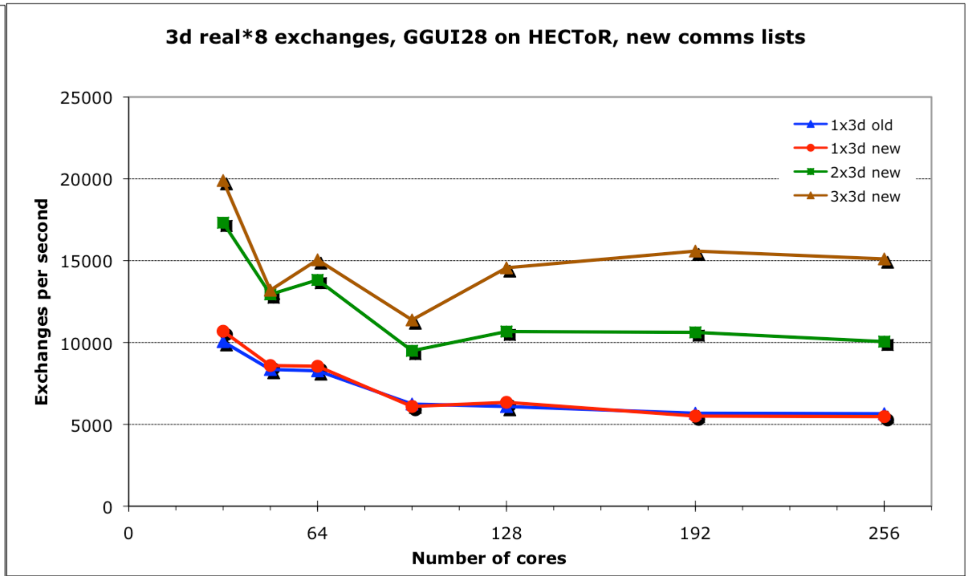
Science & Technology Facilities Council

# Halo exchange optimizations

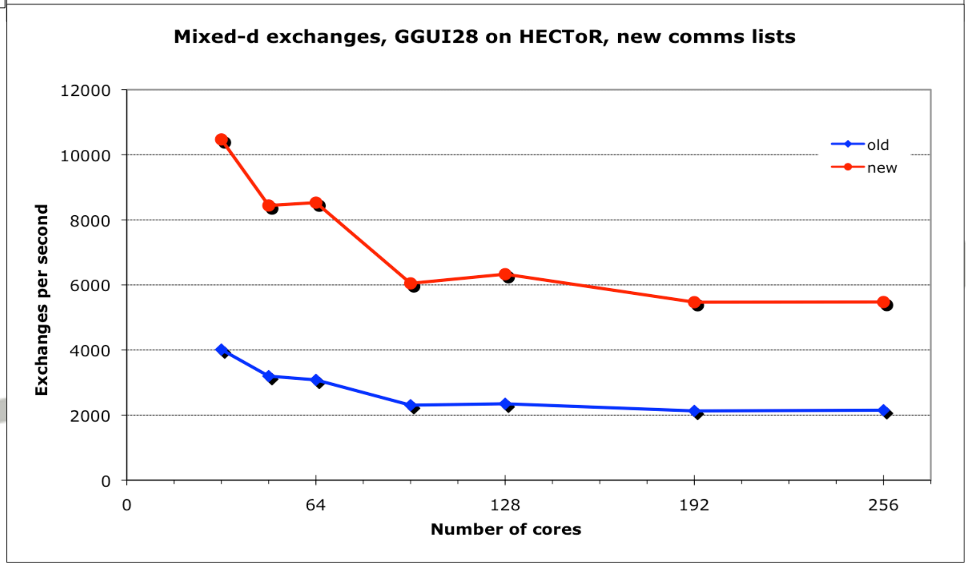
- Message combination
  - Perform exchanges on multiple arrays in one operation, reducing latency
  - Need to manually pack & unpack message buffers
    - Abandoning MPI derived datatypes
  - Requires a different API
    - Some compiler-related performance issues with Fortran pointers
- Eliminating dry points from halo messages
  - Masking, clipping, wet patches
- Pre-posting receives & rank re-ordering
  - Gave little benefit

# 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



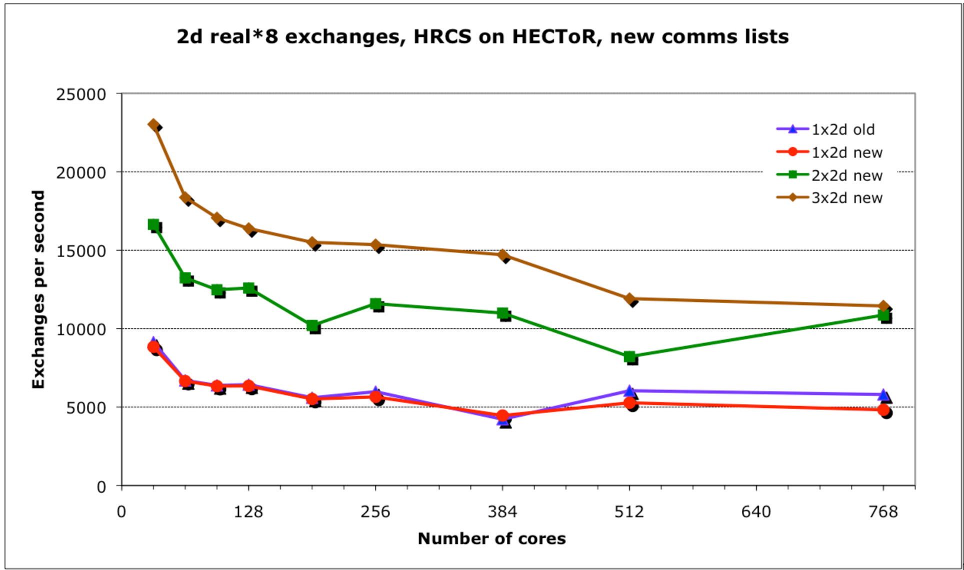
# Masking, Clipping, Wet patches

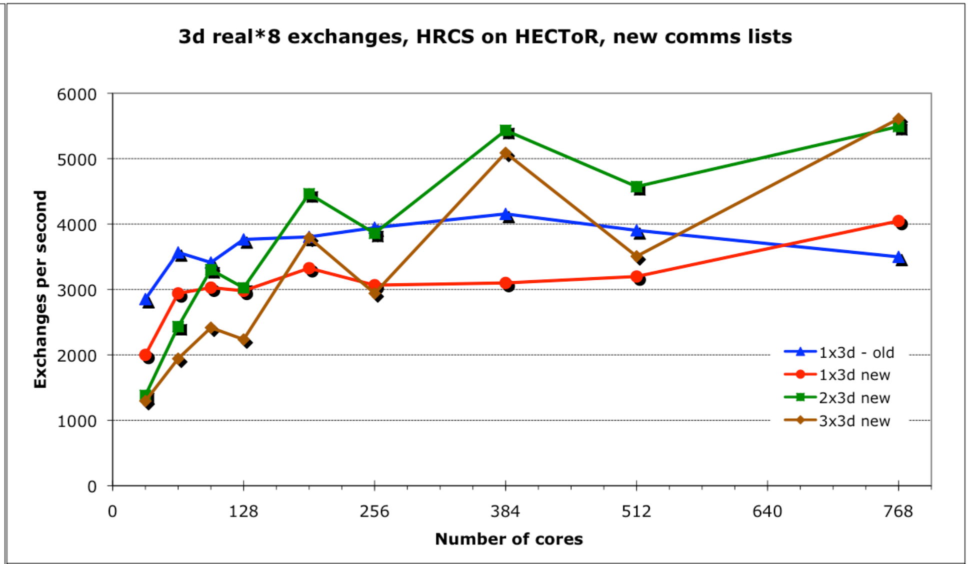
Three ways to reduce dry points in messages:

- Message masking
  - Apply wet/dry mask during pack & unpack
  - Overhead from testing mask
- Message clipping
  - If a halo patch has *exterior* rows or columns that are permanently dry, these can be *clipped* from the comms lists
  - Compatible with MPI derived datatypes and works with existing API
  - Always a good thing to do, but wins not always significant
    - Internal dry points must be important
- Wet patches
  - Change comms tables, defining multiple patches for each message
  - Friendlier than masking for pack & unpack
  - Eliminates most interior points

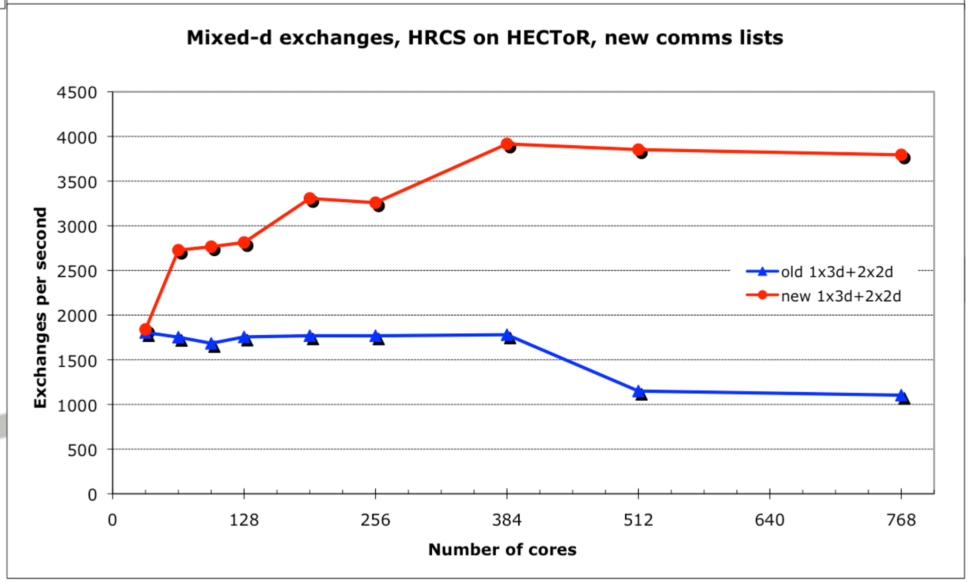
Science & Technology Facilities Council

# Results, larger domain, XT4



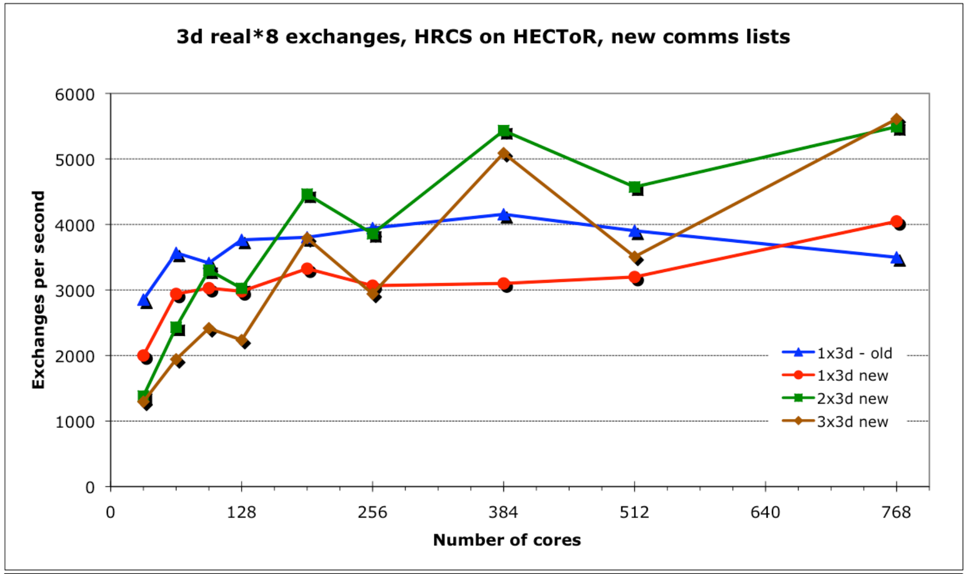

Halo exchange performance, larger HRCS domain, on HECToR, using message combination and wet patches

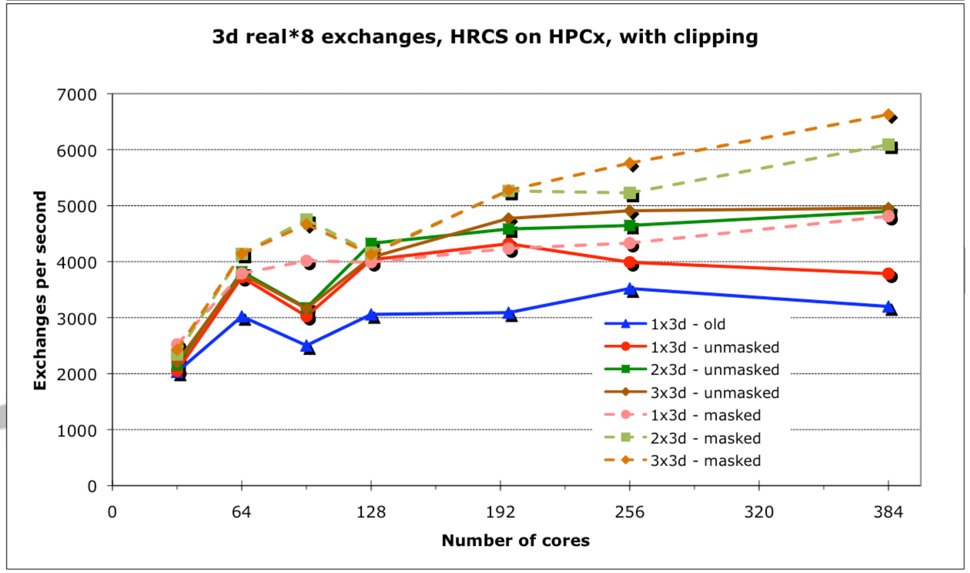


# Taking stock

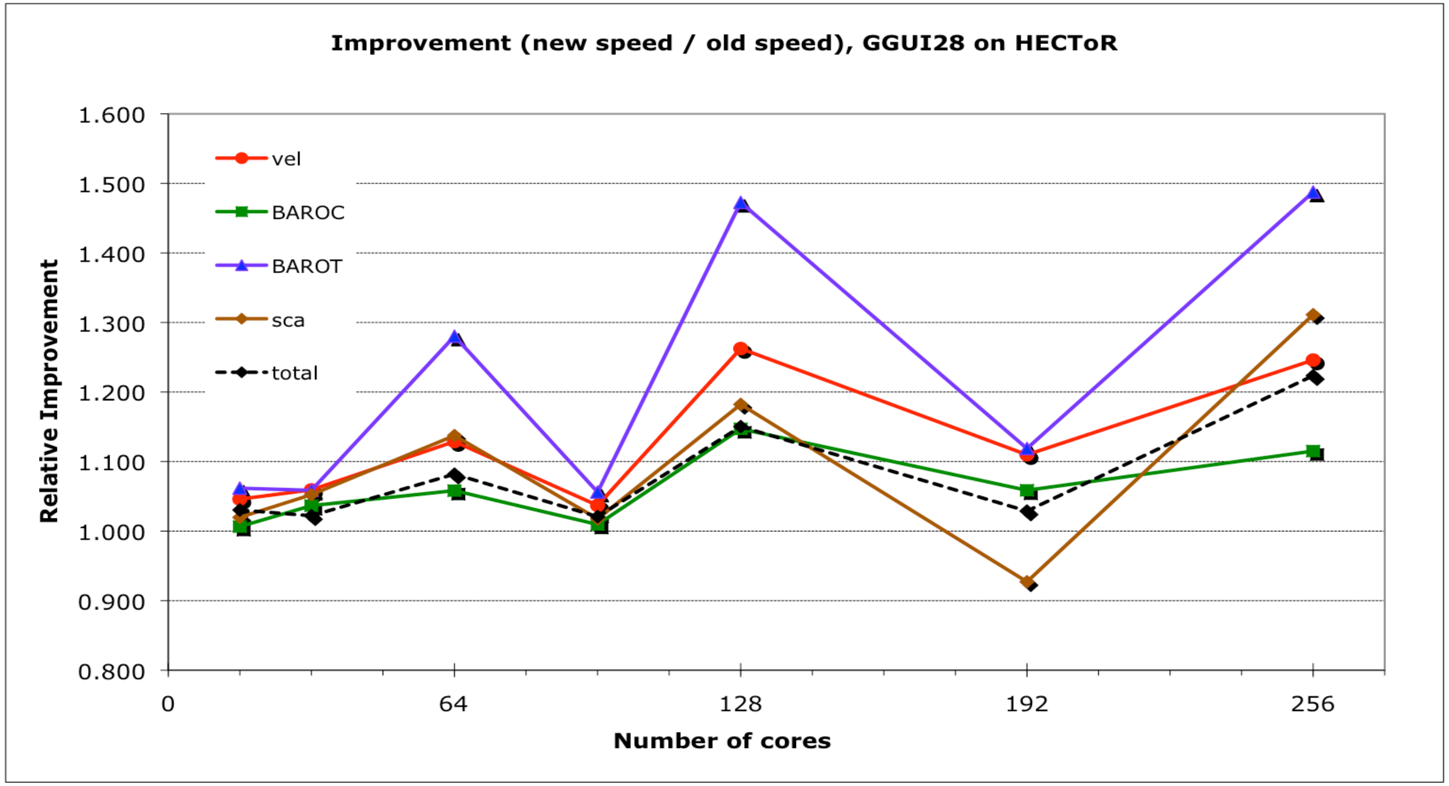
- Combining latency-limited 2d exchanges always helps
- Combining 2d and 3d exchanges usually helps
- Combining 3d arrays does not always help, and can be slower!
  - Cache issues in pack/unpack?

- Performance benefits are architecture-dependent
  - On Cray XT, manual pack/unpack can't match performance of MPI derived datatypes
  - Situation reversed on HPCx (IBM Power5 e-series)



# Effect on overall code

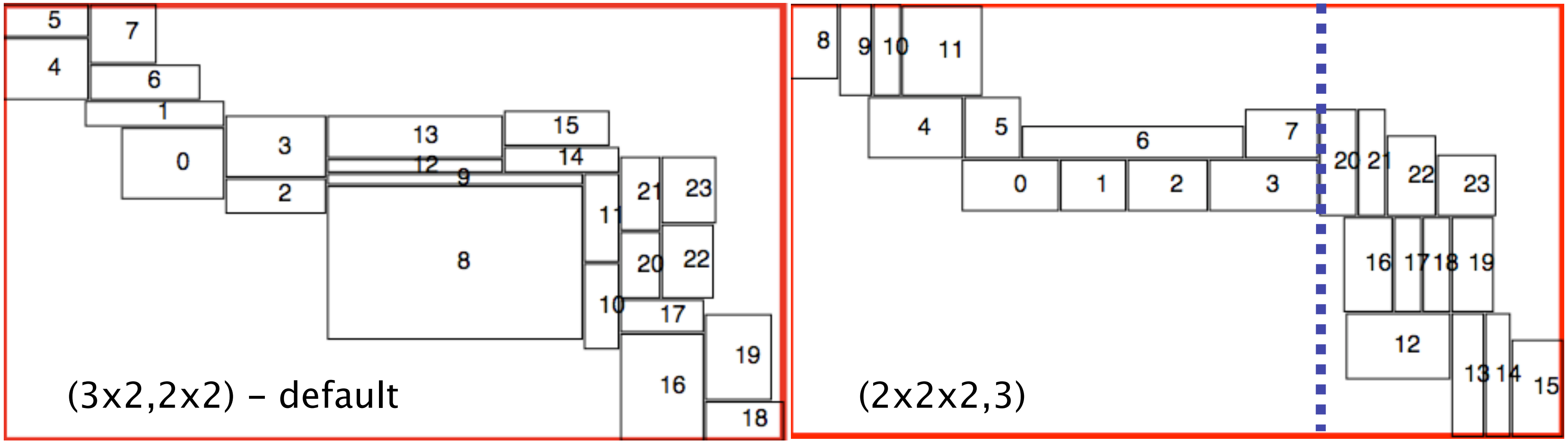



Performance improvement (relative to original) on key physics routines

Only some halo exchanges use the new routines

~50 out of ~350 in applications code

# A closer look at partitioning

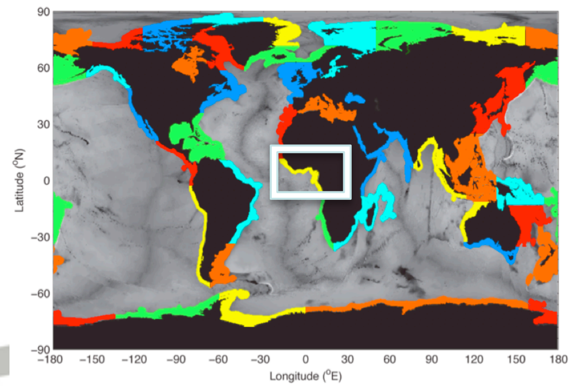



Small domain (Gulf of Guinea) on 24 processors

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

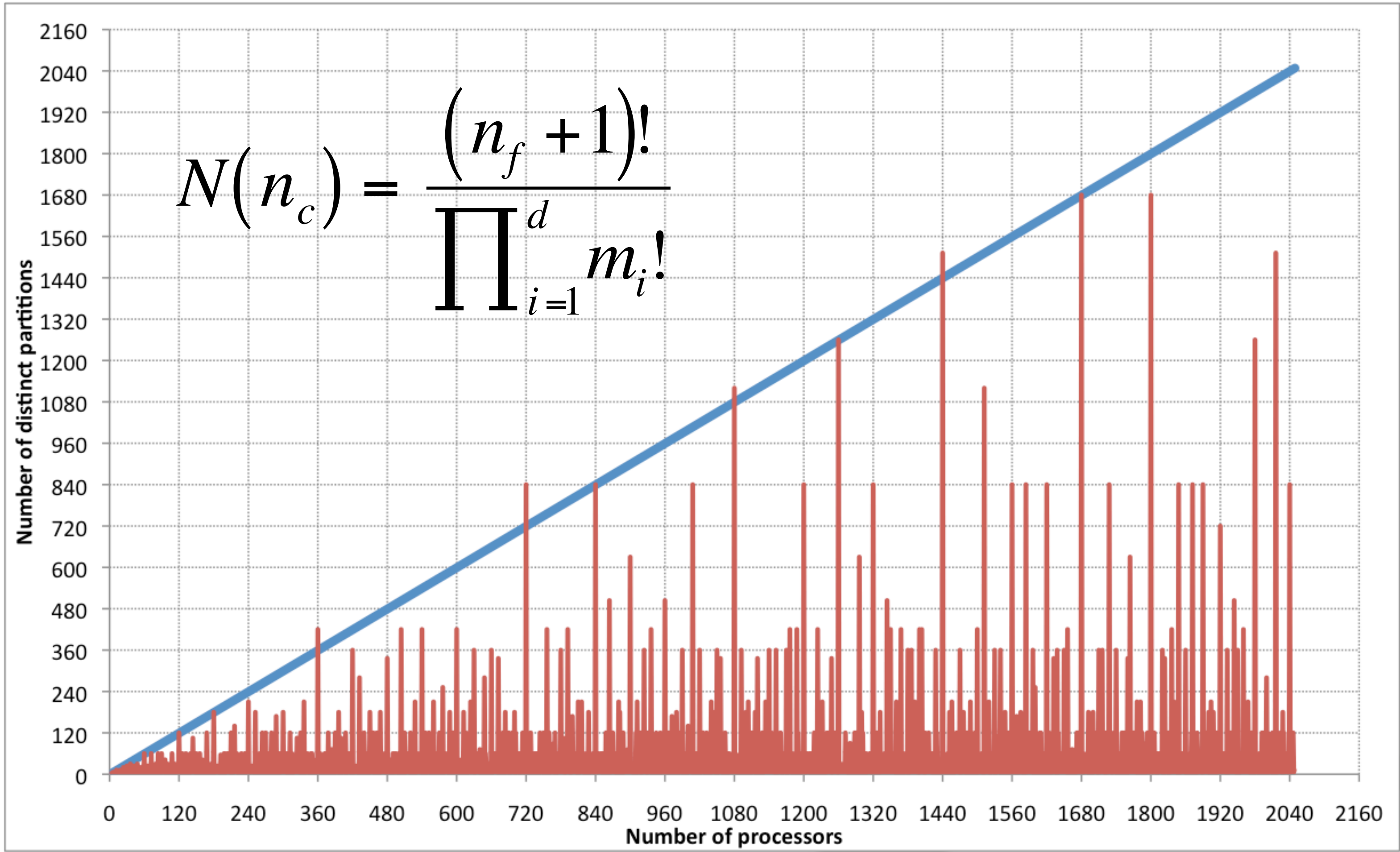
The default factorization is good for quad-core nodes, but not 6- or 12-core

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




Science & Technology Facilities Council

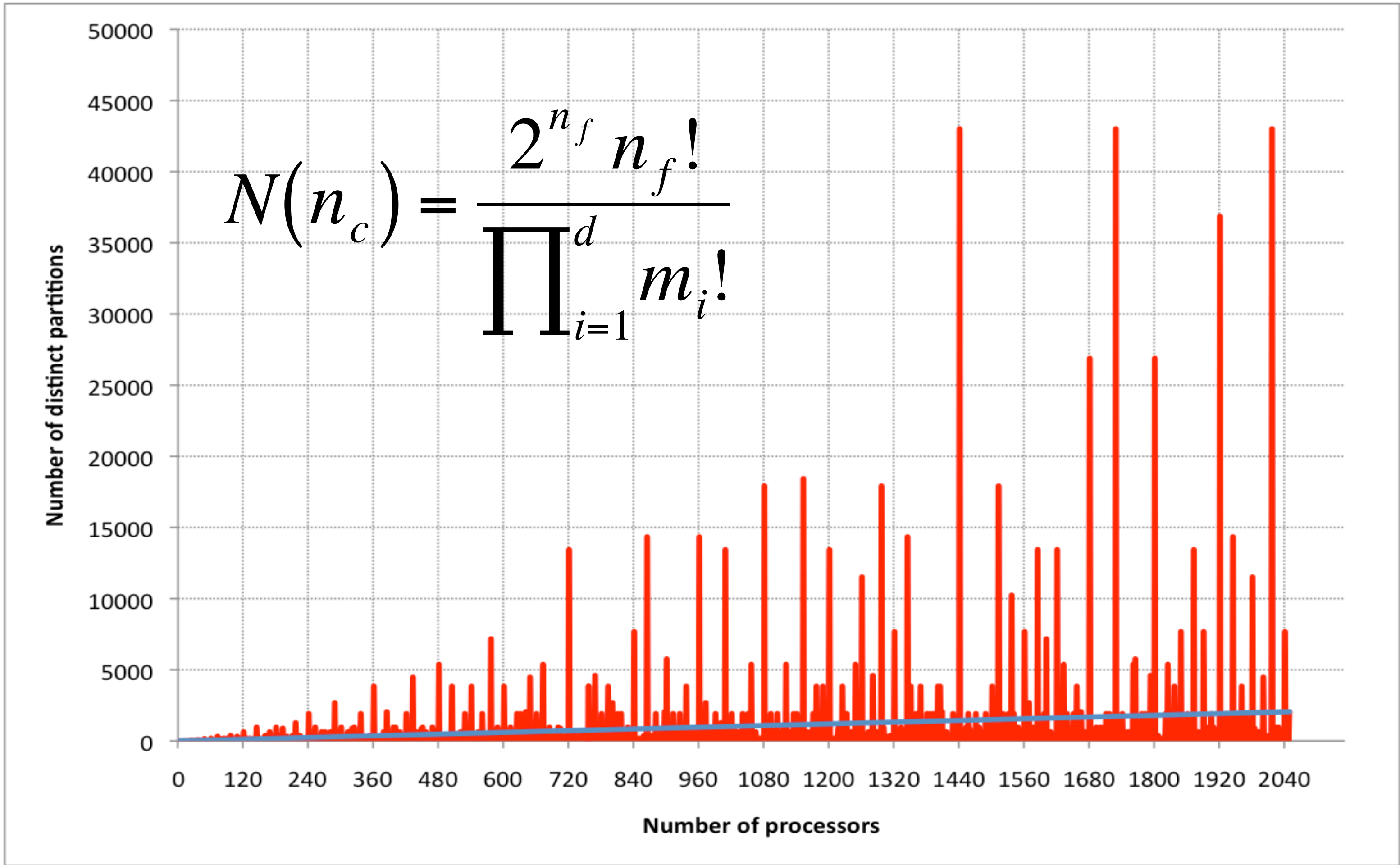
# How many distinct partitions?




$$N(n_c) = \frac{(n_f + 1)!}{\prod_{i=1}^{d} m_i!}$$

Science & Technology Facilities Council

# Aside: even more partitions




$$N(n_c) = \frac{2^{n_f}\, n_f!}{\prod_{i=1}^{d} m_i!}$$

Could reach even more partitions by slightly modifying the recursive k-section method

Science & Technology Facilities Council

# Multi-core aware partitioning

- On 6-, 12-, 24-core systems, more likely to have a factor of 3 in the processor grid
  – Usually want to reserve whole nodes
  – Many more distinct partitions compared to jobs with power-of-2 core counts
- Opportunity to
  – Improve computation and/or communications load-balance
  – Maximize communications locality
    - Intra-node messages are cheaper than inter-node.
    - I assume default (SMP) rank ordering
- Can evaluate alternative partitions in parallel
  – Need cost function, and method for visiting $n^{th}$ distinct permutation without generating all of them

Science & Technology Facilities Council

# Evaluating partitions in parallel

```
do n=rank, N-1, size
    determine the factors of the nth 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
  - I devised a hybrid method based on variable radix bases
  - Some details in paper

# Cost function

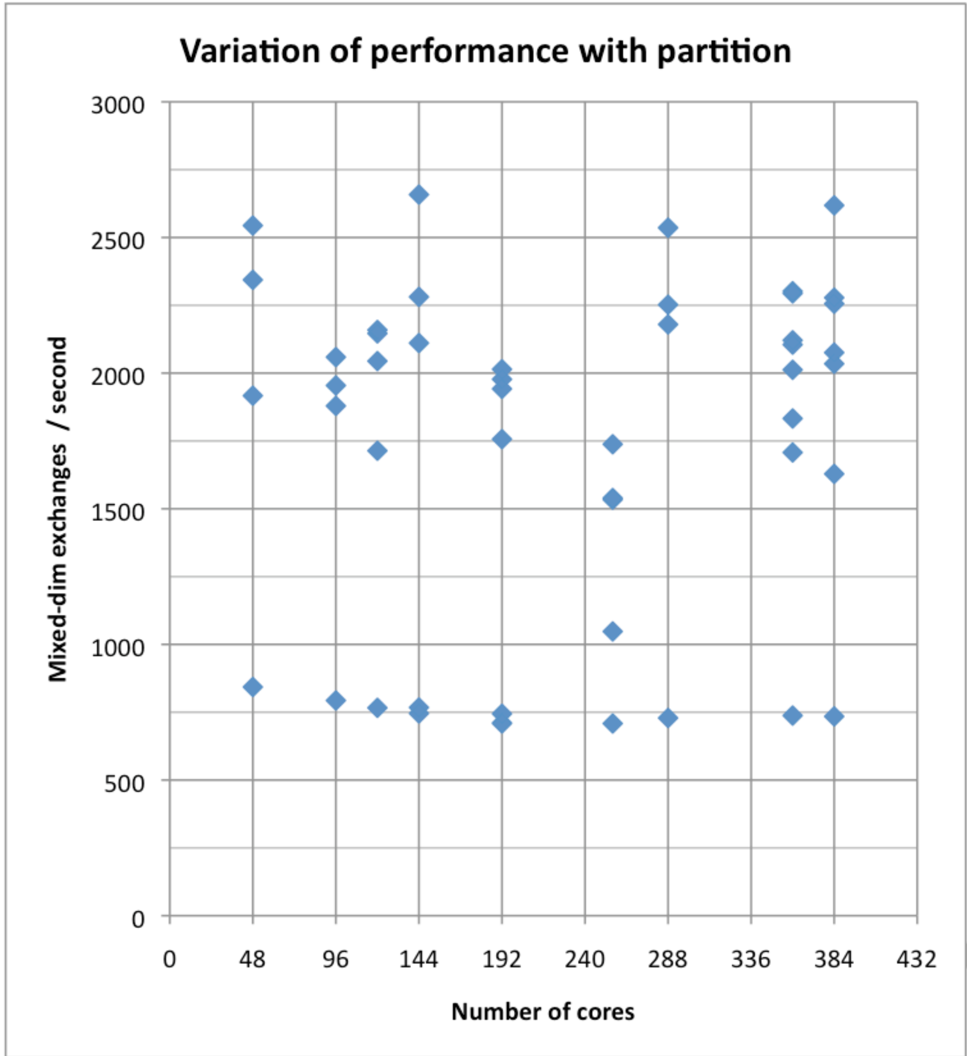
$$t \propto \max\left(c_{wet} n_{wet} + c_{dry} n_{dry} + c_{off} n_{off} + c_{on} n_{on}\right)$$

- Computation time is dominated by wet points.
  - Small overhead from dry points
- Communications time is dominated by halo exchange
- Overall run-time limited by the slowest MPI process
  - Maximum is taken over processes
- This form neglects latency
  - Latency could (and should) be added in easily enough
- The c* are tunable coefficients
  - Careful tuning is work-in-progress. I used, somewhat arbitrarily:

$$t \propto \max\left(n_{wet} + 0.05 \times n_{dry} + 5 \times n_{off} + n_{on}\right)$$

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

# Conclusions

- Message combination and dry-point elimination improves performance of halo exchange in ocean simulations
- Multi-core aware partitioning offers significant opportunities for performance and scalability improvement
  - Not doing so could lead to disappointment on systems with multiple 6-core chips/node

Science & Technology Facilities Council

# Acknowledgments

Thanks to:

- Swiss National Supercomputing Centre (CSCS) for time on Rosa (Cray XT5)
- NERC for time on HECToR
- Mike Ashworth, Andrew Porter, Kevin Roy and Jason Holt for helpful discussions

Science & Technology Facilities Council

Science & Technology Facilities Council

# The end