Science & Technology
Facilities Council

Developing hybrid OpenMP/MPI parallelism for Fluidity/ICOM

Xiaohu Guo, Gerard Gorman, Andrew Sunderland
ARC, CSE Department, STFC
AMCG, Department of Earth Science and Engineering,
Imperial College London

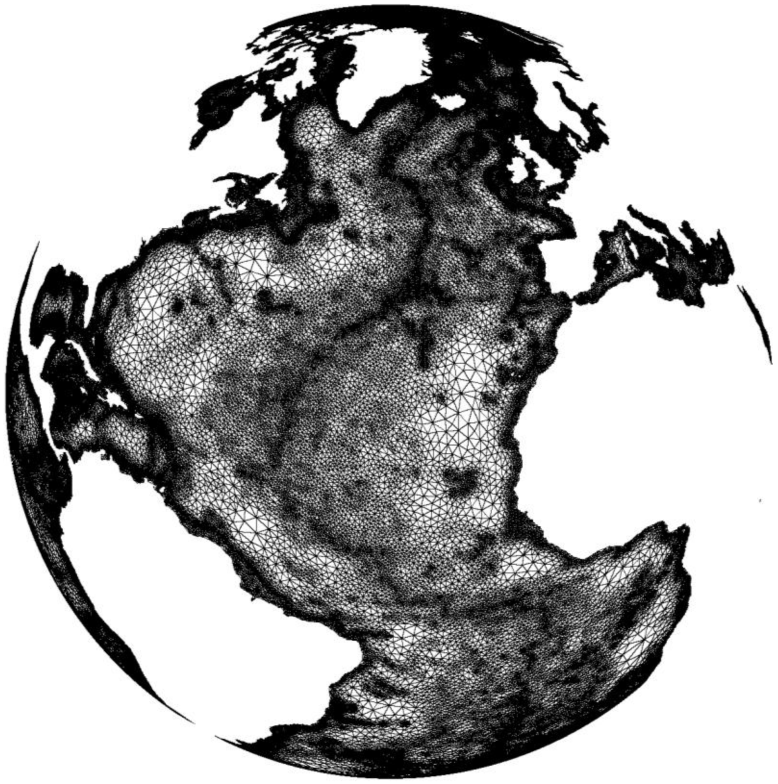
Science & Technology Facilities Council

Contents

- Introduction of current dCSE project
- A overview of the Implementation
- Case Study: Experiences and Results analysis
- Conclusions and Future work

Fluidity

- General purpose fluid dynamics framework
- Ocean modelling (Fluidity-ICOM)
- Features:
  - Unstructured mesh
  - Multiple discretisations, CG,DG,CV
  - MPI parallelized
    - Optimised for HECToR in previous dcse project
  - Adaptive mesh
  - User friendly interface
  - Fortran 90, C++, Python



Involved Groups

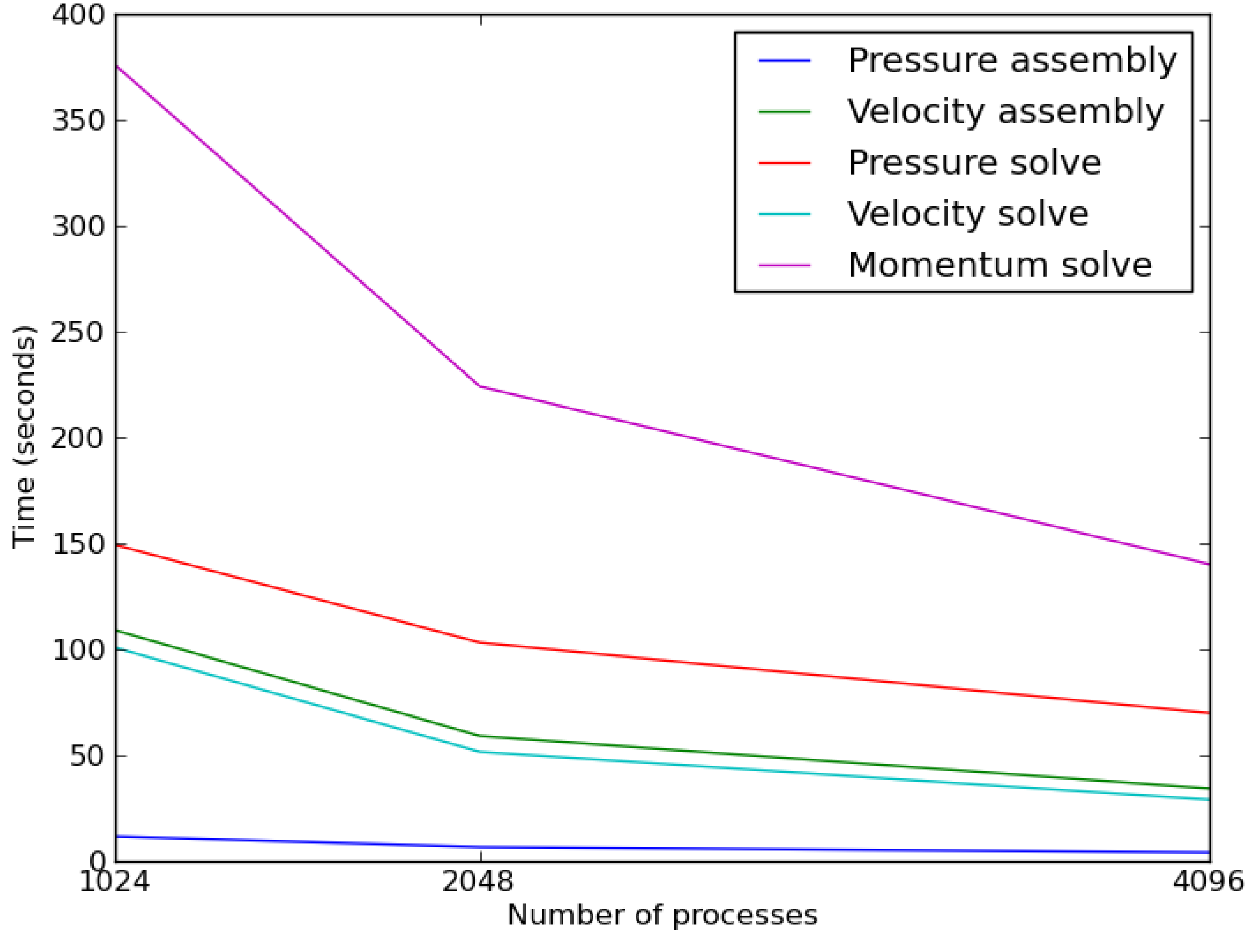
- The Applied Modelling and Computation Group(AMCG), Imperial College London
- Advance Research Computing Group(ARC), CSE Department, STFC.
- EPCC, Edinburgh
- NAG

Science & Technology Facilities Council

Project Objectives

- Further develop Fluidity-ICOM in order to run efficiently on supercomputers comprised of NUMA nodes.
- Hybrid OpenMP/MPI decreases the total memory footprint per compute node (the total size of mesh halos increases with number of partitions) and provides memory bandwidth optimization opportunities.
- The use of hybrid OpenMP/MPI will decrease the total volume of data to write to disk, and the total number of metadata operations based on the files-per-process I/O strategy.
- Reduced number of domain partitions benefits many algorithms, e.g. AMG, mesh adaptivity.

- Previous dCSE project has showed that the computational costs within Fluidity-ICOM are dominated by
  - sparse matrix assembly
  - the sparse linear preconditioners/solvers



Science & Technology Facilities Council

Fluidity ICOM Sparse Matrix Assembly

- Using element by element approach
- Sparse matrix storage formats:
  - CSR + diagonal
  - PETSC csr format
- Block assembly
- 30-40% of total computation due to higher order and DG integrations.

Science & Technology Facilities Council

An overview of the Implementation

- Working out sparse patterns (element adjacency matrix) for different numerical discretisation method, eg, DG, CG and CV

- Parallelize matrix assembly with colouring method, colouring elements according to their sparse patterns, a loop over colours is added around the main assembly loop.

- The main assembly loop over elements is parallelised using the OpenMP parallel do directive with a static schedule.

- This divides the loop into chunks of size ceiling (number_of_elements/number_of_threads) and assigns a thread to a separate chunk.

Science & Technology
Facilities Council

Pseudo Algorithm for Momentum DG assembly

```
!! generate the dual graph of the mesh
     p0_mesh = piecewise_constant_mesh(parent_mesh, "P0Mesh”)
!! the sparse pattern of the dual graph
    dependency_sparsity => get_csr_sparsity_XXX(state, p0_mesh, p0_mesh)
!! colouring dual graph according the sparsity pattern with greedy colouring algorithm
    call colour_sparsity(dependency_sparsity, p0_mesh, node_colour, no_colours)
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(clr, nnid, ele, len)
    colour_loop: do clr = 1, no_colours
     len = key_count(clr_sets(clr))
      !$OMP DO SCHEDULE(STATIC)
      element_loop: do nnid = 1, len
        ele = fetch(clr_sets(clr), nnid)
        call construct_momentum_element_dg( long parameter lists………..)
      end do element_loop
      !$OMP END DO
      !$OMP BARRIER
     end do colour_loop
  !$OMP END PARALLEL
```

Greedy Colouring Algorithm

1. Give an arbitrary ordering of the nodes
2. Find the maximum degree of the nodes, the maximum number of colours is maxdgr+1
3. Set the colouring number of the first point as 1
4. Colouring the remaining nodes with lowest unused colours
    1) Select some uncolored vertices and colour it with the new colour.
    2) Scan the list of uncoloured vertices. For each uncoloured vertex, determine whether it has an edge to any vertex already coloured with the new colour.
    3) If there is no such edge, colour the present vertex with the new colour.

Note:

- Generally, the above colouring method tries to colour as many vertices as possible with the first colour, then as many as possible of the uncoloured vertices with the second colour, and so on
- Therefore the number of elements is not balanced between each colour group.
- For OpenMP, it′s not a problem as long as each thread has enough work load.
- The performance is not sensitive to the total number of colour groups

Gyre Case Study

- The wind-driven baroclinic gyre benchmark: A gyre in oceanography is a large system of rotating ocean currents, particularly those involved with large wind movements
- Velocity Assembly: DG
- Temperature Assembly: CG
- Mesh nodes:
  - 10 million, resulting in 200 million degrees of freedom for velocity due to the use of DG
  - 28560

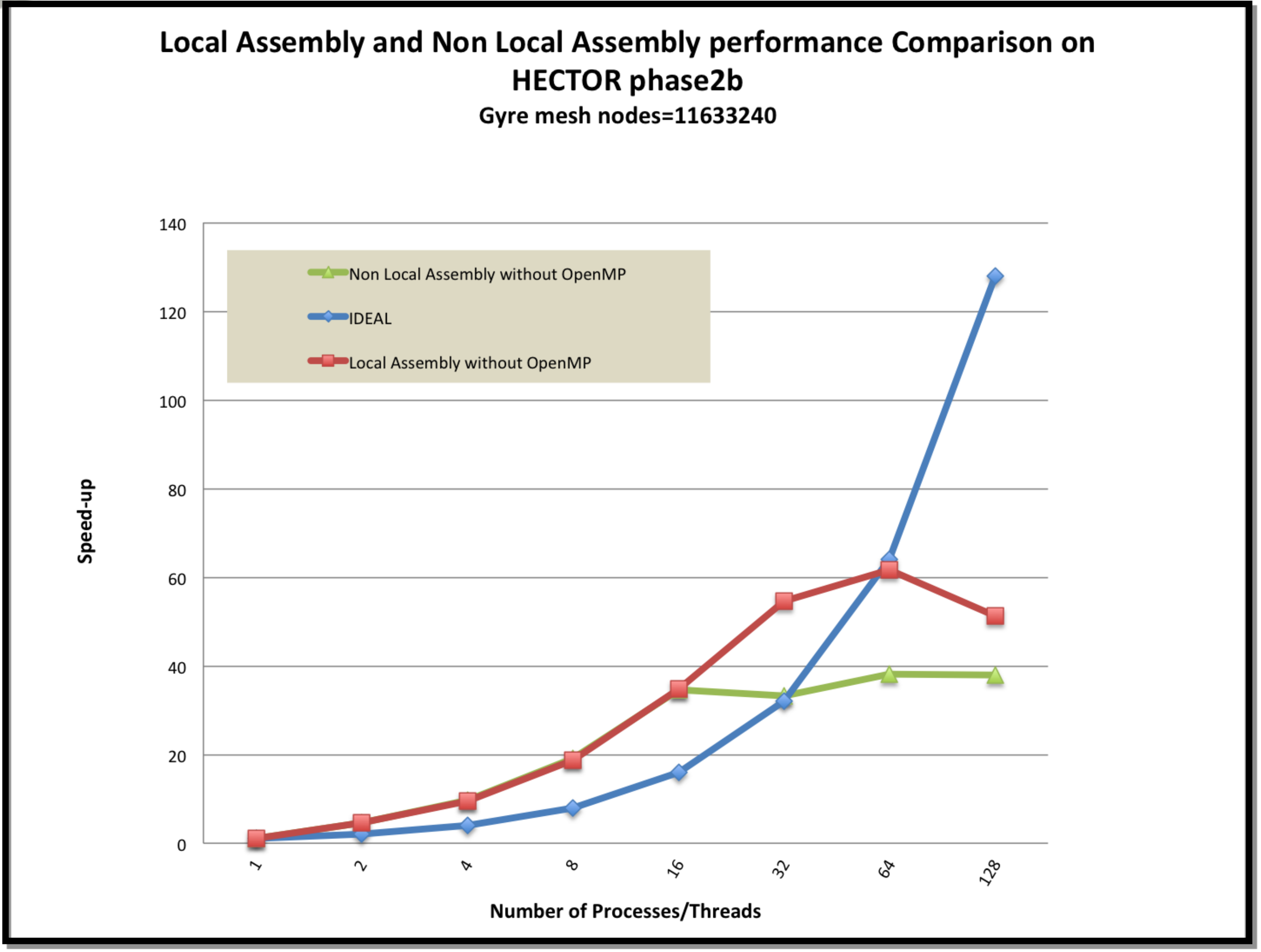
Local assembly v.s. non local assembly

- PETSC Matrix stashing: The stash is used to temporarily store inserted vec values that belong to another processor. During the assembly phase the stashed values are moved to the correct processor -- not thread safe

- When **MAT_IGNORE_OFF_PROC_ENTRIES** is set, any **MatSetValues** calls to rows that are off-process will be discarded. This makes matrix assembly much faster as no communications are needed -- recompute rather than communicate

Science & Technology Facilities Council

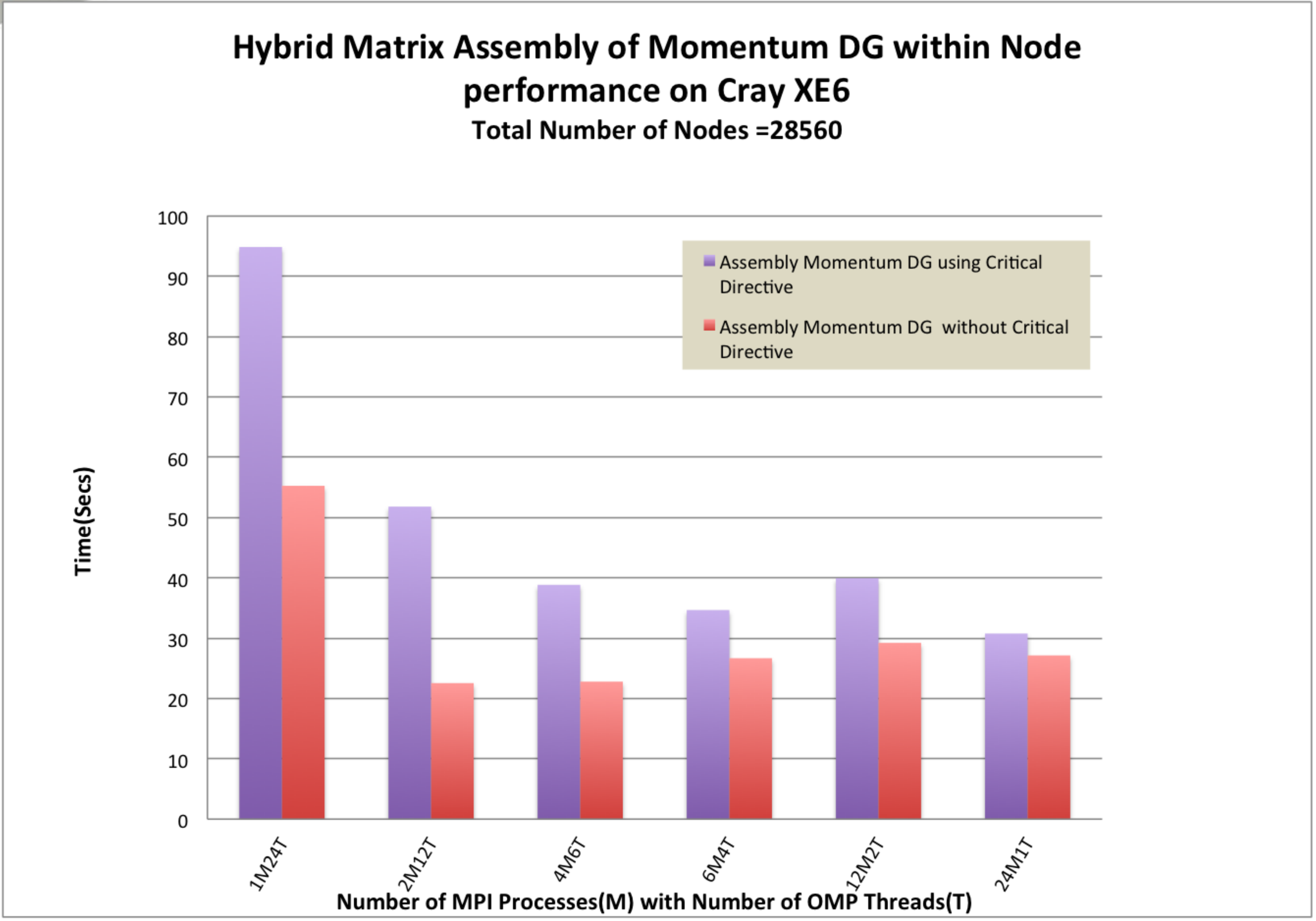
Local assembly v.s. nonlocal assembly

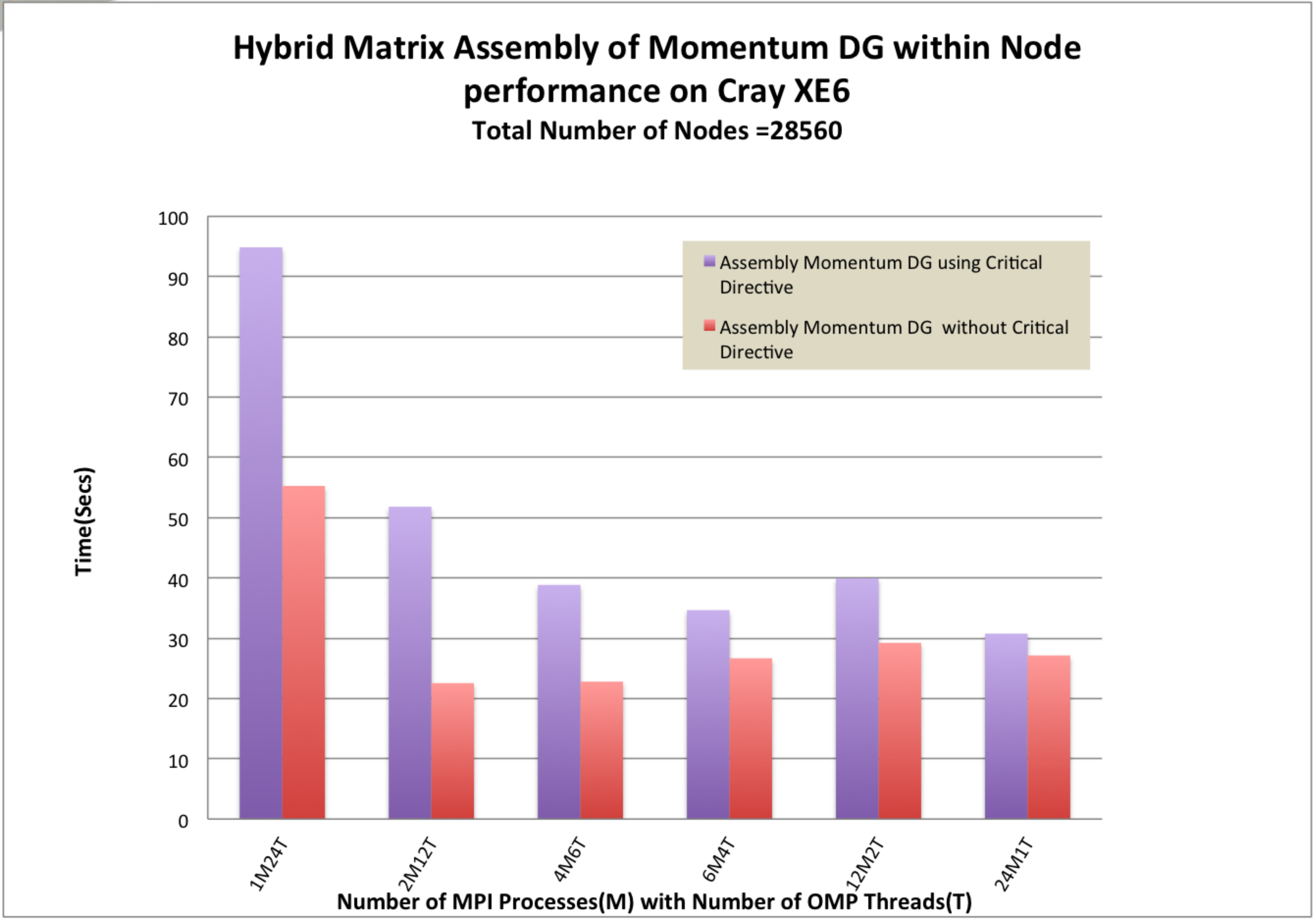


Science & Technology Facilities Council

Thread Safe Issues of Memory Reference Counting

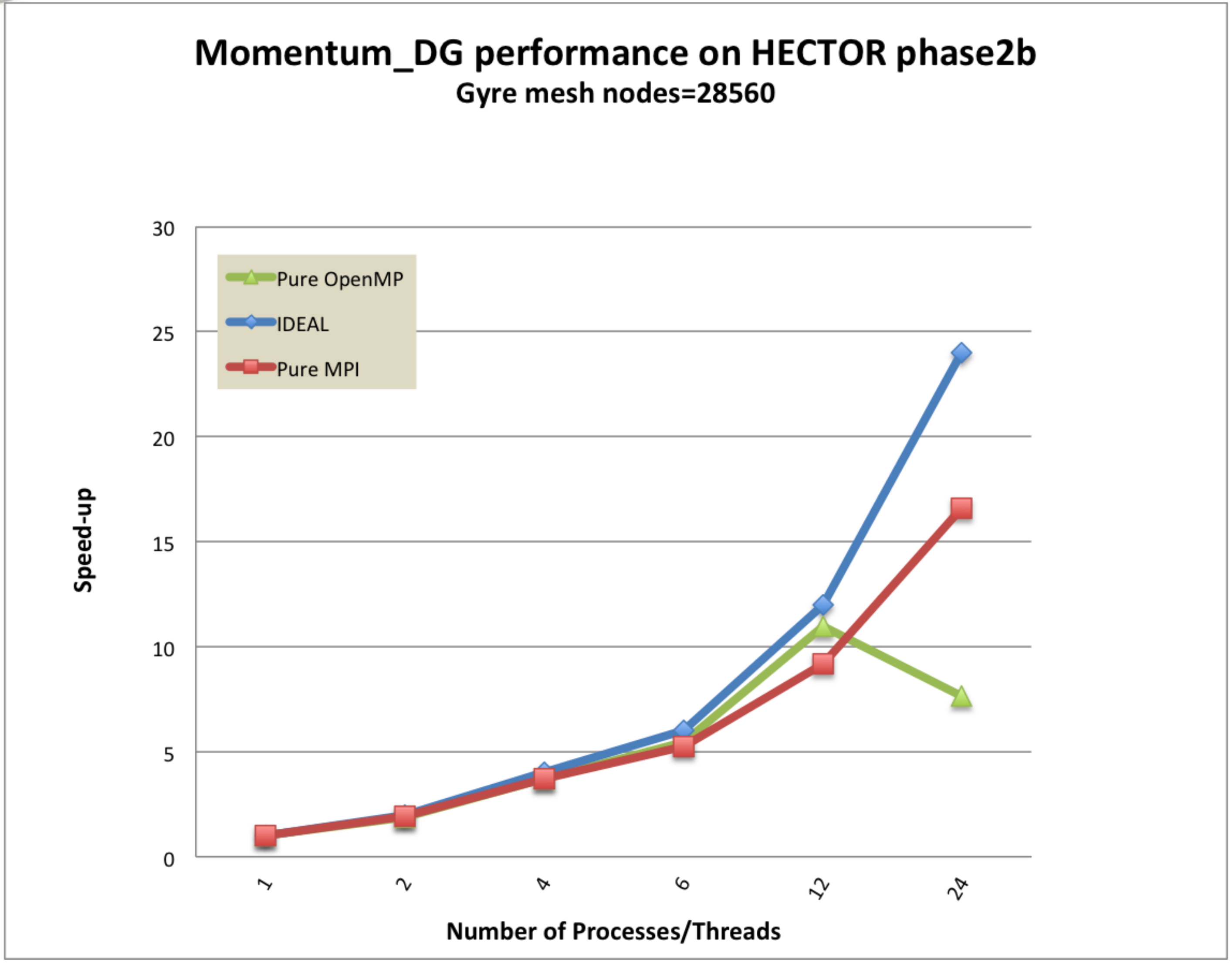
- Any defined type objects in fluidity being allocated or deallocated, the reference count will be plus one or minus one.
- If the objects counter equals zero, the objects should then be deallocated.
- In the element loop, the element-wise physical quantities should not do allocation or deallocation. (But they do, which causes racing conditions for the reference counter.)
- Solutions for the above,
  - add critical directives around reference counter.
  - Move allocation or deallocation outside of element loop


Hybrid Matrix Assembly of Momentum DG within Node performance on Cray XE6
Total Number of Nodes =28560
Assembly Momentum DG using Critical Directive
Assembly Momentum DG without Critical Directive
Time(Secs)
0
10
20
30
40
50
60
70
80
90
100
1M24T
2M12T
4M6T
6M4T
12M2T
24M1T
Number of MPI Processes(M) with Number of OMP Threads(T)


Hybrid Matrix Assembly of Momentum DG within Node performance on Cray XE6
Total Number of Nodes =28560
Time(Secs)
100
90
80
70
60
50
40
30
20
10
0
Assembly Momentum DG using Critical Directive
Assembly Momentum DG without Critical Directive
1M24T
2M12T
4M6T
6M4T
12M2T
24M1T
Number of MPI Processes(M) with Number of OMP Threads(T)

Science & Technology Facilities Council

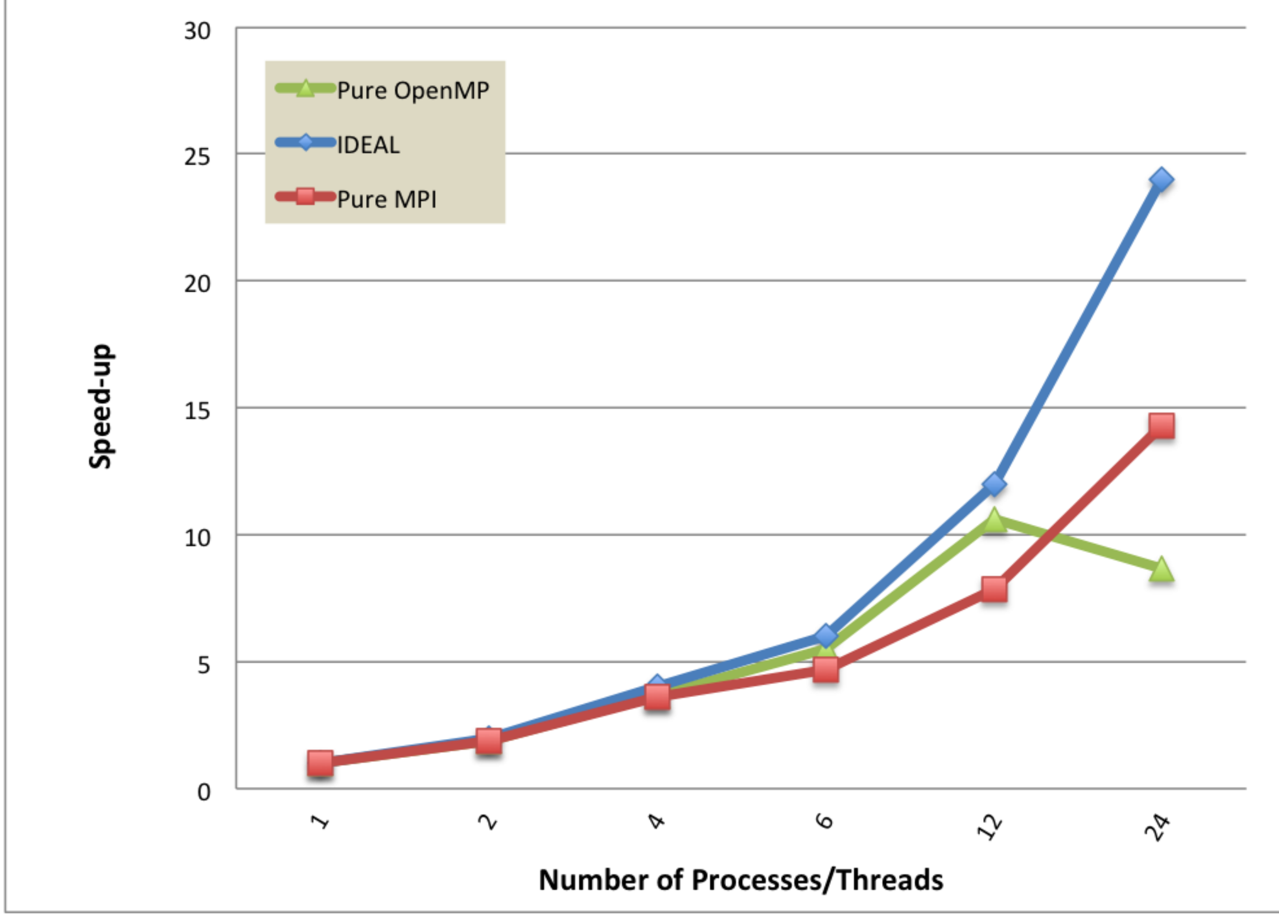
Pure MPI v.s. Pure OpenMP within node



Science & Technology Facilities Council

Advection Diffusion CG performance on HECTOR phase2b

Gyre mesh nodes=28560



Science & Technology Facilities Council

Tools for Thread error detecting

- DRD v.s. Helgrind
  - They both have a lot of false positives, also take very long time to generate results for fluidity typical runs(at least ten hours for gyre node performance test case).
  - Need pay a lot of attention to "store" operation, start with "Conflicting Store by…"
  - Very helpful for detecting racing conditions.
- Intel Thread Checker.

- Initial value for variables in a subroutine
  - Integer :: counter =0 --- acquire the save attribute automatically
  - Integer :: counter

    Counter=0

A variable with save attribute in a recursive subprogam is shared by all instances of the subprogram, be careful ! Hard to detect !

Science & Technology Facilities Council

Conclusions

- We have focused on assembly.
  - Regarding Matrix stashing, local assembly performance is better than non local assembly, This makes assembly an inherently local process.
  - Thus focus is on optimizing local (to the compute node) performance, try to avoid use mutual synchronization directives: eg. critical
  - Above performance results indicate that node optimization can be done mostly using OpenMP with efficient colouring method

- We also noticed that the code performance is not sensitive to number of colour groups.
- Work for the future includes improving memory bandwidth usage through NUMA optimisations(eg: first touch) to investigate if we can get best performance using pure openmp within nodes.
- Colouring mesh instead of colouring sparsity.
  - There are only 4 possible stencils for different numerical discretized methods for each topological mesh.
  - It's better to have a 4 element array to save colouring information for each topological mesh
- More work on solvers.

Acknowledgements

- The authors would like to acknowledge the support of a HECToR distributed Computational Science and Engineering award.
- The authors would also like to thank the HECToR and NAG support team for their help throughout this work.
- The authors would also like to thank the EPCC collaborators for their valuable contribution

Science & Technology
Facilities Council

THANKS !