Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 61 (2013) 251 – 257

Procedia Engineering

www.elsevier.com/locate/procedia

Parallel Computational Fluid Dynamics Conference (ParCFD2013)

Exploring the thread-level parallelisms for the next generation geophysical fluid modelling framework Fluidity-ICOM

Xiaohu Guo[a,*], Gerard Gorman[b], Michael Lange[b], Lawrence Mitchell[c], Michèle Weiland[c]

[a] *Science and Technology Facilities Council, Daresbury Laboratory, Daresbury Science and Innovation Campus, Warrington, WA4 4AD, UK*
[b] *Department of Earth Science and Engineering, Imperial College London,London SW7 2AZ, UK*
[c] *EPCC, The University of Edinburgh, Edinburgh, UK*

Abstract

In this paper, we highlight our progress in implementing a hybrid OpenMP-MPI version of the unstructured finite element application Fluidity-ICOM. We demonstrate that utilising non-blocking algorithms and libraries are critical to mixed-mode application so that it can achieve better parallel performance than the pure MPI version. In the matrix assembly kernels, the OpenMP parallel algorithm utilises graph colouring to identify independent sets of elements that can be assembled simultaneously with no race conditions. The TCMalloc are used here to tackle performance issues arising from automatic arrays memory allocations. The sparse linear systems defined by various equations are solved by using threaded PETSc and HYPRE is utilised as a threaded preconditioner through the PETSc interface. Since unstructured finite element codes are well known to be memory bound, particular attention has to be paid to ccNUMA architectures where data locality is particularly important to achieve good intra-node scaling characteristics. With mixed mode MPI/OpenMP, Fluidity-ICOM can now run well above 32K cores job, which offers Fluidity-ICOM capability to solve the "grand-challenge" problems.

Selection and peer-review under responsibility of the Hunan University and National Supercomputing Center in Changsha (NSCC)
Keywords: Fluidity-ICOM, OpenMP, MPI, HYPRE, PETSc, TCMalloc, NUMA

1. Introduction

Fluidity-ICOM (Imperial College Ocean Model) has the capability to efficiently resolve a wide range of scales simultaneously. This offers the opportunity to simultaneously resolve both basin-scale circulation and small-scale processes such as boundary currents, through-flows, overflows and geostrophic eddies. For Earth system and climate modellers, the underlying numerical software technology offers the opportunity to focus resolution in regions of particular importance, such as boundary currents and overflows, without increasing the computational cost above that of a conventional coarse-resolution model. As the climate change research agenda moves to addressing impacts and considering how to adapt to them, fully integrated models that can address interactions between global and localised phenomena will need to be developed. Therefore, there is an urgent need to improve both their fidelity and their capabilities to provide more confident assessments at small, regional and global scales.

* Dr. Xiaohu Guo. Tel.: +44-1925-603836 ; fax: +44-1925-603634.
E-mail address: xiaohu.guo@stfc.ac.uk

Selection and peer-review under responsibility of the Hunan University and National Supercomputing Center in Changsha (NSCC)
doi:10.1016/j.proeng.2013.08.012

Fluidity-ICOM is built on top of Fluidity[1], an adaptive unstructured finite element code for computational fluid dynamics. It consists of a three-dimensional non-hydrostatic parallel multiscale ocean model, which implements various finite element and finite volume discretisation methods on unstructured anisotropic adaptive meshes so that a very wide range of coupled solution structures may be accurately and efficiently represented in a single numerical simulation without the need for nested grids. It is used in a number of different scientific areas including geophysical fluid dynamics, computational fluid dynamics, ocean modelling and mantle convection. Fluidity-ICOM uses state-of-the-art and standardised 3rd party software components whenever possible. For example, PETSc[2] is used for solving sparse linear systems while Zoltan[3] is used for many critical parallel data-management services both of which have compatible open source licenses. Python is widely used within Fluidity-ICOM at run time for user-defined functions and for diagnostic tools and problem setup. It requires in total about 17 other third party software packages and use three languages (Fortran, C++, Python). Fluidity-ICOM is coupled to a mesh optimisation library allowing for dynamic mesh adaptivity.

The trend for HPC architectures is towards large number of lower frequency cores with a decreasing memory to core ratio. This is imposing a strong evolutionary pressure on numerical algorithms and software to efficiently utilise the available memory and network bandwidth. Using modern multi-core processors presents new challenges for scientific software such as Fluidity-ICOM, due to the new node architectures: multiple processors each with multiple cores, sharing caches at different levels, multiple memory controllers with affinities to a subset of the cores, as well as non-uniform main memory access times.

For modern multi-core architecture supercomputers, hybrid OpenMP/MPI also offers new possibilities for optimisation of numerical algorithms beyond pure distributed memory parallelism. For example, scaling of algebraic multi-grid methods is hampered when the number of subdomains is increased due to difficulties coarsening across domain boundaries. The scaling of mesh adaptivity methods is also adversely effected by the need to adapt across domain boundaries.

Portability across different systems is very critical for application software packages, and the directives based approach is a great way to express parallelism in a portable manner. It offers potential capabilities to use the same code base to explore accelerated and non-accelerator enabled systems because OpenMP is expanding its scope to embedded systems and accelerators.

Because of this, there is a growing interest in hybrid parallel approaches where threaded parallelism is exploited at the node level, while MPI is used for inter-process communications. Significant benefits can be expected from implementing such mixed-mode parallelism. First of all, this approach decreases the memory footprint of the application as compared with a pure MPI approach. Secondly, the memory footprint is further decreased through the removal of the halo regions which would be otherwise required within the node. For example, the total size of the mesh halo increases with number of partitions (i.e. number of processes). It can be shown empirically that the size of the vertex halo in a linear tetrahedral mesh grows as $O(P^{1.5})$, where P is the number of partitions. Finally, only one process per node will be involved in I/O (in contrast to the pure MPI case where potentially 32 processes per compute node could be performing I/O), which will significantly reduce the number of meta data operations on the file system at large process counts for those applications based on files-per-processes I/O strategy. Therefore, the use of hybrid OpenMP/MPI will decrease the total memory footprint per compute node, the total volume of data to write to disk, and the total number of meta data operations given Fluidity-ICOMs files-per-process I/O strategy.

In this paper, we highlight our progress in implementing a hybrid OpenMP-MPI version of the unstructured finite element application Fluidity-ICOM. We demonstrate that utilising non-blocking algorithms and libraries are critical to mixed-mode application so that it can achieve better parallel performance than the pure MPI version. In the matrix assembly kernels, the OpenMP parallel algorithm utilises graph colouring[1] to identify independent sets of elements that can be assembled simultaneously with no race conditions. The TCMalloc are used here to tackle performance issues arising from automatic arrays memory allocations. The sparse linear systems defined by various equations are solved by using threaded PETSc [3, 2] and HYPRE is utilised as a threaded preconditioner [4] through the PETSc interface. Since unstructured finite element codes are well known to be memory bound, particular attention has to be paid to ccNUMA architectures where data locality is particularly important to achieve good intra-node scaling

[1] http://amcg.ese.ic.ac.uk/index.php?title=Fluidity
[2] http://www.mcs.anl.gov/petsc
[3] http://www.cs.sandia.gov/Zoltan/

characteristics. We highlight the most common pitfalls and describe the solutions, which were also used in the process of threading and benchmarking the whole Fluidity-ICOM. With a full implementation of mixed mode MPI/OpenMP and previous several years efforts [5, 6], Fluidity-ICOM can now run well above 32K cores job, which offers Fluidity-ICOM the capability to solve the ”grand-challenge” problems.

2. MPI/OpenMP mixed-mode parallelisation of the Finite Element Matrix Assembly

Previous performance analysis [5] has already shown that the two dominant simulation costs are sparse matrix assembly (30%-40% of total computation), and solving the sparse linear systems defined by these equations. The HYPRE librarys hybrid sparse linear system solvers/preconditioners, which can be used by Fluidity-ICOM through the PETSc interface, are competitive with the pure MPI implementation. Therefore, in order to run a complete simulation using OpenMP parallelism, the sparse matrix assembly kernel is now the most important component remaining to be parallelised using OpenMP. The finite element matrix assembly kernel is expensive for a number of reasons including: significant loop nesting, where the innermost loop increases in size with increasing quadrature; many matrices have to be assembled, e.g. coupled momentum, pressure, free-surface and one of each advected quantity; indirect addressing (a known disadvantage of finite element codes compared to finite difference codes); and cache re-use (a particularly severe challenge for unstructured mesh methods). The cost of matrix assembly increases with higher order if discontinuous Galerkin (DG) discretisations are used.

For a given simulation, a number of different matrices need to be assembled, e.g. continuous and discontinuous finite element formulations for velocity, pressure and tracer fields for the Navier-Stokes equations and Stokes flow. Each of these have to be individually parallelised using OpenMP. The global matrix to be solved is formed by looping over all the elements of the mesh (or sub-domain if this is using a domain decomposition method) and adding the contributions from that element into the global matrix. Sparse matrices are stored in PETSc's CSR (Compressed sparse row) containers (these includes block-CSR for use with velocity vectors, for example, and DG). The element contributions are added into a sparse matrix which is stored in CSR format.

In order to thread the assembly loop, it is clear that both the operation assembles an element into a local matrix, and the addition of that local matrix into the global matrix must be thread safe. This can be implemented in two different approaches. The first is typically through locks, if two threads assemble into same position of matrix, it will be serialized. In the OpenMP implementation, it generally use critical directive or atomic. However, OpenMP atomic can only deal with single variables, the only option here is using critical directive. The Reference[6] gives the evaluation of the OpenMP critical directive overhead, which suggest we should avoid using the critical directive. The second is using graph colouring to break data dependencies which is one of the typical non-blocking algorithms.

2.1. Non-blocking finite element matrix assembly approach

Non-blocking finite element matrix assembly can be realised through well-established graph colouring techniques[7]-[11]. This is implemented by first forming a graph, where the nodes of the graph correspond to mesh elements, and the edges of the graph define data dependencies arising from the matrix assembly between elements. Each colour then defines an independent set of elements whose term can be added to the global matrix concurrently. This approach removes data contention, so called critical sections in OpenMP, allowing very efficient parallelisation.

To parallelise matrix assembly using graph colouring techniques, a loop over colours is first added around the main assembly loop. The main assembly loop over elements will be parallelised using the OpenMP parallel directives with a static schedule. This will divide the loop into chunks of size ceiling **(number_of_elements/number_of_threads)** and assign a thread to each separate chunk. Within this loop an element is only assembled into the matrix if it has the same colour as the colour iteration. The details can be seen in the reference [6].

3. MPI/OpenMP mixed-mode parallelisation of the Sparse Linear solvers

Fluidity-ICOM use PETSc for solving sparse linear systems. Many other scalable preconditioner/solvers can be called through PETSc interface, eg: HYPRE. Previous studies [5] have already shown that Fluidity-ICOM spends the majority of it's run time in sparse iterative linear solvers. This paper mainly investigate BoomerAMG from HYPRE as preconditioner as it has been fully threaded.

3.1. Threaded HYPRE BoomerAMG Preconditioner

BoomerAMG has two phases: setup and solve. The primary computational kernels in the setup phase are the selection of the coarse grids, creation of the interpolation operators, and the representation of the fine grid matrix operator on each coarse grid. The primary computational kernels in the solve phase are a matrix-vector multiply (MatVec) and the smoothing operator, which may closely resemble a MatVec.

For most basic matrix and vector operations, such as MatVec and dot product, OpenMP parallelisation has been applied at the loop level. In the setup phase, only the generation of the coarse grid operator (a triple matrix product) has been threaded. Both coarsening and interpolation do not contain any OpenMP statements. The solve phase (MatVec and the smoothing operator) has been completely threaded [4].

3.2. PETSc OpenMP branches

PETSc is a widely used library for the scalable solution of partial differential equations. It consists of a series of classes that implement the different components required for linear algebra in separate classes: Index Sets, Vectors and Matrices; Krylov Subspace Methods and Pre-conditioners; and Non-linear Solvers and Time Steppers. The Vector and Matrix classes represent the lowest level of abstraction and are the core building blocks of most of the functionalities.

In this paper we use a separate PETSc branch where a number of low-level matrix and vector operators in the Mat and Vec classes have been threaded using OpenMP. In particular, task-based parallelism and an explicit thread-level load-balancing scheme are utilised in this branch to optimise the parallel scalability of sparse matrix-vector multiplication [2]. However, the Krylov subspace methods and the pre-conditioners implemented in the KSP and PC classes have not been threaded explicitly since the basic algorithms, like CG and GMRES and SOR preconditioners, are based on functionality from the Mat and Vec classes, which have been threaded. Other frequently used preconditioners, such as Symmetric Over-Relaxation (SOR) or Incomplete LU-decomposition (ILU), have not been threaded yet due to their complex data dependencies. These may require a redesign of the algorithms to improve parallel efficiency [3].

4. Benchmarking and performance analysis

The benchmarking and performance tests were carried on the UK National HPC platform HECToR, which is a Cray XE6 system. It offers a total of 2816 XE6 compute nodes. Each compute node contains two AMD 2.3 GHz 16-core processors giving a total of 90, 112 cores; offering a theoretical peak performance of over 800 Tflops. There is presently 32 GB of main memory available per node, which is shared between its thirty-two cores, the total memory is 90 TB. The processors are connected with a high-bandwidth interconnect using Cray Gemini communication chips. The Gemini chips are arranged on a 3 dimensional torus.

The lock exchange test case has been used here. The lock exchange is classic CFD test problem. A lock separates two fluids of different densities (e.g. hot and cold) inside a tank; when the lock is removed, two gravity currents propagate along the tank. An up to 18.7 million vertices mesh, resulting in 448.8 million degrees of freedom for velocity, has been used for the scalability analysis on large number of cores. The benchmark starts with 2 HECToR Interlargos nodes and scales up to 1024 nodes (32768 cores). The speedup are obtained with the following formula:

$$S_p = T_l / T_p \tag{1}$$

Where T_l is the wall time with minimum l nodes required to run the test case, each node comprises 32 AMD Interlagos cores, T_p is the wall time with p nodes($P \geq l$).

4.1. Performance consideration of NUMA architecture

Many multi-core machines are typically Cache coherent Non-Uniform Memory Architectures (ccNUMA). This means that memory latency (access time) depends on the physical memory location relative to a processor. Within a ccNUMA system, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors, Therefore, it is important that memory should be allocated from mapped local memory nodes, so that software running on the machine may take advantage of ccNUMA locality

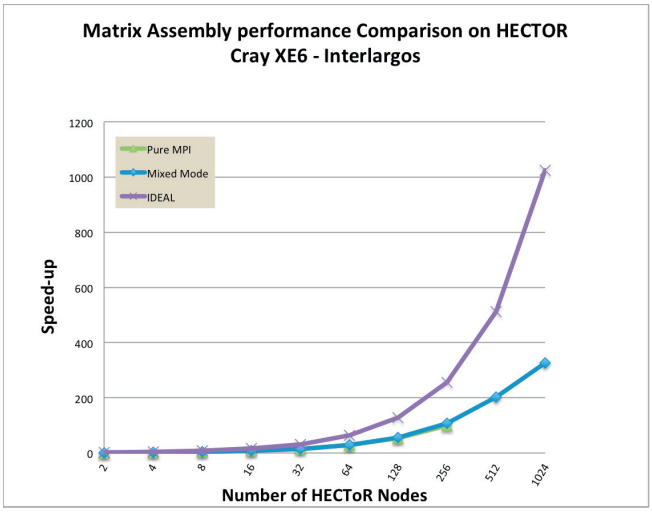

Fig. 1: Matrix Assembly Performance Comparison on HECToR XE6-Interlagos

and therefore have a optimal utilisation of memory bandwidth. To achieve this, the following methods have been employed to ensure good performance:

- First-touch initialisation ensures that page faults are satisfied by the memory bank directly connected to the CPU that raises the page fault.
- Thread pinning to ensure that individual threads are bound to the same core throughout the computation.

5. Final benchmark and performance analysis

Based on our previous efforts in developing thread-level parallelism in finite element matrix assembly kernels [6] and the PETSc linear solver library [2], we are now able to run Fluidity-ICOM in full MPI/OpenMP mixed-mode.

Fig. 1 shows that matrix assembly scales well up to 32768 cores. The speedup of the mixed-mode implementations is 107.1 compared with 99.3 for pure MPI when using 256 nodes (8192 cores). This is due to the use of local assembly which makes this part of the code essentially a local process. The hybrid mode performs slightly better than pure MPI, which can scale well up to 32768 cores.

We have also compared the mixed-mode and the pure MPI version for the pressure solve. The preconditioner HYPRE BoomerAMG and solver Conjugate Gradient are used here for solving the Poisson equation. Fig. 2 indicates that pure MPI performance starts to degrade from 64 HECoR nodes (2048 cores) onwards where the hybrid mode begins to outperform pure MPI. However, we also observed that the mixed-mode of is slower than the pure MPI version for the momentum equation solve. This is simply because we are using GMRES with Eisenstat/B-Jacobi as the preconditioner, which has not been fully threaded yet.

Fig. 3 shows the whole Fluidity-ICOM total run-time and efficiency. Clearly pure MPI runs faster up to 2048 cores. However, due to the halo size increasing exponentially with number of MPI tasks, the cost of MPI communication becomes dominant from 4K cores onwards, where the mixed mode begins to out perform the pure MPI version.

Furthermore, for high core count simulation, the I/O becomes major bottleneck for pure MPI version. Significant efficiency of I/O operations based on files-per-process strategy has been achieved by using mixed mode parallelism.

6. Summary and Conclusions

As the current OpenMP standard (3.0), which has been implemented by most popular compilers, doesn't cover page placement at all, memory-bound applications, like Fluidity-ICOM, require explicit memory placement. Hereby

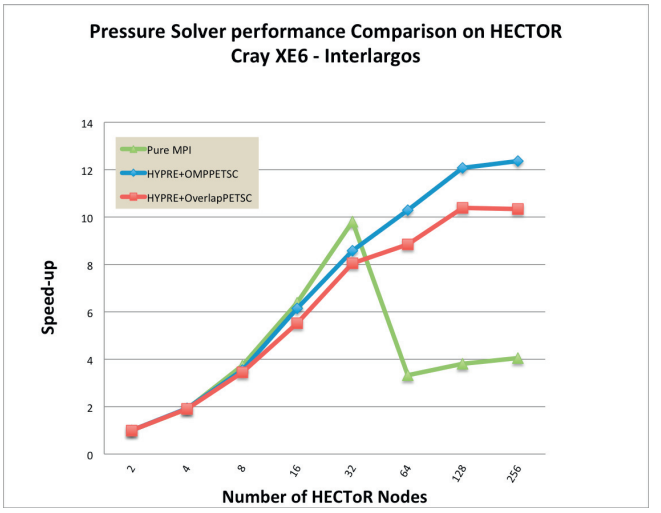

Fig. 2: Pressure Solver Performance Comparison on HECToR XE6-Interlagos

it is important to make sure that memory gets mapped into the locality domains of processors that actually accesses the data. This was achieved by implementing a first touch policy to minimize NUMA traffic across the network. Thread pinning was then used to guarantee that threads are bound to physical CPUs and maintain locality of data access.

The performance of mixed mode finite element matrix assembly kernels are generally better than pure MPI version with using non-block algorithms described above. This offers us good guidance for the future development of scalable FEM software applications.

For pressure solver, using threaded HYPRE BoomerAMG and CG and the threaded PETSc branch provides better scalibility than the pure MPI version. For other solvers, such as the momentum equation solver, pure MPI version is generally faster, due to non-threaded precondioners. However, we have shown that the overall performance of mixed mode is generally better than pure MPI version when using large number of cores.

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/NAG support team for their help throughout this work. The author would also thanks to his colleagues, Dr. Charles Moulinec, Dr. Stephen Pickles, Dr. Andrew Porter and Dr. Andrew Sunderland for their valuable suggestions and discussions.

References

[1] Welsh, D. J. A.; Powell, M. B., *An upper bound for the chromatic number of a graph and its application to timetabling problems*, *The Computer Journal*, 10(1):8586, 1967 doi:10.1093/comjnl/10.1.85

[2] Michael Lange, Gerard Gorman, Michele Weiland, Lawrence Mitchell, James Southern, *Achieving efficient strong scaling with PETSc using Hybrid MPI/OpenMP optimisation*, *submitted to ISC'13*, 2013

[3] M. Weiland, L. Mitchell, G. Gorman, S. Kramer, M. Parsons, and J. Southern, *Mixed-mode implementation of PETSc for scalable linear algebra on multi-core processors*, *In Proceedings of CoRR*. 2012.

[4] A.H. Baker, M. Schulz and U. M. Yang, *On the Performance of an Algebraic Multigrid Solver on Multicore Clusters*, *in VECPAR 2010, J.M.L.M. Palma et al., eds., vol. 6449 of Lecture Notes in Computer Science, Springer-Verlag* (2011), pp. 102-115

[5] Xiaohu Guo, G. Gorman, M Ashworth, S. Kramer, M. Piggott, A. Sunderland, *High performance computing driven software development for next-generation modelling of the Worlds oceans*, *Cray User Group 2010: Simulation Comes of Age (CUG2010)*, Edingburgh, UK, 24th-27th May 2010

[6] Xiaohu Guo, G. Gorman, M Ashworth, A. Sunderland, *Developing hybrid OpenMP/MPI parallelism for Fluidity-ICOM - next generation geophysical fluid modelling technology*, *Cray User Group 2012: Greengineering the Future (CUG2012)*, Stuttgart, Germany, 29th April-3rd May 2012

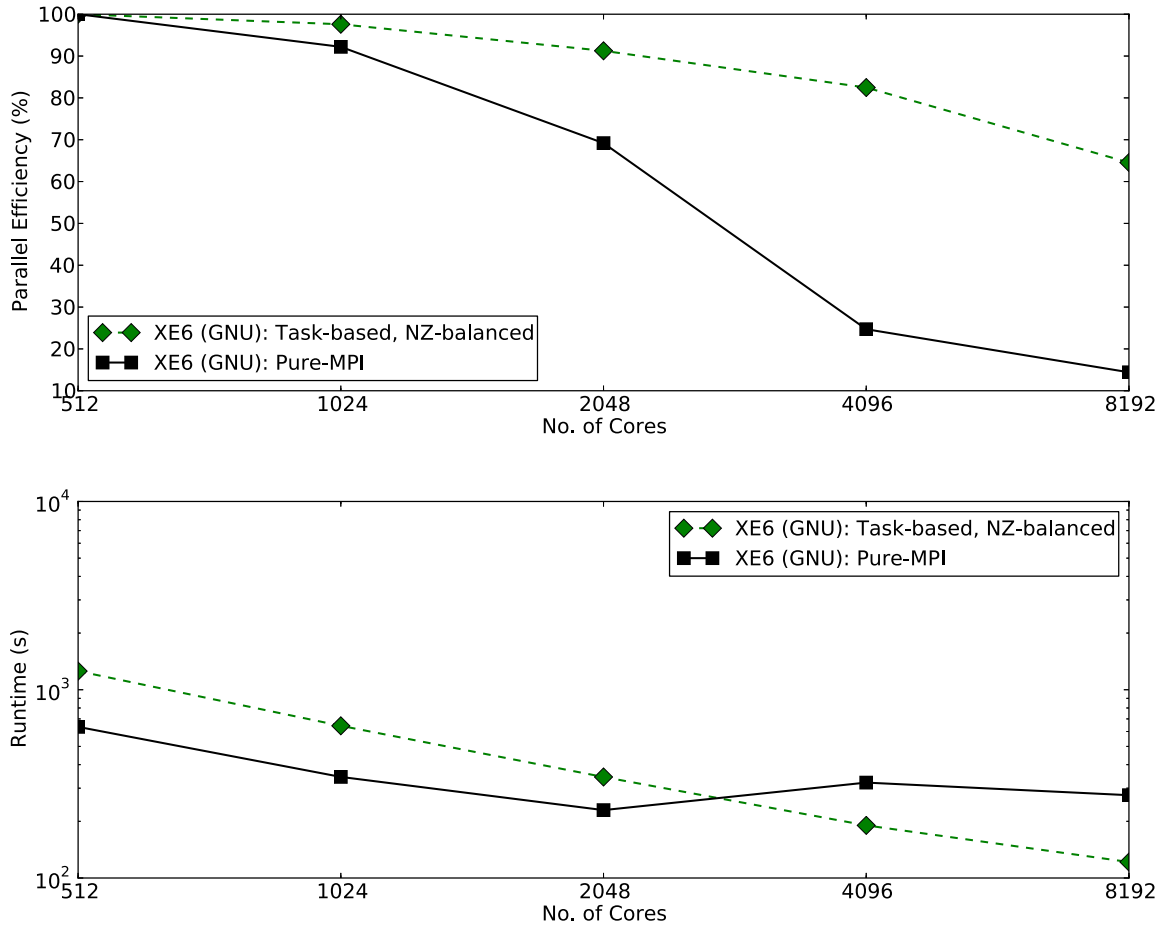

Fig. 3: Strong scaling results for the whole Fluidity-ICOM up to 256 XE6 nodes (8192 cores). All hybrid modes use 4 MPI ranks per node and 8 threads per rank.

[7] P. Berger, P. Brouaye, J.C. Syre, *A mesh coloring method for efficient MIMD processing in finite element problems*, in: *Proceedings of the International Conference on Parallel Processing*, ICPP'82, August 24-27, 1982, Bellaire, Michigan, USA, IEEE Computer Society, 1982, pp. 41-46.

[8] T.J.R. Hughes, R.M. Ferencz, J.O. Hallquist, *Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients*, *Comput. Methods Appl. Mech. Engrg*. 61(2), (1987a), 215-248.

[9] C. Farhat, L. Crivelli, *A general approach to nonlinear finite-element computations on shared-memory multiprocessors*, *Comput. Methods Appl. Mech. Engry*. 72(2), (1989), 153-171.

[10] D. Komatitsch, D. Michaa, G. Erlebacher, *Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA*, *J. Parallel Distrib. Comput*. 69, (2009), 451-460.

[11] C. Cecka, A.J. Lew, and E. Darve, *Assembly of Finite Element Methods on Graphics Processors*, *Int. J. Numer. Meth. Engng* 2000, 1-6.