ParCFD'2020
32^{nd} International Conference on Parallel Computational Fluid Dynamics
May-11-13 2020, Nice FRANCE

FLEXIBLE DATA STRUCTURES FOR SCALABLE CFD CODES ON EMERGING ARCHITECTURES

Francesco GAVA*, **Ghislain LARTIGUE*** **AND Vincent MOUREAU***

* CORIA
76081 Saint Étienne-du-Rouvray, France
e-mail: ghislain.lartigue@coria.fr

Key words: Data structures, Heterogeneous architectures, Versatility, Graph, Scalability

Abstract. Supercomputing hardware is becoming increasingly diversified. Multi-physics Computational Fluid Dynamics (CFD) codes must adapt to these new heterogeneous machines if large simulations are to be performed efficiently. We propose a flexible and lightweight graph-like data structure capable of decoupling different needs of the code, such as cache blocking and linear system solution. Furthermore this data structure can also improve parallel performances reducing communication time through multi-layer ghosts.

1 INTRODUCTION

Looking at the evolution of the Top500 [1], one can see that the heterogeneity of the architectures has increased dramatically. These machines have gone from having only mono-cores CPUs in the early 2000s to a completely diverse combinations of multi-core CPUs, MICs and GPUs in more recent years. The computational power has also grown and we are now on the verge of the so-called exascale era. Furthermore, the number of applications for these machines has increased exponentially, consequently they have become more generalist and it is harder to have codes that can fully take advantage of all these configurations. In an environment with such dissimilar architectures, multi-physics CFD codes need to have data structures that can adapt easily to any of these possible hardware in order to exploit them fully.
Here, we propose a flexible graph-like data structure whose objective is to make CFD codes more versatile, decoupling their different kernels and adapting them to the hardware on which they are executed.

2 MOTIVATION

Low-Mach number incompressible CFD codes usually use iterative linear solvers to find a solution to the Poisson's equation. These algorithms have typically an extremely low arithmetic intensity, consequently such codes find their performances often limited by memory accesses rather than computational power. The hardware memory hierarchy needs then to be exploited efficiently in order to obtain reasonable performances [2].
Organising the data in blocks that could fit the cache brings then non-negligible improvements to the code return time, consequently most CFD codes should have data structures

that exploit cache-blocking. Unfortunately, the memory structure of GPUs, for example, is different from that of ordinary CPUs and cache blocking is no longer useful and could even be detrimental. The above mentioned data structure must then be versatile enough to adapt to this two different scenarios, possibly with optimal performances in both cases. The code used in this work is the parallel LES solver YALES2. Its entire data structure is based on a double domain decomposition (DDD): first the grid is divided in sub-domains, one for each process, then each sub-domain is split again in groups of element whose size is sufficiently small for the data to fit into cache memory [3].
To solve the momentum equation, YALES2 uses a prediction-correction method for the velocity, which implies the solution of a Poisson's equation for the pressure. A Deflated Preconditioned Conjugate Gradient (DPCG) algorithm is used to solve the resulting linear system [4]. The DPGC is a multi-grid method: the Poisson's equation is solved first on an auxiliary coarser grid and then this solution is used to precondition the system on the main grid. Usually convergence is found with only a few steps on the fine grid, however the number of iterations on the coarser one can become extremely large. Performances are consequently driven by the solution of the deflation system.
In YALES2, the deflation grid is also based on the DDD: as shown in Figure 1, each node of the coarse grid corresponds to a group of elements. The RHS and operators on the

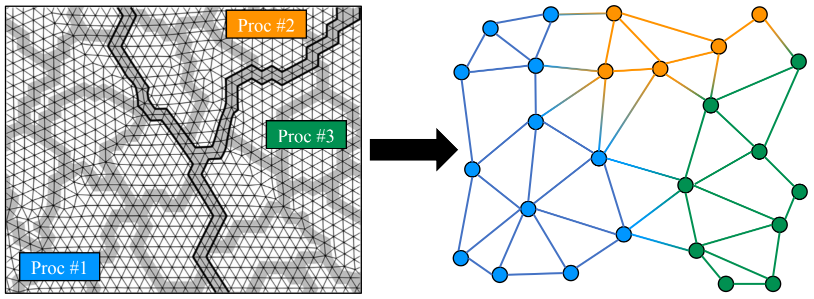

Figure 1: Example of DDD with groups limited by grey areas (left) and corresponding deflation grid/graph (right)

auxiliary grid are computed from those on the main one to build an equivalent linear system: for each data, the values on the nodes of each group on the fine grid are interpolated to give the value for that group on the coarse grid, and then, once the solution on the deflation grid is found, the value on each group is projected back on all its nodes on the fine grid.
In order for the entire data needed for the resolution of the system to fit in a L2 cache of 512kB, a group should be made of a few hundred elements. However, our tests have shown that on realistic test cases, the best performances of the linear solver are obtained for much larger group sizes, typically between 5000 and 7500 elements: the resolution of the linear system on a coarser grid converges faster, but a further increase in the groups size is detrimental to the preconditioning of the main linear system.
The code has then two incompatible performance constraints: the cache blocking asks for rather small groups, while the Poisson's solver needs coarser meshes. It is important to

underline that a typical deflation mesh consists of only a few hundred nodes per process, hence its computational cost is very low. However, as shown in Figure 2, even if this particular algorithm requires only one point-to-point (P2P) and one collective communication per iteration, with a large number of processes the scalability of this algorithm can become far from ideal due to the fact that communication cost becomes preponderant. The introduction of a new data structure in the code could allow to decouple the two constraints and to have a deflation grid independent of the DDD, which will be only based on the cache size, making it more versatile. In Section 3 we show that this new data structure also allows to enhance parallel performances.

3 GRAPH DATA STRUCTURE

A graph-like data structure seems particularly adapted for the above mentioned needs: the analogy between a grid and a graph is trivial (Figure 1) and only the connectivity between neighbour vertices is needed in the DPCG algorithm. Furthermore, there is an extensive quantity of already available and optimized algorithms to manipulate graphs. Parallel data exchanges on the boundaries among processes are dealt with a classical overlapping ghost structure.

In Section 2 we have seen that the typical size of a deflation grid is of a few hundred elements per process and that communication can quickly become preponderant. This data structure has been thought in such a way that multiple layers (nl) of ghosts vertices can be added on each process, extending the available stencils and allowing the algorithm to work with only a P2P communication each nl iterations, as shown in Figure 2.

It is important to remember that this algorithm works on a very small grid. The size of the data that is typically exchanged between processes consists of only few hundred Bytes per ghost layer. Consequently, the communication cost is mainly due to network latency and load imbalance. Although computational cost and message size increase proportionally to the number of layers, their effect remains sufficiently small to be negligible in comparison to the aforementioned factors. In Figure 2 we can see a trace obtained by TAU [5] over few iterations of the algorithm with 1 and 4 ghost layers on 14 processes. While the cost of collective communications is mainly due to the load imbalance, we notice that the P2P communication cost is paramount. Even though the single data exchange is slightly costlier when using multiple layers, the gain in the other iterations shows the efficiency of this methodology.

4 CONCLUSION AND PERSPECTIVES

We have argued that more flexible data structures are needed in multi-physics CFD codes if they are to be adapted to the new heterogeneous supercomputing architectures and we have shown that a simple graph-like data structure could be sufficient for all those algorithms in which only the connectivity between nodes is needed. Although the developments are still in a validation phase, the measures shown in Figure 2 are very promising. We aim to demonstrate the efficiency of this new data structure on large industrial cases. Architecture aware partitioning is fundamental for good parallel perfor-

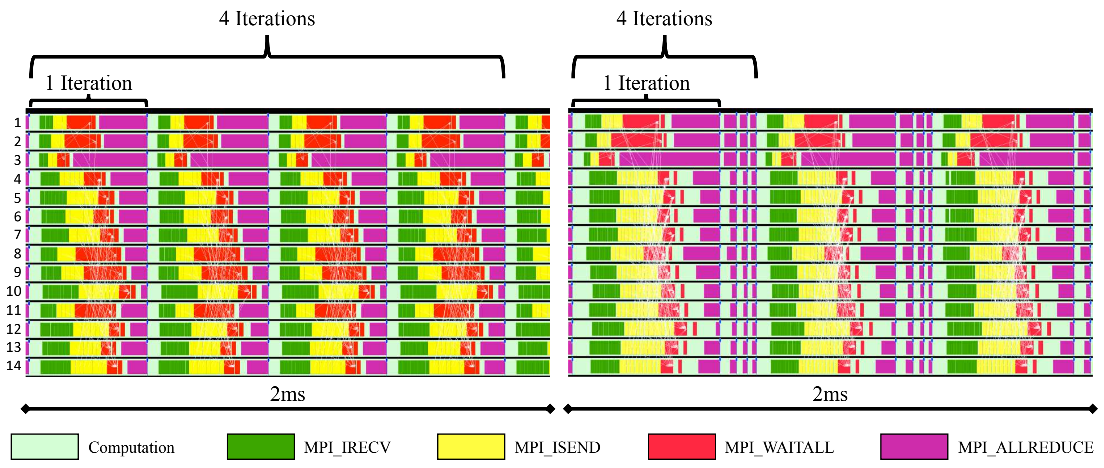

Figure 2: Trace of a few deflation iterations: single ghost layer $nl = 1$ (left) vs multiple ghost layers $nl = 4$ (right). One iteration is contained between two MPI_ALLREDUCE calls.

mances, consequently we are also going to use a hardware-aware partitioner for the grid in order to minimize the number of neighbours, hence the quantity of exchanged messages, to be able to fully exploit shared memory systems and have better load-balancing across processes. We have repeatedly underlined the fact that the mesh size per process is very small. This would allow each process to treat reasonably larger graphs. This pushes us to create a gather-scatter system for the graphs, in order to have a larger graph per process and in the meantime reduce the number of processes participating to the computation. This will allow better scalability when the cost of collective communication could become important.

REFERENCES

[1] https://www.top500.org

[2] Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F. Latency, bandwidth, and concurrent issue limitations in high-performance CFD. (2000) Web. doi:10.1016/B978-008043944-0/50783-6.

[3] Moureau V., Domingo P., and Vervish L. Design of a massively parallel CFD code for complex geometries. Une algorithmique optimisée pour le supercalcul appliqué à la mécanique des fluides numérique, Comptes Rendus Mécanique (2011)

[4] Malandain, M. Simulation massivement parallèle des écoulements turbulents à faible nombre de Mach. (2013) HAL Id : tel-00801502

[5] Shende S. and Malony A. D. , "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, SAGE Publications, 20(2):287-331, Summer 2006