SFC based multi-partitioning for accurate load balancing of CFD simulations

Ricard Borrell, J C Cajas, Lucas Mello Schnorr, Arnaud Legrand, Guillaume Houzeaux

Tenth International Conference on Computational Fluid Dynamics (ICCFD10), Barcelona,Spain, July 9-13, 2018 **ICCFD10-380**

SFC based multi-partitioning for accurate load balancing of CFD simulations

R. Borrell[a], J.C. Cajas[a], L. Schnorr[b], A. Legrand[c] and G. Houzeaux[a]
Corresponding author: ricard.borrell@bsc.es

[a] Barcelona Supercomputing Center (BSC), Barcelona, Spain.
[b] Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil.
[c] Université Grenoble Alpes, Inria, CNRS, Grenoble INP, Grenoble, France.

Abstract: In the context of multi-physics simulations on unstructured and heterogeneous meshes, generating well-balanced partitions is not trivial. The computing cost per mesh element in different phases of the simulation depends on various factors such as its type, its connectivity with neighboring elements or its layout in memory with respect to them, which determines the data locality. Moreover, if different types of discretization methods or computing devices are combined, the performance variability across the domain increases. Due to all these factors, evaluate a representative computing cost per mesh element, to generate well-balanced partitions, is a difficult task. Nonetheless, load balancing is a critical aspect of the efficient use of extreme scale systems since idle-times can represent a huge waste of resources, particularly when a single process delays the overall simulation.
In this context, we present some improvements carried out on an in-house geometric mesh partitioner based on the Hilbert Space-Filling Curve. We have previously tested its effectiveness by partitioning meshes with up to 30 million elements in a few tenths of milliseconds using up to 4096 CPU cores, and we have leveraged its performance to develop an autotuning approach to adjust the load balancing according to runtime measurements. In this paper, we address the problem of having different load distributions in different phases of the simulation, particularly in the matrix assembly and in the solution of the linear system. We consider a multi-partition approach to ensure a proper load balance in all the phases. The initial results presented show the potential of this strategy.

Keywords: Space-Filling Curve, SFC, Mesh Partitioning, Multi-partitioning, Geometric Partitioning, Parallel computing, Autotuning.

1 Introduction

Larger supercomputers allow the simulation of more complex phenomena with increased accuracy. Eventually, this requires finer geometric discretizations with larger numbers of mesh elements. In this context, and extrapolating to the Exascale paradigm, meshing operations such as generation, deformation, adaptation or partition, become a critical issue within the simulation workflow. In this paper, we focus on mesh partitioning. In particular, we present some improvements carried out on an in-house parallel mesh partitioner based on the Hilbert Space-Filling Curve.

In the context of multi-physics simulations, the partition of the mesh determines the work load to be executed by each parallel process. In general, a proper partition consists of a well-balanced distribution, which also minimizes the communication requirements among parallel processes. Communications produce an overhead which degrades the scalability of the algorithm. On the other hand, the load imbalance may not degrade the overall parallel performance, as long as the slowest parallel process shows good parallel efficiency, but derives on idle time of the faster parallel processes. In many cases, the waste of resources produced by the imbalance can be much worse than the communications overhead.

Mesh partitioning is traditionally formulated as a graph partitioning problem, which is a well-studied NP-complete problem generally addressed using multilevel heuristics composed of three phases: coarsening, partitioning, and un-coarsening. Publicly available libraries such as ParMetis [?] or PT-Scotch [?] implement different variants of them. These libraries provide parallel mesh partitioning but with a limited performance and a decreased quality of partitions in some cases [?]. On the other hand, geometric partitioning techniques obviate the topological interaction between mesh elements and perform its partition according to their spatial distribution. A Space-Filling Curve (SFC) is a continuous function used to map a multi-dimensional space into a one-dimensional space with good locality properties. The idea of geometric partitioning using SFC consists in mapping the mesh elements into a 1D space and then easily divide the resulting segment into equally weighted sub-segments. A significant advantage of the SFC partitioning is that it can be computed very fast and it is easy to parallelize, especially when compared to graph partitioning methods. However, while the load balance of the resulting partitions can be guaranteed, the data transfer between the resulting subdomains, measured in terms of edge-cuts in the graph partitioning approach, cannot be explicitly measured and thus neither be minimized.

In this paper, we present improvements on the SFC-based mesh partitioning algorithm that we have previously presented in [?]. On that work, we showed some computing experiments performed on the Blue Waters supercomputer, from the National Center for Supercomputing Applications (NCSA) at the University of Illinois (U.S.A). We demonstrated that a mesh of 30M elements can be partitioned in few cents of seconds using up to 4K CPU-cores. We asserted as well that the solution achieved is independent (disregarding round-off errors) of the number of parallel processes used to compute it. We have leveraged the performance of the partitioner to develop an auto-tuning approach to adjust the load balancing according to runtime measurements. With this approach we avoid the overwhelming problem of generating a representative formula to approximate the computing cost of each element in different conditions. We show some illustrative results of this approach on Section **??**. In this paper, we address the problem of having different load distributions in different phases of the simulation, particularly in the matrix assembly and in the solution of the linear system. We consider a multi-partition approach to ensure a proper load balance in all the phases. We present some initial results that will be extended in the conference presentation.

The SFC based partitioner developed in this paper and the physics solvers used in the test cases are both integrated into Alya [?, ?]: the high-performance computational mechanics code developed at the Barcelona Supercomputing Center. The physics solvable with the Alya system includes incompressible/compressible flow, solid mechanics, chemistry, particle transport, heat transfer, turbulence modeling, electrical propagation, etc. Alya aims at massively parallel supercomputers [?]; its parallelization includes both the MPI and OpenMP frameworks, as well as heterogeneous system options including accelerators. Alya is one of the twelve simulation codes of the Unified European Applications Benchmark Suite (UEABS) of PRACE and thus complies with the highest standards in HPC.

The structure of the paper is as follows: Section 2 contains a short overview of the Hilbert SFC; in Section 3 we summarize our in-house algorithm for parallel partitioning based on the Hilbert SFC. Section **??** contains the auto-tuning strategy to optimize the partition according to runtime measurements. Section **??** presents some initial estimations of the benefits achievable with the multi-partitioning approach. Finally, general conclusions are outlined in Section **??**.

2 Hilbert Space Filling Curve

A Space-Filling Curve (SFC) is a continuous function used to map a multi-dimensional space into a one-dimensional space with good locality properties. There are many possible definitions of SFC based on different mapping options, among them the well-known Peano [?] and Hilbert [?] approaches. Sagan *et al.* [?] give a complete overview of the different versions. In this paper, the Hilbert SFC is selected for its good locality preservation, however, switching to another mapping option would be straightforward in our implementation.

The Hilbert SFC is defined using a geometric recursion. For the 2D case, on the *pth* level of the recursion a discrete function is obtained:

$$h : \{1, ..., 2^{2p}\} \longrightarrow \{1, ..., 2^p\}\{1, ..., 2^p\} \tag{1}$$

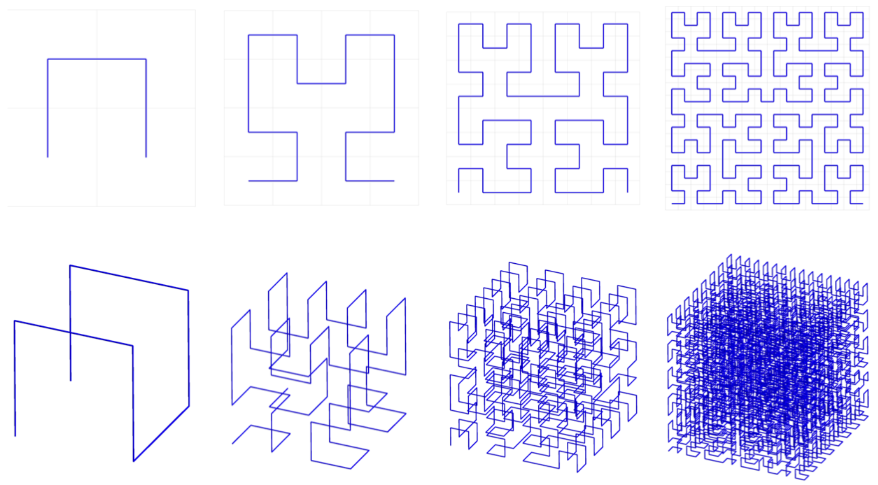


Figure 1: Hilbert SFC recursive generation, 2D (top) and 3D (bottom)

determining the ordering of the curve through a $2^p \times 2^p$ Cartesian grid. We represent the four initial steps of this recursion in Figure 1 (top). On the first step, the curve has a $\sqcap$ shape traversing a 2×2 grid. On the second step, we refine this curve so that it traverses the 4×4 grid. Then, we decompose this 4×4 grid into four 2×2 sub-grids, where the curve takes the shapes $\sqsupset$, $\sqcap$, $\sqcap$ and $\sqsubset$, respectively. Following a specific refinement pattern for each of these basic 2×2 shapes, we obtain the third level, and we continue this process until we achieve the desired level of refinement. Therefore, the process of generating a Hilbert curve consists in recursively applying a specific refinement pattern to each of the basic 2×2 shapes generated at each level of the recursion. In other words, the SFC recursion is determined by the decomposition applied to each of the former 2×2 shapes.

We can implement this algorithm by cyclic lookups to two arrays: the first storing the orderings which define the basic 2×2 shapes, the second its refinement patterns, also referred as orientations. Luitjens *et al.* [**?**] provide details of this implementation approach. The 3D case follows the same idea: here, we define the basic shapes on $2 \times 2 \times 2$ sub-grids. We depict the first four steps of the 3D recursion in Figure 1 (bottom). Note that a Hilbert SFC definition extends to any rectangular or cubic domain by using a grid with 2^p elements on each axis. Besides, the grid does not necessarily have to be uniform.

We also can generate a Hilbert SFC in parallel following a multilevel approach. An initial coarse grid is defined, and the SFC-index of each of its elements, hereafter referred as coarse bins, determines the rank of the parallel process continuing the recursion within it. If the proper orientation is followed within each coarse bin, the resulting SFC generated in parallel is the same that would be obtained sequentially. This process is illustrated in Figure 2. The parallel generation of an SFC has two advantages: it is faster than the sequential version, and, since there is more memory available, a higher level of refinement can be achieved. This second aspect determines the granularity of the balancing process.

3 Overview of the SFC based mesh partitioning algorithm

We illustrate the principal steps of the SFC-based mesh partitioning algorithm in Figure 3. The idea is straightforward. We start by defining a grid on a bounding box containing all the mesh elements. Then, we associate a weight with each bin of the grid according to the elements that it contains. Finally, we traverse the grid following the SFC ordering and, each time we reach the target weight per subdomain, we create a new subset of the partition.

We explain in detail the parallel version of this algorithm in previous works [**?**]. The two-level strategy adopted consists in defining a coarse grid within the bounding box that encloses the geometry, and then perform an SFC partition within each coarse bin separately. Two ingredients are necessary:

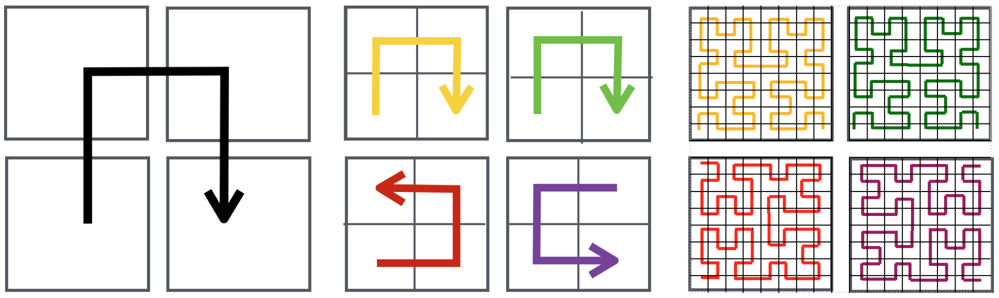
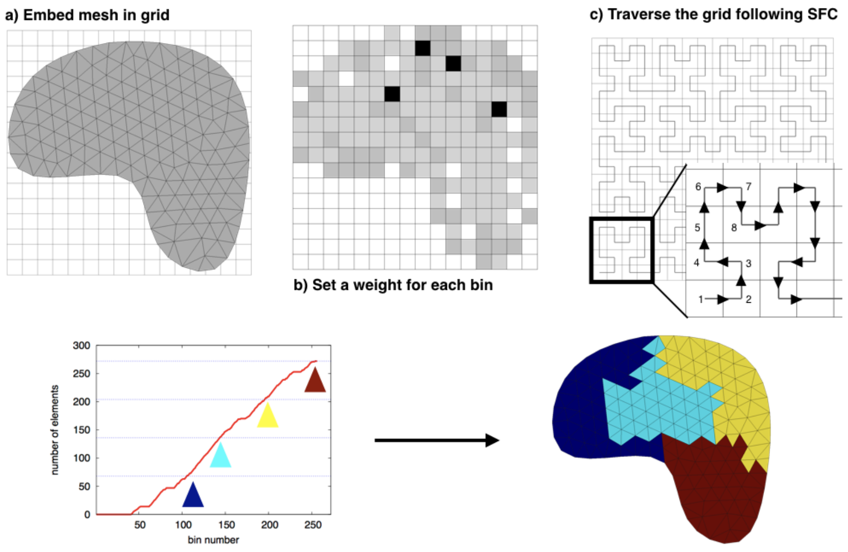


Figure 2: Hilbert SFC parallel generation. a) Initial coarse grid definition, b) orientation of each parallel process to continue the recursion, c) recursive generation of SFC in each parallel process.



a) Embed mesh in grid
b) Set a weight for each bin
c) Traverse the grid following SFC
1
2
3
4
5
6
7
8
number of elements
bin number

Figure 3: Sequential algorithm for mesh partitioning using the Hilbert SFC. a) Embed mesh in a Cartesian grid, b) Define wights of the grid bins, c) Traverse grid in the SFC order and define subdomains.

- The generation of a Hilbert SFC in parallel, obtained as the result of joining the local SFCs generated within each coarse bin (as described in Section 2);
- Define a partition in each coarse bin such that the resulting overall partition is coherent.

We illustrate this parallelization strategy in Figure 4. In this particular example, we divide the mesh into five parts employing four parallel processes. We associate each MPI rank to the coarse bin with the same SFC-index. Therefore the overall SFC traverses the domain in ascending order of MPI rank. In these conditions, if a parallel process obtains: i) the total weight accumulated in the SFC grid; and ii) the weight accumulated on the previous coarse bins (i.e., lower MPI ranks); then it can generate a local partition coherent with the global one. For example, in Figure 4, the process with rank 2, having the aforementioned information, generates a local partition that completes the second subdomain and starts the third one.

We outline the steps of the parallel partitioning algorithm in Algorithm 1. For more details, we invite the reader to read our previous publication [**?**]. As initial condition, we assume that the mesh cells are distributed among P parallel processes. Regarding the data transfers, there are three collective communications. In step 1 a `MPI_Allreduce` communication is used to evaluate the limits of the bounding box enclosing the mesh. In

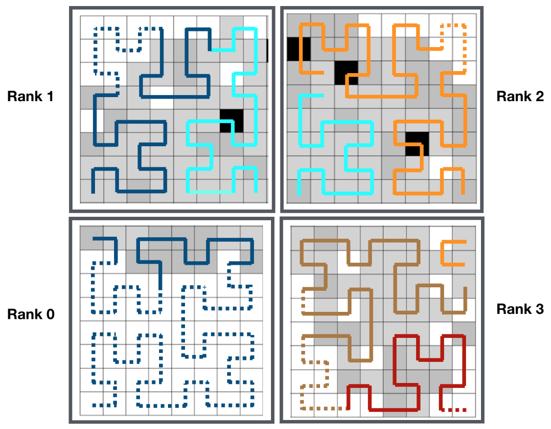


Figure 4: Parallel mesh partitioning using SFC.

step 2 a `MPI_Alltoall` is used to set up the communication requirements for the bins weight redistribution and, finally, in step 3 a `MPI_Allgather` is used to obtain the weight distribution across the coarse bins on each parallel process. Moreover, two point-to-point communications are necessary: to redistribute the weight of the bins (step 2), and to obtain the result on each parallel process according to the elements it initially holds (step 5). As explained in the following subsection, for the range of experiments carried out on Blue Waters supercomputer (using up to 4K CPU-cores and for meshes with up to 30M elements), the computation of the partition is dominated by the cost of the point-to-point communications of steps 2 and 5.

Algorithm 1 Parallel SFC-based partitioning

```
1: Definition of the bounding box and the Cartesian grids to be used
2: Evaluation and redistribution of fine-grid bins weight
3: Gather the coarse-grid weights distribution on all processes
4: Local partition based on SFC
5: Redistribution of the result of the local-bins partition
6: Infer mesh partition
```

Apart from round-off errors, the sequential and parallel executions of the algorithm produce the same partition if the fine grid resulting from joining the local grids is the same than the one used for the sequential partition. However, since a larger grid can result in a better-balanced partition, with the parallel execution, we can produce more accurate results. We observe an opposite behavior with graph partitioners, where sequential and parallel results are generally different, and the parallelization can worsen the result [**?**].

Finally, re-partitioning and load balancing based on SFC have a compelling advantage: the partition adjustments consist in moving the cutting points along the 1D segment defined by the SFC. Therefore, changes occur in the extremes of the sub-segments obtained from the former partition and, in general, most of the mesh elements remains in the same subdomain. In this way, Adaptive Mesh Refinement (AMR) frameworks [**?**] can exploit well this property.

Performance issues

As mentioned above, we presented in our previous publication [**?**] a set of numerical experiments carried out on the Blue Waters supercomputer. We considered two meshes used in production simulations. The first one was the simulation of a sniff flow in the respiratory system [**?**]. The associated heterogeneous and anisotropic mesh was composed of 17.7M cells. Figure 5 depicts the details of this case study. The second case under consideration was the numerical simulation of a swirling combustor [**?**]. The associated mesh, composed of prisms, tetrahedral, and pyramids has 28.9M cells. The CPU-cores used for the numerical experiments ranged from 8 up to 4096. In both cases, we observed that, at increasing the number of MPI threads, the computing costs become dominated by the point-to-point communications required in the steps 2 and 5 of

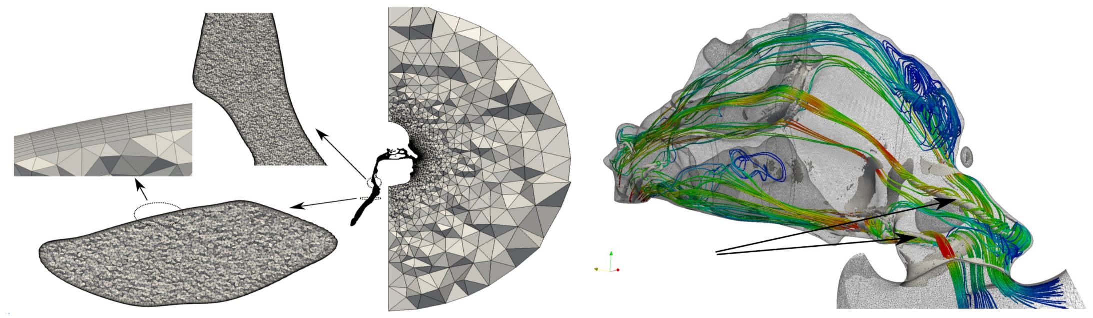


Figure 5: Geometric discretization of the respiratory system (left). Streamlines colored by velocity magnitude and iso-surface of Q-criterion pointing by arrows in the nasal cavities and throat (right).

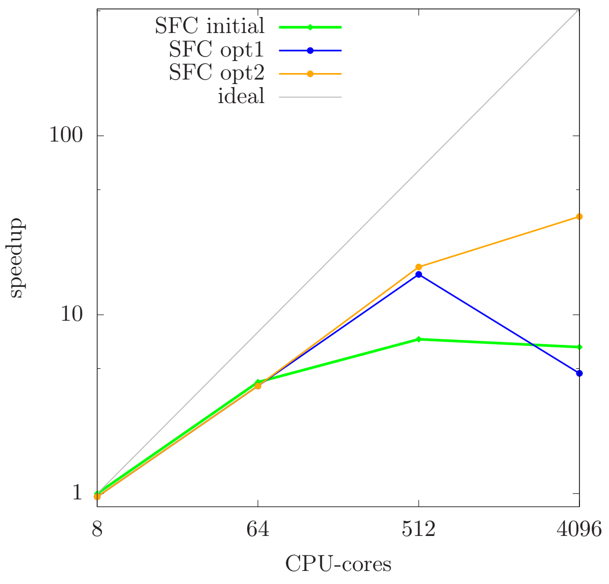


Figure 6: Strong speedup of the initial version of the partitioning algorithm and two optimizations: overlapping communications and computations (opt1), overlapping and initial redistribution (opt2). Mesh of the respiratory system (17.7M).

Algorithm 1, which end up representing around 90% of the execution time. Nonetheless, the executions were fast; we could perform both partitions in less than 50 milliseconds on 4096 CPU-cores. On the light of these results, we focused on the optimization of the two redistribution steps. We considered two ideas:

- *Overlapping of communications and computations.* In the redistribution performed on the step 2 of Algorithm 1, the object of redistribution are the contributions to the SFC grid bins weight. Therefore, if the same bin contains two mesh elements, their contribution is summed up by the source process before sending it to the corresponding target, *i.e.*, the process accounting for the portion of the bounding box where such bin is located. In our initial version of the algorithm, we first evaluated the *collisions*. Then, we shared the communication requirements between the parallel processes (`MPI_Alltoall`). Finally, we performed the point-to-point communications. The optimization consisted in overlapping the evaluation of collisions with the point-to-point communications. For this purpose, instead of sharing the exact communication requirements, we used an upper bound of them that we can evaluate in advance [**?**].

- *Intermediate redistribution to reduce communications costs.* Note that in the point-to-point communi-

#CPU	initial version	opt. 1	opt. 2
8	1,31	1,37 (104%)	1,37 (104%)
64	0,31	0,33 (106%)	0,33 (106%)
512	0,18	0,078 (43%)	0,071 (39%)
4096	0,20	0,28 (140%)	0,037 (18%)

Table 1: Partitioning time (sec.) with different optimizations of the partitioning algorithm. Opt. 1: overlapping communications and computations. Opt 2: overlapping + initial redistribution. Respiratory system mesh (17.7M).

cations data corresponding to all of the mesh elements is potentially moved. The initial distribution of elements may come from a parallel read of the mesh file, and then the bin weight contributions are redistributed according to its spacial location. So it represents a *3D* communication pattern, in contrast to the *2D* interface updates used to operate with a field in a domain decomposition. We recall that we do not send the elements, but the contributions to bin weights. Therefore, the higher the number of collisions of elements in the same bin, the less data needs to be transferred. To foster this situation, we interleaved an initial redistribution of mesh elements between subsets of processes using the `MPI_Alltoallv` function. The objective was to group elements of the same region favoring collisions. This optimization reduced the communication requirements for the overall point-to-point redistribution and also reduced the communications contention derived from the anisotropy of the mesh.

We illustrate the strong scaling of the initial algorithm and the two optimizations in Figure 6. Note that we directly applied the second optimization on the algorithm resulting from the first optimization. The respective computing times are presented in Table **??**. The overlapping strategy (blue line) improved the scalability up to 512 CPU-cores but was not helpful with 4096 CPU-cores, where the computation time becomes insignificant versus the communications and there is no room for the overlapping. On the other hand, the data redistribution proposed in the second optimization reduced significantly the cost of the point-to-point communications producing a relevant performance improvement up to 4096 CPU-cores.

4 Auto-tuning the Compute Load Balance of Partitions

As we have argued before, evaluating an accurate estimation of the computing cost associated with the mesh elements is an overwhelming problem. Many factors may influence the compute time of a single mesh element, such as its type, its topological interaction with other elements of the mesh, its layout in memory with respect to its neighboring elements, the characteristics of the device being used, etc. Our approach to adjust the partition consists on measuring its compute time during runtime and balance the load accordingly.

Let N be the number of mesh elements, and P the number of subsets or subdomains desired for the partition. The objective number of elements for the $i'th$ subset of the partition, noted as W_i, is defined as

$$W_i = c_i^k \frac{N}{P} \tag{2}$$

where c_i^k is the correction coefficient derived from runtime measurements in the $k'th$ optimization step. Initially the coefficients $c_i^0 = 1$, for all the subsets. We define the coefficients for the following steps in two sub-steps. First the coefficients $\tilde{c}_i^{k+1}$ are defined as

$$\tilde{c}_i^{k+1} = c_i^k \left((1 - \alpha_i) + \alpha_i \frac{\bar{t}}{t_i} \right) \qquad \forall i \in \{1, ..., P\} \tag{3}$$

where t_i is the elapsed time spent by process i on the part of the execution being considered, $\bar{t}$ is the average time for all processes and α_i is a relaxation factor. Note that this relaxation factor can be different for each partition. We use this to *froze* the partition subsets which elapsed time is close to the average. For example we can used the following criteria

$$\alpha_i = \begin{cases} 0, & \mid 1 - \frac{t_i}{t} \mid < 0.02 \\ 0.5, & \mid 1 - \frac{t_i}{t} \mid \geq 0.02. \end{cases}$$

In this case, the coefficient correction remains unmodified for the partition subsets with a deviation with respect to the average lower than 2%. Finally we need to ensure that the sum of the correction coefficients is P such that $\sum_i W_i = N$. In order to rescale the values of $\tilde{c}_i^{k+1}$, we use the following definitions. The Ω_a defined as

$$\Omega_a = \{i : \alpha_i \neq 0\} \tag{4}$$

are the partition subsets being adjusted, and C_a and C_f defined as

$$\begin{aligned} C_a &= \sum_{i \in \Omega_a} \tilde{c}_i^{k+1} \\ C_f &= \sum_{i \notin \Omega_a} \tilde{c}_i^{k+1} = \sum_{i \notin \Omega_a} c_i^k \end{aligned} \tag{5}$$

are the sum of the coefficients of the partition subsets being adjusted and the sum of the ones remaining to be fixed, respectively. Then, the scaling factor for the coefficients adjusted is

$$\Gamma = \frac{N - C_f}{C_a}. \tag{6}$$

Therefore, the coefficients are defined as

$$c_i^{k+1} = \begin{cases} c_i^k, & i \notin \Omega_a \\ \Gamma \tilde{c}_i^{k+1}, & i \in \Omega_a \end{cases}$$

Figure **??** shows the elapsed time spent by each parallel process on the matrix assembly phase of the time integration. This result corresponds to the case illustrated in Figure 5, executed using 384 CPU-cores. A significant imbalance is observed across the parallel processes with both the partitions generated with METIS (red line) and SFC (black line). In this case the imbalance comes from the mesh heterogeneity. The blue line presents the result after 10 iterations of the partition optimization algorithm described above, note that most of imbalance has been canceled. Figure **??** (left) shows for the same case the maximum and average time between all processes for the successive iterations of the optimization algorithm. We observe how the maximum time converges to the average and thus the imbalance is significantly reduced. In particular, the initial imbalance (evaluated as maximum time divided by the average) is 1.53 and, after 12 iterations, it reduces to 1.02. Finally, Figure **??** (right) compares the cost of an overall time-step for three different partitions: the METIS and SFC partitions without optimization, and the SFC partition obtained after the auto-tuning process. We observe that the non-optimized SFC partition outperforms METIS in some cases but performs similarly for 192 CPU-cores. However, when we add the auto-tuning process the resulting partition outperforms the METIS one by a percentage ranging between 15% and 20%.

5 Preliminary results of the Multi-Partitioning Strategy

Numerical simulations can be structured in different repetitive phases. In particular, in the respiratory system simulation under consideration, the time integration is divided in different time-steps which in turn can be divided into two main phases, the matrix assembly and the linear solver. Both phases have different computing signatures. The assembly is based on three nested loops where the outer loop traverses the elements of the mesh and does not require any communication. The linear solver is dominated by the sparse matrix vector product kernel and requires both point-to-point and all-reduce communications. This dissimilarity on the computing signatures makes it difficult to generate a well-balanced partition for the two phases simultaneously. We consider in this work a multi-partitioning approach where an specific partition

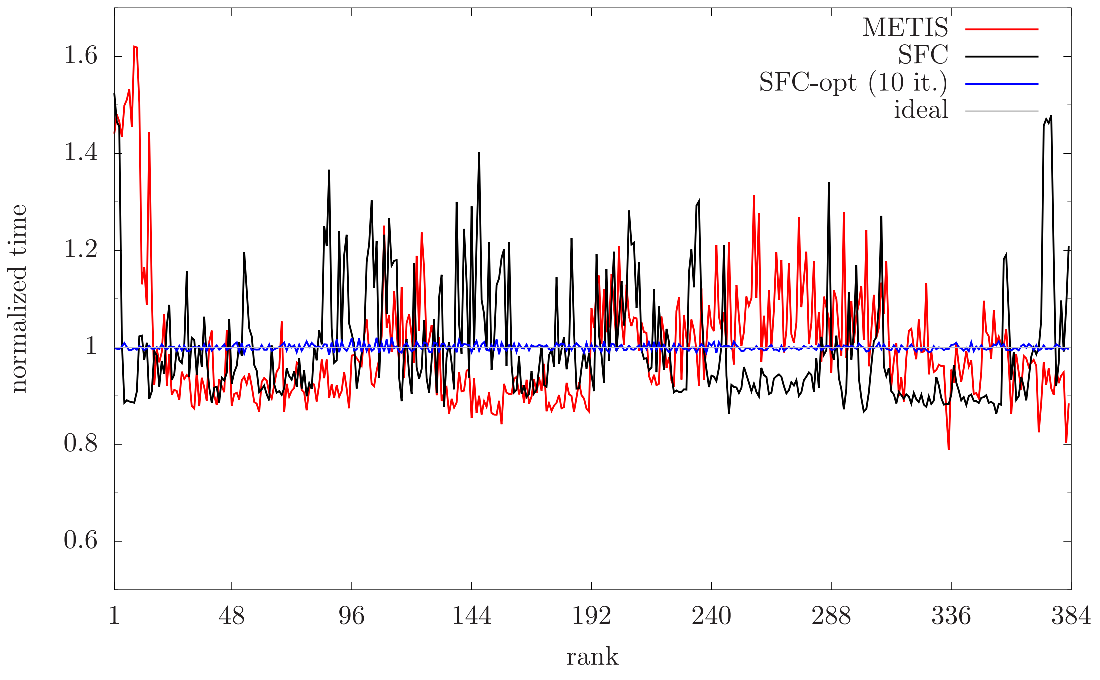


Figure 7: Normalized time per rank, defined as the elapsed time per rank divided by the average time across all ranks, for the assembly phase of the respiratory system simulation (28.9M elements mesh). Simulation performed using 384 CPU-cores. Comparison between METIS partition (red), non-optimized SFC partition (black) and optimized SFC partition (blue).

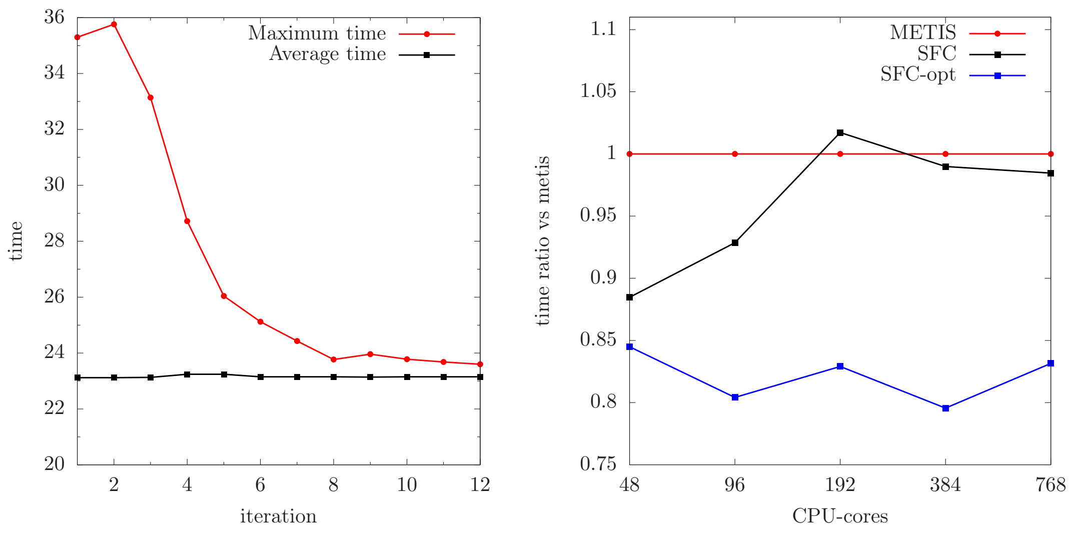


Figure 8: Left: maximum and average time across all ranks for the different iterations of the optimization algorithm. Right: Ratio of the time-step cost for the optimized SFC (blue) and non-optimized SFC (black) vs METIS. Mesh of the respiratory system (28.9M).

is carried out for each phase. Figure **??** shows an initial estimation of the benefits of this approach for the respiratory system simulation using 192 CPU-cores. Three optimization criteria (on the X axis of each facet) are compared, the first one considers the imbalance of the assembly phase, the second one the solver phase and the last one considers the overall time-step. Considering as a base-line the partition optimizing the overall time-step, the partition optimized for the assembly reduces its cost by 6.5%, while the partition optimized for the solver reduces its cost 2.3%. Combining the best partitioning for each phase might bring lower the overall time step cost by 6.1%. In this case, if a single partition is used the best choice is the one optimized for the assembly because this phase represents 65% of the time step.

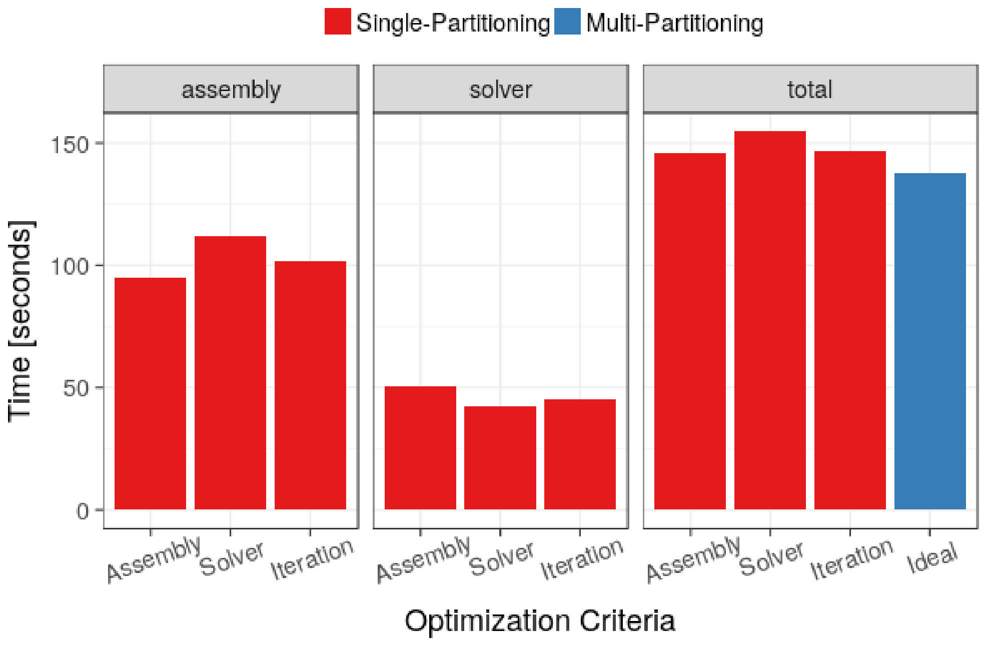


Figure 9: Elapsed time accumulated over a total of 15 time-steps and for the corresponding assembly and solver phases. In blue the estimated total time for and ideal multi-partitioning implementation. Mesh of the respiratory system (28.9M).

6 Concluding Remarks

In this paper, we have presented some improvements carried out on an in-house geometric mesh partitioner based on the Hilbert Space-Filling Curve. An overview of the parallel algorithm has been presented detailing the critical aspects to be considered for performance. This algorithm has been previously tested on meshes with up to 30M elements in the Blue Waters supercomputer, where the partitions are calculated in few tenths of milliseconds. We took advantage of the low cost of the partitioning process to develop an iterative auto-tuning strategy to improve the quality of the partitions, in terms of load balancing, from real measurements of the corresponding parallel execution. This iterative strategy and some preliminary results have been presented in this manuscript. Considering a simulation of the respiratory system on a heterogeneous unstructured mesh of 30M elements, we show how the imbalance is reduced from 50% to 2% in twelve iterations of the auto-balance algorithm.

Finally, we address the problem of having different load distributions in different phases of the simulation, particularly in the matrix assembly and in the solution of the linear system. We consider a multi-partition approach to ensure a proper load balance in all the phases. A first estimate of the benefits of this strategy has been evaluated for the partition of the aforementioned 30M mesh in 192 parts, the reduction of the time per iteration is ≈6%. This study will be extended in order to better discern the suitability of this strategy.

Acknowledgements

This work has been financially supported by the Ministerio de Economia, Industria y Competitividad, of Spain (TRA2017-88508-R). We thank the EU H2020 Programme and the MCTI-Brazil under the HPC4E Project (grant 689772), the FAPERGS Green Cloud (16/488-9) and MultiGPU/Multi-core (16/354-8) projects, the CNPq 447311/2014-0, and the CAPES/Cofecub 899/18. Some experiments were carried out at the Grid'5000 platform, with support from Inria, CNRS, RENATER and several other organizations. The authors would also like to thank the Industry Program and the Blue Waters sustained petascale computing project at the National Center for Supercomputing Applications (NCSA), and PRACE preparatory access projects funded in part by the EU Horizon 2020 research and innovation programme (2014-2020) under the grant agreement 653838. J.C. Cajas acknowledges the financial support of the 'Consejo Nacional de Ciencia y Tecnología (CONACyT, México)' grant number 231588 290790.

References

[1] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[2] Cédric Chevalier and François Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *CoRR*, abs/0907.1375, 2009.

[3] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *J. Parallel Distrib. Comput.*, 48(1):71–95, January 1998.

[4] R. Borrell, J.C. Cajas, D. Mira, A. Taha, D. Koric, M. Vázquez and G. Houzeaux Parallel mesh partitioning based on space filling curves. *Computers and Fluids*,DOI: 10.1016/j.compfluid.2018.01.040, 2018.

[5] G. Houzeaux, M. Vzquez, R. Aubry, and J.M. Cela. A massively parallel fractional step solver for incompressible flows. *Journal of Computational Physics*, 228(17):6316 – 6332, 2009.

[6] G. Houzeaux, R. Aubry, and M. Vázquez. Extension of fractional step techniques for incompressible flows: The preconditioned orthomin(1) for the pressure schur complement. *Computers & Fluids*, 44(1):297–313, 2011.

[7] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, E. Dering Burness, J. M. Cela, and M. Valero. Alya: Multiphysics engineering simulation towards exascale. *J. Comput. Sci.*, 14:15–27, 2016.

[8] G. Peano. *Sur une courbe, qui remplit toute une aire plane*, pages 71–75. Springer Vienna, Vienna, 1990.

[9] David Hilbert. *Über die stetige Abbildung einer Linie auf ein Flächenstück*, pages 1–2. Springer Berlin Heidelberg, Berlin, Heidelberg, 1970.

[10] Hans Sagan. *Space-Filling Curves*. Springer New York, New York, NY, 1994.

[11] J. Luitjens, M. Berzins, and T. Henderson. Parallel space-filling curve generation through sorting. *Concurrency Computation Practice and Experience*, 19(10):1387–1402, 2007.

[12] 0. Antepara, O. Lehmkuhl, R. Borrell, J. Chiva and A. Oliva. Parallel adaptive mesh refinement for large-eddy simulations of turbulent flows. *Computers and Fluids*, 110:48–61, 2015.

[13] H. Calmet, A. M. Gambaruto, A. J. Bates, M. Vázquez, G. Houzeaux, and D.J. Doorly. Large-scale cfd simulations of the transitional and turbulent regime for the large human airways during rapid inhalation. *Computers in biology and medicine*, 69:166–180, 2016.

[14] H. Calmet, C. Kleinstreuer, G. Houzeaux, A.V. Kolanjiyil, O. Lehmkuhl, E. Olivares, and M. Vzquez. Subject-variability effects on micron particle deposition in human nasal cavities. *Journal of Aerosol Science*, 115(Supplement C):12 – 28, 2018.

[15] Simon Gövert, Daniel Mira, Jim B. W. Kok, Mariano Vázquez, and Guillaume Houzeaux. The effect of partial premixing and heat loss on the reacting flow field prediction of a swirl stabilized gas turbine model combustor. *Flow, Turbulence and Combustion*, Sep 2017.