Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Enabling Space Filling Curves parallel mesh partitioning in Alya

R. Borrell[a], J.C. Cajas[a], G. Houzeaux[a] and M.Vazquez[a]

[a]Barcelona Supercomputing Center
C/Jordi Girona 29, 08034-Barcelona, Spain
ricard.borrell@bsc.es, juan.cajas@bsc.es,guillaume.houzeaux@bsc.es,mariano.vazquez@bsc.es

Abstract

Larger supercomputers allow the resolution of more complex problems that require denser and thus also larger meshes. In this context, and extrapolating to the Exascale paradigm, meshing operations such as generation, deformation, adaptation/regeneration or partition/load balance, become a critical issue within the simulation workflow. In this paper we focus on the mesh partitioning, presenting the work carried out in the context of a PRACE Preparatory Access Project to enable a Space Filling Curve (SFC) based partitioner in the computational mechanics code Alya. In particular, we have run our tests on the MareNostrum III supercomputer of the Barcelona Supercomputing Center. SFC partitioning is a fast and scalable alternative to the standard graph based partitioning and in some cases provides better solutions. We show our approach at implementing a parallel SFC based partitioner. We have avoided any computing or memory bottleneck in the algorithm, while we have imposed that the solution achieved is independent (discounting rounding off errors) of the number of parallel processes used to compute it.

1. Introduction

Alya is a high performance computational mechanics code to solve coupled engineering problems. The physics solvable with the Alya system include: incompressible/compressible flow, solid mechanics, chemistry, particle transport, heat transfer, turbulence modeling, electrical propagation, etc. Alya is designed for massively parallel supercomputers [1]. Its parallelization includes both the MPI and OpenMP models to take advantage of the distributed and the shared memory paradigms. Accelerators like GPU are also exploited at the iterative solver level to further enhance the performance of the code. Recently, dynamic load balancing techniques have been introduced as well to better exploit the computational resources at the node level. Alya is one of the twelve simulation codes of the Unified European Applications Benchmark Suite (UEABS) and thus complies with the highest standards in HPC. The code has been tested on the largest supercomputers worldwide, where it has proven to efficiently scale up to 100K CPU-cores for industrial applications. The aim of this paper is to broaden the scalability of the mesh-partitioning module.

Mesh partitioning is traditionally based on graph partitioning, which is a well-studied NP-complete problem generally addressed by means of multilevel heuristics composed of three phases: coarsening, partitioning, and un-coarsening. Different variants of them have been implemented in publicly available libraries such as Metis [2] and Scotch [3]. Parallel formulations have also been developed (ParMetis [4], PT-Scoch [5]), however they present limitations on the parallel performance and the quality of the solution at a growing number of partitioning processes [6]. This lack of scalability makes graph-based partitioning a potentially critical bottleneck in the simulation workflow. However, taking into account the evolution of the computing HPC systems, any potential bottleneck becomes an effective bottleneck if not addressed in time. Motivated by these circumstances, we present a fully parallel geometric partitioning alternative implemented in Alya.

Geometric partitioning techniques obviate the topological interaction between mesh elements and perform its partition according to their spatial distribution. If we consider as unitary element the mesh cell, then its mass center can be used to determine its spatial location. 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, i.e. it tries to preserve the proximity of elements in both spaces. The idea of geometric partitioning using SFC is to map 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 techniques. However, while the load balance of the resulting partitions can be guaranteed, the data transfer between the resulting subdomains, measured by means of edge-cuts in the graph partitioning approach, cannot be explicitly measured and thus neither be minimized.

In this paper we present our implementation of a parallel SFC based partitioner in Alya. We have avoided any computing or memory bottleneck that could limit the scalability of the algorithm, imposing that the solution achieved is independent (discounting rounding off errors) of the number of parallel processes used to compute it. Section 2 presents a short overview of the Hilbert SFC, Section 3 explains the parallel SFC mesh partitioner implemented in Alya, Section 4 presents some computing results, before concluding in Section 5.

2. **Hilbert Space Filling Curve and SFC-based mesh partitioning**

There are many alternative definitions of SFC on the 2D and 3D spaces, representing different mapping options. Among them the well-known Peano [7] and Hilbert [8] approaches. A complete overview is given in [9]. In this paper we use the Hilbert SFC, but changing the mapping option would be straightforward in our implementation.

The Hilbert SFC is defined by means of a geometric recursion. For the 2D case, on the p'th level of the recursion a discrete function is obtained:

$$h:\{1,\ldots,2^{2p}\} \rightarrow \{1,\ldots,2^{p}\}\times\{1,\ldots,2^{p}\},$$

which covers all the cells of a $2^p \times 2^p$ uniform grid. The four initial steps of this recursion are represented in Figure 1 (top). On the first step, the curve has an "∩" shape set up on a 2×2 grid. This is further refined as shown in the second step. At this point we have a 4×4 grid which can be decomposed into four 2×2 sub-grids, where the curve has the shapes "⊃", "∩", "∩" and "⊂", respectively. Following a specific decomposition for each of these 2×2 shapes we obtain the third level, and this process is continued until the desired level of refinement. Therefore, to generate the Hilbert curve it is only necessary to recursively apply the decomposition pattern of each of the basic 2×2 shapes. This can be implemented by cyclic lookups on two arrays, one storing the orientation of the basic shapes the second its decomposition. Details of this implementation approach can be found in [10]. The same idea is followed in the 3D case, where the basic shapes are defined on 2×2×2 sub-grids, the first four steps of the recursion are depicted in Figure 1 (bottom). Note that the Hilbert SFC can be used in any rectangular or cubic domain by just applying a scaling factor to the axes.

The process of partitioning an unstructured mesh using an SFC, is divided in five steps:

1. *Define a bounding box.* A bounding box for the mesh is defined from the maximum and minimum coordinate values of the nodes.

2. *Embed mesh elements into a cartesian grid*. First a grid within the bounding box is defined, then for each mesh element a *weight* is added to the grid bin containing it. A mesh partition can be based on the distribution of nodes or cells. In this paper we use cells and we associate each cell with the bin that contains its mass centre. We define the *weight* of a cell as its number of nodes.

3. *Sort the grid bins using the SFC order*. Using a Hilbert SFC poses a restriction: the cartesian grid must have a power of two of elements in each direction. If this number is referred as n, the steps of the Hilbert recursion required to cover the whole grid are $\log_2(n)$.

4. *Divide the set of bins in equally sized sub-sets.* First the objective weight is defined as the total weight accumulated in the grid bins divided by the desired number of partitions, P. Then the grid is traversed following the SFC order until the objective weight is reached. The bins traversed compose a subset of

the partition. Each time a subset is defined, the objective weight is re-evaluated to redistribute the discrete errors as much as possible.

5. *Define the mesh partition from the bins partition.* This step is straightforward; the partition index of each mesh element is equal to the partition index of the bin containing it.

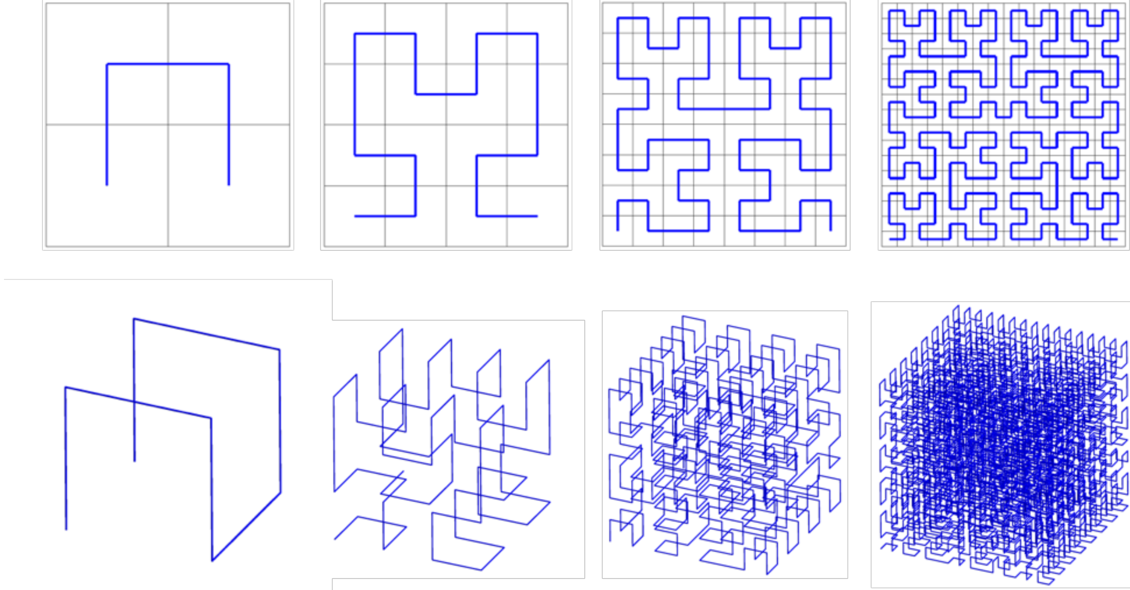


Fig. 1. Representations of the Hilbert curve recursive generation, 2D (top) and 3D (bottom).

3. Parallel SFC partition implemented in Alya

The parallel extension of the algorithm outlined in the previous section has the following steps:

1. *Parallel mesh read*. Each parallel process reads a portion of the mesh file. Namely a continuous "chunk" of cells and the corresponding nodes. If the algorithm is applied on a re-partition or load balancing process this step is not necessary, actually each parallel process will already hold a part of the mesh.
2. *Bounding box.* A bounding box containing the mesh is defined, this requires a collective reduce operation in order to find the maximum and minimum value for each of the node coordinates.
3. *Sub-boxes distribution and sorting*. The bounding box is divided into equally sized sub-boxes, one for each of the P_{sfc} parallel processes. Each process will be in charge of the portion of mesh contained in its corresponding sub-box (which can be an empty set). The sub-boxes are themselves indexed using the Hilbert SFC order, this way each parallel process can continue the recursion to compute the piece of SFC covering its sub-box independently. To do this it is only necessary to apply the recursive process defined above starting with the proper orientation/shape, which is determined by the index of the sub-box. Joining all the independently processed local SFCs a coherent Hilbert SFC is obtained for the overall bounding box. Using the Hilbert SFC poses a restriction: the number of partitioning processes must be a multiple of 2^d, where d is the domain dimension. Therefore, the number of partitioning processes, $P_{sfc,}$ is set as the largest multiple of 2^d lower than P, the total number of processes. At this step is also decided the depth of the Hilbert recursion within the sub-boxes, referred as q_l. In consequence the depth of the overall SFC is $q = log_2(P_{sfc}^{1/d})\ q_l$.
4. *Bin weights evaluation and distribution.* Each parallel process traverses its sub-list of mesh elements and assigns weights to the corresponding bins. Note this sub-list comes from the parallel mesh read (step 1) so is not related with the sub-boxes distribution (step 3). In order to accumulate the bins weight while traversing the cells sub-list an array would be required having the same size as the complete grid. However, to avoid possible memory limitations, this operation is performed in different sub-steps:

a. Count the number of cells of the sub-list belonging to each sub-box
b. For each sub-box (other than the own one) containing elements of the sub-list: i) allocate a buffer according to the number of sub-list elements contained in it; ii) store in it the bins with non-null weight; iii) send the buffer to the process associated with the sub-box. On this step, an array with the same dimension as the size of the local sub-grids is used in order to accumulate the weights corresponding to the sub-box being considered.
c. The partitioning processes sum-up the weights received for the bins of its sub-grid.

5. *Local partition*. Before each parallel process can perform its local partition independently, we need the weight distribution through the sub-boxes to set-up the local partitions. This is obtained by means of a collective all-gather communication involving one real value per parallel process. From this information each parallel process can evaluate: i) its initial partition index; ii) the number of partitions it must generate (this can be any real number from 0 to P); iii) the maximum relative weight that it must assign to its first partition subset. For instance, for a particular parallel process, these initial conditions could be: i) the initial partition index is 56; iii) the number of partitions is 3.65 ii) the maximum relative weight of the first subset is 0.34. Under these conditions the relative weights of the subsets of the resulting local partition would be: 0.24, 1, 1, 1, 0.41; and the corresponding partition indexes: 56,57,58,59,60. The process associated to the next sub-box, will start with the partition index 60 and a maximum relative weight for the first partition sub-set of 0.59, its number of partitions will be evaluated as the relative weight contained on its sub-box with respect to the total weight of the mesh.
6. *Mesh partition.* Once a partition index is assigned to each bin, the partition index of each mesh element is fixed as the one of the bin containing it. This requires the point-to-point communication opposite to the on made in step 4, this way each parallel process obtains the partition indices of the bins containing cells of its sub-list.
7. *Data redistribution.* Each parallel process distributes its part of mesh according to the partition result by means of point-to-point communications.

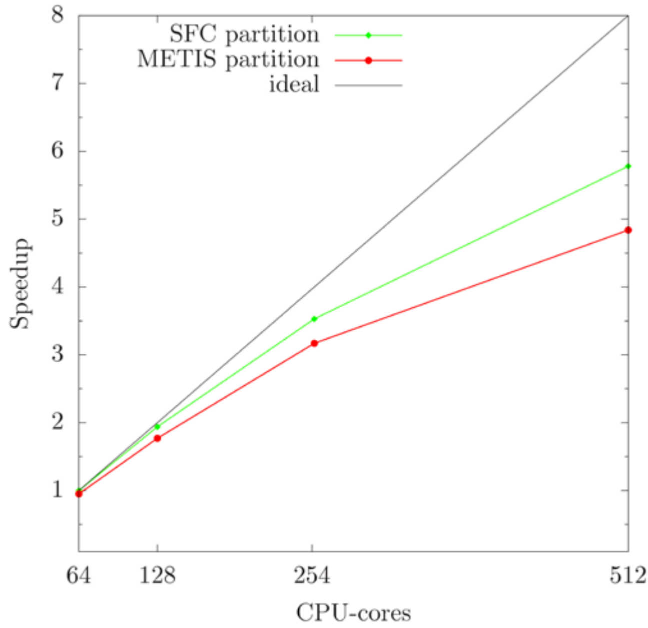
Note that in no step of the algorithm more than 1/P of the mesh or the box grid elements are used. Therefore, the algorithm does not have any memory or computational bottleneck. Regarding data transfers, there are two collective low-weighted communications in steps 1 and 5, and three point-to-point communications in steps 4, 6 and 7.

The sequential and parallel executions of the algorithm produce an almost identical partition if the grid used to discretize the bounding box is the same. However, since a larger grid can result in more accurate results in terms of final load balance, this advantage can be used in parallel executions. This represents an opposite behaviour to that of the parallel graph partitioners, which in general generate poorer partitions at increasing the number of partitioning processes. In case of identical grids, there can still be a minor difference between the parallel and sequential SFC algorithms. This is because of the objective weight re-evaluation used to distribute the discrete errors on steps 4 and 5 of the sequential and parallel algorithms, respectively. In the sequential case the re-evaluation is performed P-1 times and affects all the domain; in the parallel case it is performed locally on the sub-box of each partitioning process and the number of repetitions depends on the portion of mesh contained in it.

Finally, a nice property of SFC based partitioning when used for load balancing is that the partition changes occur at the extremes of the 1D sub-segments obtained when the SFC is broken. Therefore, in many cases, most of data of the previous partition remains in the former subdomain.

4. **Preliminary results**

Tests and validations of the algorithm were performed on the MareNostrum III supercomputer of the Barcelona Supercomputer Center. In this section we present a comparison of the quality of the partitions obtained with our algorithm vs the ones obtained with the sequential k-way graph algorithm implemented in Metis [3], using its default parameters. In particular, we have performed the partitions considering the mesh cell as unitary element, and we consider the scalability of the whole time step. The pre-processing stage, and in particular the partitioning is not considered, we expect to generate comparisons with ParMetis in following works. We show results for two different test cases. Despite a careful study it is still required to better understand the results obtained and optimize the algorithm accordingly, these tests show the readiness and applicability of the new partitioner in Alya.



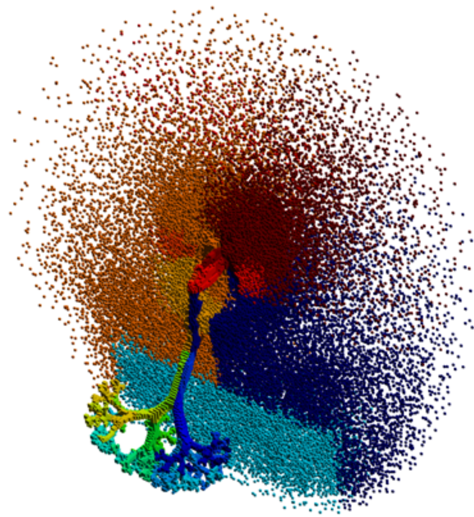
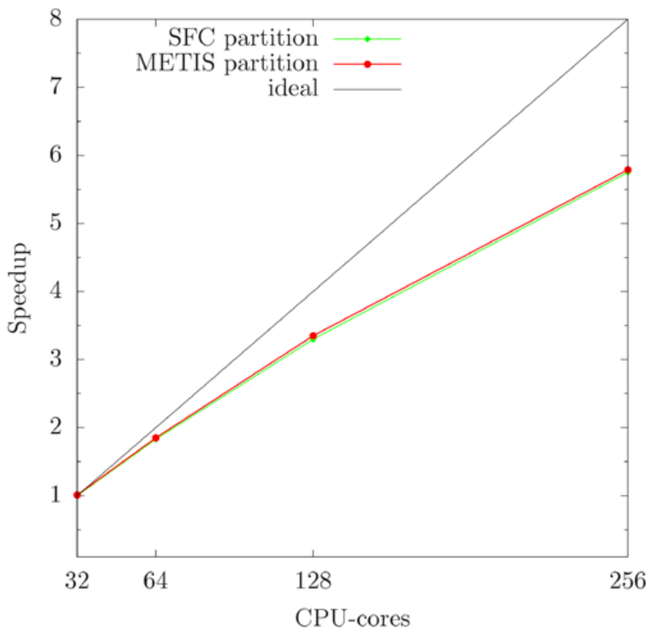


Fig. 2. Simulation of the respiratory system airflow with a 17M cells mesh. Left: Strong speedup of the time-integration using SFC and Metis partitions. Right: Representation of the bins partition (output of step 5 of the parallel algorithm).

Figure 2 compares the strong scalability achieved with both partitions for the simulation of the airflow through the respiratory system. The domain is discretized by means of a highly anisotropic 17M mesh with a high density in the nasal cavity and the trachea. The parallel performance is clearly better with the SFC partition. With 1024 CPUs it outperforms Metis by 16%. A better balance achieved with the SFC partition causes this difference. Figure 3 shows an equivalent test for a solid mechanics simulation of a fusion reactor component on a 100K elements mesh. In this case both partitions perform very similarly.



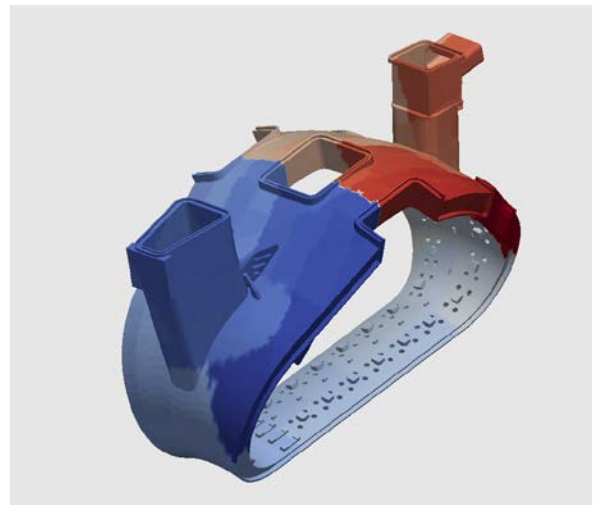


Fig. 3. Solid mechanics simulation of a fusion reactor component, 100K cells mesh. Left: Strong speedup of the time-integration using SFC and Metis partitions. Right: Representation of a mesh partition.

5. **Summary and Outlook**

This paper is the result of a PRACE Preparatory Access Project, our aim was to develop and implement from scratch a new SFC-based partitioner. Our initial motivations were the limitations of standard graph partitioners in terms of parallel performance and quality of the solution, at growing the number of parallel process used for the partition. We have presented with details a fully scalable implementation, without any memory or computational bottleneck, and that keeps or even increases the quality of the partition at growing the number of partitioning processes. We have shown some preliminary results comparing the quality of the partitions achieved with both approaches, but a more comprehensive study is still necessary in order to draw any conclusion. In any case, the SFC seems to be a competitive approach for parallel mesh partitioning and its integration in Alya overcomes some bottlenecks of standard libraries at the largest scale.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838.

References

[1] Vázquez, M., Houzeaux, G., Koric, S., Artigues, A., Aguado-Sierra, J., Arís, R., Mira, D., Calmet, H., Cucchietti, F., Owen, H., Taha, A., Burness, E. D., Cela, J. M., Valero, M. Alya: Multiphysics engineering simulation toward exascale, Journal of Computational Science 14 (2016) 15-27.
[2] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," SIAM J. Sci. Comput., vol. 20, no. 1, pp. 359–392, Dec. 1998.
[3] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," in Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM), ser. Supercomputing '96. Washington, DC, USA: IEEE Computer Society, 1996.
[4] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, ser. HPCN Europe 1996. London, UK, UK: Springer-Verlag, 1996, pp. 493–498.
[5] F. Pellegrini, "PT-Scotch and libScotch 5.1 User's Guide," Aug. 2008, 76 pages User's manual.
[6] A. Artigues and G. Houzeaux. Parallel Mesh Partitioning in Alya. Technical report 202, PRACE White paper, 2015.
[7] Peano, G., "Sur une courbe, qui remplit toute une aire plane". Mathematische Annalen 36, 157–160 (1890).
[8] Hilbert, D., "Über die stetige abbildung einer linie auf ein flächenstück". Mathematische Annalen 38, 459–460 (1891).
[9] Sagan, H.: Space-Filling Curves. Springer, New York (1994).
[10] Campbell P M, Devine K D, Flaherty J E, Gervasio L G, Teresco J D. "Dynamic octtree load balancing using space-filling curves". Technical Report CS-03-01, Williams Collge Department of Computer Science, 2003.
[11] Niedermeier, R., Reinhardt, K., Sanders, P.: Towards optimal locality in mesh- indexings. Discrete Applied Mathematics 7, 211–237 (2002).