Proceedings of Machine Learning Research vol 245:1–8, 2024 2024 International Conference on Machine Learning and Intelligent Computing

Reordering CAE Matrix using Hierarchical Clustering

Shanhong He[*] SZHESH@HOTMAIL.COM
Suzhou Nuclear Power Research Institute Co, ltd, Suzhou, 215000, China
National Engineering Research Center for Nuclear Power Plant Safety & Reliability, Suzhou, 215000, China

Yahui Yang
Suzhou Nuclear Power Research Institute Co, ltd, Suzhou, 215000, China
National Engineering Research Center for Nuclear Power Plant Safety & Reliability, Suzhou, 215000, China

Lihong Tang
Suzhou Nuclear Power Research Institute Co, ltd, Suzhou, 215000, China
National Engineering Research Center for Nuclear Power Plant Safety & Reliability, Suzhou, 215000, China

Tao Zhang
Suzhou Nuclear Power Research Institute Co, ltd, Suzhou, 215000, China
National Engineering Research Center for Nuclear Power Plant Safety & Reliability, Suzhou, 215000, China

Xiaolong Jiang
Suzhou Nuclear Power Research Institute Co, ltd, Suzhou, 215000, China
National Engineering Research Center for Nuclear Power Plant Safety & Reliability, Suzhou, 215000, China

Editors: Nianyin Zeng and Ram Bilas Pachori

Abstract

Traversing a high-dimensional mesh using 1-D trajectory is a widely used technique in the fields of CAE computing and data management. Such a trajectory is widely known as a space-filling curve. Nevertheless, most of the space filling curves, such as Z-curve or Hilbert curve, are designed for structured meshes. Therefore, it is vital to design an effective space-filling curve for unstructured meshes. In this paper, we propose a space-filling curve for unstructured mesh by considering the original problem as a graph clustering problem. We generate a hierarchical clustering schema and uses depth-first search to traverse the hierarchical schema. In this way, the sequence of the depth-first search naturally be-comes a 1-D trajectory. Compared with traditional space-filling curves, our solution can effectively handle traversing problems on unstructured meshes.

Keywords: CAE, Matrix Reordering, Hierarchical Clustering

1. Introduction

Matrix ordering is important in the field of CAE (Computer Aided Engineering). A CAE application uses algorithms to simulate a physical experiment, such as a ship sailing in the water. During this process, physical parameters of the ship and the water body is computed using algorithms such as the SIMPLE algorithm (Acharya et al., 2007a). In this case, analysts can understand and evaluate the performance of a ship design without con-ducting real-life physical experiments.

In a typical CAE experiment, the mesh model of the environment (the ship, the water body, etc.) is represented using a matrix. To explain this, we introduce a simple ex-ample. Figure 1 demonstrates a 2-D, 3×4 mesh. When representing the mesh using a matrix, the principle is to use a 1-D trajectory to traverse the whole mesh, so that the cell order of the trajectory decides which row/column the matrix corresponds to. Such a trajectory is also known as the space-filling curve (Moon et al., 2001). In the figure we demonstrate two possible trajectories, a default trajectory (left) and a trajectory reoredered using the RCM (Reverse Cuthill-Mckee) algorithm (Azad et al., 2017) (right). The bitmaps underneath the trajectories are the shape of the corresponding matrices. Here we only consider the distribution of NNZs (Non-zeroes) of the matrices, therefore we use bitmaps, with black cells representing NNZs and white cells representing zeroes. One may notice that the distribution of the NNZs matches the pattern of the adjacency matrices of the meshes. For example, in the left trajectory, the first cell has two neighbors 2 and 5, therefore the first row of the matrix has 3 NNZs, corresponding to 1st, 2nd and 5th cells. This principle applies to all cells. The adjacency information is needed since the principle of a CAE application is to simulate a physical process by iteratively compu-ting the interactions between adjacent cells, such as pressure, momentum, etc.

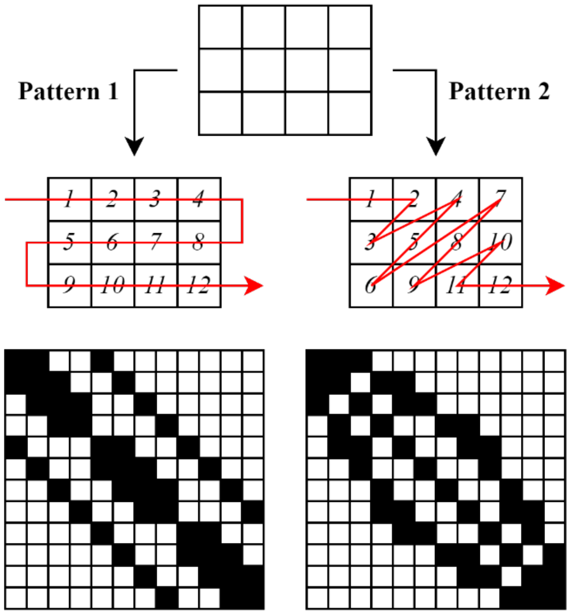

Figure 1: Two ordering patterns of the same mesh.

Nevertheless, we can observe that the ordering pattern of the cells strongly influences the shape of the corresponding matrices, and subsequently the features of the matrices such as local density, data continuity, bandwidth, etc. Different matrix shapes could lead to performance variations when CAE tasks are executed. For example, when performing SpMV (Spares Matrix-Vector multiplication), matrices with more compact local NNZ distributions would lead to higher efficiency (Elafrou et al., 2017).

There are many existing space-filling curves for ordering structured meshes, such as the gray-code curve, linear scan, Z-curve, Hilbert curve, etc. (Jagadish, 1990) Among these curves, the Hilbert curve has been widely adopted for the regular structured meshes (Acharya et al., 2007b). Figure 2 demonstrates the trajectory of a typical Hilbert curve. From the figure we can ob-serve that it can smoothly traverse a structured mesh i.e. mesh composed of rectangles or hyper-rectangles. However, all of the aforementioned curves can hardly apply to a unstructured mesh, such as the mesh displayed in Figure 3. Meanwhile, such unstructured meshes are quite common in CAE applications. Therefore, how to properly traverse an unstructured mesh has become a challenge.

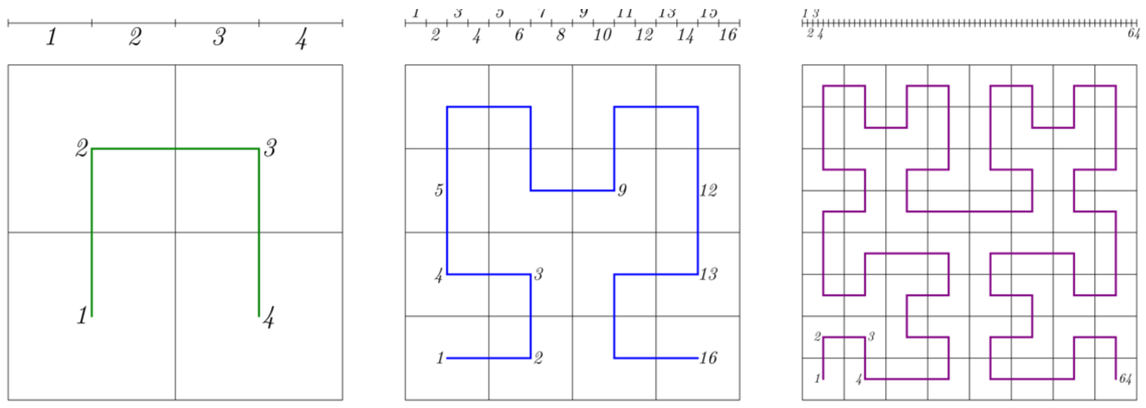

Figure 2: An example of a Hilbert curve.

To solve this problem, we introduce a space-filling curve generator for both structured and unstructured meshes. To do this, we consider a mesh (either structured or unstructured) as a graph, with nodes representing cells and edges representing the adjacency information. Then, we apply a hierarchical graph clustering algorithm to generate an agglomerative tree. Finally, we adopt the depth-first search on the tree and consider the search sequence as the output trajectory. Since the trajectory decides the ordering of the corresponding matrix, we can reorder the adjacency matrix according to the space-filling curve we generated. Compared with previous solutions, our algorithm is hardly dependent on the structure of the mesh, which greatly improves the usability of our algorithm.

The rest of the paper is organized as follows. In Section 2 we introduce our algorithm in detail, in Section 3 we conduct numerical evaluations. We conclude this paper in Section 4.

2. Solution

2.1. Intuition

In this section, we will introduce the principle of our algorithm. To generate a space filling curve for either a structured or unstructured mesh, we need to first evaluate the correlation between cells. In Section 1 we discussed that the basic principle of the CAE simulations is dependent on the adjacency relationship between cells. In this case, the correlation of cells will be closely related to the geometrical distribution of the cells. Specifically, given two cells c_1 and c_2, suppose we denote their correlation as $\mathrm{Cor}\,(c_1, c_2)$, then we have

$$\mathrm{Cor}\,(c_1, c_2) \propto Euc\,(c_1, c_2) \tag{1}$$

where $\mathrm{Euc}\,(c_1, c_2)$ represents the Eu-clidean distance between c_1 and c_2.

Based on the conclusions above, we can represent the correlation between cells using a weighted graph. A graph-based representation is a widely used approach (Osei-Kuffuor et al., 2015; Taghibakhshi et al., 2022). The advantage of a graph representation is that it omits the basic mesh structure, therefore can be applied to arbitrarily shaped meshes. To convert a mesh to a graph, a valid approach is to consider each cell as a node, and the adjacency information as edges. Since on the graph, a node is equivalent to a cell, in the rest of the paper, the words node and cell can be used interchangeably.

To preserve the geometric information of the original mesh, so that equation 1 is satisfied, we need to put weight on the edges. The weight of an edge represents the distance between the geometric centers of the adjacent cells (see Figure 4).

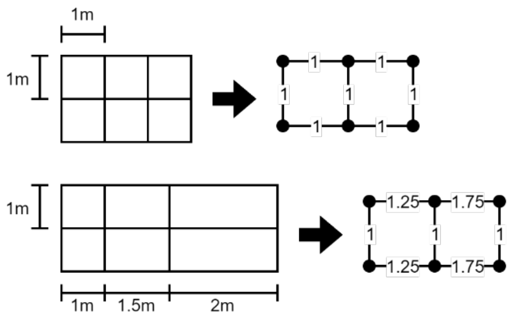

Figure 3: Representing a mesh using a weighted graph.

2.2. Hierarchical Clustering

After converting the mesh to a graph, the next objective is to design an algorithm to generate a 1-D node trajectory based on the graph. Given two nodes c_1 and c_2, let $w\left(c_1, c_2\right)$ be the edge weight between these two nodes, and I_1 and I_2 be their indices on the trajectory. Ideally, the indices should satisfy the following constraint:

$$w\left(c_1, c_2\right) \propto \left|I_1 - I_2\right| \tag{2}$$

Equation 2 indicates that, cells that are geometrically closer to each other should also be located closer on the trajectory. Unfortunately, this constraint can never be fully satisfied. The reason is that converting a mesh to a 1-D trajectory is fundamentally a dimensionality reduction process. A cell in a n-dimensional mesh would have 2^n neighbors. Meanwhile, an index on a trajectory only has 2 neighbors. Therefore, when traversing a n-dimensional mesh using a 1-D trajectory, a cell will inevitably lose 2^{n-1} of the original neighbors, which breaks the constraint of equation 2. Nevertheless, the constraint of equation 2 can still be used as a guide-line for how to properly traverse the graph.

Here we adopt the hierarchical clustering algorithm (Murtagh and Contreras, 2012) (HC). HC is a simple yet effective algorithm, it iteratively merges the most correlated nodes/clusters into a new cluster, until the whole data set is merged into a single cluster. If we apply HC to the graph based on the edge weight, and iteratively merge nodes with smallest edge weight, we will eventually generate a tree-shaped structure (a dendrogram). Figure 5 demonstrates the result of applying HC to a mesh introduced in Figure 4.

There is an interesting feature of the dendrogram. Nodes of higher correlation (smaller weights) will likely be located in the same branch of the tree. The higher the correlation, the lower the branch will be. This feature is determined by the principle of HC since it will always try to merge cells of

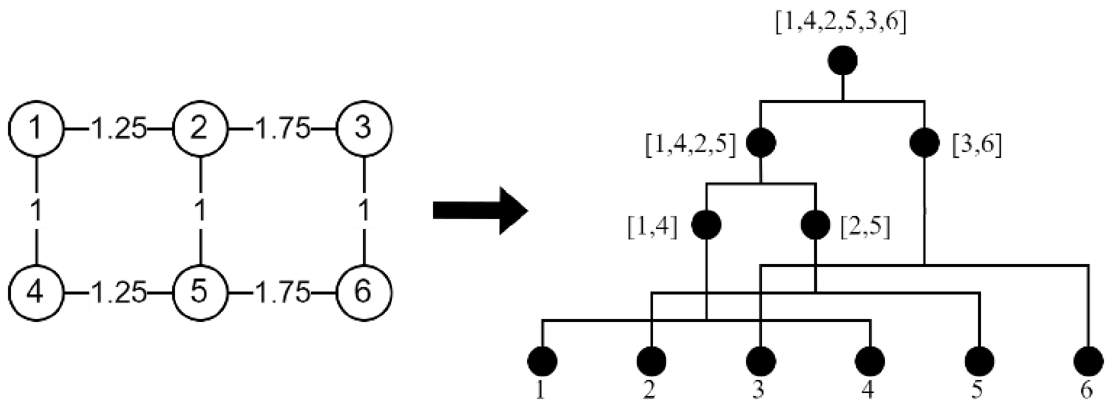

Figure 4: HC on a weighted graph and the generated dendrogram.

the highest correlation. Using this feature, we can re-arrange the 3-D grid model into a 1-D array by further applying the depth-first-search (Tarjan, 1971) (DFS) algorithm to the dendrogram. DFS will always try to visit the lowest level of the dendrogram, and nodes belonging to the same branch. This mechanism accords with the principle of the construction of the dendrogram. Therefore, we use the traversal sequence of DFS as the 1-D representation of the grid. The node sequence generated using DFS is marked on the top level of the dendrogram in Figure 5. So far we have introduced the overall process of our algorithm. The pseudo-code of the algorithm is included in Algorithm 1.

3. Experiment

3.1. Experimental Setup

We conduct experiments on a PC with Intel Core i7 2.20GHz CPU and 32GB RAM, NVIDIA GeForce RTX 3060 graphic card. We evaluate the performance of the reordering algorithm by conducing SpMV on both the default ordered matrix and reordered matrix. The dataset used for SpMV are collected from the SuiteSparse (Davis and Hu, 2011) data set. We choose 5 SpMV task with the size of the matrix ranging from 10^2 to 10^6, and perform SpMV 100 times, each time with randomly generated vectors. We record the overall time consumption and compute the average time consumption per SpMV. We use the Numpy (Bressert, 2012) and Cupy (Okuta et al., 2017) package to implement the SpMV operation. The numpy utilizes CPU for the computation and Cupy uses GPU.

3.2. Results

The experimental results are demonstrated in Figures 5 and 6. Figure 5 demonstrates the average time consumption of performing SpMV on CPU using Numpy. From the result we can see that the efficiency gap between the default SpMV and reordered SpMV is almost indistinguishable on small scale matrices (10^2 and 10^3). On matrices of larger scales (10^4 to 10^6), the performance gap gradually widens. This phenomenon demonstrates the effectiveness of our algorithm.

Algorithm 1 Reordering using Hierarchical Clustering

```
Input: A mesh data $M$.
Output: A 1-D array $S$ of cell sequence of $M$.
    Description:
    //Initialization
 1: Initialize an empty node set $V$ and an empty edge set $E$;
 2: Generate a weighted graph $G = \langle V, E \rangle$ based on $M$;
    //Hierarchical clustering
 3: Initialize an empty dendrogram $D$;
 4: $\mathbf{D}$.leaf nodes $= \mathbf{V}$;
 5: While $|V| > 1$ do:
 6: Find smallest edge $e = \langle v_1, v_\mathbf{2} \rangle$ in $E$;
 7: $V$.remove($v_1, v_\mathbf{2}$);
 8: $v'$=Merge($v_1, v_\mathbf{2}$);
 9: $V$.insert($v'$);
10: $D$.insert($v'$);
11: D.$v_1$.parent $= v'$;
12: D.$v_2$.parent $= v'$;
13: End While
    //Depth first Search
14: $S = $ DFS$(D)$
15: return $S$
```

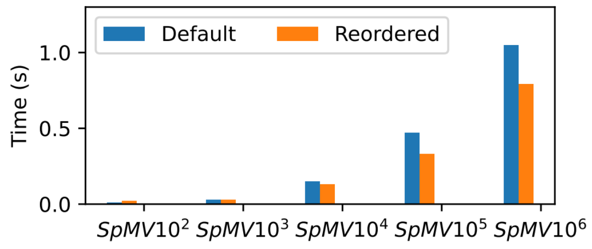

Figure 5: Experimental results on CPU using Numpy.

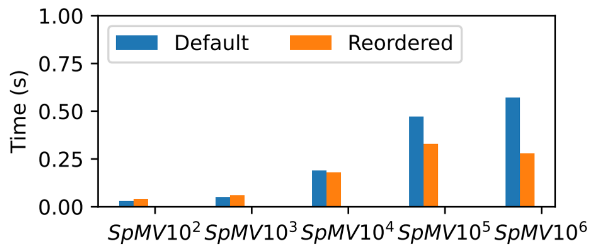

Figure 6: Experimental Results on GPU using Cupy.

Figure 6 demonstrates similar result on GPU, except for two differences. First, on smaller matrices, the time consumption on GPU is larger on CPU. Second, the performance gap between default SpMV and reordered SpMV widens on large scale matrices.

3.3. Analysis

To understand the performance of the experimental results. We need to understand the difference of execution plans of SpMV on CPU and GPU. Generally speaking, a CPU has less cores with more powerful computational capacity, while a GPU has significantly more cores each with relatively limited capacity. Hence, when a SpMV task is executed on CPU, it will be split up into multiple sub-tasks and will be distributed among all cores. When executed on GPU, the data will first be transferred to GPU RAM, then split into significantly more sub-tasks and are distributed among all GPU cores. Therefore, there are two fundamental differences between execution plans on CPU and GPU: First, the number of tasks on CPU is much smaller than on GPU. Second, the on-GPU computation requires an extra step of data exchange be-tween RAM and GPU RAM.

The differences above lead to the performance variations of SpMV on CPU and GPU. The reason that small scale SpMV executes less efficiently on GPU than on CPU is that, the extra data exchange between RAM and GPU RAM brings extra efficiency cost. When the scale of the matrix is small, the performance of GPU computation cannot compensate for the extra efficiency cost between RAM and GPU cache.

The reason that the performance gap between the default SpMV and reordered SpMV is more distinguishable on GPU is that, more sub-tasks on GPU leads to smaller data segmentation per task. In this case, data locality within each segment will play a more important role. As our reordering algorithm brings higher locality to the matrix, the reordered SpMV will show better performance on GPU than on CPU.

4. Conclusion

In this paper we study the matrix reordering problem, especially in the field of CAE analysis. We introduce the principle of CAE experiments and the importance of the ordering of the matrix. We briefly surveyed the existing ordering algorithms based on space-filling curves, and pointed out that they are not applicable to unstructured meshes. In this case, we highlight the problem of generating a ordering algorithm on both the structured and unstructured meshes.

To do this, we represent the mesh structure using a weighted graph, and adopt hierarchical clustering to generate a dendrogram for the mesh cells. Then, we adopt the DFS algorithm to traverse the dendrogram and uses the traversal trajectory as the reordered trajectory. By studying the principle of CAE experiment, we theoretically analyze the necessity and validity of using hierarchical clustering and DFS for the task. To further prove the effectiveness of our algorithm, we conduct experiments on various SpMV task using both CPU and GPU. Experimental results show that our reorder-ing algorithm can help accelerate SpMV tasks.

References

S. Acharya, B. R. Baliga, K. Karki, J. Y. Murthy, C. Prakash, and S. P. Vanka. Pressure-Based Finite-Volume Methods in Computational Fluid Dynamics. *Journal of Heat Transfer*, 129(4): 407–424, 01 2007a. doi: 10.1115/1.2716419.

S. Acharya, B. R. Baliga, K. Karki, J. Y. Murthy, C. Prakash, and S. P. Vanka. Pressure-Based Finite-Volume Methods in Computational Fluid Dynamics. *Journal of Heat Transfer*, 129(4): 407–424, 01 2007b. ISSN 0022-1481. doi: 10.1115/1.2716419.

Ariful Azad, Mathias Jacquelin, Aydin Buluç, and Esmond G. Ng. The reverse cuthill-mckee algorithm in distributed-memory. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 22–31, 2017. doi: 10.1109/IPDPS.2017.85.

Eli Bressert. *SciPy and NumPy: An Overview for Developers*. O'Reilly Media, 2012.

Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011. doi: 10.1145/2049662.2049663.

Athena Elafrou, Georgios Goumas, and Nectarios Koziris. Performance analysis and optimization of sparse matrix-vector multiplication on modern multi- and many-core processors. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 292–301, 2017. doi: 10.1109/ICPP.2017.38.

H. V. Jagadish. Linear clustering of objects with multiple attributes. *SIGMOD Rec.*, 19(2):332–342, may 1990. doi: 10.1145/93605.98742.

Bongki Moon, H. v. Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Trans. on Knowl. and Data Eng.*, 13(1):124–141, jan 2001. doi: 10.1109/69.908985.

Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery*, 2(1):86–97, 2012. doi: https://doi.org/10.1002/widm.53.

Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman. Cupy : A numpy-compatible library for nvidia gpu calculations. 2017.

Daniel Osei-Kuffuor, Ruipeng Li, and Yousef Saad. Matrix reordering using multilevel graph coarsening for ilu preconditioning. *SIAM J. Sci. Comput.*, 37(1):A391–A419, jan 2015. doi: 10.1137/130936610.

Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning, 2022.

Robert Tarjan. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 114–121, 1971. doi: 10.1109/SWAT.1971.10.