Domain Decomposition Methods under Constraints in CFD Simulations

Pratik Suryawanshi, Akshat Shukla, Arla Gracia Diengdoh

Project Co-ordinator: Dr. Maximilian Bauer

Modeling Seminar 2024

Contents

1 Introduction

1.1 Problem Statement

Computational Fluid Dynamics (CFD) simulations are widely used in various engineering disciplines to analyze and solve complex fluid flow problems. A common challenge we face while performing large-scale CFD simulations is managing the computational resources required. This challenge becomes even more prominent in simulations involving highly refined meshes, where the number of computational cells can easily reach into the millions or billions. A critical aspect of efficiently handling such simulations lies in the decomposition of the computational domain, or mesh, into smaller subdomains. This process, known as *domain decomposition*, plays a vital role in parallel computing environments, where multiple processors work simultaneously to solve the problem.

The problem we face is to partition the CFD mesh in a way that the communication overhead between the subdomains is minimized, with respect to predetermined geometric partitions. The objective is to reduce the communication load across these interfaces, thus improving computational efficiency. The goal of this study is to explore, implement and investigate various domain decomposition methods under constraints typical in CFD simulations.

A sample simulation from STAR-CCM+, a commercial CFD solver, is illustrated in Figure 1, showing the complexity of the meshes that need to be partitioned effectively. The initial problem was explained in the context of a scenario where certain parts of the simulation are moving while others remain stationary. As an example, in the case of a wheel attached to a car, the wheel represents the moving part, while the part connected to chasis itself is stationary. This creates a need to develop a partitioning scheme that takes into account both the dynamic nature of the moving components and the stationary regions, ensuring efficient communication between these during the simulation.

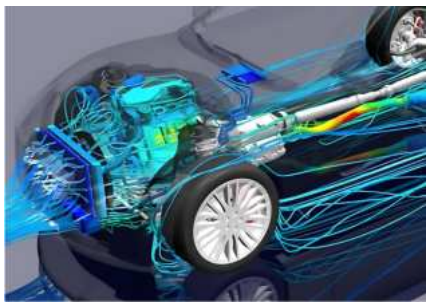


Figure 1: Sample simulation from STAR-CCM+ showing complex mesh structure

1.2 Objectives

To address the problem of domain decomposition in CFD simulations, this project focuses on achieving the following key objectives:

- **Develop a Partitioning Algorithm**: We aim to develop and test a partitioning algorithm that ensures interface continuity while reducing communication overhead between subdomains.
- **Ensure Computational Efficiency**: The algorithm should be computationally efficient, allowing for large-scale simulations to be performed within a reasonable timeframe.
- **Evaluate Existing Methods**: The project will evaluate existing methods for graph partitioning, particularly focusing on the use of tools like *METIS* and *GMSH*, and assess their performance in minimizing communication loads while maintaining balanced partition sizes.
- **Improve Load Distribution**: A key metric for evaluation will be the distribution of computational load across processors. A well-partitioned mesh should result in an even workload distribution, avoiding bottlenecks.

This study is motivated by the need to improve computational efficiency in large-scale simulations, which are increasingly common in industries such as aerospace, automotive, and energy. By developing a more efficient partitioning method, we aim to reduce simulation runtimes and improve the accuracy of the simulations by ensuring smoother transitions between subdomains.

2 Graph Theory and Tools

2.1 Overview of Graph Theory

Graph theory forms the foundation for partitioning meshes in computational simulations. A graph $G = (V, E)$ consists of vertices V and edges E, where vertices represent computational cells or mesh points, and edges indicate connections between them. This structure helps divide the mesh into subgroups, minimizing the number of edges (or "cuts") between them.

Minimizing cuts reduces communication between different subdomains in parallel computing, directly lowering communication overhead. In mathematical terms, given a graph G, the goal is to partition the vertex set V into k disjoint subsets while minimizing the sum of edge weights between these subsets.

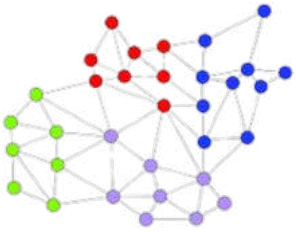


Figure 2: Example of a graph partition into two sets.

2.2 Graph Partitioning Methods

Graph partitioning aims for balanced partitions with minimal inter-partition communication. Common methods include *multilevel recursive bisection* and *multilevel k-way partitioning*. Both involve three stages:

1. **Coarsening**: Reduce the graph by collapsing vertices and edges.
2. **Partitioning**: Partition the coarsened graph.
3. **Uncoarsening**: Map the partition back to the original graph and refine it.

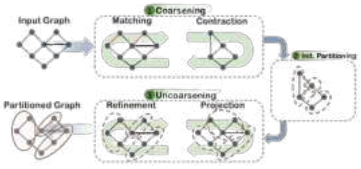


Figure 3: Multilevel graph partitioning process.

2.3 Tools Used (METIS and GMSH)

2.3.1 METIS

METIS is a highly efficient tool for partitioning large graphs and meshes. It uses a multilevel k-way partitioning algorithm to reduce communication overhead. METIS' key features include:

- **Scalability**: Handles very large meshes and graphs.
- **Multilevel Approach**: Ensures minimal edge cuts and balanced partitions.
- **Flexibility**: Partitions graphs into any number of parts.

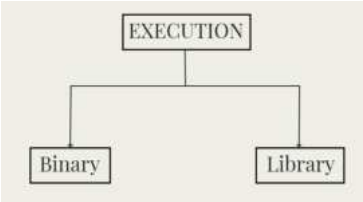


Figure 4: Schematic of METIS execution process.

2.3.2 GMSH

GMSH is an open-source 3D mesh generator, used for generating and visualizing meshes. It supports file formats like `.vtk`, commonly used for visualizing computational meshes. GMSH integrates with METIS, creating a smooth workflow for partitioning and analyzing complex mesh structures.

3 Initial Approaches and Attempts

In our initial attempts to partition the mesh, we began by converting the original mesh data stored in the `.vtk` (Visualization Toolkit) file format into the `.graph` format required by METIS. The `.vtk` format is widely used in scientific computing for storing mesh data, and its structure allows for storing various types of data, including points, cells, and associated attributes. For more details on the structure of `.vtk` files, refer to the VTK File Structure Guide.

The conversion from `.vtk` to `.graph` involved mapping the mesh nodes and connectivity information to a format compatible with METIS. The `.graph` format represents the mesh as a graph, where the vertices correspond to mesh nodes and the edges indicate connections between them. The specifics of the `.graph` file structure are outlined in the METIS Manual.

3.1 Challenges in Visualization

After converting the `.vtk` files to `.graph`, we attempted to visualize the resulting graph representation. However, as shown in Figure 5, the initial visualization made it difficult to discern the location and connectivity of the nodes due to the lack of clear geometric information. The complexity of the mesh and the abstract representation in the graph format led to confusion in understanding the spatial arrangement of the nodes.

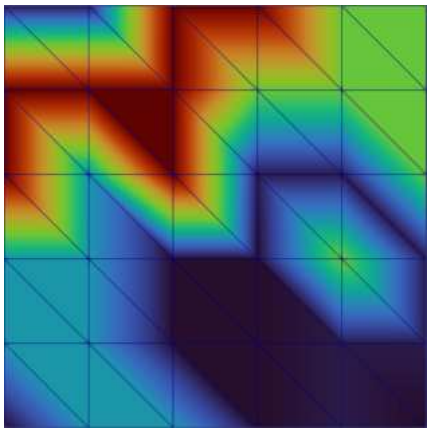


Figure 5: Initial visualization of the mesh after conversion from `.vtk` to `.graph`.

3.2 Development of a Custom Renderer

To address the limitations of the initial visualization, we developed a custom rendering tool using OpenGL. The goal was to provide a more intuitive visualization of the mesh structure by incorporating the geometric data from the original `.vtk` file. Our renderer allows for interactive exploration of the mesh, enabling users to zoom, pan, and rotate the mesh to better understand its spatial characteristics.

The custom renderer provides several features:

- **Interactive Navigation**: Users can explore the mesh interactively, making it easier to locate specific nodes and understand their connections.
- **Geometric Integration**: By using data from the original `.vtk` file, the renderer ensures that the visualized mesh accurately reflects the original geometry.

The development of this custom tool significantly improved the visualization process, allowing for a more efficient analysis of the mesh partitioning and aiding in troubleshooting issues during the conversion process.

4 Multi-level Partitioning Theory

4.1 Theorem and Background

Multi-level partitioning theory plays a crucial role in mesh partitioning, especially when dealing with domains that have distinct geometric characteristics. The aim is to partition the mesh in a way that reduces the number of elements on the interface between partitions, thus minimizing communication overhead in parallel computations.

The following theorem provides a foundation for geometric partitioning in higher-dimensional spaces:

Theorem 1. *Let $G = (V, E)$ be a (αk)-overlap graph in d-dimensions with n nodes, where:*

$$d \in \mathbb{N} \quad \textit{and} \quad n \in \mathbb{N}$$

Then there exists a vertex separator V_s such that $V = V_1 \cup V_s \cup V_2$, where:

- *The size of the separator $|V_s|$ is small relative to the total number of vertices $|V|$.*
- *The separator V_s separates the graph into two disjoint subgraphs V_1 and V_2, which do not share any vertices.*
- *The communication between V_1 and V_2 is minimized, as the separator isolates the partitions from each other.*

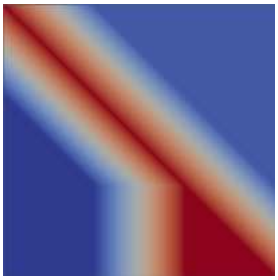
Note: In our specific case, there is no overlap between the subdomains, making the (αk) condition irrelevant.

4.2 Partitions Without and With a Geometric Separator

To illustrate the impact of geometric partitioning, Figures 6a and 6b show partitions of a mesh without and with a geometric separator, respectively. The geometric separator itself, however, serves as an approximate location where we wish the partition to occur, aligning with the geometry of the domain.

The use of geometric separators is particularly advantageous in 3D meshes, where minimizing the interface surface area can greatly improve the efficiency of parallel computations. This method of partitioning can be used in conjunction with other techniques, such as graph-based partitioning, to achieve a balance between partition quality and computational performance.

(a) Partition without a geometric separator, leading to a higher number of interface elements.



(b) Partition with a geometric separator, reducing the number of interface elements and communication overhead.

Figure 6: Comparison of partitions with and without geometric separators.

4.3 Strategic Pivot

The strategic pivot was made with the question: Why opt for METIS if the geometric separator of the mesh is already known?

Objective Realignment – The new focus shifted towards minimizing communication with the geometrically separated sub-domain. This approach aimed to reduce communication overhead by leveraging the known geometric properties of the domain to create more efficient partitions.

4.4 METIS Partitioning Objectives

Minimizing the Edgecut

Consider a graph $G = (V, E)$, and let P be a vector of size $|V|$ such that $P[i]$ stores the number of the partition that vertex i belongs to. The *Edgecut* N_{Edgecut}, of this partitioning is defined as the number of edges (u, v) for which $P[u] \neq P[v]$. If the graph has weights associated with the edges, then Edgecut is defined as the sum of the weights of these straddling edges.

$$N_{\text{Edgecut}} = \sum_{\substack{(u,v)\in E \\ P[u]\neq P[v]}} w(u, v), \tag{1}$$

where $w(u, v)$ represents the weight of the edge between vertices u and v.

4.5 Minimizing the Total Communication Volume

Let $G = (V, E)$ be a graph, and let P be a vector of size $|V|$ where $P[i]$ denotes the partition number to which vertex i belongs. Define $V_b \subset V$ as the set of interface (or border) vertices, where each vertex $v \in V_b$ is connected to at least one vertex in a different partition.

For each vertex $v \in V_b$, let $N_{\text{adj}}[v]$ represent the number of distinct partitions (other than

$P[v]$) to which the vertices adjacent to v belong. The *total communication volume* $N_{\text{total.v}}$, of this partitioning is defined as:

$$N_{\text{total.v}} = \sum_{v \in V_b} N_{\text{adj}}[v]. \tag{2}$$

4.6 Multilevel k-way Partitioning

A k-way partition P_m of the coarsest graph $G_m = (V_m, E_m)$ is computed, which divides V_m into k parts. The multilevel k-way partitioning algorithm consists of three main phases:

- **Coarsening**: The original graph is reduced to a smaller graph by collapsing vertices and edges, forming a hierarchy of coarser graphs $G_0, G_1, \ldots, G_m$.
- **Partitioning**: A k-way partition is performed on the coarsest graph G_m.
- **Uncoarsening**: The partitions are successively mapped back to the original graph, refining the partitions at each level to optimize the partition quality.

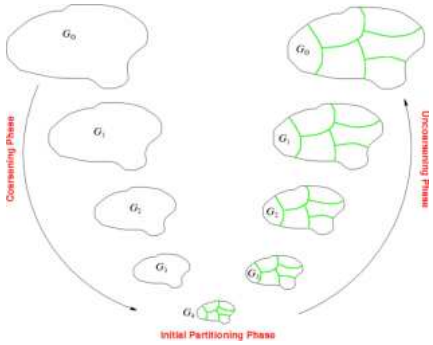


Figure 7: Multilevel k-way Partitioning process, showing coarsening, partitioning, and uncoarsening phases.

4.7 Multilevel R-Bisection

A 2-way partition P_m of the graph $G_m = (V_m, E_m)$ is computed, which partitions V_m into two parts, each containing approximately half of the vertices of G_0. The multilevel R-bisection algorithm follows a similar three-phase process as multilevel k-way partitioning:

- **Coarsening**: The graph is reduced by collapsing vertices and edges, creating a sequence of coarser graphs $G_0, G_1, \ldots, G_m$.
- **Partitioning**: A 2-way partition is performed on the coarsest graph G_m.

- **Uncoarsening**: The partition is projected back onto the finer graphs, with refinements applied at each level to improve partition quality.

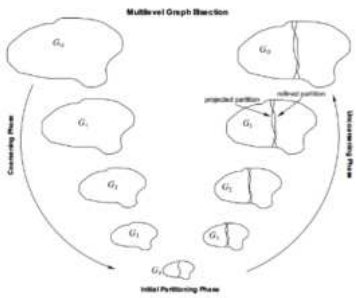


Figure 8: Multilevel R-Bisection Partitioning process, showing coarsening, partitioning, and uncoarsening phases.

4.8 METIS Partitioning Routines

`METIS_PartMeshDual`

This function is used to partition a mesh into k parts based on a partitioning of the mesh's dual graph.

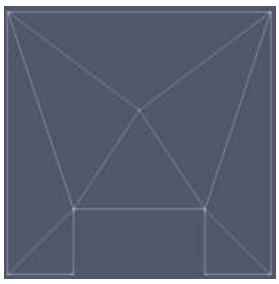
`METIS_PartMeshNodal`

This function is used to partition a mesh into k parts based on a partitioning of the mesh's nodal graph.

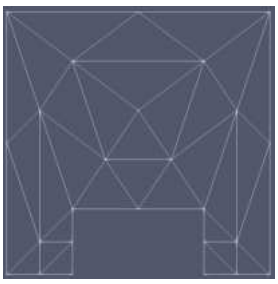
5 Meshes and Partitioning Outcomes

5.1 Geometrically Separated Meshes

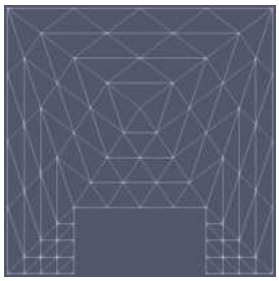
The figures below show various meshes that were used in the partitioning analysis. These include L1, L2, and L3 geometrically separated 2D meshes, as well as a transfinite 3D cube mesh.



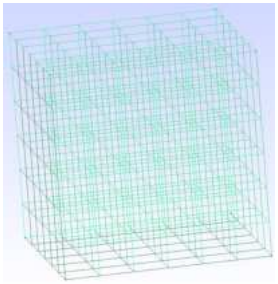
(a) L1 Geometrically Separated 2D Mesh.



(b) L2 Geometrically Separated 2D Mesh.



(c) L3 Geometrically Separated 2D Mesh.



(d) Transfinite 3D Cube Mesh.

Figure 9: Comparison of geometrically separated 2D meshes and a 3D cube mesh.

5.2 Algorithmic Combinations for Partitioning

Figure 10 illustrates the different algorithmic combinations used for partitioning each mesh. These combinations include Dual and Nodal approaches, along with K-way and R-Bisection methods for weighted and unweighted configurations.

Figure 10: Algorithmic Combinations for Partitioning in Single Partitioning Scheme. Over 300 files were generated per mesh.

5.3 Partitioning Results

The partitioning results for various configurations are presented below.

(a) L1, Dual K-way, 5 Partitions.

(b) L3, Dual K-way, 90 Partitions, Unweighted.

(c) L3, Nodal K-way, 100 Partitions, Unweighted.

(d) L3, Dual R-Bisection, 120 Partitions, Weighted.

Figure 11: Comparison of different partitioning methods across various partition sizes.

The figure shows that key partitioning methods fail at specific points. The first failure occurs at 40 partitions, where the Nodal method fails. At 90 partitions, the K-way method fails, and by 130 partitions, the Nodal method generates invalid files. Beyond these points, partitioning methods are gradually eliminated due to inefficiencies or failures.

6 Communication Metric and Stitching Outcomes

6.1 Communication Metric

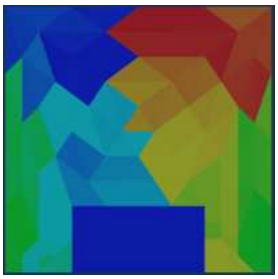
The communication metric calculates the following communication quantities within the mesh:

1. Communication Metric for the ‘Stitched’ Mesh
2. Total Communication Load
3. Average Communication per Partition (normalized)
4. Maximum Communication Load

6.2 Stitching Outcomes

Figures 12a and 12b show examples of partitioning for L3 mesh after stitching, with configurations for unweighted and weighted Dual R-Bisection into 120 partitions.

(a) L3, Dual R-Bisection, 120 Partitions, Unweighted, Stitched.



(b) L3, Dual R-Bisection, 120 Partitions, Weighted, Stitched.

Figure 12: Comparison of L3 Dual R-Bisection 120 Partitions, Unweighted and Weighted, Stitched.

6.3 Communication Analysis

The following diagrams illustrate the communication flows and metrics across different partition configurations, showing how the communication loads vary for different partitioning strategies.

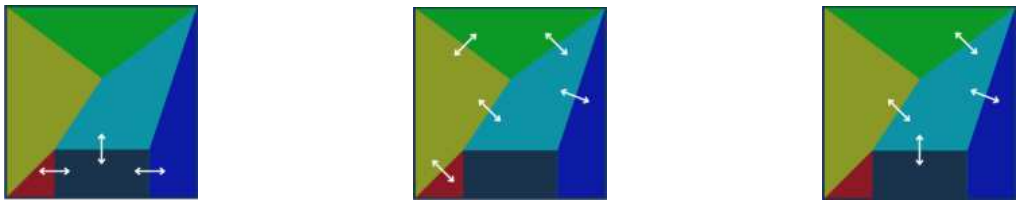


Figure 13: Examples of communication directions and flows in different partition configurations.

6.4 Communication Load Graph for Stitched Mesh

The graph in Figure 14 presents the communication load across a range of partition numbers for different partitioning schemes. It compares the stitched communication loads for Dual and Nodal approaches using K-way and R-Bisection methods, with both weighted and unweighted configurations.

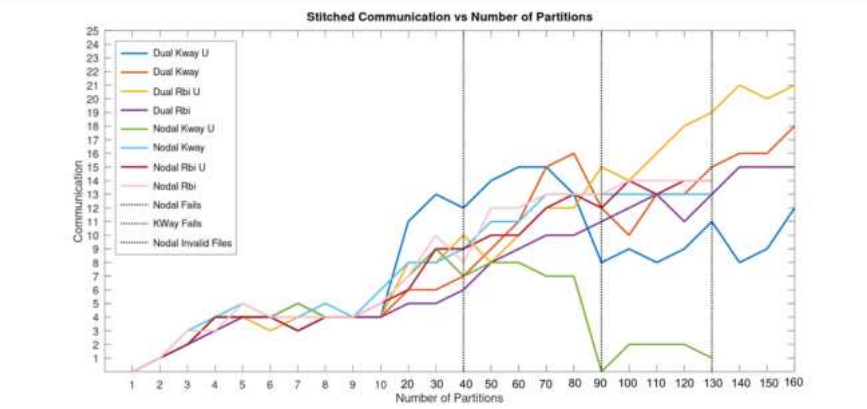


Figure 14: Communication load vs. number of partitions for the stitched mesh.

7 Communication Load Analysis

7.1 Stitched Communication Load Graphs

The figures below represent the communication load in the stitched mesh with varying numbers of partitions. The analysis compares the communication results for different partitioning schemes such as Dual R-Bisection and Nodal configurations, considering both weighted and unweighted settings.

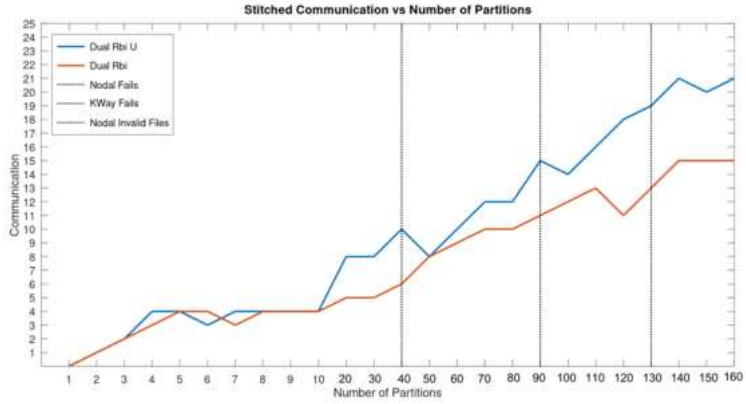


Figure 15: Stitched Communication vs. Number of Partitions for the Stitched Mesh.

7.2 Total Communication Load Graphs

The total communication load across different numbers of partitions is shown in Figures 16 and 17. The graphs display trends for various partitioning techniques and assess the maximum communication load reached at each step.

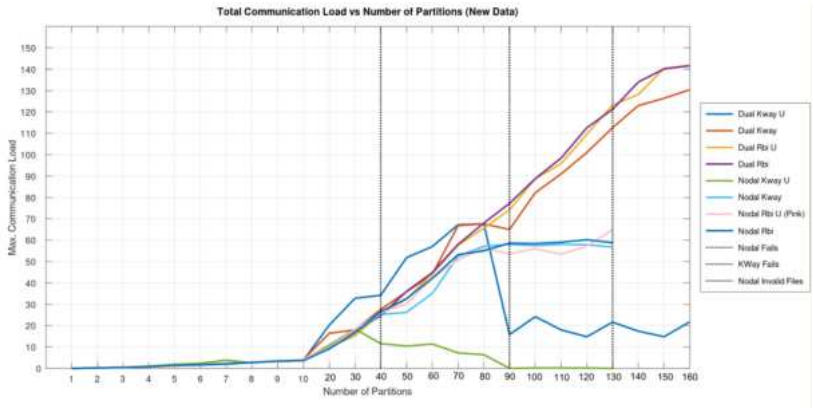


Figure 16: Total Communication Load vs. Number of Partitions.

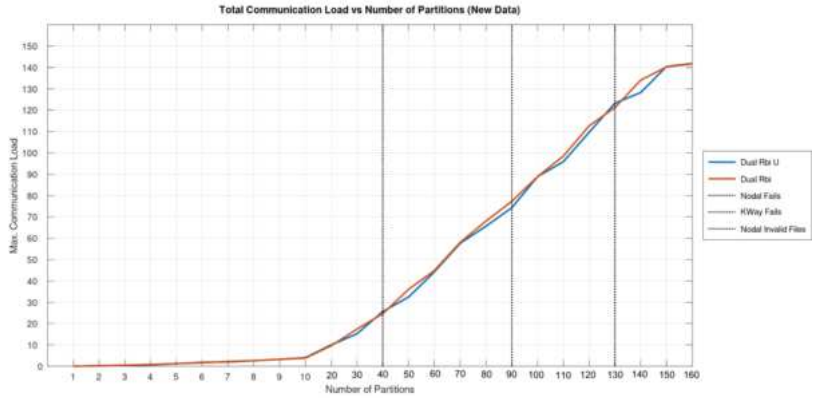


Figure 17: Total Communication Load vs. Number of Partitions.

7.3 Average Communication Load Graphs

Figures 18 and 19 demonstrate the average communication load per partition for different configurations, revealing the effectiveness of various partitioning strategies.

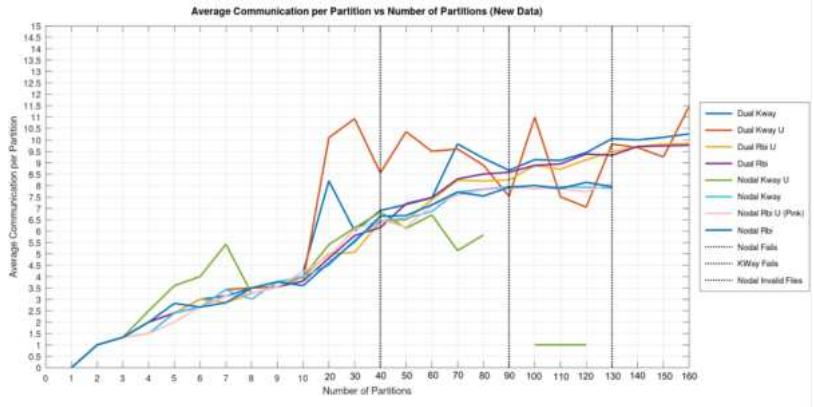


Figure 18: Average Communication per Partition vs. Number of Partitions.

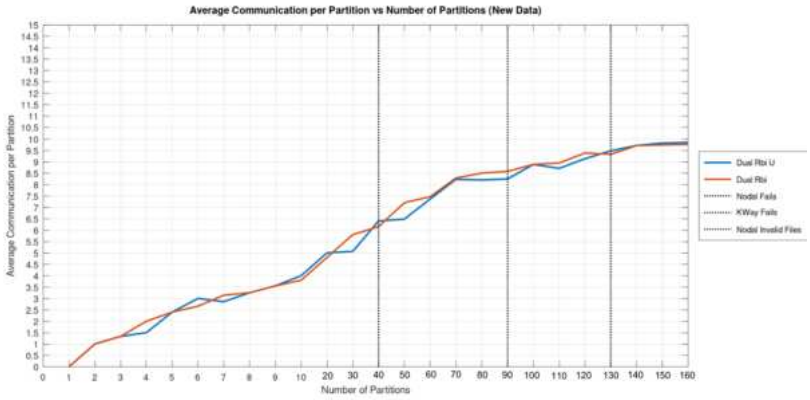


Figure 19: Average Communication per Partition vs. Number of Partitions.

7.4 Maximum Communication Load Graphs

The maximum communication load encountered during partitioning is visualized in Figures 20 and 21. These figures show the critical points where communication overhead becomes significant.

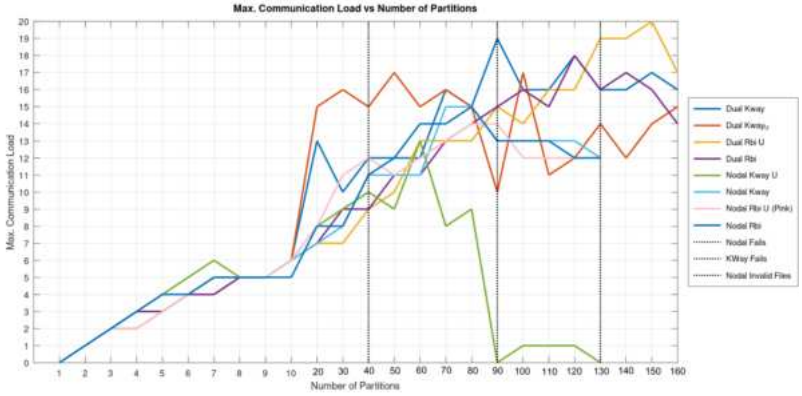


Figure 20: Maximum Communication Load vs. Number of Partitions.

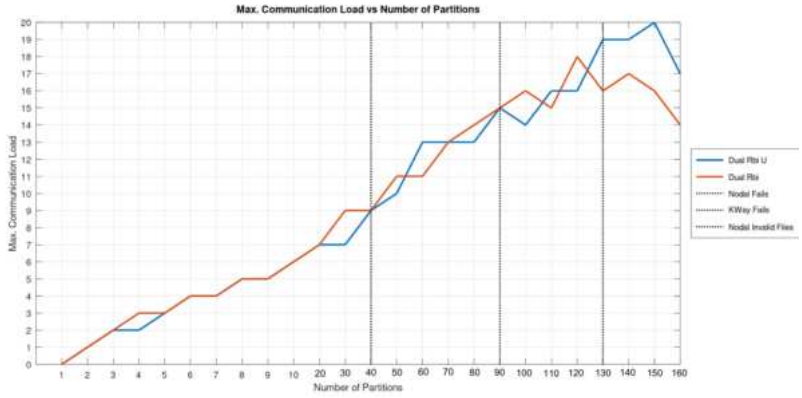


Figure 21: Maximum Communication Load for Different Partition Counts.

7.5 Interpretation

The presented graphs provide insights into the behavior of communication load across different partitioning methods, as well as a clear comparison between weighted and unweighted strategies. As discussed earlier, beyond approximately 10 partitions, the communication load behavior becomes increasingly erratic. Several partitioning methods are gradually excluded due to failures or inefficiencies as the number of partitions increases. The results clearly indicate that the Dual Recursive Bisection (RBi) methods—both weighted and unweighted—outperform the others in terms of communication efficiency, consistently exhibiting lower communication overhead as the number of partitions increases. This suggests that the weighted RBi offers a more robust and efficient solution for managing communication load in large-scale partitioning scenarios.

8 Drawbacks of Current Methodology

The current methodology presents certain challenges and limitations:

- **Difficulty** – Issues related to mapping the mesh to a graph.
- **Time Consumption** – Executing multiple commands adds complexity.
- **File Handling** – Generating multiple files for each partitioning task.

9 Data Formats and Ease of Parsing

9.1 File Format Selection for Mesh Mapping

In the process of mapping meshes to graphs, we considered several file formats to identify the most suitable one for our partitioning analysis. The following file formats were evaluated based on their ability to support mesh data and the ease with which the data could be parsed:

- `.vtk` – Visualization Toolkit format
- `.obj` – Wavefront object format
- `.blend` – Blender format
- `.msh` – Gmsh format
- `.stl` – Stereolithography format
- `.vtu` (XML-based format for VTK)

After a thorough analysis of the data structures and parsing methods required for each format, we decided to proceed with the `.vtk` file format. The primary reasons for this decision were:

- **Ease of Data Parsing**: The `.vtk` format provides a straightforward way to extract mesh data, such as node coordinates and element connectivity, which is essential for graph representation.
- **Support for Various Data Types**: It is versatile and widely used in computational geometry and visualization, with support for different types of meshes, including structured and unstructured grids.
- **Community and Tooling**: The `.vtk` format is well-supported by numerous libraries and tools, making it easier to integrate into the workflow for data processing and visualization.

While other formats like `.obj` and `.stl` are commonly used in computer graphics, they lack sufficient support for mesh connectivity and metadata required for advanced partitioning algorithms. Formats such as `.blend` and `.vtu` involve additional complexities in data extraction or require specialized software, making them less practical for our analysis.

Ultimately, using `.vtk` enabled us to streamline the mesh-to-graph mapping process and focus on refining the partitioning techniques.

10 Troubleshooting

10.1 Issues with Partitioning Algorithms

During the partitioning process, several challenges were encountered with specific algorithms:

- **Nodal Approach Failures**: The nodal partitioning algorithm consistently failed when the number of partitions exceeded 30. This issue led to crashes or the generation of incomplete partition files.
- **K-Way Failures**: The K-Way partitioning approach encountered errors when the number of partitions went beyond 90. Beyond this threshold, the algorithm failed to produce valid partition files.
- **Invalid Files with Nodal Method**: After 90 partitions, the nodal approach not only failed but also started generating invalid or corrupted files that could not be processed further.

10.2 Platform-Dependent Issues: Windows vs. Linux

Another significant troubleshooting aspect involved platform-specific issues:

- **Linux Limitations**: While attempting to construct the geometry on a Linux platform, the process consistently encountered failures, making it impossible to complete the task.
- **Switch to Windows**: To overcome the limitations faced on Linux, we shifted the geometry construction process to a Windows platform. This switch proved successful, allowing us to proceed with the intended workflows without further issues.

10.3 Improved Partitioning Pipeline

We implemented an improved partitioning pipeline to streamline the mesh-to-graph conversion and reduce manual intervention. Below is a comparison between the pre-existing pipeline and our improved version:

- **Pre-existing Pipeline**:
  1. Takes in a `.vtk` file.
  2. Converts it to a graph file for METIS routines.
  3. Runs METIS partitioning algorithms, generating part files.
  4. Converts the part files back to `.vtk` format.

5. Outputs the partitioned `.vtk` file.

- **Improved Pipeline**:

1. Takes in a `.vtk` file.
2. Uses C++ APIs directly to partition the mesh.
3. Outputs the partitioned `.vtk` file.

The improved pipeline eliminates several intermediate steps, resulting in a more efficient workflow. Figures 22 and 24 illustrate the two approaches.

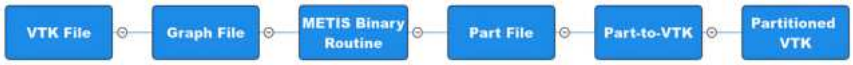


Figure 22: Pre-existing partitioning pipeline.

Figure 23: Improved partitioning pipeline.

10.4 Rendering Issues with 3D Structures

While using ParaView to render the volume of a 3D structure, the software frequently failed to display the geometry correctly. As a workaround, we developed a custom rendering solution using OpenGL, which allowed for accurate visualization of the 3D mesh.

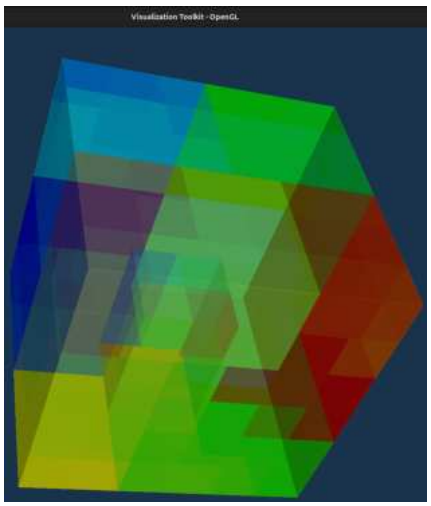


Figure 24: Improved partitioning pipeline.

11 Conclusion

The partitioning approaches applied in this project resulted in several key outcomes and optimizations:

11.1 Optimized Communication

The use of Dual, R-Bisection, and Weighted methods significantly minimized communication between the geometrically separated domains and other sub-domains. These approaches effectively reduced inter-domain communication, which is crucial for improving parallel computing efficiency in CFD simulations.

11.2 Performance Gains

A notable reduction in inter-domain communication was achieved compared to traditional partitioning methods. The partitioning algorithms provided substantial improvements in balancing the computational load while ensuring efficient data transfer between partitions.

11.3 Algorithm Outcome

The algorithms developed and implemented in this project generated valid partitions that met the specified constraints for the various meshes analyzed. These partitions were achieved with high accuracy, enabling effective domain decomposition for CFD applications.

12 Scope for Future Work

The following areas are identified for future exploration and potential improvements:

- **Multiple Geometrically Separated Partitions Analysis**: Extend the analysis to involve multiple geometrically separated regions and evaluate the impact on partitioning and communication metrics.
- **Slotted 3D Mesh Generation**: Develop and implement a method for generating slotted 3D mesh geometries to improve flexibility in partitioning complex domains.
- **Renderer Optimization**: Enhance the current rendering system, particularly focusing on optimizing the visualization of partitioned meshes and large-scale data.
- **Solver's Runtime Analysis**: Conduct a detailed runtime analysis of the solver to further optimize the partitioning algorithms and computational efficiency.

References

- Karypis, G., & Kumar, V. (1998). METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. University of Minnesota.
- Thompson, J., Soni, B. K., & Weatherill, N. P. (1999). Handbook of Grid Generation. CRC Press.
- STAR-CCM+ User Guide. (2020). Siemens Digital Industries Software.
- Geuzaine, C., & Remacle, J. F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11), 1309–1331.
- TU Chemnitz. (Year). *A Graph Survey.*