University of Innsbruck

Institute of Computer Science
Research Group DPS
(Distributed and Parallel Systems)

Topology Aware Data Organisation for Large Scale Simulations

Master Thesis

Supervisor: Herbert Jordan, PhD

Markus Walzthöni, Bsc. (1018830)

markus.walzthoeni@uibk.ac.at

Innsbruck
5 June 2017

Abstract

Partitioning meshes for large scale simulations with sufficient quality is a hard task and even advanced methods have a high time and memory complexity. We present methods to reorganize and partition meshes for large scale simulations with geometry topology aware methods. Compared with more elaborate methods the presented methods should be able to structure data with minimal effort in memory space and computation time. This thesis defines the necessary data structures, functions and algorithms to perform the transformations as well as quality metrics to compare the results. The thesis concludes with a set of experiments evaluating the quality of the selected methods as well as the state-of-the-art approach and compares the outcome of the experiments with each other.

Contents

List of Figures

1. Introduction

The field of computer science covers a wide range of different sub-fields i.e. logic, databases, artificial intelligence or security. In this thesis we focus on computational physics, more precisely the field of physically based simulations. The goal of these simulations is to model real world physical behaviour. To accomplish this, a model is established and algorithms and functions from numerical mathematics are used to approximate the desired behaviour. The model is typically represented by a mesh.

Meshes are data structures consisting of sets of nodes of different kinds and edges connecting the nodes. Different properties can be associated to nodes like positions, temperature, pressure, etc. These meshes take an important role in many fields in computer science. They are used as e.g. models for computer graphics, information networks or finite state machines. In this thesis we take a closer look at meshes for physically based simulations, in particular for finite element simulations.

These simulations consist of a mesh, associated attributes and a kernel. The kernel is an algorithm, which is applied on each single node in the mesh. Depending on the type of the node the algorithmic steps of the kernel can vary. The kernel simulates the change of the properties of the nodes over discretized time points. Typically there is a fixed time step Δt between two discretized points in time $t_{i+1} = t_i + \Delta t$. The computation from one time point to an other is called *iteration*. For further understanding, here is a small and simplified example of a temperature propagation simulation in a 2 dimensional space:

Example 1.1. The mesh consists of two node-types, cells and faces. The space is divided into areas of equal size. These areas are called cells. A face is connected with two spaces. The resulting mesh has two node-types (cells and faces) and two edge-types (cells to faces and faces to cells). The mesh is shown in Figure 1.1, where circles refers to cells and rectangles to faces. Additionally the cells have a attribute of temperature. The kernel consist of one step. in this step the kernel determines the new temperature of a cell, depending on the temperatures of the nodes connected to the neighbouring faces. E.g. two neighbouring cells have a temperature difference. So in one step of the simulation a part of the temperature/energy of the cell with the higher temperature will be transferred to the one with lower temperature.

This mesh with its initial step t_0 is shown in Figure 1.1. Figure 1.2 shows the

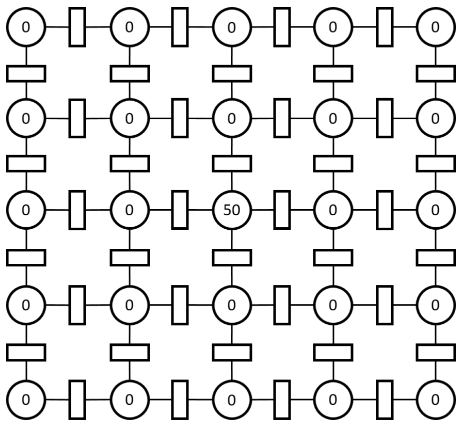

Figure 1.1.: Initial mesh

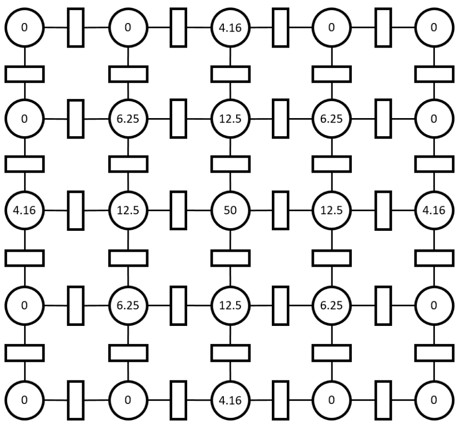

Figure 1.2.: After 3 steps

mesh after the third iteration of the kernel.

One important criterion of these simulations is the execution time. To minimize the execution time different approaches can be considered:

1. To compute the data of step t_{i+1} only data of step t_i is used. While the kernel is computing the data of step t_{i+1}, the list of properties accessed is data independent. This leads to a great opportunity for parallelizing and distributing the computation of the kernel.

2. Optimization of the organization of the data in the memory. The memory hierarchy of a modern computer is aimed at exploiting the principles of locality. Hence, data organized in the right order can lead to a decrease in execution time.

To use large scale parallel systems for the computation of the simulation the meshes and their calculations have to be separated into sub-meshes and computed on different systems. This brings up the need to partition meshes.

The *partitioning* of a mesh is defined as splitting up a mesh into a particular number of sub-meshes. A sub-mesh of a mesh includes a sub-set of the nodes of the original mesh and all edges, whose start and end point is in the node-set of the sub-mesh. The resulting *cuts* of a partition are all edges not covered by the sub-meshes of the partition. The set of edges from one sub-mesh to its neighbouring sub-meshes is called the *node-closure* of a sub-mesh.

The simulation of the sub-meshes are then computed on different nodes of a cluster. Before computing the attributes for a point in time, information from other sub-meshes and their attributes could be required. The nodes, which hold the mandatory information, are given by the closure of a sub-mesh. To compute a time step the attributes of the closure first need to be transferred to the node calculating the sub-mesh. When the transfer of the data is done, the next computational step can be performed. Performing a data transfer and a computational step is called a *cycle*.

What can be noticed is that a node of a cluster has to wait until all neighbouring sub-meshes has finished their computation to start a new cycle. Thus, to speed up the computation we need to minimize the time of a cycle. Since the longest running cycle becomes the bottleneck of the simulation, the goal is to minimize the maximal execution time of a single system. The execution time of one cycle consist of the time it takes to transfer data and to compute the kernel.

First lets take a look on the computation step. Lets assume, that the computational power of all nodes in a cluster computing a sub-mesh is about the same. Hence the execution time of a kernel applied to a node of the mesh on all nodes is about the same. To minimize the overall execution time of the computation phase we need to distribute the number of nodes of a mesh equally to all computing units. So one goal of the methods presented in this thesis is to distribute the number of nodes of a mesh evenly to all sub-meshes of a partition.

The next part is the time to transfer the required data. For simplification lets assume, that sending a data unit takes a constant amount of time, despite of network topology, installed hardware, used protocols and other elements differing elements. The data to be transferred is given by the attributes of the nodes of the node-closure of the sub-mesh. Therefore, the optimization target for the data transfer is that the total amount of data to be sent should be minimal. Keeping in mind that the duration of the computation phase of all systems is about the same, we need to minimize the maximal transfer time of all participant systems to minimize the cycle duration of all systems.

Consequently the *optimal* partition of a mesh is a partition where the maximal computation and transfer time of an executing system are kept minimal. So the properties of a good partition are:

- Evenly distribute the nodes over all sub-meshes

- Minimize the number of edges in the cuts of a partition

- Evenly distribute the cuts over all closures of the sub-meshes

But even the problem of finding a partition with evenly distributed nodes over the sub-meshes and the minimal number of edges in the cut is proven to be *NP-complete* [8].

So solving this problem by checking each single possibility is no option and to solve this problem in a reasonable amount of time heuristics are used. Standard heuristic algorithms creating partitions, like the Kernighan-Lin algorithm[12], have $\mathcal{O}(n^2 \log n)$ complexity. More advanced approximation algorithms such as the *k-way* algorithm used in the *METIS* library are able to calculate *good* partitions in $\mathcal{O}(n \log n)$. But this upper limit may be still be too slow for very big problems. In this thesis we attempt to investigate the quality of suitable heuristically methods for meshes exhibiting spatial coordinates for (some) of the node-types of a mesh. Suitable methods to partition meshes are methods, which give a result in $\mathcal{O}(n)$ with no or tiny memory overhead. To accomplish this we exploit the structure of the meshes used for these simulations. Typically nodes for these simulations have attributes like positions in $\mathbb{R}^2$ or $\mathbb{R}^3$ and there are rarely connections to nodes which are not a direct spatial neighbour. Possible approaches are based on mapping spatial coordinates to coordinates/values along space-filling curves

An other important factor for minimizing the execution time of a simulation is addressing the organization of the data structures of a mesh and the corresponding attributes in the memory. Facing the problem of optimal memory organization of the data to minimize the execution time of a simulation we have to take a closer look on the principle of *locality of reference*, specifically the principle of *spatial-/memory-* and *temporal*-locality. Spatial or memory locality states that if some data is required at a certain point of the computation, it is likely that neighbouring memory locations will be used in a short period of time afterwards. Temporal locality denotes that if data from a particular memory location is used, that there is a high probability the same data will be used again in a short period of time afterwards. As kernels of simulations typically calculate the next time-step with the attributes of neighbouring nodes, these neighbouring elements should also be placed close together in the memory.

Caches typically exploit these presented principles very efficiently. So if the organization of the data is in a fashion that neighbouring nodes are close to each other, hardware caches can exploit the resulting data locality and will speed up the simulation. More detailed information is presented in Section 3.5.

1.1. Motivation

The main motivation for this thesis is to investigate on methods to partition meshes

- in a minimal amount of time
- requiring no additional memory

Beside these 2 restrictions the quality of the partition should be still sufficient. Both bullet points are important when either dealing with big meshes or limited main memory. State-of-the-art *k-way* partitioning methods have a $\mathcal{O}(n\log(n))$ complexity. But even this could take too much time, when dealing with meshes holding billions of nodes. An other problem occurs, when the mesh almost exceeds the main memory. All non-heuristic partitioning methods need additional memory, which may be not available. These state-of-the-art approach i.e. need typically about $\mathcal{O}(n + (\frac{n}{k})^2)$ additional memory 4.2.2.

1.2. Objectives

The objectives for this master thesis are the following:

- Finding suitable $\mathcal{O}(n)$-methods to partition meshes exhibiting spatial information
- Defining a suitable set of quality metrics for evaluating identified methods
- Evaluation and comparison of the implemented methods in a set of test cases

The quality metric should give the reader a better understanding, if a method is creating a suitable partition. Therefore metrics have to established which measure the quality of the cuts, the cohesion of the sub-meshes, and the balance of the number of contained nodes of the sub-meshes. This will be accomplished with statistical analysis and metrics i.e. evaluating the coherence of the created sub-meshes.

1.3. Challenges

The first challenge is to establish a work-flow which will partition a given mesh. This work-flow uses different algorithms to perform the necessary steps to:

1. Creation of a mesh
2. Applying a method to partition the mesh
3. Reordering the whole structure to increase locality
4. Testing the outcome of the partition

To accomplish this lots of intermediate steps and additional structures will be needed. The intermediate steps and the used data structures will be accurately described in the thesis.

An other challenge is to find out, if there are methods which are faster than state-of-the-art methods for the computation of a partition. These methods should also produce suitable solutions compared to the state-of-the-art method.

An important factor is the quality of the outcome of a partitioned mesh. The quality of an outcome includes:

- The partitioned sub-meshes should be balanced in terms of the number of nodes they contain
- The number of edges in cuts, which separate the sub-meshes should be as minimal as possible
- The closures of the computed sub-meshes should be as minimal as possible
- Increased locality of the nodes to provide higher cache-locality and cache-efficiency

1.4. Overview

The first chapter after this is devoted the related work. In Chapter 2 articles and papers handling similar topics are presented.

The next part of the thesis given in Chapter 3 defines the basic structures, utility-functions and algorithms. The Chapter clarifies the descriptions given in this Chapter and explains how a meshes, attributes, mesh structures, kernels, partitionings, partitions, cuts etc. are defined. A part of the Chapter explains what a good partitioning is and how the quality of a partition is measured.

Chapter 4 of this thesis is devoted to the partitioning of a mesh. To partition a mesh and reorder it according to the partitioning a work-flow is presented and each step explained in detail. The last part of this chapter is giving an overview of the used partitioning method used in the thesis.

In Chapter 5 we present a comparison of different partitioning methods, test results and quality metrics. The tests will include measurements of the computation time to partition the meshes and the execution time of the reordered meshes used in various simulations. This section should give an overview about the advantages and disadvantages of the used methods establishing a partition compared to state-of-the-art methods.

The last chapter of this thesis is concluding the experimental section and presenting future work.

2. Related Work

Because the mesh partitioning is a big topic in computer science there are several contributions related to the topic of this thesis. A lot of papers also try to find partitioning methods which have linear time complexity regarding the number of nodes of a mesh. Another category of contributions finds ways to organize the memory in a fashion that improves the execution time.

Most contributions accomplish linear time complexity for partitioning using space-filling curves. The most popular curves are the Hilbert-3D-, the Sierpinski- and the Z-curve i.e. used by Stefan Schamberger and Jens-Michael Wierum et al. [16] and Hui Liu et al. [14]. Schamberger et al. [16] additionally uses the $\beta\Omega$-curve presented by Wierum et al. [18]. In all of these contributions the Sierpinski-curve gave the least promising results. This curve is therefore not used in this thesis. This thesis will cover the 2D and 3D variants of the Hilbert-, the Z-curve, the Peano-curve and as state-of-the-art partitioning method the k-way algorithm of the METIS-library.

The meshes used for evaluation differ in the mentioned contributions of Schamberger et al. [16] and Liu et al. [14]. Schamberger et al. [16] uses a variation of 2D and 3D meshes used for finite element simulations and a standard 100×100 grid for the partitioning with a range of 4253 to 320194 nodes and 12000 to 3.7 million edges. Instead Liu et al. [14] apply the mentioned partitioning methods on a 3D cylinder with 2.5 million nodes and a 2D irregular structure with holes and 3.7 million nodes. For the evaluation of the partitioning method this thesis will present a wider variety of meshes. The presented mesh-types are structured meshes / grids, unstructured meshes with randomly arranged nodes fulfilling the Delaunay triangulation shown in Definition 3.25, and unstructured meshes with random nodes and random edges between the nodes. The used mesh-shapes are 2D and 3D variations of all mesh-types. From each mesh-type a mesh is created with different numbers of nodes, to get a broad overview.

To evaluate the partitions Schamberger et al. [16] uses the number of edges in the cut and the number of boundary nodes of the single sub-meshes. Different to this the quality metrices used in [14] focus on the distribution of the nodes to the sub-meshes, the maximal communication overhead by one single system and the overall communication of all systems. The paper of Liu et el. [14], similar to this thesis, also uses the distribution as a metric to evaluate the spreading of the nodes and the number of edges in the node-closure (Definition 3.37). Additional to the other contributions this thesis utilize to analyze the created sum-meshes of a partitioning Coverage value [3] (Definition 3.50).

All contributions, including this thesis, use the k-way algorithm from the METIS-library [11] as state-of-the-art partitioning method to compare their solutions with.

Addressing the organization of the data in memory two interesting papers, one by Yoon et al. [19] and the other by Dennis et al. [6], are related to this thesis. The main purpose of Yoon et al. [19] is to investigate metrics which help in predicting the number of cache-misses and cache-hits as a function of the organization of the data of a mesh in memory. Nevertheless, this thesis also presents detailed execution time and cache related measurements of structures organized with space-filling curves. Dennis et al. [6] uses the space-filling curves to organize a two dimensional structure of the world map to speed up their finite difference simulation of the world's oceans.

Both contributions try to accomplish their goal with different approaches. Yoon et al. [19] partitions with the help of the $\beta\Omega$-curve, Hilbert-curve, Z-curve, H-order, row-by-row- and diagonal-by-diagonal-layout. Dennis et al. [6] instead uses a combination of the Hilbert-2D-, a variation of the Peano- and the Cinco-curve to organize the data of their mesh. The best results have been provided by Hilbert- and the $\beta\Omega$-curve.

The sizes of the meshes of both papers are quite small. Dennis et al. [6] used meshes consisting of around 115000 nodes in a 2D structure. The basic structures organized in Yoon et al. [19] have 4000 to 60000 nodes with a regular 2D structure and one experiment is done with a 3D structure consisting of 150000 nodes. The biggest data structure this thesis has $1 \cdot 10^9$ nodes. An overview of the structure-types used in this thesis can be found in a detailed description given in Section 5.1.1.

The quality metrics used in Yoon et al. [19] are cache-related measurements. Dennis et al. [6] are only interested in the final computation speed. The comparison of the individual data organization methods is done by the execution time of the restructured meshes running a simulation kernel. This thesis uses the same metric to evaluate the organization of the data structures.

3. Definitions

This chapter is devoted to formalize the informal the descriptions of Chapter 1. The structures, operations on the structures, challenges and task from the introduction will be defined.

The initial step is to introduce the mesh through a formal definition.

3.1. Mesh

To get a sufficient definition we first need to define *nodes* and *edges*.

Definition 3.1 (Nodes)**.** Let $n \in \mathbb{N}$ be the number of different node-types and $V_1, \ldots, V_n$ be sets of *nodes* of different node-types. The *number of nodes* in a set V_i of nodes is denoted by $|V_i|$. $V^* = \{V_1, \ldots, V_n\}$ is a set of sets of nodes of different node-types. $|V^*| = \sum_{1 \leq i \leq n} |V_i|$ denotes the sum of the number of nodes of all sets. Let $\mathbb{V}$ be the set of all sets of sets of nodes of different node-types.

Definition 3.2 (Edges)**.** Let $V_1, \ldots, V_n$ be sets of nodes. Let $1 \leq i, j \leq n$. An *edge* is defined as a pair $(a, b) \in V_i \times V_j$. Let $E_{ij} \subseteq V_i \times V_j$ be an *edge-set* describing a set of edges from nodes in V_i to nodes in V_j of *edge-type* $V_i \times V_j$. The number of edges in an edge-set is given by $|E_{ij}|$. Let $\mathbb{E}$ be the set of all sets of different edge-types.

As our mesh definition will provide multiple edge-sets of the same edge-type we need further structures to describe those.

Definition 3.3 (Relations)**.** Let $V_1, \ldots, V_n$ be sets of nodes. Let k be a $\mathbb{N}^{n \times n}$ matrix. Let $1 \leq i, j \leq n$ and k_{ij} denoting the number of different edge-sets of the edge-type $V_i \times V_j$. The *relation-set* is a set of edge-sets of the same kind and defined as $R_{ij} = \{E_{ij}^1, \ldots, E_{ij}^{k_{ij}}\}$. The number of different edge-set in a relation-set is given by $k_{ij} = |R_{ij}|$. Let $R^* = \{R_{11}, \ldots, R_{1n}, \ldots, R_{n1}, \ldots, R_{nn}\}$ be the set of all sets of relation-sets. Let $\mathbb{L}$ be the set of all sets of different relation-types.

Definition 3.4 (Mesh)**.** Let V^* be a set of sets of nodes. Let R^* be a set of relation-sets. A *mesh* M is defined as $M = (V^*, R^*)$. Let $\mathbb{M}$ be the set of all meshes.

To elaborate the initial Example 1.1 from the Introduction in Chapter 1 on page 3, lets define that mesh structure.

Example 3.5 (Mesh definition)**.** This example consists of two different node-types, namely the types of *Cell* and *Face*. These two are given as V_1 and V_2. So $V^* = \{V_1, V_2\}$. Inferring from these node-types we get 4 different edge-types $V_1 \times V_1,\ V_1 \times V_2,\ V_2 \times V_1,\ V_2 \times V_2$. In the initial Example 1.1 we only need 1 edge-set in the relations of $V_1 \times V_2$, and $V_2 \times V_1$ and none of the other. So we get

$$k = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

So our resulting $R^* = \{\{\}, \{E^1_{12}\}, \{E^1_{21}\}, \{\}\}$ where E^1_{12} and E^1_{21} are edge-sets. Concluding we get our mesh $M = (V^*, R^*) = (\{V_1, V_2\}, \{\{\}, \{E^1_{12}\}, \{E^1_{21}\}, \{\}\})$

To work with meshes additional utility functions are required. To access the data of the mesh, functions to gather different sets - node-sets, relation-sets or edge-sets - are required.

Definition 3.6 (Node-set operator)**.** Let $M = (V^*, R^*)$ be a mesh. Let $n = |V^*| \in \mathbb{N}$ be the number of different node-types in V^* and $1 \leq i \leq n$. Let

$$\begin{aligned} \text{nodeset}_i \colon \mathbb{V} &\longrightarrow V^* \\ \{V_1, \dots, V_n\} &\longmapsto V_i \end{aligned}$$

be the function to extract V_i from M.

Definition 3.7 (Relation-set operator)**.** Let $M = (V^*, R^*)$ be a mesh. Let $n = |V^*| \in \mathbb{N}$ be the number of different node-types in V^*. For $1 \leq i, j \leq n$ let

$$\begin{aligned} \text{relationset}_{ij} \colon \mathbb{M} &\longrightarrow R^* \\ (V^*, \{R_{11}, \dots, R_{nn}\}) &\longmapsto R_{ij} \end{aligned}$$

be the function returning the relation-set R_{ij}.

Definition 3.8 (Edge-set operator)**.** Let $M = (V^*, R^*)$ be a mesh. Let $n = |V^*| \in \mathbb{N}$ be the number of different node-types in V^*. For $1 \leq i, j \leq n$ and $1 \leq k \leq |R_{ij}|$ let

$$\begin{aligned} \text{edgeset}_{ij,k} \colon R^* &\longrightarrow R_{ij} \\ R_{ij} &\longmapsto E^k_{ij} \end{aligned}$$

be the function returning the k-th edge-set from the relation-set R_{ij}.

An other important utility for a mesh is to get the neighbours of a certain edge-type of a given node.

Definition 3.9 (Partial neighbourhood of a node)**.** Let $M = (V^*, R^*) = (\{V_1, \ldots, V_n\}, R^*)$ be a mesh and $v \in V_i \subseteq V^*$ be a node of this mesh. Let R_{ij} be the relation-set of edge-type $V_i \times V_j$ containing k_{ij} edge-sets. Let $W^* = \{W_1, \ldots, W_n\}$ be a set of node-sets with $W_i \subseteq V_i$ for $1 \leq i \leq n$. Let $1 \leq l \leq k_{ij}$ Then

$$\begin{aligned} \mathrm{neigh}_{ij} \colon \mathbb{M} \times V_i &\longrightarrow W^* \\ (M, v) &\longmapsto W_j = \{w | (v, w) \in \bigcup_{1 \leq l \leq k_{ij}} \mathrm{edgeset}_{ij,l}(\mathrm{relationset}_{ij}(M))\} \end{aligned}$$

is the set containing all nodes from V_j being connected to v with an edge of type $V_i \times V_j$.

The next step is to get all neighbours from a node of certain type.

Definition 3.10 (Neighbourhood of a node)**.** Let $M = (V^*, R^*)$ be a mesh and $v \in V_i \subseteq V^*$ be a node of this mesh. Let $n = |V^*|$ be the number of different node-types in V^*. Then

$$\begin{aligned} \mathrm{neigh} \colon \mathbb{M} \times V_i &\longrightarrow \mathbb{V} \\ (M, v) &\longmapsto W^* = \bigcup_{1 \leq j \leq n} neigh_{ij}(v) \end{aligned}$$

be the set of sets containing the nodes connected with v.

We now have a sufficient structure to define nodes of different types with the appropriate edges. But to compute a physically based simulation we need more. We need the possibility to define properties for certain node-types as temperature, pressure or positions. The nodes of a mesh should have the ability to *hold* attributes.

Definition 3.11 (Attributes)**.** Let V^* be a set of sets of node-types. A attribute A is a function with the domain of a certain node-type $N \in V^*$ and the image of an arbitrary set B.

$$A \colon N \longrightarrow B$$

Let $\mathbb{A}$ be the set of all attributes.

Example 3.12 (Attributes)**.** Let $M = (V^*, R^*)$ be the mesh from Example 3.5. We would like that the nodes in V_1 have the attribute of temperature. The temperature in this example both values in the set of $\mathbb{R}$. So the attribute is defined as

$$\begin{aligned} \mathrm{A}_1 \colon V_1 &\longrightarrow \mathbb{R} \\ v &\longmapsto A_1(v) := \text{temperature of cell } v \end{aligned}$$

Similar we can now also introduce attributes like acceleration, velocity, pressure, positions, ids or measures of length. To simplify the handling of all attributes a sufficient structure is established.

Definition 3.13 (Attribute-set)**.** Let $M = (V^*, R^*)$ be a mesh. Let $A_1, \dots, A_m$ be attributes of the mesh. A *attribute-set* $A = \{A_1, \dots, A_m\}$ is defined as a set holding these attributes. The number of all attributes of a attribute-set is given by $|A|$.

Definition 3.14 (Attribute operator)**.** Let $A = \{A_1, \dots, A_m\}$ be a attribute-set. For $1 \leq i \leq m$ let

$$\begin{aligned} \text{attribute}\colon \mathbb{A} \times \mathbb{N} &\longrightarrow A \\ (\{A_1, \dots, A_m\}, i) &\longmapsto A_i \end{aligned}$$

be the function to extract A_i from A.

This thesis will focus on meshes $M = (V^*, R^*)$ with spatial attributes. Spatial attributes or positions in a m-dimensional space are typically defined as $A\colon V^* \to \mathbb{R}^m$. W.l.o.g all spatial attributes of a given mesh have the same dimension.

Definition 3.15 (Spatial attribute operator)**.** Let $M = (V^*, R^*)$ be a mesh. Let $A = \{A_1, \dots, A_m\}$ be an attribute-set associated with the mesh M. Let $I = \{i \mid 1 \leq i \leq m \wedge A_i \text{ is a spatial attribute of } M\}$ Then

$$\begin{aligned} \text{spatial-attributes}\colon \mathbb{A} \times 2^{\mathbb{N}} &\longrightarrow \mathbb{A} \\ (A, I) &\longmapsto B = \{\text{attribute}(A, i) \mid i \in I\} \end{aligned}$$

Definition 3.16 (Structured and unstructured mesh)**.** Let $M = (V^*, R^*)$ be a mesh, $A = \{A_1, \dots, A_m\}$ an attribute-set of M and $S = \text{spatial-attributes}(M, A)$ the set of all spatial attributes of A. A *structured mesh* or *grid* is a mesh where the positions of all nodes $n \in (V_i)_{i \in I}$ are uniformly distributed in each dimension. All other meshes with spatial attributes will be stated *unstructured mesh.*

Example 3.17. Let $M = (V^*, R^*) = (\{V_1\}, \{\{E_{11}\}\})$ be a mesh with $V_1 = \{(x, y) \mid 1 \leq x, y \leq 5\} \subseteq \mathbb{N}^2$. Let

$$\begin{aligned} A_1\colon V_1 &\longrightarrow \mathbb{R}^2 \\ x &\longmapsto x \end{aligned}$$

be an attribute of V_1. M is a grid or structured mesh. Figure 3.1 shows the nodes with their spatial distribution.

But we also cover different unstructured meshes with special properties. One property of interest is when nodes are triangulated to other node which are close to them. This property ensures that no edge cross over an other. To get a mesh with this property we use the *n-dimensional Delaunay triangulation.*

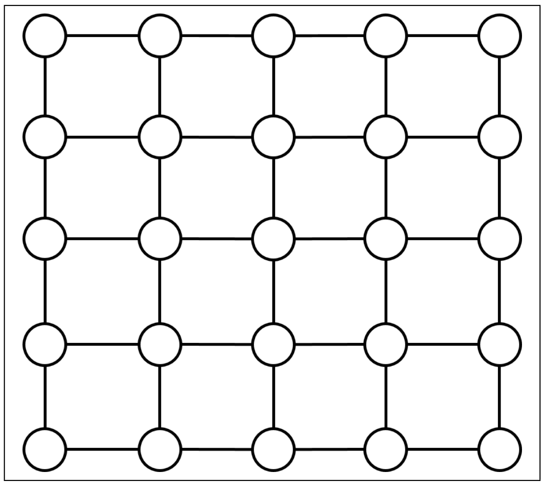

Figure 3.1.: Grid with 25 nodes

Definition 3.18 (n-simplex)**.** Let $u_0, \dots, u_n \in \mathbb{R}$ be a be linear independent points in a n-dimensional space. A *n-simplex* is a n-dimensional set of points defined as

$$S = \{\alpha_0 u_0 + \dots + \alpha_k u_k \mid \sum_{i=0}^{k} \alpha_i = 1 \text{ and } 0 \leq i \leq k \colon \alpha_i \geq 0\}$$

Example 3.19. A 2 dimensional simplex is a triangle and a 3 dimensional simplex is a tetrahedron.

Definition 3.20 (n-contact)**.** Let $a_1, \dots, a_n$ the linear independent points in a n-dimensional space. A *n-contact* is defined as the a set of points

$$\text{n-contact} = \{\alpha_1 a_1 + \dots + \alpha_k u_k \mid \sum_{i=1}^{k} \alpha_i = 1 \text{ and } 1 \leq i \leq k \colon \alpha_i \geq 0\}$$

Definition 3.21 (n-D triangulation)**.** Let $M = (V^*, R^*)$ be a mesh, $A = \{A_1, \dots, A_m\}$ an attribute-set of M and $I \subseteq \{i \mid 1 \leq i \leq |V^*|\} \times \{j \mid 1 \leq j \leq m\}$ with $|I| \geq 1$. $(x, y) \in I$ denote to a node-types V_x in V^* with a corresponding n-dimensional spatial attribute A_y. A n-dimensional *triangulation* [7] of a mesh M is a set of n-Simplex τ, such that

- The set of all points

  $$\bigcup_{(i,j) \in I} \bigcup_{1 \leq k \leq |V_i|} A_j(v_k), \quad v_k \in V_i$$

  is equal to all points from the set of n-Simplex

- Convex hull of the set of points

  $$\bigcup_{(i,j) \in I} \bigcup_{1 \leq k \leq |V_i|} A_j(v_k), \quad v_k \in V_i$$

  is equal to $\bigcup_{T \in \tau} T$

- $\forall T, U \in \tau \mid T \neq U$ the intersection of $T \bigcap U$ is either a n-contact or empty

Definition 3.22 (n-ball)**.** A *n-ball* with radius r is a set of points $S^n = \{x \in \mathbb{R}^n \mid ||x||_2 \leq r\}$

Example 3.23. A 1-ball with radius r is a set of points of an interval $[-r, r]$ forming a line from $-r$ to r, a 2-ball with radius r is the area bounded by a circle.

Definition 3.24 (n-circumsphere)**.** The *n-circumsphere* in an n-dimensional space of a triangle is defined as the (n)-ball that passed through all three vertices of the triangle. The center of this (n)-ball is the cross of all lines that are orthogonal and passing the midpoint of the edges connecting the triangle.

Definition 3.25 (n-dimenional Delaunay triangulation)**.** Let $M = (V^*, R^*)$ be a mesh with a finite number of nodes and $I \subseteq \{i \mid 1 \leq i \leq |V^*|\} \times \{j \mid 1 \leq j \leq m\}$ with $|I| \geq 1$. $(x, y) \in I$ denote to a node-types V_x in V^* with a corresponding n-dimensional spatial attribute A_y. A triangulation of a finite point set

$$P = \bigcup_{(i,j) \in I} \bigcup_{1 \leq k \leq |V_i|} A_j(v_k), \quad v_k \in V_i$$

is called a *n-dimensional Delaunay triangulation* [7], if the n-circumsphere of every triangle is empty, that is, there is no point from P in its interior.

Definition 3.26 (Unstructured mesh with close neighbourhood)**.** Let $M = (V^*, R^*)$ be a unstructured mesh with a finite number of nodes and $I \subseteq \{i \mid 1 \leq i \leq |V^*|\} \times \{j \mid 1 \leq j \leq m\}$ with $|I| \geq 1$. $(x, y) \in I$ denote to a node-types V_x in V^* with a corresponding n-dimensional spatial attribute A_y. M is an *unstructured mesh with close neighbourhood* if the nodes

$$\bigcup_{(i,j) \in I} \bigcup_{1 \leq k \leq |V_i|} A_j(v_k), \quad v_k \in V_i$$

and edges in $\{R_{ij} \mid i, j \in I\} \subseteq R^*$ fit the definition of a n-dimensional Delaunay triangulation.

Example 3.27. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a unstructured mesh with close neighbourhood with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$. Figure 3.2 shows the mesh with its spatial distribution.

3.2. Simulation

With the previous definition various mesh structures with corresponding attributes can be defined.

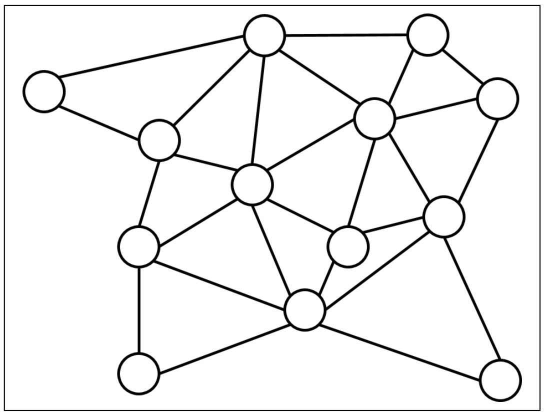

Figure 3.2.: Unstructured mesh with close neighbourhood inhabiting 13 nodes

A physically based simulation typically consists of a mesh structure with attributes and a kernel executing the simulation. The simulation itself is discretized over time. This means, that the simulation starts at a time t_0. The task of the kernel is to approximate the attributes of the mesh for the next time t_1, occurring after a time step Δt after t_0. The approximation is based on the attributes P^{t_0} of the mesh at the time t_0. The new calculated values are the attribute-set P^{t_1}. The calculation step for all node is also called the *computation phase.*

These simulations have to be computed on a computer. As there are various types of architectures and compositions of computers we need a general definition.

Definition 3.28 (Node, Cluster)**.** In this thesis we refer to a *computing node* as a hardware unit with at least one central processing unit and random access memory. Additional the computing node can have a mass storage device, I/O-devices or a graphic processing unit. A distributed system consisting of more than one node is called a *cluster.*

Example 3.29 (Kernel)**.** Elaborating on the Example 1.1 from the Introduction, we define a function applied to each node of the mesh which calculates the temperature diffusion from a time t_i to t_{i+1}. Let $M = (V^*, R^*)$ be the mesh defined in Example 3.5 with the attributes defined in Example 3.12 with the attribute-set $A^t = \{A_1\}$ at time t. Let

$$NC(v) := \{c \in V_1 \mid \forall f \in neigh(v)\colon \forall c \in neigh(f)\colon c \neq v\}$$

be an auxiliary function. Let

$$A^{t+1} = A^t(v) + \sum_{c \in NC(v)} A^t(c) - A^t(v)$$

be the function applied to all nodes of M.

The following Figures 3.3 to 3.6 illustrate the iterative process of the heat diffusion kernel. To show the spread of the temperature the node in the middle has a constant value of 50.

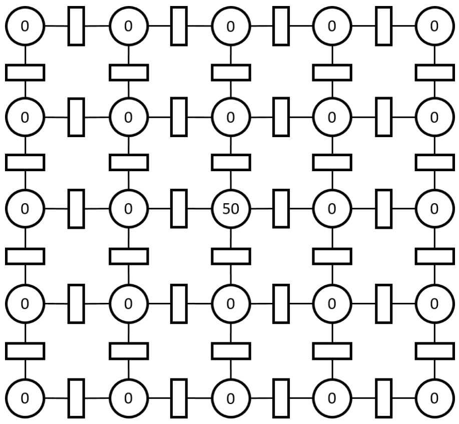

Figure 3.3.: Initial mesh

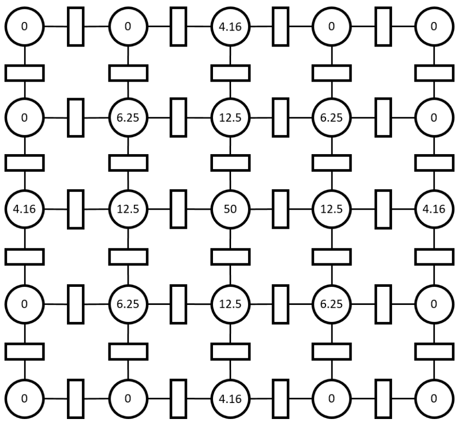

Figure 3.4.: After 3 steps

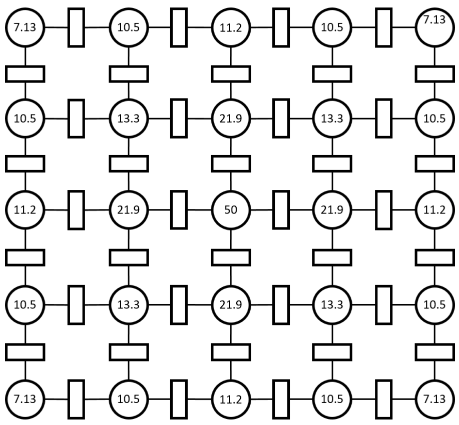

Figure 3.5.: Mesh after 8 steps

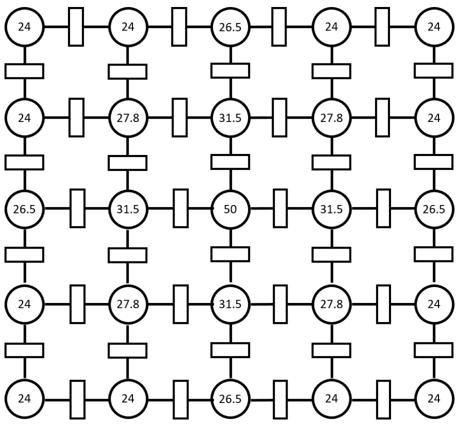

Figure 3.6.: Mesh after 18 steps

These simulations have interesting properties.

The data used to compute a solution for a time t_{i+1} is given by the data from the attribute-set of time t_i. While computing the attribute-set of time t_{i+1} there are no writing operations on the data of t_i. Also there is no attribute in the attribute-set of t_{i+1} depending on an attribute of the same time. Therefore the computation of the attribute-set of time t_{i+1} can be executed in a parallel fashion on a cluster. From this property we gain advantages. One is the acceleration of the simulation, using the parallel computing possibilities of one computing node or the computation power of a cluster or a combination of both.

Also the dependencies of the computation of a single node n is locality-bound. Most computational kernel functions use the information from the direct neighbours $\text{neigh}(v)$ to calculate the attributes of the next step. It provides the possibility to split the mesh and its data structure up into numerous parts and compute the single parts on a cluster. To perform the computational kernel functions for the nodes on the boundary of such a part, the information for other parts has to be gathered. This property can be used when the mesh and its attributes are exceeding the capacity of the memory of a computing node. However, this brings up the need to partition a mesh.

3.3. Partition of a mesh

Definition 3.30 (Partition)**.** A *partition* $P = \{M_1, \dots, M_k\}$ of a mesh $M = (V^*, R^*)$ with $n = |V^*|$ is the resulting set of splitting up a mesh into $k \in \mathbb{N}$ parts $M_1, \dots, M_k$ having the properties

- For $1 \leq i \leq n$

  $$M_i = (V_i^*, R_i^*) = (\{V_1^i, \dots, V_n^i\}, \{R_{11}^i, \dots, R_{1n}^i, \dots, R_{n1}^i, \dots, R_{nn}^i\})$$

  is a mesh. $R_{xy}^i \subseteq R_i^*$ is defined as a relation-set of edge-sets of edges connecting nodes of node-type x with nodes of type y. ${}^iE_{xy}^l$ is defined as an l-th edge-set of the relation-set R_{xy}^i.

- For $1 \leq i \leq k$ the disjoint set union of the node-sets of all sub-meshes

  $$V_i^1 \biguplus \cdots \biguplus V_i^k = V_i$$

  equals the node-set of the original mesh.

- For $1 \leq u, v \leq n, 1 \leq i \leq k$ and $1 \leq l \leq |R_{uv}^i|$ the edge-sets

  $${}^iE_{uv}^l = \{(x, y) \in E_{uv}^l \mid x \in V_u^i,\ y \in V_v^i\}$$

  contains all edges of the original mesh where the outgoing an incoming nodes are in a set of the set V_i^*.

The parts $M_1, \dots, M_k$ are called *sub-meshes*.

Let $\mathbb{P} = 2^{\mathbb{M}}$ be the set of all different partitions.

To demonstrate the creation of a partition of a mesh, lets look on the following example.

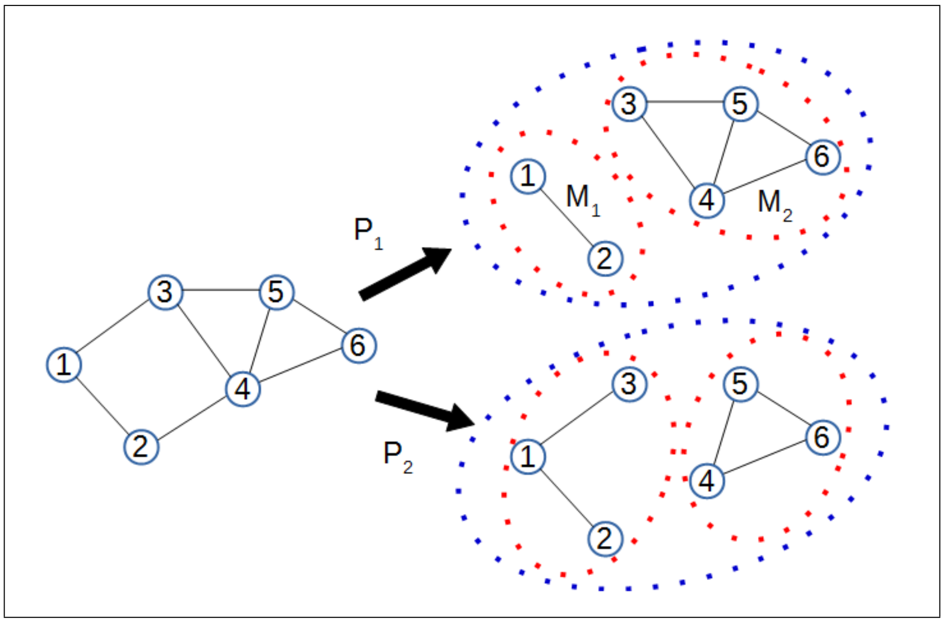

Figure 3.7.: Two partitions of a mesh

Example 3.31. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$.

Displayed in Figure 3.7 are 2 different results for a partition of a mesh.

1. Partition $P_1 = \{M_1, M_2\}$:

   - $M_1 = (V_1^*, R_1^*)$ with $V_1^* = \{\{1, 2\}\}$ and $R_1^* = \{\{\{(1, 2)\}\}\}$
   - $M_2 = (V_2^*, R_2^*)$ with $V_2^* = \{\{3, 4, 5, 6\}\}$ and $R_2^* = \{\{\{(3, 4), (3, 5), (4, 5), (4, 6), (5, 6)\}\}\}$

2. Partition $P_2 = \{M_1, M_2\}$:

   - $M_1 = (V_1^*, R_1^*)$ with $V_1^* = \{\{1, 2, 3\}\}$ and $R_1^* = \{\{\{(1, 2), (1, 3)\}\}\}$
   - $M_2 = (V_2^*, R_2^*)$ with $V_2^* = \{\{4, 5, 6\}\}$ and $R_2^* = \{\{\{(4, 5), (4, 6), (5, 6)\}\}\}$

Not included by a partition are the edges between the sub-meshes of a partition.

Definition 3.32 (Edge-set cuts)**.** Let $M = (V^*, R^*)$ be a mesh with $n = |V^*|$ and P a partition of M. Let $R_{ij} \in R^*, 1 \leq i, j \leq n$ a relation-set. Let $k \in \mathbb{N}^{n \times n}$ be a matrix, which entries k_{ij} denoting the number of edge-sets in a relation-set

$R_{ij}, 1 \leq i, j \leq n$. Then

$$\begin{aligned} \text{edge-cuts}_{ij,l} \colon \mathbb{M} \times \mathbb{P} &\longrightarrow \mathbb{E} \\ (M, P) &\longmapsto F = \{E^l_{ij} \setminus \biguplus_{1 \leq m \leq |P|} {}^m E^l_{ij}\} \end{aligned}$$

Definition 3.33 (Relation-set cuts)**.** Let $M = (V^*, R^*)$ be a mesh with $n = |V^*|$ and P a partition of M. Let $k \in \mathbb{N}^{n \times n}$ be a matrix, which entries k_{ij} denoting the number of edge-sets in a relation-set $R_{ij}, 1 \leq i, j \leq n$. Then

$$\begin{aligned} \text{relation-cuts}_{ij} \colon \mathbb{M} \times \mathbb{P} &\longrightarrow \mathbb{E} \\ (M, P) &\longmapsto F = \bigcup_{1 \leq m \leq |R_{ij}|} \text{edge-cut}_{ij,m}(M, P) \end{aligned}$$

Definition 3.34 (Cuts)**.** Let $M = (V^*, R^*)$ be a mesh with $n = |V^*|$ and P a partition of M. The *cuts* of a partition are represented by

$$\begin{aligned} \text{cuts} \colon \mathbb{M} \times \mathbb{P} &\longrightarrow \mathbb{E} \\ (M, P) &\longmapsto S = \bigcup_{1 \leq i,j \leq |V^*|} \text{relation-cuts}_{ij}(M, P) \end{aligned}$$

expressing the edges between the sub-meshes $M_1, \ldots, M_k$.

Example 3.35. We calculate the cuts of a partition based on the defined mesh and their partitions of Example 3.31. For P_1 we have $\text{cuts}(P_1) = \{(1, 3), (2, 4)\}$. For P_2 we have $\text{cuts}(P_2) = \{(2, 4), (3, 4), (3, 5)\}$.

Definition 3.36 (Closure)**.** Let $M = (V^*, R^*)$ be a mesh, $n_V = |V^*|$ and $P = \{M_1, \ldots, M_m\}$ a partition of M. Let $1 \leq k, l \leq n_V$ and $1 \leq m \leq |R^i_{kl}|$. The *closure* of a sub-mesh M_i is defined as the set of indices of the neighbouring sub-meshes

$$\text{closure}_{M_i} = \{j \mid 1 \leq j \leq m,\ j \neq i,\ x \in V^i_k,\ y \in V^j_l,\ \exists (x, y) \in E^m_{kl}\} \subseteq \mathbb{N}$$

Definition 3.37 (Node-closure)**.** Let $M = (V^*, R^*)$ be a mesh and $P = \{M_1, \ldots, M_m\}$ a partition of M. The *node-closure* of a sub-mesh $M_i \in P$ is defined as the set of all nodes connected in

$$\begin{aligned} \text{node-closure} \colon P &\longrightarrow \mathbb{V} \\ M_i &\longmapsto W = \Big\{ \bigcup_{v \in V^i_j} \text{neigh}(v) \Big\}_{1 \leq j \leq |V^*_i|} \setminus V^*_i \end{aligned}$$

Example 3.38. We calculate the closure and the node-closure of the partition P_1 based on the defined mesh and their partitions of Example 3.31. The closure for M_1 is $\text{closure}_{M_1} = \{2\}$. The closure for M_2 is $\text{closure}_{M_2} = \{1\}$. The node-closure for M_1 is $\text{node-closure}_{M_1} = \{\{3, 4\}\}$. The node-closure for M_2 is $\text{node-closure}_{M_2} = \{\{1, 2\}\}$.

The initial motivation to create partitions of meshes was, to parallelize the computation. Computing a partitioned mesh on a cluster works different than on a single node. Lets start with a small example:

Example 3.39. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$. V_1 inhabits 6 nodes $v_1, \dots, v_6$ and the connections are define in the edge-set $E_1 1 = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6)\}$ where the bidirectional edges are not listed. The node-type V_1 has a attribute of temperature

$$\begin{aligned} Temp \colon V_1 &\longrightarrow \mathbb{R} \\ v &\longmapsto Temp(v) := \text{temperature of node } v. \end{aligned}$$

which is initialized at time t_0. The partition $P = \{M_1, M_2, M_3\}$ illustrated in Figure 3.8 consist of three sub-meshes, each containing 2 nodes. The kernel for this simulation is kept simple. The temperature attribute of each node n_i will be updated with the mean of the sum of its own temperature and the temperature of the neighbouring nodes.

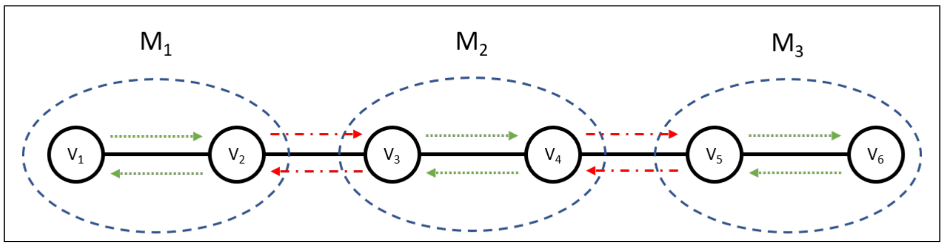

Figure 3.8.: Dotted arrows mark available attributes, dash/dotted arrows indicate necessary data transfer

So to calculate the temperature for time t_1 the node v_2 will need the temperature of node v_3 and v_1 to compute the mean temperature. To compute each node of the sub-mesh M_2 information from $v_2 \in M_1$ and $v_5 \in M_3$ will be needed and the calculations for M_3 can be done with the information of $v_4 \in M_2$ in addition to the information present in the local sub-mesh. As a result the computation of a sub-mesh M_i will need the information of it node-closure(M_i). For the special case, that the computation only need information from the direct neighbours, the nodes contained by the node-closure$_{M_i}$ have the sufficient information.

What can be seen in this example is, that before every computation step of the kernel the computing node needs to gather information from the computing nodes calculating the sub-meshes from the closure. To keep the information from the attributes of the sub-meshes, we also need to adapt the attributes of a single sub-mesh.

Definition 3.40 (Attribute of a sub-mesh)**.** Let $M = (V^*, R^*)$ be a mesh and $P = \{M_1, \dots, M_m\}$ a partition of the mesh. For a sub-mesh $M_i = (V_i^*, R_i^*)$, $1 \leq$

$i \leq m$ with the closure $\text{closure}_{M_i} = C_i$ a attribute with the image of an arbitrary type B is defined as

$$\begin{aligned} A^i \colon V_i^* \ \cup \ (\bigcup_{c \in C_i} V_c^*) &\longrightarrow B \\ v &\longmapsto A^i(v) \end{aligned}$$

${}^nA^i$ refers to a attribute of a sub-mesh M_i at a certain step n.

An other observation from the last example is, that before a computation phase a data transfer is mandatory. The combination of a data transfer and subsequent computation phase is called a *cycle*. To perform a data transfer for a sub-mesh, the computation phase of all neighbouring sub-meshes has to be done.

Concluding to this a adaptation on the kernel has to be made, if it should be able to compute a partition of a mesh on a cluster.

Example 3.41. Lets look on the kernel-function defined in Example 3.29 and find out what we need to change that a computing node can apply the kernel-function to all nodes of a sub-mesh $M_i = \{V_i^*, R_i^*\}$.

Let $C = \text{closure}(M_i)$ be the set of all indices of the closure of M_i. The attributes of the nodes of the meshes indexed by C has to to be gathered from the computing nodes executing the kernel-function to the sub-meshes of the closure C. Let $1 \leq m \leq |C|$. Let $1 \leq n \leq |V_m^*|$. Let $v_o \in V_n^m$. Then the function

$${}^tA^i(v_o) = \text{attribute}({}^tA^m, v_o)$$

is gathering the attributes of all node-types.

Then the same kernel is applied to the nodes od the sub-meshes.

$$NC(v) := \{c \in V_1^i \mid \forall f \in neigh(v) \colon \forall c \in neigh(f) \colon c \neq v\}$$

$${}^{t+1}A^i(v) = {}^tA^i(v) + \sum_{c \in NC(v)} {}^tA^i(c) - {}^tA^i(v)$$

Distributing the computation on a cluster can bring benefits when it comes to computation time. The computation time of one calculation step can be reduced by the number of participating computing nodes. This gives a great opportunity for speed-up. The downside is, that after every computational step each computing node has to gather data from other nodes computing the sub-meshes of its closure. An other problem is, that the transfer of data can only be done, when the processing of the data in the last step is done. Therefore a computing node processing a sub-mesh has to wait for the data transfer until the nodes computing the sub-meshes of closures are done. This brings up the need for synchronization.

So the question arises what is a *good partition* in respect to the given benefits and disadvantages of the distributed computation of a simulation. Broadly speaking a good partition is a partition which minimizes the execution time of the distributed simulation. The execution time of the whole simulation is depending on the time it takes to compute one cycle. All computing nodes have to wait until the last node computing a sub-mesh of the closure has finished its computation, hence the execution time of a cycle is the maximal execution time of all computing nodes. Therefore the goal of a partition is to minimize the maximal execution time of a cycle carried out on a computing node.

The time a computing node takes to compute on cycle is depending on a lot of different factors. The estimation of the time for transferring the data is depending on the amount of data, the partitions of the sub-meshes of the closure are holding, the network topology, on which the different computing nodes are connected, the protocols used to transfer the data and many other. The calculation of computation time of one cycle is depending on the nodes processor, the memory layout, other processes running on the same node, the operating system managing the node, the number of nodes of the sub-mesh to calculate, and many other factors. Accounting to all of these factors would exceed the scope of this master thesis. Hence, assumptions are made to simplify the modeling of the execution time of a cycle.

Definition 3.42 (Computation time factor)**.** All computation time dependent factors are summarized to a single factor indicating the computation time per node of a computational node. This factor is indicated by C_T.

The factor C_T displaying the mean computation time for a single node of a mesh on a computation node.

Definition 3.43 (Data transfer time factor)**.** All data transfer time dependent factors are summarized to a single factor indicating the data transfer time per node of between two computational nodes. This factor is indicated by T_T.

The factor T_T could display the mean data transfer time for a single node of a mesh to an other computation node of a cluster.

Definition 3.44 (Cycle time)**.** Let $M = (V^*, R^*)$ be a mesh and $P = \{M_1, \dots, M_m\}$ a partition of the mesh. The number of all nodes of a sub-mesh $M_i = (V_i^*, R_i^*)$

is given by $n_i = |V_i^*|$. The indexes of the sub-meshes of the closure of a single sub-mesh M_i is given by $I_i = \text{closure}_{M_i}$. The execution and transfer time of a single computing node is given by C_T and T_T. Concluding, the execution time of a cycle can be processing on the sub-mesh M_i can be approximated by

$$\text{cycle-time}_{M_i} = \left(\sum_{j=1}^{n_i} |V_j^i|\right) \cdot C_T + \left(\sum_{j \in I_i} \sum_{k=1}^{|V_j^*|} |V_k^j|\right) \cdot T_T$$

For simulations computing a kernel using only the direct neighbours of a node to compute the next step the cycle-time can be reduced to:

$$\text{cycle-time}_{M_i} = \left(\sum_{j=1}^{n_i} |V_j^i|\right) \cdot C_T + |\text{node-closure}(M_i)| \cdot T_T$$

Concluding a *good partition* is a partition $P = \{M_1, \dots, M_m\}$ which minimizes the maximal cycle time

$$\min(\max_{1 \le j \le m}(\text{cycle-time}_{M_j})).$$

3.4. Quality Metrics for Partitions

To compare different partitions quality metrics are necessary. These quality metrics should give an overview how good a partition is, in respect to the quality of the cuts, the cohesion of the sub-meshes, the balance of the number of contained nodes of the sub-meshes and many more. This will be achieved with a set of statistical analysis tools and other metrics described below.

3.4.1. Node Distribution

A interesting metric is, how the nodes of a mesh M are distributed over the sub-meshes of a partition $P = \{M_1, \dots, M_m\}$. This property is important, since the computing nodes applying a kernel the same number of nodes of a sub-mesh they have the same computational load. Hence their computation time will also be about the same. This helps reducing the maximal execution time of a cycle's computation phase.

Definition 3.45 (Node distribution)**.** Let $M = (V^*, R^*)$ be a mesh and $P = \{M_1, \dots, M_m\}$ a partition. Let $n = (n_1, \dots, n_m) = (|V_1^*|, \dots, |V_m^*|)$ be a m-tuple where each entry n_i expresses the number of nodes of the corresponding sub-mesh M_i. Measuring the property of distribution of the nodes in a partition is done with the*mean* value and the *standard deviation*. The standard deviation σ defined as

$$\sigma = \sqrt{s^2}$$

where variance s^2 is determined by

$$s^2 = \frac{\sum_{1 \le i \le m} (n_i - \overline{n})^2}{|P| - 1}$$

where $\overline{n}$ is defined as the mean of all values n_i.

Example 3.46. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$. Let $P = \{M_1, \ldots, M_5\}$ be the partition of M illustrated in Figure 3.9.

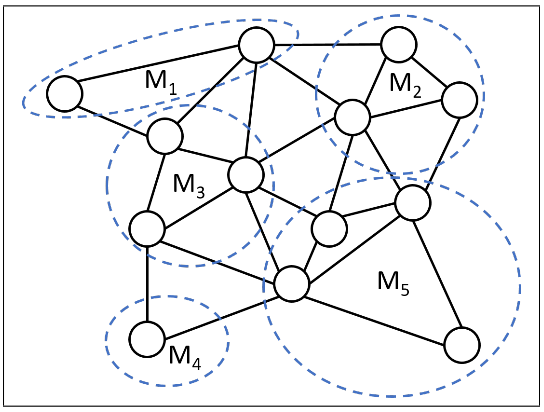

Figure 3.9.: Partition of M with 5 sub-meshes

To calculate the standard deviation of the node distribution first the mean has to be known

$$\overline{n} = \frac{\sum_{v \in V^*} |v|}{|P|} = \frac{2 + 3 + 3 + 1 + 4}{5} = 2.6$$

The next step is to compute the variance

$$s^2 = \frac{\sum_{v \in V^*} (|v| - \overline{n})^2}{|P| - 1} = 1.3$$

Then the standard deviation of the node distribution is

$$\sigma = \sqrt{s^2} = \sqrt{1.3} = 1.14$$

The standard deviation is used as a metric for the node distribution, as it is giving a better overview on the statistical distribution of sub-mesh sizes than just comparing the maximal to the minimal number of nodes.

Example 3.47. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$. Let $P = \{M_1, \ldots, M_{10}\}$ be a partition of M. The sub-meshes M_1 to M_9 inhabit each 5 nodes. The sub-mesh M_{10} has only one node.

Comparing the maximal to the minimal value would give as a result the value 4.

The mean and the standard deviation provide as a result

$$\overline{n} = 4.6$$
$$\sigma = 1.26$$

which illustrates a more accurate picture of the present sub-mesh size distribution.

3.4.2. Node-closure Distribution

An other important metric is the distribution of nodes of the node-closure. Many kernel functions only depend in the attribute of their direct neighbours. So the necessary amount of data to send during the transfer phase is proportional to the number of nodes in the node-closure.

Definition 3.48 (Node-closure distribution)**.** Let $M = (V^*, R^*)$ be a mesh and $P = \{M_1, \ldots, M_m\}$ a partition. Let $c = (c_1, \ldots, c_m)$ be a m-tuple where each entry c_i expresses the number of nodes in the node-closure of the corresponding sub-mesh M_i. Measuring the distribution of the nodes in the node-closure of the sub-meshes will be performed by the *mean* value and the *standard deviation* as defined in Definition 3.45.

Example 3.49. Let M be a mesh and $P = \{M_1, \ldots, M_5\}$ a partition of M from Example 3.46. To calculate the standard deviation of the closure distribution first the mean has to be known

$$\overline{e} = \frac{\sum_{i=1}^{|P|} |\text{node-closure}(M_i)|}{|P|} = \frac{5+6+8+2+7}{5} = 5.6$$

The next step is to compute the variance

$$s^2 = \frac{\sum_{i=1}^{|P|} (|\text{node-closure}(M_i)| - \overline{e})^2}{|P|-1} = 5.3$$

Then the standard deviation of the closure distribution is

$$\sigma = \sqrt{s^2} = \sqrt{5.3} = 2.3$$

3.4.3. Coverage

Definition 3.50 (Coverage)**.** Let $M = (V^*, R^*)$ be a mesh and $P = \{M_1, \dots, M_m\}$ a partition. The *Coverage* [3] value of a partition P is the fraction of the number of edges in the sub-meshes of a partition and the total number of all edges.

$$\text{coverage}(P) = \frac{\sum_{i=1}^{|P|} \sum_{1 \le j,k \le |V_i^*|} \sum_{l=1}^{|R_{jk}|} |{}^{i}E_{jk}^{l}|}{\sum_{1 \le i,j \le |V^*|} \sum_{k=1}^{|R_{ij}|} |E_{ij}^{k}|}$$

Intuitively a high value of the coverage(P) is indicating a better quality of a partition P. A higher value indicates, that lesser edges are in the cuts(M,P) and more edges are inhabited inside the sub-meshes. But this metric is insensitive to the distribution of nodes in the sub-meshes of a partition. I.e. this metric gives the highest values when a partition consist of 2 sub-meshes, where one sub-mesh inhabits only one node with one edge to the neighbouring cluster. Hence when using this metric the node-distribution and deviation has to be taken into account to get a complete overview.

Example 3.51. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$. Two different partitions P_1 and P_2 are created, illustrated in Figure 3.10 and Figure 3.11.

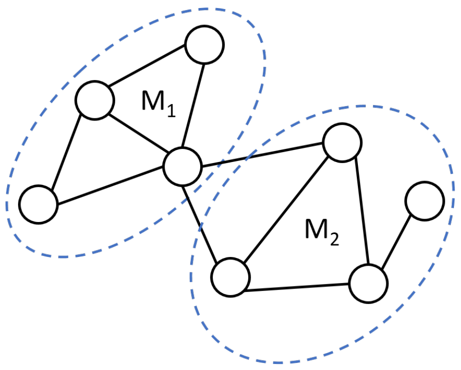

Figure 3.10.: Partition P_1

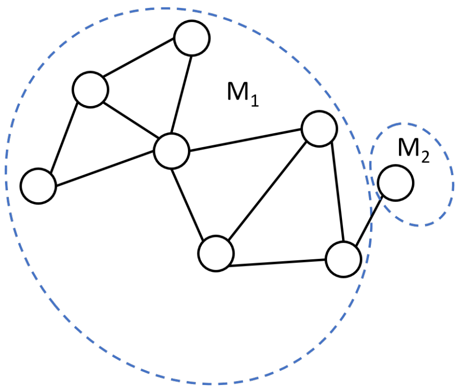

Figure 3.11.: Partition P_2

The Coverage value of P_1 is

$$\text{Coverage}(P_1) = \frac{5+4}{11} = 0.818.$$

The coverage value of P_2 is

$$\text{Coverage}(P_1) = \frac{10+0}{11} = 0.909.$$

3.4.4. Summary of Quality Metrics for Partitions

In Section 3.4 4 different quality metrics are presented. The presented metrics provide an overview to important characteristics of a partition.

Metric	Objective	Value range
Node distr.	Distr. of the nodes in the sub-meshes	
Node-closure distr.	Distr. of the nodes in the node-closures	
Coverage value	# of edges compared to total # of edges	[0,1]

3.5. Locality of Reference

Locality of reference [5] is stating that data references recently used or adjacent storage locations to the recently used data, are likely accessed in future steps. These two principals are also called:

- Temporal locality
- Spatial locality

Temporal locality states that references to a memory location is grouped by time. This means that there is a high probability that data used at a specific point in time is used in the near future again. Therefore the data recently used should be kept in the cache memory, as there is a high likelihood that it will be used again.

Spatial locality references the property of programs that memory locations located close to recently accessed locations are more likely to be referenced in the near future than other memory locations. Hence it would be beneficial to pre-fetch certain memory references, as it will be likely that these will be used in the near future.

Caches typically exploit these program characteristics. To gain maximum advantage out of the hardware exploitation, the data of the mesh and its attributes has to be organized in a fashion, that data used in the near future is close to data used currently by the kernel. To achieve this, we will reorder the mesh and its attributes according to their sub-mesh index, as it is assumed, that nodes with the same sub-mesh index lie close to each other.

3.6. Quality Metrics for Data Organization

Measuring the quality of a data organization is complicated. The *optimal* data layout of a mesh and its attributes is depending on the kernel of the simulation. The kernel decides which elements are used during the computation. So one data organization could work perfectly fine with a kernel, but not with an other kernel accessing different elements. Hence, it would undermine the credibility using only metrics just focusing on the data organization itself.

An other problem comes up, as it is hard to model the caches [1]. Their internal structures, the different techniques used in caches e.g. pre-fetching and their hierarchical structure make it nearly impossible to model them accurately.

An expressive metric, however, is to actually measuring the execution time of a simulation. This gives a good comparison of the different organization methods of the data structure. Hence this is the method chosen to measure and evaluate the organization of mesh data in memory.

4. Mesh Partitioning

This chapter is denoted to the process of creating a partition of a mesh. The first section explains the architecture and the intermediate steps of the work-flow forming the framework for all partitioning methods investigated by this thesis. Section 4.2 covers the methods used to create partitions. In this section further explanations of the used methods are provided.

4.1. Overview / Architecture

This section is all about the process of creating a partition of a mesh. First we introduce a work-flow forming a framework for a class of partitioning methods. Subsequently we give further information to all intermediate structures needed to establish the work-flow and give examples for a better understanding of each intermediate step. Section 4.2 is investigating on methods used to partition the meshes.

4.1.1. Work-flow Overview

As illustrated, Figure 4.1 outlines the overall framework for the class of methods to be investigated. The first step is a input step. In the input step meshes with different structures and node-types can be assigned. The next step it to partition the selected node-types of the mesh. The result will be a new attribute for a selected node-types mapping each node of this type to the index of the sub-mesh it should belong to - according to the selected partitioning method. In the next step the remaining node-types and their elements have to get an index of a sub-mesh. This is happening in a subsequent propagation step. Following to that, the mesh will be reordered according to the given position of its assigned sub-mesh.

4.1.2. Intermediate Steps

Lets take a closer look on the work-flow and its intermediate steps. This section is devoted to define all the essential steps to create a partition of a mesh and

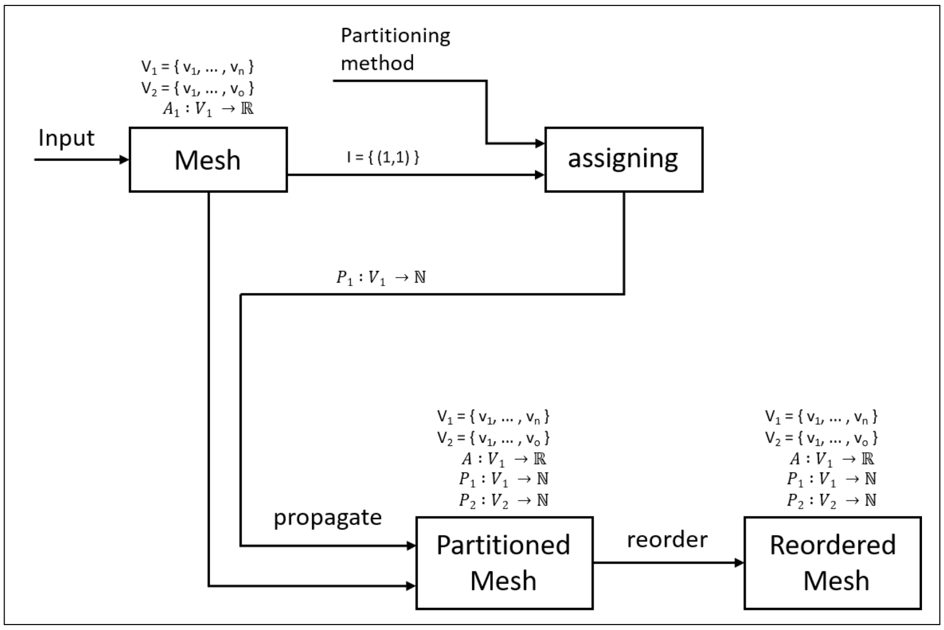

Figure 4.1.: The workflow used to partition the meshes

modify it according to its properties.

Loading a file and filling a mesh structure covered in Definition 3.1 is the first step of the workflow.

4.1.3. Assigning

The next step is to assign the selected nodes of the mesh a number. The assigned number i to a certain node corresponds to the index of the sub-mesh M_i of the partition to be created. This process is called *assigning*. The assigned index of the sub-mesh is called *partition-number*. Therefore a function is established taking a mesh and a set of indices and a partition method. This function returns a set of attributes with the domain of the selected node-sets of the mesh and the image stating the partition-number.

Definition 4.1. Let $M = (V^*, R^*)$ be a mesh and A correspond to the set of all attributes assigned to the node-types of M. Let

$$I \subseteq \{(i, j) \mid 1 \leq i \leq |V^*| \land 1 \leq j \leq |A|\}$$

a set of tuples of selected indices of node-types, where the first element of the tuple corresponds to node-types which are intended to get a partition-number assigned and the second element of the tuple the corresponding attribute index. Let $P = \{P_i\}_{(i,j) \in I}$ be an attribute-set. The following algorithm illustrates, how the selected node-types get a partition-number assigned.

```
function Assigning($M, A, I, P, f$)

    for all $(i, j) \in I$ do
        $a \leftarrow$ attribute$(A, j)$
        $p \leftarrow$ attribute$(P, i)$
        for all $v_k \in V_i$ do $p(v_k) = f(v_k, a_j))$
        end for
    end for
end function
```

The result from the procedure `Assigning` is then stored the resulting attribute-set P.

Example 4.2. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type $V_1 = \{v_1, \ldots, v_9\}$ and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}\}$. Let $1 \leq i \leq |V_1|$. The mesh has a 2 dimensional spatial attribute

$$\begin{aligned} a\colon V_1 &\to \mathbb{R}^2 \\ v_i &\longmapsto (i\%3, i/3) \end{aligned}$$

The function

```
function $f(v, a)$
    $(x, y) \leftarrow a(v)$
    $result \leftarrow$ floor$(y) * 3 +$ floor$(x)$
    return $result$
end function
```

take a node and returns the corresponding partition-number. The result for some M and a is shown in Figure 4.2.

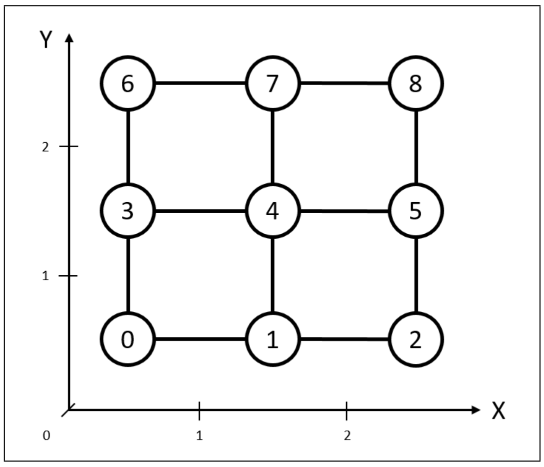

Figure 4.2.: Mesh with partition-numbers

4.1.4. Propagation

So far only selected node-types have a partition number assigned, however, all nodes from the remaining node-sets need one too. Hence a mechanism is needed, which distributes the partition-numbers from the nodes of the selected node-types to the remaining nodes. This process is called *propagation* of partition-numbers.

Definition 4.3 (Propagation Algorithm)**.** Let $M = (V^*, R^*)$ be a mesh and A correspond to the set of all attributes assigned to the node-types of M. Let $I \subseteq \{(i, j) \mid 1 \leq i \leq |V^*| \wedge 1 \leq j \leq |A|\}$ a set of tuples of indices, where the first element of the tuple corresponds to node-types which are intended to get a partition-number assigned and the second element of the tuple to the spatial attribute to be considered. Let $P = \{P_1, \dots, P_{|V^*|}\}$ be a set of attributes devoted to map the nodes of the different node-types to a partition-number. For $(i, j) \in I$ the attributes P_i have already been assigned with partition-numbers. Let $v_m \in \text{nodeset}_m(V^*)$. Let $P_m = \text{attribute}(P, m)$. Let $D^i(v_m) = \{d \in \text{nodeset}_k(\text{neigh}(v_m)) | 1 \leq k \leq |\text{neigh}(v)| \wedge {}^iP_k(d) \text{ is defined}\}$. Then

$$
{}^{i+1}P_m(v_m) = \begin{cases} {}^iP_m(v_m) & \text{if } {}^iP_m(v_m) \text{ is defined} \\ \min_{d \in D^i(v_m)}({}^iP_m(d)) & \text{where } D^i(v_m) \neq \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}
$$

propagates the partition number iteratively until all nodes are assigned. This will create sub-meshes with a strong connectivity, as nodes share their partition-number with other nodes directly connected to them.

Example 4.4. Let $M = (V^*, R^*)$ be a mesh illustrated in Figure 4.3. The node-types indexed in set $I = \{1\}$ are partitioned according to some partition method. The Figures 4.3, 4.4, 4.5 and 4.6 illustrate how the partition-numbers are propagated over the whole mesh.

4.1.5. Reordering

The last step to the work-flow is assigned to the organization of the data in the mesh. With the reorganization of the mesh the access to the elements should be accelerated by exploiting the capabilities of the CPUs caches. The data should be organized in a way, that the principles of locality are fulfilled. Already known is, that executing the kernels for a specific node is likely to involve the data associated to neighbouring nodes connected through an edge. The methods creating the partition yield partition-numbers that should give the same or similar partition-numbers to nodes spatially located close to each other. Partition-numbers with a bigger difference refer to nodes which are not

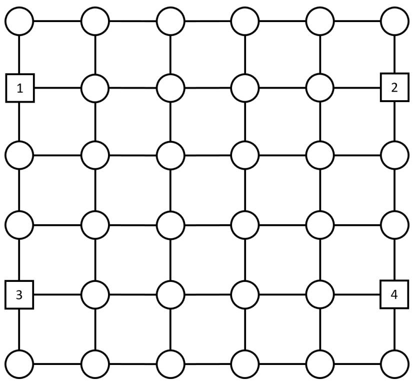

Figure 4.3.: Initialized mesh with partition-numbers

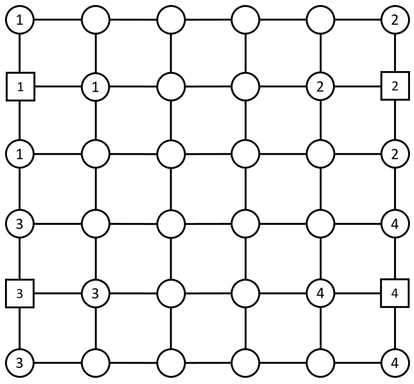

Figure 4.4.: Propagation after 1 propagation-step

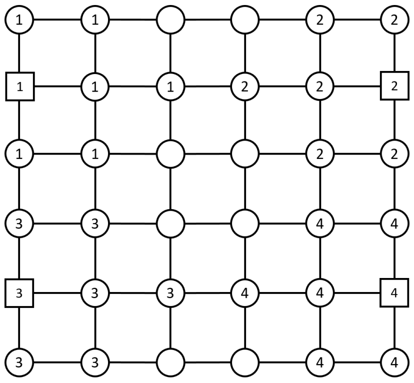

Figure 4.5.: Propagation after 2 propagation-steps

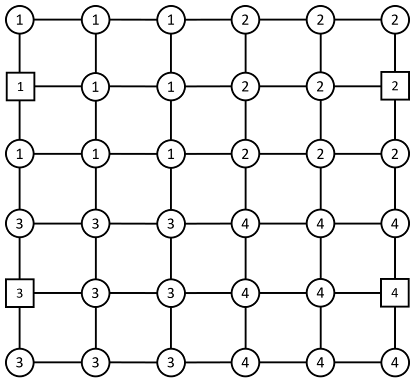

Figure 4.6.: Propagation after 3 propagation-steps

likely to be related. Sorting the nodes according to their partition-number should therefore increase the locality.

At this point the partition-numbers of all nodes of the mesh are known. The last step of the work-flow is the *reordering* of the mesh. The reordering is a realized by all nodes according to their partition-number.

It is assumed, that the executing program is mapping the memory locations according to the node-numbers in ascending order. Hence the reordering of the nodes and their attributes is processed by a renaming step.

Definition 4.5. To reorder a mesh $M = (V^*, R^*)$ for each node-set V_i of a mesh a list of pairs P_i is created, where the the i-th position of the list the first entry of the pair is the number of the node in the node-set and the second entry is the nodes partition-number. Let $P = \bigcup_{1 \le i \le |V^*|} P_i$. These lists will be sorted according to the second entry. The next step is to create a new mesh

$N = (W^*, S^*)$ with a set of node-sets $W^* = \{W_1, \dots, W_n\}$ where $n = |V_i|$. The set of relations need to be transformed according to the results of P_i.

```
procedure RenamingRelations($M, N, P$)
    for $1 \leq i, j \leq n$ do
        for $1 \leq k \leq |\text{relationset}_{ij}(M)|$ do
            edgeset-old ← edgeset$_{ij,k}(M)$
            edgeset-new ← edgeset$_{ij,k}(N)$
            for all $(v_1, v_2) \in$ edgeset-old do
                new-pos$_1$ ← position of $v_1$ in $P_i$
                new-pos$_2$ ← position of $v_2$ in $P_j$
                edgeset-new ← (edgeset-old$_{\text{new-pos}_1}$, edgeset-old$_{\text{new-pos}_2}$)
            end for
        end for
    end for
end procedure
```

After renaming the nodes and the corresponding relations the attributes have to be changed. To accomplish this an algorithm similar to the one presented to reorder the relations is introduced. Let A be the attribute-set of mesh M and B the attribute-set of N.

```
procedure ReorderAttributes($M, A, B, P$)
    $n \leftarrow |A|$
    for $1 \leq i \leq n$ do
        attributeset-old ← attributeset$_i(A)$
        attributeset-new ← attributeset$_i(B)$
        $V$ ← node-set according to attributeset$_i(A)$
        for all $v \in V$ do
            new-pos ← position of $v$ in $P_i$
            attributeset-new($v_{\text{new-pos}}$) = attributeset-old($v$)
        end for
    end for
end procedure
```

After this step the renamed version of the mesh M is in N.

Example 4.6. Let $M = (V^*, R^*) = (\{V_1\}, \{R_{11}\})$ be a mesh with one node-type and one edge-type E_{11} in the corresponding relation-type $R_{11} = \{E_{11}^1\}$. The node-set consist of 3 nodes. The edge-set is defined as $E_{11} = \{(v_1, v_2), (v_2, v_3)\}$. The node-set V_1 has a attribute a with an image in $\mathbb{N}$ and a attribute $p\colon V_1 \to \mathbb{N}$ representing the partition number of a node. The nodes of V_1 are mapped to the following values in attribute a: $a(v_1) = 5$, $a(v_2) = 6$, $a(v_3) = 7$. The nodes of V_1 are mapped to the following values in attribute p: $p(v_1) = 2$, $p(v_2) = 3$, $p(v_3) = 1$. The Figure 4.7 shows the mesh, the structure and the partition-numbers of the nodes.

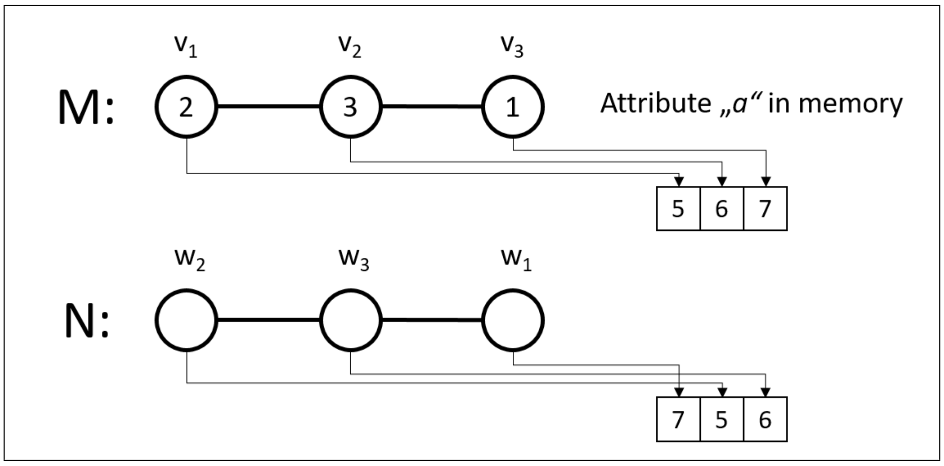

Figure 4.7.: Mesh M and N

The mesh $N = (W^*, S^*)$ is the reordered mesh M according to the partition-numbers given. $W^* = \{W_1\}$ is a set of node-sets with one node-set, which has 3 elements. $S^* = \{S_{11}\}$, $S_{11} = \{E^1_{11}\}$ with $E^1_{11} = \{(w_1, w_3,), (w_2, w_3)\}$. The nodes of W_1 are assigned to the following values: $a(w_1) = 7$, $a(w_2) = 5$, $a(w_3) = 6$.

4.2. Partitioning Methods

This section is devoted to the different methods used to create a partition of a mesh. The first methods to be covered are the space-filling curves in Section 4.2.1. These curves enable a fast assignment of a position in a n-dimensional space to a number.

In Section 4.2.2 the state-of-the-art partition method is presented. This method is called *k-way partitioning*. The state-of-the-art k-way partitioning technique is utilized in Section 4.2.2 to establish a reference line for the partitioning schemes evaluated in this thesis.

4.2.1. Space-Filling Curves

Space-filling curves are continuous curves in a n-dimensional space. The theoretical fundamentals have been described by George Cantor in 1878. In 1887 the mathematician Camille Jordan gave a precise description of space-filling curves in $\mathcal{I}^2$:

A curve (with endpoints) is a continuous function whose domain is the unit interval $\mathcal{I} = [0, 1]$.

So the research was focused on searching for functions describing curves in the form of $c\colon \mathcal{I} \to \mathcal{I}^2$. In 1879 a proof was found, that those curves can't be bijective. So the research focused on finding curves, which provide a continuous surjective mapping from $\mathcal{I}$ to $\mathcal{I}^2$.

In the year 1890 the mathematician Giuseppe Peano was the first one constructing such a curve. Since then the space-filling curves are sometimes also called the *Peano-curves*. This name also refers to the curve created by Peano, which will be discussed in Section 4.2.1. Since then a whole variety of curves have been presented. But not all of them are sufficient for the purpose of partitioning. We will present curves with a high potential to create a good partition and give an order for a satisfiable data organization. Curves which have already proven to deliver poor results e.g. the Sierpinski-curve are not taken into account.

Generating Space-filling curves is an iterative process. Each curve has a base-form, in which the curve should fit the assigned space. With every iterative step the space is divided according to the definition of the curve and the curve itself get refined, so the curve fit into the newly divided space.

Hilbert 2D curve

The *Hilbert 2D* curve is a steady curve in a 2 dimensional space. The structure was invented by David Hilbert [10] in 1891. He was also the first mathematician who came up with the idea, that those curves are best understood when they are graphically illustrated.

Definition 4.7. The *Hilbert 2D* curve is defined by an algorithm [4] calculating the pre-image of the function describing the space-filling curve:

```
**procedure** ROTATE($n, a, b, x, y$)
    **if** $x = 0$ **then**
        **if** $y = 0$ **then**
            $a \leftarrow n - a - 1$
            $b \leftarrow n - b - 1$
        **end if**
        Swap($a, b$)
    **end if**
**end procedure**

**function** CALCULATECURVEPOSISTION($n, x, y$)
    result $\leftarrow 0$
    $a \leftarrow n/2$
    **for** $a > 0$ **do**
        $u \leftarrow (x \,\&\, a) > 0$
        $v \leftarrow (y \,\&\, a) > 0$
```

```
            result ← result + $a^2 \cdot ((3u) \oplus v)$          ▷ $\oplus$ stand for XOR
            ROTATE($a, x, y, u, v$)
            $a \leftarrow a/2$
        end for
        return result
    end function
```

To get a position on the Hilbert 2D curve the procedure `CalculateCurvePosistion` is called. The call has the parameters n indicating $log_2 n$ recursive steps, x, y indication the position in a 2D space.

In Figure 4.8 the base case of the Hilbert 2D curve is illustrated. The first iterative step is shown in Figure 4.9 and Figure 4.10. Figure 4.11 and Figure 4.12 illustrate the construction of the second iteration. The third iteration is displayed in Figure 4.13.

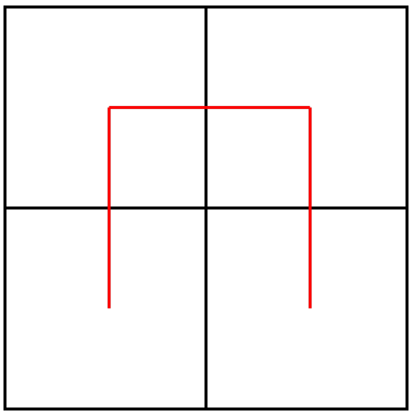
Figure 4.8.: Base case of Hilbert 2D curve

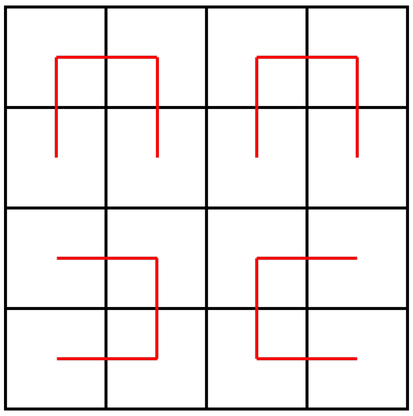
Figure 4.9.: First iteration shown with base cases

Hilbert-3D-curve

The *Hilbert 3D* curve is space-filling curve inspired by the classic Hilbert 2D curve. According to Haverkort [9] there are 10694807 different possibilities to construct a curve based on the properties of the Hilbert 2D-curve. An other paper [2] states that there are only 1536 unique structurally different solutions. The base pattern of the Hilbert 2D curve is preserved in the Hilbert 3D curve. The base case of the Hilbert 3D curve is illustrated in Figure 4.14. The second iteration is shown in Figure 4.15.

Definition 4.8. A number given in the fashion of $0_8 t_1 t_2 t_3 \ldots$ states a number

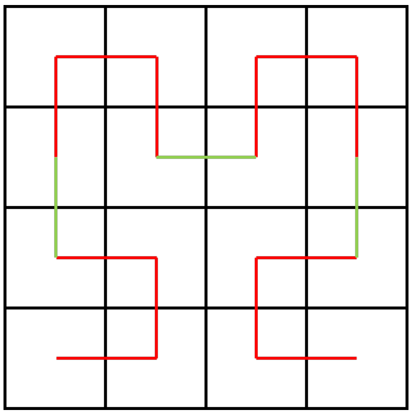
Figure 4.10.: First iteration of the curve connected

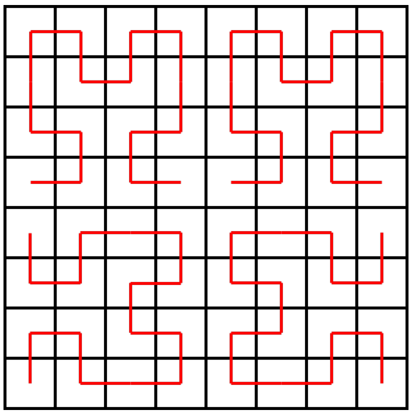
Figure 4.11.: Second iteration of the Hilbert 2D curve

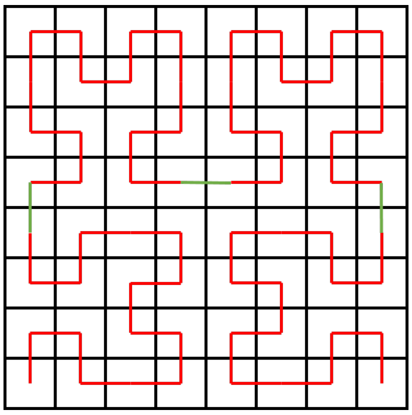
Figure 4.12.: Second iteration connected

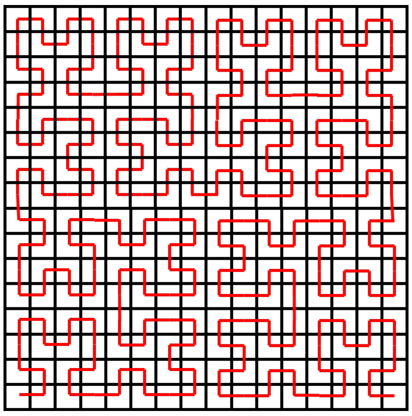
Figure 4.13.: Shape of Hilbert 2D after third iteration

between 0 and 1 in octal representation. The function $f_{\text{hilbert3D}}: \mathcal{I} \longrightarrow \mathcal{I}^3$ with

$$f_{\text{hilbert3D}}(0_8 t_1 t_2 t_3 \dots) = H_{t_1} \circ H_{t_2} \circ H_{t_3} \circ \cdots \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

with operators H_i defined as

$$H_0\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}x+0\\\frac{1}{2}z+0\\\frac{1}{2}y+0\end{pmatrix}\qquad H_1\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}z+0\\\frac{1}{2}y+\frac{1}{2}\\\frac{1}{2}x+0\end{pmatrix}$$

$$H_2\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}x+\frac{1}{2}\\\frac{1}{2}y+\frac{1}{2}\\\frac{1}{2}z+0\end{pmatrix}\qquad H_3\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}z+0\\-\frac{1}{2}x+\frac{1}{2}\\-\frac{1}{2}y+\frac{1}{2}\end{pmatrix}$$

$$H_4\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-\frac{1}{2}z+1\\-\frac{1}{2}x+\frac{1}{2}\\\frac{1}{2}y+\frac{1}{2}\end{pmatrix}\qquad H_5\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}x+\frac{1}{2}\\\frac{1}{2}y+\frac{1}{2}\\\frac{1}{2}z+\frac{1}{2}\end{pmatrix}$$

$$H_6\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-\frac{1}{2}z+\frac{1}{2}\\\frac{1}{2}y+\frac{1}{2}\\-\frac{1}{2}x+1\end{pmatrix}\qquad H_7\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}x+0\\-\frac{1}{2}z+\frac{1}{2}\\-\frac{1}{2}y+1\end{pmatrix}$$

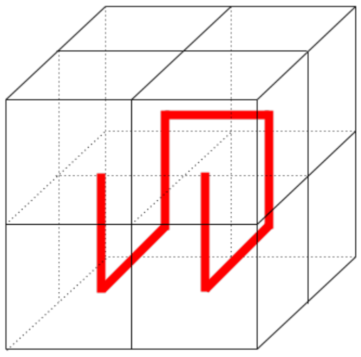
Figure 4.14.: Base case of the Hilbert 3D curve

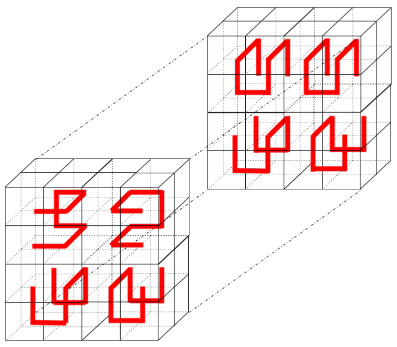
Figure 4.15.: Hilbert 3D curve with one iteration

Z-curve

The *Z-curve*, also called Lebesgue curve, was introduced by Guy Macdonald Morton in 1966. The calculation of a point on the Z-curve is straight forward. The number of a point is calculated by the bitwise interleaving the binary coordinates. This is shown in Figure 4.16 and Figure 4.18.

Definition 4.9. The n-dimensional *Z-curve* is a space-filling curve defined as a function [15] $f_{\text{Z-curve}}\colon \mathcal{I} \to \mathcal{I}^n$. Calculating the pre-image of a n-dimensional coordinate is defined by following algorithm:

```
function Z-CURVE($n, x[m]$)
    result $= 0$
    for $1 \leq i \leq n$ do
        $y[m] \leftarrow (0, \dots)$
        for $1 \leq j \leq m$ do
            if $x[j] > 0.5$ then
                $y[j] \leftarrow 1$
            end if
            result $\leftarrow$ SHIFTLEFT($x, 1$)
            result $=$ result & $y[j]$
        end for
        for $1 \leq j \leq m$ do
            if $y[j] = 1$ then
                $x[j] \leftarrow (x[j] - 0.5) \cdot 2$
            else
                $x[j] \leftarrow x[j] \cdot 2$
            end if
        end for
    end for
    return result
end function
```

The parameters for the call of `Z-curve` are the number of iterative steps n and the position $x \in \mathcal{I}^m$.

The base case of the Z-curve is illustrated in Figure 4.16 Figure 4.18 shows the first, Figure4.18 the second iteration step. Figure 4.16 and Figure 4.18 also illustrate how the result of the function `Z-curve` is bitwise created. The numbers in these Figures are given in binary representation.

Peano 2D curve

The *Peano 2D* curve was the first space-filling curve invented by Giuseppe Peano in the year 1890. Different to the other presented structures, this curve splits per iteration the space into 9 more fine grain pieces.

Definition 4.10. A number given in the fashion of $0_3 t_1 t_2 t_3 \dots$ states a number in ternary representation. The function $f_{\text{Peano2D}} \colon \mathcal{I} \longrightarrow \mathcal{I}^2$ with

$$f_{\text{Peano2D}}(0_3 t_1 t_2 t_3 \dots) = P^x_{t_1} \circ P^y_{t_2} \circ P^x_{t_3} \circ P^y_{t_4} \circ \cdots \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

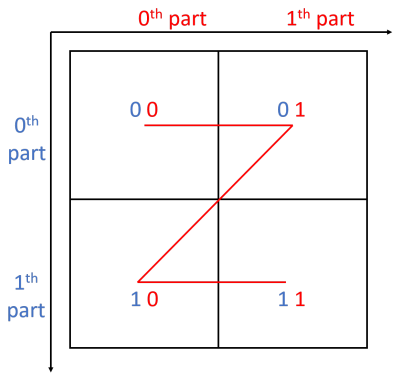

Figure 4.16.: Base case of the Z-curve

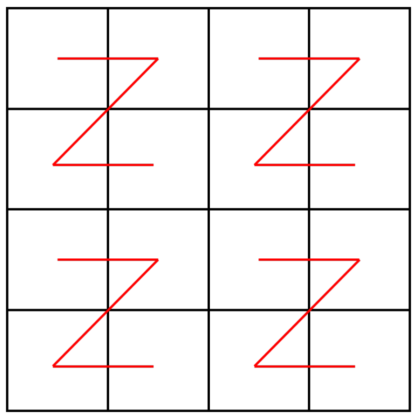

Figure 4.17.: Z-curve with one iteration

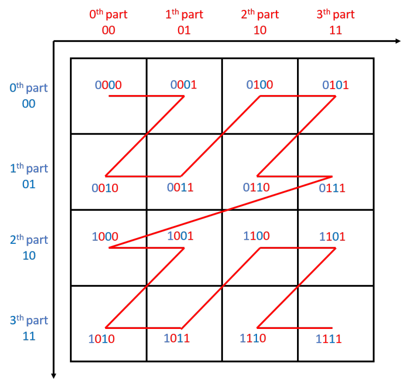

Figure 4.18.: Bitwise interleaving after 2 steps

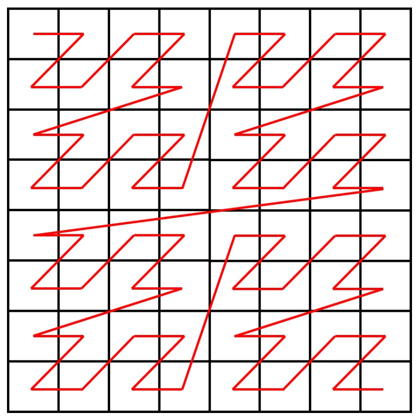

Figure 4.19.: Z-curve after two iterations

with operators $P^x_{t_{2i+1}}$ and $P^y_{t_{2i}}$ defined as

$$P^x_0 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x+0 \\ \frac{1}{3}y+0 \end{pmatrix} \qquad P^x_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x+1 \\ \frac{1}{3}y+\frac{1}{3} \end{pmatrix}$$

$$P^x_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x+0 \\ \frac{1}{3}y+\frac{2}{3} \end{pmatrix} \qquad P^y_0 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x+0 \\ y+0 \end{pmatrix}$$

$$P^y_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x+\frac{1}{3} \\ -y+1 \end{pmatrix} \qquad P^y_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x+\frac{2}{3} \\ y+0 \end{pmatrix}$$

For a better understanding of the Peano curve the Figures 4.20, 4.21 and 4.22 show an illustration of the first 2 iterations.

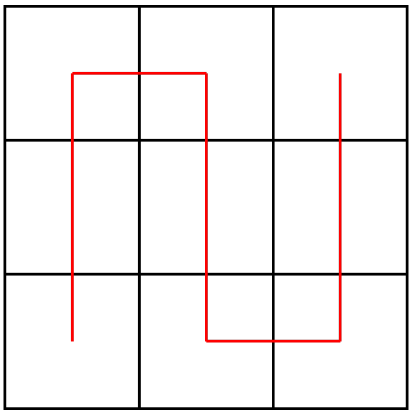
Figure 4.20.: Base case of the Peano curve

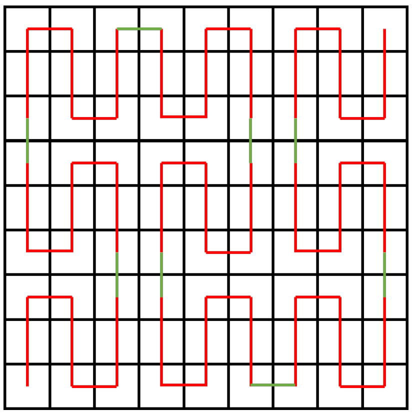
Figure 4.21.: Peano curve with one iteration

Peano 3D curve

The *Peano 3D* curve is inspired by the Peano 2D curve. The structure of the original Peano curve is preserved and reused. The unit cube $\mathbb{I}^3$ if divided in 27 equal parts with 3 planes each consisting of 9 parts. Each plane is constructed by a Peano curve and all three are connected to each other to preserve the property of a continuous curve.

Definition 4.11. The *Peano 3D curve* is a function $f_{Peano3D} : \mathcal{I} \to \mathcal{I}^3$ describing a space-filling curve. Calculating the pre-image is defined as a an iterative algorithm:

```
function GETPOSITION($x$, length)
    if $x < \frac{\text{length}}{3}$ then
        return 0
    else if $x > 2 \cdot \frac{length}{3}$ then
        return 2
    end if
    return 1
end function

function PEANO-2D(($x, y$), mirrored, direction, length)
    field ← 0
    x-pos ← GETPOSITION($x$, 1)
    y-pos ← GETPOSITION($y$, 1)
    field ← x-pos · 3
```

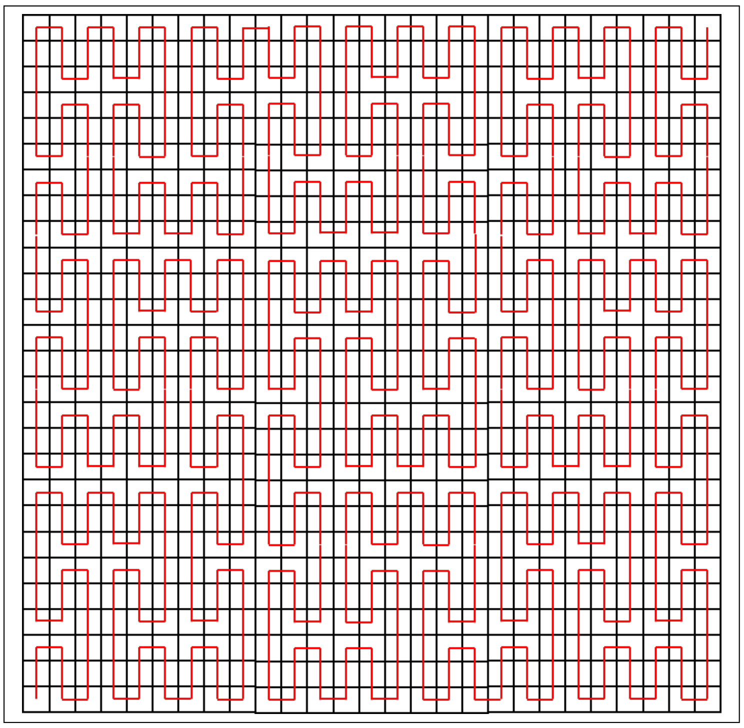

Figure 4.22.: Peano 2D curve after two iterations

```
    y-num ← 0
    if x-pos! = 1 then
        y-num ← y-pos
    else
        y-num ← 2 − y-pos
    end if
    if mirrored = 1 then
        y-num ← 2 − y-num
    end if
    field ← field + y-num
    if direction = 1 then field ← 8 − field
    end if
    return field
end function

procedure Peano-3D(($x, y, z$), mirrored, direction, length, result)
    x-pos ← getPosition($x$, length)
    y-pos ← getPosition($y$, length)
    z-pos ← getPosition($z$, length)
    direction-2D ← direction
    if ()mirrored $\oplus$ (z-pos mod 2) = 1) then
        direction-2D ← 1 − direction-2D
    end if
    x-pos-2D $\leftarrow \frac{3 \cdot x}{\text{length}}$
    y-pos-2D $\leftarrow \frac{3 \cdot y}{\text{length}}$
    xy-num $\leftarrow$Peano-2D((x-pos-2D, y-pos-2D), mirrored, direction-2D)
    z-num $\leftarrow 0$
    if mirrored $= 0$ then
        z-num $\leftarrow 9 \cdot$ z-pos
    else
        z-num $\leftarrow 9 \cdot (2 -$ z-pos$)$
    end if
    result $\leftarrow$ (xy-num + z-num) $\cdot 3 \cdot$ lenght
    if length $> 0$ then
        $a \leftarrow x \cdot \frac{\text{x-pos} \cdot \text{length}}{3}$
        $b \leftarrow y \cdot \frac{\text{y-pos} \cdot \text{length}}{3}$
        $c \leftarrow z \cdot \frac{\text{z-pos} \cdot \text{length}}{3}$
        mirrored$_{new} \leftarrow$ mirrored $\oplus$ (xy-num mod 2)
        direction$_{new} \leftarrow$ direction $\oplus$ (x-pos mod 2) $\oplus$ ()y-pos mod 2)
        Peano-3D($(a, b, c)$, mirrored$_{new}$, direction$_{new}$, length/3, result)
    end if
end procedure
```

The procedure `Peano-3D` is called with the parameters (x, y, z), mirrored, direction, length, result. (x, y, z) is the position to map on the Peano 3D curve. The parameters mirrored and direction state if the Peano 3D curve is either mirrored or traversed in the other direction. Typically these two parameters are initially kept 0. The last parameter length denotes the length of the cube, which the Peano 3D curve fills.

The Peano 3D curve is illustrated in Figure 4.23

4.2.2. Multi-Level / k-way Partitioning

The *multi-level* partitioning is the state-of-the-art approach to create partitions of meshes. These methods are fast and deliver good results for a partitioning. These multi-level methods are applying different phases illustrated in Figure 4.24. Nodes and edges of the mesh are pooled together. Typically the first phase is a coarsening stage. There are different methods to coarsen a mesh e.g. by spatial attributes or neighbourhood relationships. Finishing with a reasonable amount of nodes m the next phase starts. In this phase the reduced mesh is partitioned with an expansive method, which deliver very good results. In this phase the mesh is partitioned into k different meshes. After the partition for the coarsened mesh is created, the partition-numbers are propagated back to the nodes of the refined mesh. If the final amount of the nodes in the

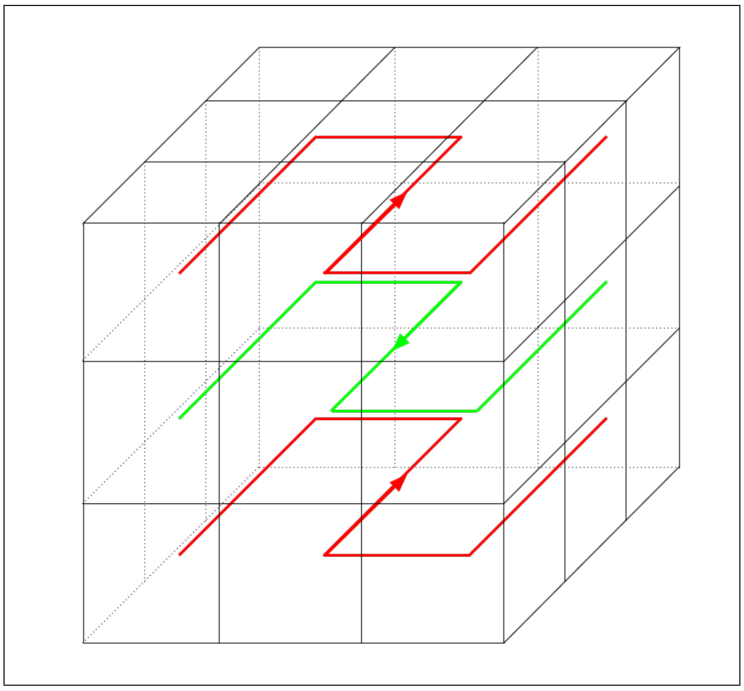

Figure 4.23.: The Peono-3D-curve base case

generated sub-meshes is not well distributed, a final phase can be introduced, which is responsible for rebalancing the sub-meshes.

The complexity is of this methods is typically $\mathcal{O}(n + k\log(k))$ where n denotes to the number of nodes and k is the number of sub-meshes of the partition. Depending on the methods to coarsen the mesh and to partition the reduced mesh the complexity can vary. The memory complexity is typically $\mathcal{O}(n + m^2)$ where m states the amount of nodes after the coarsening. During the coarsening phase the algorithm need to remember its root nodes, to propagate the computed partition-numbers to them. The partitioning methods used in the second phase typically have $\mathcal{O}(m^2)$ memory complexity [13].

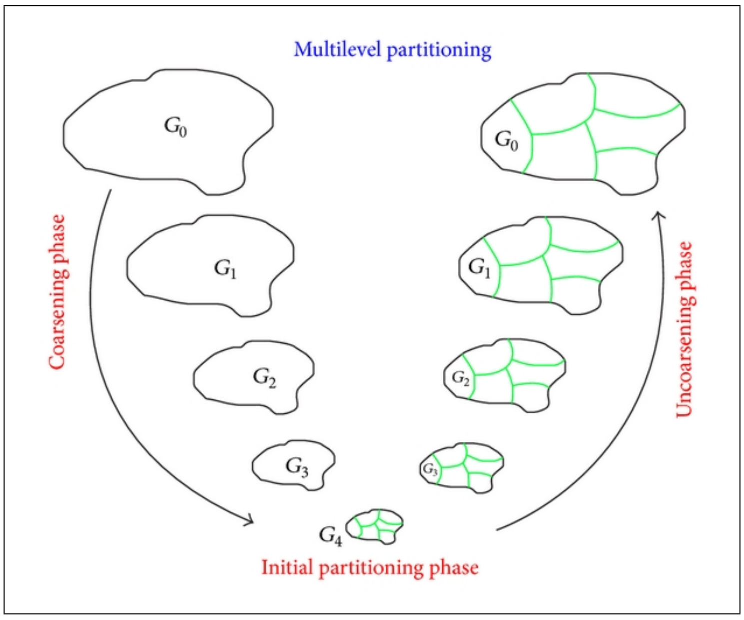

Figure 4.24.: The phases of a multi-level partitioning illustrated [20]

5. Experimental Section

In this chapter all experimental results and their evaluations are presented. The first part of this chapter will elucidate the setup of the experiments. The used mesh structures are specified as well as the used methods to create the partitions. The system on which the performance tests will be executed is listed in the Appendix.

5.1. Experiment Setup

5.1.1. Meshes

The first mesh type will be a simple *grid structure* or structured mesh as defined in Definition 3.16 with one node-type. This regular structure is widely used for simulations with less details in the model. It describes the uniformly distributed spatial discretization of a real-world model. For the experiments we use grids with 40, 400 Million and 1 Billion nodes. The 2 dimensional versions of the meshes are forming a square. The 3 dimensional versions will form a cube.

The next mesh type is the unstructured mesh with close neighbourhood defined in Definition 3.26 with one node-type. In this section this mesh-type is called *delaunay mesh*. There will be different variations of this mesh type with which the experiments will be proceeded. For the experiments we use meshes with 40, 400 Million and 1 Billion nodes. The confining shapes will be the same as for the structured mesh. To construct the meshes of this type the required amount of nodes are created. These nodes get random two or three dimensional coordinate within the confining shape. The next step is to let the `TETGEN`-tool [17] do the Delaunay triangulation for the created meshes. The result of the tool are the edges of the delaunay triangulated mesh. These meshes are then used for the execution of the experiments.

The third category of meshes used are *random meshes* described in Definition 3.16 with one node-type. These meshes do not have a restriction on how the edges might be connected. Similar to the other 2 cases the shape of the structures will be the same. The nodes will be connected in a random fashion over the mesh with no restrictions. Each node will have 3 connections to other nodes. Finally the mean number of edges per node is 6.

5.1.2. Partitioning methods

The used partitioning methods have been defined in Subsection 4.2. The main focus is on creating partitions with space-filling-curves. These are

- Hilbert 2D curve
- Hilbert 3D curve
- Peano 2D curve
- Peano 3D curve
- Z-curve

As a reference and state-of-the-art partitioning method the k-way-partitioning algorithm from the METIS-library [11] is used.

To compute partition numbers of 3D coordinates with 2D space-filling curves, the coordinates are projected on a plane p with the properties

$$p = \langle \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rangle$$

5.1.3. Data organization methods

To test the organization of the data structure the reordering step from Definition 4.5 is used. The nodes, edges and attributes will be sorted in ascending order according to the partitioning-number of the nodes. The resulting meshes then run in a simulation with a simple kernel to measure the execution time. Since all kernels are executing the same arithmetical operations in the same order, the only factor varying the performance is the efficiency of the cache utilization.

5.1.4. Evaluation Methods

To evaluate the created partition of the meshes we use the quality metrics introduced in Section 3.4. Those have been the

- Coverage value
- Node distribution deviation

- Mean nubmer of nodes in the node-closure

To evaluate the data structure organization of a mesh the execution time of the mesh with a simple kernel will be measured. Therefore an attribute for the only node-type is established. The initial attribute value for each node will be set randomly. The kernel used will be the same as in Example 3.29. It will calculate the mean value of the neighbours attributes.

5.2. Evaluation of the Partitioning

To provide a better overview the results of the evaluation are separated by the different mesh types and the dimension of the coordinates of the nodes.

The figures presented provided for each mesh-type illustrate the measured quality metrics to the corresponding number of partitions. The coverage value is stated in %. The node distribution error states the deviation in absolute number. The node closure distribution states the mean value of number of nodes in the node-closure of the computed sub-meshes.

As the measured results for all different mesh-types with different sizes correlate with little deviation - the coverage value, the node distribution deviation and the number of nodes in the node closure changes in a linear fashion with different slopes - only the results for the meshes with 40 Million nodes are presented.

The average of the number of nodes in the sub-meshes is the total number of nodes divided by the number of sub-meshes. Hence this information is not displayed in Section 5.2.

The deviation of the node-closure distribution is in all measured cases smaller than 0.005%. Hence this information is not displayed in Section 5.2.

5.2.1. 2D Grid Structure

The results for partitioning a 2D Grid with $2^1, \ldots, 2^7$ sub-meshes with 40 Million nodes with the presented methods are shown.

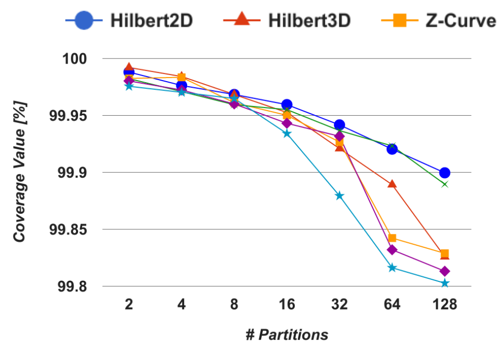

Figure 5.1.: Coverage

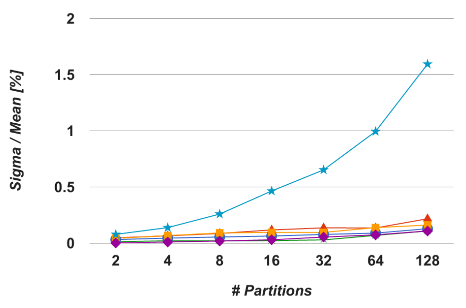

Figure 5.2.: Node distribution error

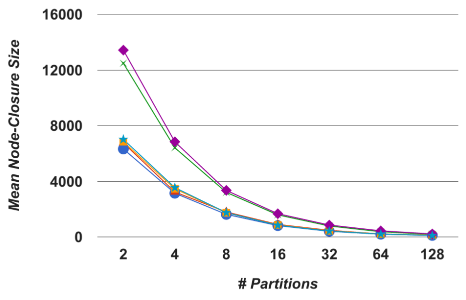

Figure 5.3.: Node closure distribution

In Figure 5.1 we can see that no partition created by a space-filling curve cut more than 0.2% of all edges when computing 128 sub-meshes.

Illustrated in Figure 5.2 that the number of nodes in the sub-meshes is nearly evenly distributed. The node distribution deviation of the space-filling curve partitioned meshes is not exceeding 0.22% of the corresponding mean value.

According to the measured results, also the 2D curves are performing slightly better than the 3D curves.

As we can see in Figure 5.3 the Peano curves are not performing as well as the others. This is caused by the construction of the partitions. To compare the partitions created by the Peano curve with all other curves, we have to create $2, 4, 8, \ldots$ sub-meshes per partition. The highest assigned partition number after partitioning with the Peano 2D with p iterations is $\max_n = 3^{2p} - 1$. To

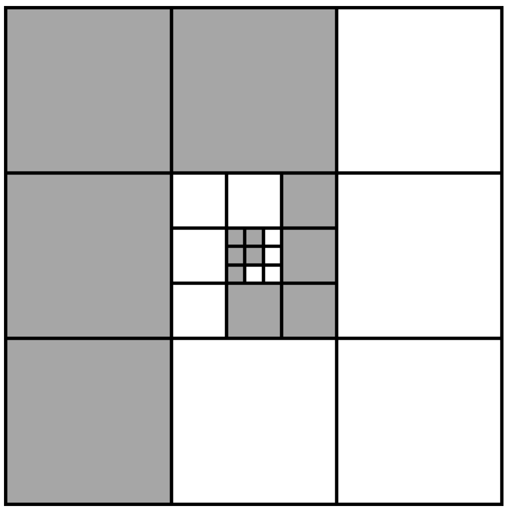

Figure 5.4.: Partition with 2 sub-meshes created by Peano 2D

create $i \in \{2^{j+1} \mid j \in \mathbb{N}\}$ partitions a node with a partition number n is assigned to the sub-mesh m when $\frac{m \cdot \max_n}{i} \leq n < \frac{(m+1) \cdot \max_n}{i} - 1$. Illustrated in Figure 5.4 the nodes inhabited in sub-mesh$_0$ are the nodes in the grey area. The nodes in the white area are assigned to sub-mesh$_1$. We can see in Figure 5.4 that the contact line, on which the nodes are separated, is longer than the length of the line halves the mesh. Hence the mean number of nodes in the node-closure is higher.

5.2.2. 3D Grid Structure

The results for partitioning a 3D Grid with $2^1, \ldots, 2^7$ sub-meshes with 40 Million nodes with the presented methods are shown.

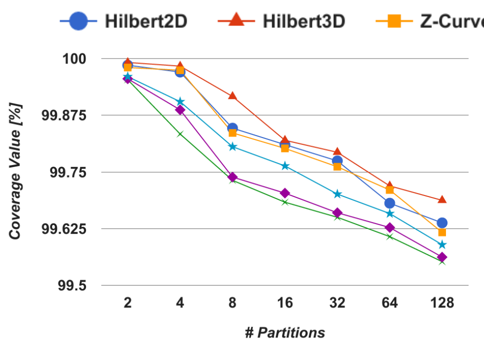

Figure 5.5.: Coverage

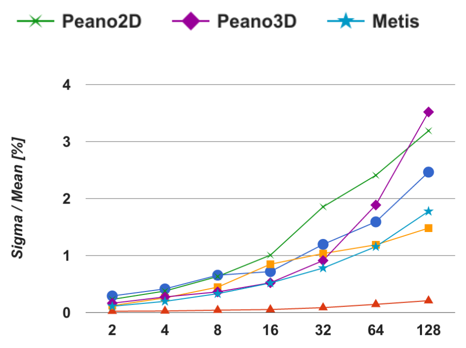

Figure 5.6.: Node distribution error

In Figure 5.5 we can see that there is no partitioning method which cut more

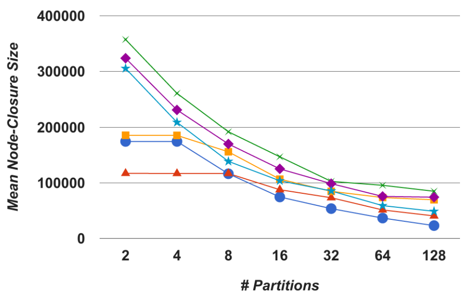

Figure 5.7.: Node closure distribution

than 0.5% of all edges when computing 128 sub-meshes. The partitions computed by the space-filling curves deliver better coverage values than the one computed by the METIS library, as exploiting the spatial domain is of advantage when splitting structured grid.

Again the node distribution deviation is not exceeding 4% of the corresponding mean value. This indicates, that the nodes are well distributed over the sub-meshes.

The best results for partitioning these meshes are delivered by the Hilbert 3D curve.

5.2.3. 2D Delaunay Mesh Structure

The results for partitioning a 2D delaunay triangulated mesh with $2^1, \ldots, 2^7$ sub-meshes with 40 Million nodes with the presented methods are shown.

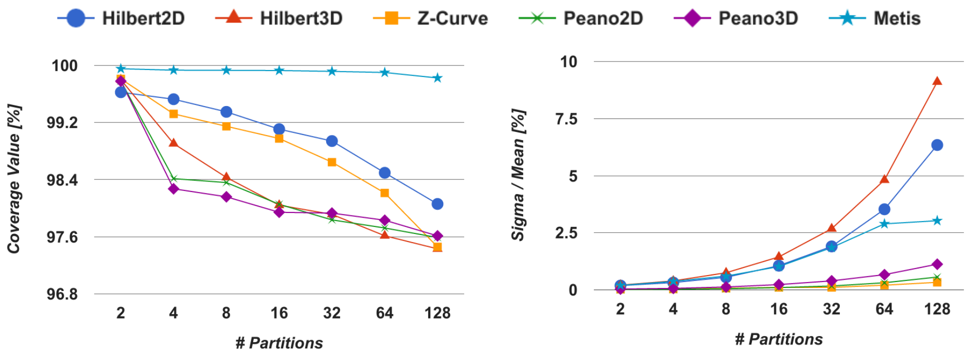

Figure 5.8.: Coverage

Figure 5.9.: Node distribution error

In Figure 5.8 we can see that the total number of edges in the cut of the

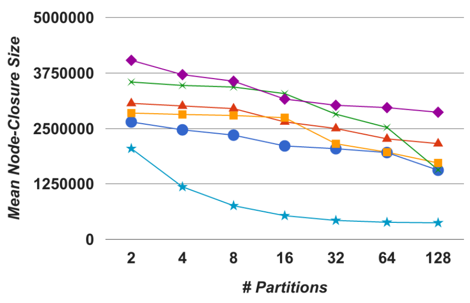

Figure 5.10.: Node closure distribution

partitions is minimal when partitioned with the METIS library. The best results of the space-filling curves are achieved with the Hilbert 2D curve.

The deviation of the node distribution shown in Figure 5.9 is not as good, as in the case presented in Section 5.2.1. The Hilbert 3D partitioning with 128 sub-meshes has an error of 9.1% corresponding to the mean. The Peano 2D curve and the Z-curve have a maximal error of 0.5%.

Looking at the number of nodes in the node-closure in Figure 5.10 we can see, that the Hilbert 2D curve performed best. Compared with the state-of-the-art partitioning method METIS the number of nodes in the closure is halved.

5.2.4. 3D Delaunay Mesh Structure

The results for partitioning a 2D delaunay triangulated mesh with $2^1, \dots, 2^7$ sub-meshes with 40 Million nodes with the presented methods are shown.

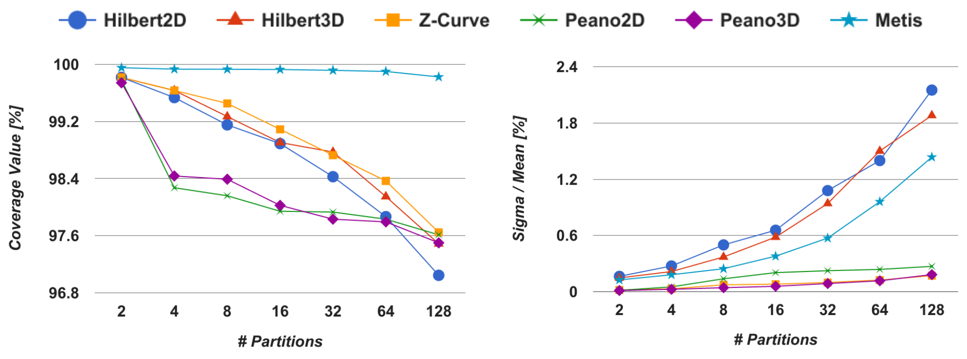

Figure 5.11.: Coverage

Figure 5.12.: Node distribution error

In Figure 5.11 is shown, that the coherency of the METIS partitioned meshes is

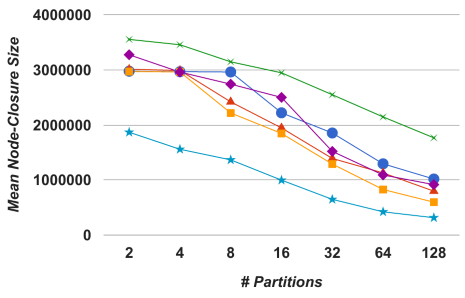

Figure 5.13.: Node closure distribution

higher than the ones partitioned with the space filling curves. The space-filling curve closest to the METIS library is the Z-curve.

Shown in Figure 5.12 is the deviation of node distribution. The Hilbert curves are performing not as well as the other space-filling curves. The maximal error of the Hilbert 2D curve is 2.15%. The best performance are stated by the Peano curves with a maximal error of 0.27%. The overall best performance for this mesh-type has the Z-curve.

5.2.5. 2D Random Mesh

The results for partitioning a 2D random mesh with $2^1, \ldots, 2^7$ sub-meshes with 40 Million nodes with the presented methods are shown. Each node of the mesh is connected to 3 random neighbours. Finally the mean number of edges per node is 6.

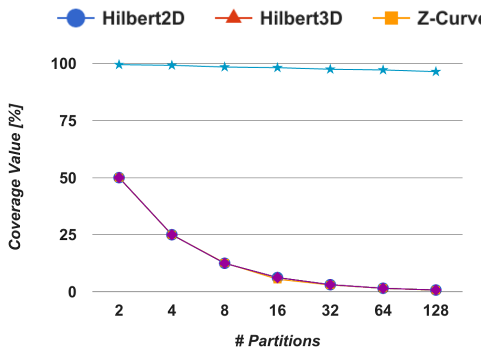

Figure 5.14.: Coverage

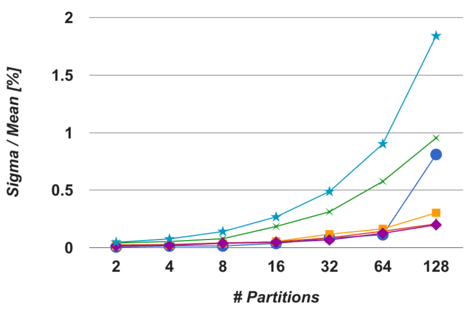

Figure 5.15.: Node distribution error

In Figure 5.14 we can determine that the space-filling curves cut half already in the first step. Only 50% of all edges of the mesh are inside a sub-mesh.

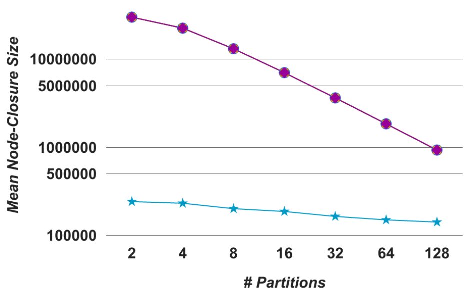

Figure 5.16.: Node closure distribution

With each consecutive step the number of edges in the sub-meshes are cut in half again. This lead to poor results for the coverage value. Only the METIS library is able to compute good structured partitions.

Nevertheless the number node is well distributed over the sub-meshes. The node distribution deviation for the space-filling curve partitioned meshes is not exceeding 1% of the corresponding mean value.

5.2.6. 3D Random Mesh

The results for partitioning a 3D random mesh with $2^1, \ldots, 2^7$ sub-meshes with 40 Million nodes with the presented methods are shown. Each node of the mesh is connected to 3 random neighbours. Finally the mean number of edges per node is 6.

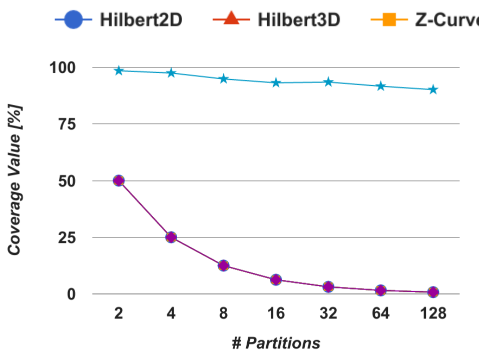

Figure 5.17.: Coverage

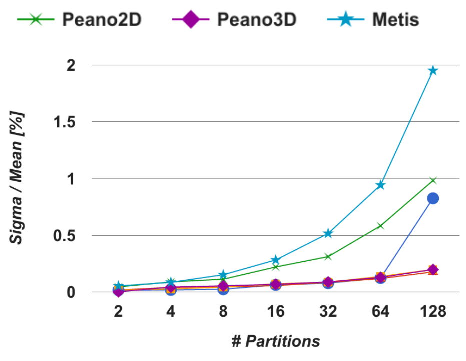

Figure 5.18.: Node distribution error

The evaluation of the partitioning of 3D unstructured meshes is similar to the 2D unstructured meshes presented in Section 5.2.5.

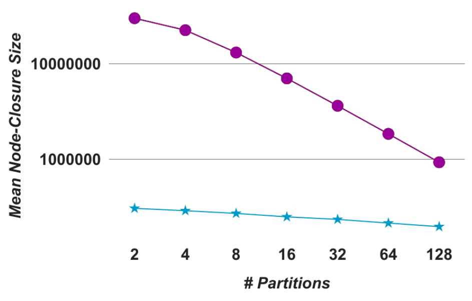

Figure 5.19.: Node closure distribution

In Figure 5.17 we can determine that partitioning unstructured meshes with space-filling curves cut half of all edges when creating 2 sub-meshes. When creating 4 sub-meshes only 25% of all edges are inside the sub-meshes.

5.3. Evaluation of Computation Time

Subsequent to the partitioning the meshes, the nodes have been reordered according to their partition number. To get good results for the reordering for the space-filling curves the partition numbers are number in the interval $[0, 2^63 - 1]$. To reorder the METIS partitioned meshes, 1024 partitions have been created and been reordered according to their partition number. Then simulations with these reordered meshes are performed. The kernel for the simulation is displayed in Example 3.29. Each simulation was executed with 20 time steps.

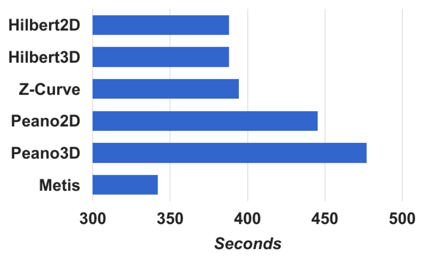

Figure 5.20.: Reordered 2D grid structured meshes execution times

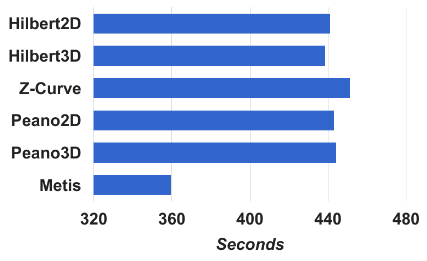

Figure 5.21.: Reordered 3D grid structured meshes execution times

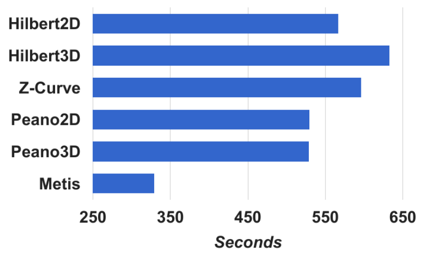

Figure 5.22.: Reordered 2D random meshes execution times

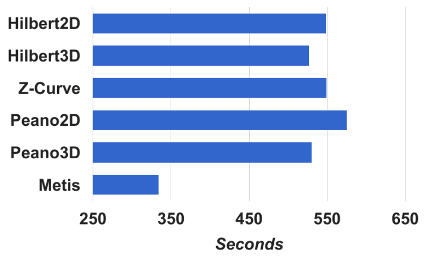

Figure 5.23.: Reordered 3D random meshes execution times

In Figure 5.20 the execution times for the reordered 2D grid structured meshes are displayed. We can see that the Z-curve and the Hilbert curves have the same performance. The Peano curves show an increase of maximal 23% compared to the other space-filling curves. The METIS reordered mesh is execution the simulation 15% faster than the Hilbert curves and the Z-curve.

Figure 5.21 is illustrating the execution times for the reordered 3D grid structured meshes. The execution times of the space-filling curve reorganized meshes differs only about 2%. The state-of-the-art approach is executing 24% faster than all the other methods.

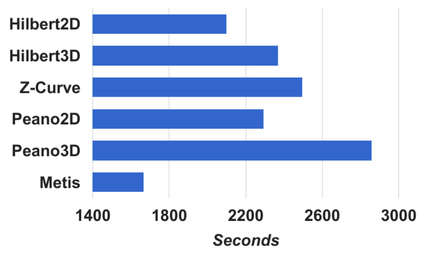

Figure 5.24.: Reordered 2D delaunay meshes execution times

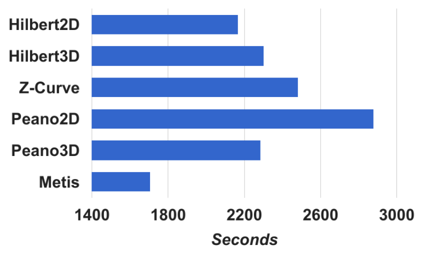

Figure 5.25.: Reordered 3D delaunay meshes execution times

Illustrated in Figure 5.22 and Figure 5.23 we can see that the space-filling curve partitioned random meshes show poor results. The reordered mesh according to the METIS partitioning is 48% faster than the Hilbert 3D approach and 38% faster than the Peano 2D reordered mesh.

In Figure 5.24 and Figure 5.25 the execution times of the 2D and 3D delaunay reordered meshes are presented. The best results for the space-filling curves in both cases are shown by the Hilbert 2D curve. The Hilbert 2D curve reordering performed 9% faster in the 2D case and 6% faster in the 3D case than the second best space-filling curve reordered mesh. Remarkable is that this is the only case, where the 2D space-filling curve provide better results than its 3D version on a 3D data structure. Again the METIS approach achieve the best results. It is 21% and 27% faster than the Hilbert 2D curve reordered mesh in the 2D and 3D case.

6. Conclusion and Future Work

6.1. Conclusion

As shown in Section 5 space-filling curves compute acceptable results for partitioning grid and delaunay structured meshes.

For partitioning grids with space-filling curves the number of nodes per sub-mesh is balanced and coverage values for partitions constructed with space-filling curves get similar results as the partitions created with a k-way algorithm.

Partitioning delaunay triangulated meshes with space-filling curves show some deficits. The number of nodes in the node-closure is nearly double as high compared to the k-way partitioning. The best results for the 2D delaunay triangulated meshes are provided by the Hilbert curves. For the 3D case the Z-curve and the Hilbert 3D curve achieve the best results.

When it comes to partitioning unstructured meshes, the only useful approach is to partition the meshes with a k-way algorithm. The partitions computed with space-filling curves have a deficit when it comes to number of nodes in the node-closure and number of edges in the cut of the partitioning.

The execution times of a simulation with the reordered structured meshes and a kernel show that for the space-filling curves the z-curve and the Hilbert curves perform best, but underlie the METIS reordered mesh.

The best results executing a simulation with delaunay triangulate meshes are achieved by using a Hilbert reordered mesh as data structure.

The unstructured meshes reordered according to the space-filling curves achieve poor results.

Conclusion: Using low-cost space-filling curves for partitioning meshes is a fast and easy way to get a partitioning for a mesh with quality in node distribution of the sub-meshes, coverage and the distribution of nodes to the node closures, but compared to the reordering of the state-of-the-art partitioning method it has huge deficits.

6.2. Future Work

The presented partitioning methods with space filling curves are only tested on artificially created meshes. Testing these partitioning methods with real-world meshes is missing. This includes meshes of different sizes and shapes i.e. circles or irregular structures is missing.

Further investigation should be done on the reason, why METIS reordered meshes dominate the execution time measurement, even so the only 1024 different partition numbers are used.

The simulation with the reordered meshes was executed on a single system. How simulations of the computed sub-meshes executed on a distributed memory system was not investigated by this thesis either. The quality metrics of this thesis has been derived by a model for computing sub-meshes on a cluster. With the execution of the computed partitions on a cluster the impact of the measured quality metrics could be investigated.

Bibliography

[1] Anant Agarwal, J Hennessy, and M Horowitz. An analytical cache model. *ACM Transactions on Computer Systems (TOCS)*, 7(2):184–215, 1989.

[2] Jochen Alber and Rolf Niedermeier. On multi-dimensional hilbert indexings. In *Computing and Combinatorics: 4th Annual International Conference, COCOON'98, Taipei, Taiwan, RoC, August 1998. Proceedings*, pages 1–1. Springer, 1998.

[3] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Engineering graph clustering: Models and experimental evaluation. *ACM Journal of Experimental Algorithmics*, 12(1.1):1–26, 2007.

[4] Arthur R Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Transactions on Computers*, 100(4):424–426, 1971.

[5] Peter J Denning. The locality principle. *Communications of the ACM*, 48(7):19–24, 2005.

[6] John M Dennis. Inverse space-filling curve partitioning of a global ocean model. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.

[7] Welzl Gaertner, Hoffmann. Computational geometry. University Lecture, 2013.

[8] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[9] Herman Haverkort. An inventory of three-dimensional hilbert space-filling curves. *arXiv preprint arXiv:1109.2323*, 2011.

[10] David Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. In *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes*, pages 1–2. Springer, 1935.

[11] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[12] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[13] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[14] Hui Liu, Kun Wang, Bo Yang, Min Yang, Ruijian He, Lihua Shen, He Zhong, Zhangxin Chen, et al. Dynamic load balancing using hilbert space-filling curves for parallel reservoir simulations. In *SPE Reservoir Simulation Conference*. Society of Petroleum Engineers, 2017.

[15] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.

[16] Stefan Schamberger and Jens-Michael Wierum. Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In *International Conference on Parallel Computing Technologies*, pages 165–179. Springer, 2003.

[17] Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)*, 41(2):11, 2015.

[18] Jens-Michael Wierum. Definition of a new circular space-filling curve $\beta\omega$-indexing. 2002.

[19] Sung-Eui Yoon and Peter Lindstrom. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1213–1220, 2006.

[20] Lei Zhang, Guoxin Zhang, Yi Liu, and Hailin Pan. Mesh partitioning algorithm based on parallel finite element analysis and its actualization. *Mathematical Problems in Engineering*, 2013, 2013.

A. Additional data

A.1. Target Architecture used in Experiment Section

Listed are the relevant technical specifications of the target architecture:

CPU:

CPU Type	Intel(R) Xeon(R) CPU E5-4650
Clock Frequency	2.70 GHz

L1 Data Cache:

Total size	32 KB
Line size	64 B
Number of Lines	512
Associativity	8

L1 Instruction Cache:

Total size:	32 KB
Line size	64 B
Number of Lines	512
Associativity	8

L2 Unified Cache:

Total size	256 KB
Line size	64 B
Number of Lines	4096
Associativity	8

L3 Unified Cache:

Total size	20480 KB
Line size	64 B
Number of Lines	327680
Associativity	20

Main Memory:

Memory Size	256 GB

Compiler:

Name	g++
Version	5.1.0
Used Flags	-O3
	-std=c++14

The execution time measurements are performed in the system with a single threaded on the specified architecture.