COUPE: A MESH PARTITIONING PLATFORM

Cédric Chevalier[1,2] Hubert Hirtz[1,2] Franck Ledoux[1,2] Sébastien Morais[1,2]

[1] *CEA, DAM, DIF, F–91297, Arpajon, France*
[2] *LIHPC - Laboratoire en Informatique Haute Performance pour le Calcul et la simulation - DAM Île-de-France, University of Paris-Saclay*
{*Cedric.Chevalier, Hubert.Hirtz, Franck.Ledoux, Sebastien.Morais*}*@cea.fr*

ABSTRACT

This paper presents Coupe, a mesh partitioning platform. It provides solutions to solve different variants of the mesh partitioning problem, mainly in the context of load-balancing parallel mesh-based applications. From partitioning weights ensuring balance to topological partitioning that minimizes communication metrics through geometric methods, Coupe offers a large panel of algorithms to fit user-specific problems. Coupe exploits shared memory parallelism, is written in Rust, and consists of an open-source library and command line tools. Experimenting with different algorithms and parameters is easy. The code is available on Github: `https://github.com/LIHPC-Computational-Geometry/coupe`.

Keywords: mesh partitioning, multi-threading, geometric partitioning, component-based programming

1. INTRODUCTION

For numerical simulation-based analysis, High-Performance Computing (HPC) solutions are nowadays a standard. Numerous solvers run in parallel, and large multi-physics codes are built to take advantage of HPC cluster architectures. Large-scale numerical simulations that run on large-scale parallel computers require the simulation data to be distributed across the computing units (GPU, CPU, or any type of core) to exploit these architectures efficiently. Each unit must process a fair share of the work according to its computing capability. The straightforward way to achieve a "good" load balance is to model each job by its cost and to partition all the jobs between the computing units relatively. The problem with two identical computing units is relative to the classical number partitioning problem [1].

A large category of simulation codes is based on discrete numerical models that rely on the Finite Element Methods or the Finite Volume Methods. In both cases, the geometrical study domain noted Ω must be spatially split into a set of simple atomic elements, called *cells*, that geometrically partition Ω. This set of cells is called a *mesh*. Depending on the numerical methods, those meshes will be very structured, like an entirely regular grid of cubes, mainly structured or fully unstructured. In the latter case, cells are generally simplices (triangles in 2D and tetrahedra in 3D) or generic polyhedra.

The mesh and the numerical and physical data attached to the mesh cells must be partitioned between the different computing units. The goal of the partitioning stage can drastically differ between applications. For load balancing simulations, several tools exist [2, 3, 4, 5] and solve complex graph or hyper-graph partitioning problems[6, 7, 8].

Among the tools mentioned, many have a resolution approach based on topological algorithms and do not take advantage of the geometric information associated with the mesh. Moreover, these tools are mainly focusing on minimizing the communication costs while keeping the imbalance of the solution below a given

maximal imbalance. They do not minimize the load imbalance as an objective, and do not take the memory load and memory capacity of the computing units into account.

In this paper, we present Coupe a mesh partitioning platform that aims to fill the gaps mentioned above. For this purpose, algorithms under developments as well as well known algorithms for geometrical, topological and number partitioning are available. These algorithms are either direct or refinement algorithms and can be easily chained together.

The remainder of this paper is structured as follows. In section 2 we define the Mesh Partitioning Problem and more precise sub-problems which can be of interest, for large-scale applications, depending on what objectives have higher priority. Then, in section 3, we concisely present the main practical algorithms for each partitioning problem. On top of those algorithm we mention two refinement algorithms under development which are used in our experiments. Section 4 focuses on Coupe and discuss its software choices, architecture, and technical differences with other partitioning tools. Moreover, we also present the multiple tools provided to easily experiment partitioning apparoches with Coupe, and other partitioners. Finally, in section 5, we experimentally evaluate our tool by combining multiple kind of algorithms and compare their results to Scotch and Metis.

2. MESH PARTITIONING AND LOAD BALANCING

Previously we have briefly introduced the need to break down the elements of the mesh into several subsets that different computing units will handle.

Load balancing is paramount for any large-scale application: the higher the number of computing units, the most likely one computing unit will have to wait for another, locally stopping the computation.

This paper focuses on mesh-based applications and how solving a Mesh Partitioning Problem can enable high scalability. Mesh Partitioning works with numerical simulation solvers [9] as well as mesh generation software [10].

Giving a precise definition of the mesh partitioning problem is difficult due to the number of constraints and objectives one wants to satisfy. It depends on how the application uses the mesh, the programming paradigm, the data structure layouts, and what kind of hardware performs the computations. We can start a skeleton problem given in definition 1.

Definition 1 (Mesh Partitioning Problem) *Given a mesh $\mathcal{M}$ for which each cell c has a computation cost w_c, find a family $\Pi = (C_i)_{0 \leq i < k}$ of subsets of $\mathcal{M}$ that satisfies:*

- *Partition: each cell c of $\mathcal{M}$ is in exactly one set $C_i \in \Pi$;*
- *Balance: the sum of the weights w_c of the elements c of each set C_i of Π is (approximately) the same ;*
- *Over-cost: the cost (communication, ...) induced by the decomposition Π of $\mathcal{M}$ is minimal.*

The family $\Pi = (C_i)_{0 \leq i < k}$ is named a partition and its elements C_i are called parts.

The "over-cost" objective is often reduced to the communication costs for updating distributed computations. However, it can also represent other data distribution effects, such as a change in the numerical convergence (for domain decomposition linear solvers, for example) or the time spent accessing common resources for shared memory architectures.

In practice, the Mesh Partitioning Problem of definition 1 can be declined in more precise sub-problems, depending on what objectives have higher priority. We will always satisfy the "partition" constraint, but the other two objectives will be modified depending on the chosen model.

2.1 Balance oriented partitioning

The most straightforward Mesh Partitioning Problem reduction focuses only on the "balance" objective. In this case, mesh topology is ignored, and the mesh cells are considered independent between them.

The problem is now the Number Partitioning Problem, a well studied NP-Hard problem [11, 12].

Definition 2 (Number Partitioning Problem) *Given a mesh $\mathcal{M}$ for which each cell c has a computation cost w_c, find a family $\Pi = (C_i)_{0 \leq i < k}$ of subsets of $\mathcal{M}$ such that:*

- *Partition: each cell c of $\mathcal{M}$ is in exactly one set $C_i \in \Pi$;*
- *Balance: Minimize the maximum of the sum of the weights w_c of the elements c of each set C_i of Π.*

This model's main issue ignores any relation or computational dependency between cells. That means that it is very likely that adjacent cells can belong to different parts. In practice, this leads to high communication costs, and when mesh data structure duplicates cells that are parts on a part boundary (ghost cells), there is high memory consumption.

2.2 Geometric partitioning

A simple way to avoid this adjacency issue is to exploit the geometry of the domain. The geometric domain Ω is cut, and the mesh $\mathcal{M}$ is distributed accordingly. This geometric approach leads to more locality for the parts, keeping them almost always connex.

The issue with this approach is that to minimize the imbalance between parts, it is likely that the excellent geometrical domains should be over-cut. The continuous domain can be discomposed using only straight lines [13, 14], but the discrete domain cannot.

In practice, a new Mesh Partitioning Problem is solved, as presented by definition 3. The "balance" objective becomes a constraint: partitioner user fixes a tolerance ε to allow a certain amount of imbalance, and the algorithm satisfies this constraint.

Definition 3 (Balanced-Partitioning) *Given a mesh $\mathcal{M}$ for which each cell c has a computation cost w_c, and an imbalance tolerance ε, find a family $\Pi = (C_i)_{0 \leq i < k}$ of subsets of $\mathcal{M}$ such that:*

- *Partition: each cell c of $\mathcal{M}$ is in exactly one set $C_i \in \Pi$;*
- *Balance: Ensure that the maximum of the sum of the weights w_c of the elements c of each set C_i of Π is less than $1 + \varepsilon$ the average one ($\frac{\Sigma}{k}$).*

Note that definition 3 does not rely on any geometrical notion: this problem is often solved using geometric methods, as we will see later, but it is not required. The "balance" constraint here only focuses on the weight of the heaviest part, but sometimes the lightest part is checked and compared to $1 - \varepsilon$ times the average.

2.3 "Communication" optimized partitioning

With the previous definition, there is still no explicit way to compare different partitions in terms of induced parallelism for the application.

The mesh partition defines the required communications between computing units on distributed memory systems. With Bulk Synchronous Parallel [15] programming, it translates into two communication overcosts. The number of messages and the data transfer volume are metrics for communication consumption. The other cost is synchronization: computing units must wait for the slowest. This wait is not strictly a communication cost but is due to the imbalance of the computations.

In practice, communication cost models use the mesh's topological properties, *i.e.* how cells and other mesh entities are connected. This topological information is embedded into a simpler topological structure such as a graph or a hyper-graph to be partitioned [6, 16, 17]. The primary metrics used to evaluate partition quality are the edge and the $\lambda - 1$ cut. The edge cut is the sum of the weights of the graph edges that link vertices that belong to two different parts. The $\lambda - 1$ cut is a more refined measure, defined on a hyper-graph by being the sum of the product of the weight of a hyper-edge and the number λ of different parts linked by this hyper-edge minus one.

Definition 4 (Communication-Minimized Partitioning) *Given a mesh $\mathcal{M}$ for which each cell c has a computation cost w_c, and an imbalance tolerance ε, find a family $\Pi = (C_i)_{0 \leq i < k}$ of subsets of $\mathcal{M}$ such that:*

- *Partition: each cell c of $\mathcal{M}$ is in exactly one set $C_i \in \Pi$;*
- *Balance: Ensure that the maximum of the sum of the weights w_c of the elements c of each set C_i of Π is less than $1 + \varepsilon$ the average one ($\frac{\Sigma}{k}$).*
- *Communication: Minimize the modeled communication cost (edge-cut, $\lambda - 1$ cut, ...).*

2.4 "Memory" optimized partitioning

Even if definition 4 is the most popular model, some issues still have to be addressed: no direct control over the memory consumption induced by the partition, and the evaluation of the communication cost becomes inaccurate. Applications rely on complex communication semantics, mixing point-to-point, global and pseudo-global communications, or even asynchronous one-sided patterns. No graph or hyper-graph can accurately model such complex patterns. Moreover, non-BSP applications can hide asynchronous communication times by computing. All these aspects are essential to efficiently exploit exascale computers [18].

Memory footprint depends even more on the application implementation. Distributed codes often duplicate boundary vertices to save some communication costs. This duplication has a cost which can be modeled [7] as shown in definition 5.

Definition 5 (Memory-aware Partitioniong) *Given a mesh $\mathcal{M}$ for which each cell c has a computation cost w_c and a memory occupation ω_c, and a list of memory capacity $(M_i)_{0 \leq i < k}$, find a family $\Pi = (C_i)_{0 \leq i < k}$ of subsets of $\mathcal{M}$ such that:*

- *Partition: each cell c of $\mathcal{M}$ is in exactly one set $C_i \in \Pi$;*
- *Balance: Minimize the maximum of the sum of the weights w_c of the elements c of each set C_i of Π;*

- *Memory: Ensure that for each part C_i, the sum of the memory occupation of the elements of C_i and the ghosts are less than M_i.*

An important change between this definition 5 and more classical approaches such as definition 3 or definition 4 is that the "balance" is an objective and not a constraint.

2.5 Other specificities

All previous models assumed that only the mesh partition changes the application's computing properties. However, a numerical impact can exist if the application uses the partition as a base for a domain decomposition solver [19]. Therefore, specific partitioners that focus on the parts' geometric and topological aspect ratio have been designed [20, 21].

Even for linear solvers, if the application relies on the partition to order the matrices, convergence properties can be affected [22].

It is also worth to mention that for the sake of simplicity all these models are for initial partitioning, however most of them are extended to deal with re-partitioning for dynamic load balanced computations [6, 9].

3. ALGORITHMS

In section 2, we have presented many ways to define a Mesh Partitioning Problem. All these different models, except for definition 3, are NP-Hard discrete problems.

This section will present the main practical algorithms for each model. Descriptions will be kept concise as very good surveys [23] or books [24] are available.

3.1 Direct algorithms

Some algorithms can start from scratch to construct a partition of the mesh $\mathcal{M}$.

For the Number Partitioning Problem, a greedy algorithm that takes the heaviest element and puts it into the part of the least weighted is a fast approach. Differencing algorithm, also known as Karmarkar-Karp[25], can lead to better results [26]. However, both methods lead to highly fragmented partitioning when one looks at the resulting mesh.

On the opposite, geometric algorithms such as Recursive Coordinates Bisections [27], multi-jagged[28], or space-filling curves algorithms [29, 30] produce very compact parts [16]. The idea is to travel across the mesh following an axis and find swinging elements up to the weight constraint. These elements are pivot points that define the geometry of the partition. However, the path along the mesh might not allow matching the weight constraints. For example, if the heaviest elements are potential pivots, algorithms can end up in a state where stopping the part here is not enough, but stopping after the next element will be too much.

Graph and hyper-graph partitioners are often a better compromise to achieve good load balance while keeping the communication cost low. Most of the current partitioners such as Kahip [31], Metis [4], Patoh [3], Scotch [5], or Zoltan[2] rely on multi-level algorithms [32, 33]. The idea is to apply a very simple greedy algorithm on a smaller graph and then refine the obtained partition. During different steps named levels, the graph is coarsened by grouping pairs of vertices together, selecting the ones most likely to end up in the same part. Once the graph is small enough, *i.e.* around a few hundred vertices, the greedy initial partitioning is used. Then, the method projects the result, level by level. The projected partition is refined at each level, considering the higher precision of the upper levels graphs.

3.2 Refinement algorithms

Refinement algorithms, *i.e.* processes that take a partition and optimize it according to a partitioning model, are the other class of algorithms commonly used to compute a partition.

Multi-level schemes require such optimization methods. Several approaches exist, but most practical tools rely on either local refinement or diffusion algorithms. Local refinement methods move a vertex from one part to another [34] or exchange pairs of vertices [35]. They produce high-quality results but are challenging to parallelize. We have implemented a modified version of [34] called **Arcswap** which can serve as an example of how Coupe can be used to implement new algorithms. The original algorithm proceeds by moving vertices between parts, one by one, so that the quality of the solution improves the most at every step. A specific table is used to retrieve in constant time a vertex whose move improves the solution the most and great care is taken to also enable constant time updates of the table each time a vertex is moved. By its greedy nature, this algorithm is very much sequential and thus mainly used in multilevel partitioning, where the input size has been drastically reduced. Thus, the idea behind Arcswap is to drop the greedy requirement and the table, and to move any vertex with a positive gain. Diffusion-based methods [36, 37] are more parallel than local refinement methods but do not have a direct effect on the cut.

Close to diffusion methods, a K-means [38] algorithm optimizes a partition geometrically.

On the opposite, we also have implemented a modified version of the greedy algorithm for the Number Partitioning Problem [12] that improves the balance of an initial partition, moving elements between parts. This algorithm, named **VNBest**, consists of iteratively moving a number from the heaviest part to another, selecting the number that most decreases the imbalance. VNBest takes advantage of the relationship between the numbers and the current imbalance to select the best move efficiently.

3.3 Algorithms composition

We saw that mesh partitioning can relate to many problems with many practical algorithms.

Combining partitioning models or algorithms can lead to better, more adapted results. Scotch has a highly customizable way to choose algorithms with its strategy strings: one can chain algorithms using conditionals and tune their parameter accordingly. However, the partitioning problem remains the same for all algorithms: Scotch solves graph partitioning.

For Mesh Partitioning, being able to change the chosen model at each link of the chain is of great interest. For example, one can start with a geometric partition generated from scratch by a fast algorithm, apply a refinement algorithm to optimize the load balance, and then finish with another optimization focusing on minimizing the cut. With this scheme, it should be possible to obtain a pretty well-shaped partition, thanks to the geometric decomposition, but well-balanced and with low communication volume, thanks to the refinement algorithms.

Composing algorithms that work on different problems and with different kinds of inputs is a software challenge. It is hard to do robustly, so we designed Coupe as a mesh partitioning framework with this particular feature in mind.

4. COUPE: A PLATFORM DEDICATED TO MESH PARTITIONING

Coupe was primarily started from scratch following a paradigm shift on partitioning a mesh.

First, the most widely used partitioning tools, except for Zoltan, target arbitrary graphs or other topological structures. Although these structures allow the expression of relationships between mesh elements, they remove important information associated with the mesh: the geometrical information about its elements. In our case, we focus on mesh partitioning and want to leverage such information.

Second, partitioning a mesh does not necessarily require distributing the mesh on several nodes. Indeed, the memory capacity of current machines is often large enough to work locally on the whole mesh, allowing us to use multithreading and GPU acceleration as a way to scale.

Third, the classical tools, built around MPI, follow an approach mainly based on multi-level partitioning. Although this approach is well proven in practice, we believe that it is worth experimenting with inherently scalable algorithms that have the potential to reach a higher threshold of concurrency [28, 38].

Finally, this new platform is intended to host different types of partitioning algorithms that could be coupled for mesh partitioning, as well as different metrics that are not directly treated by the classical tools, e.g., the ghost cells [7].

4.1 Rust as primary development language

A critical distinguishing design decision was to choose Rust [39] as the development language. While building for distributed architectures led software developers to mainly write code in C or Fortran whose support is required by the MPI standard, Coupe is not bounded to this interface. However, it still requires low-level access to hardware that these programming languages offer to fully use CPU cores and GPU accelerators. Moreover, Coupe is intended to host different types of partitioning algorithms to experiment with new partitioning approaches and address practical partitioning problems. Among the proposed algorithms, we can find those dedicated to:

- geometrical or topological partitioning;
- number or vector partitioning, which can be used to obtain balanced initial partitions;
- optimizing an existing partition.

Being convinced of the interest in composing these different algorithms, it is crucial that their implementation does not interfere with others and that the source code, as a whole, is highly modular. The technological challenge induced by the modularity above-mentioned and the extensive use of multi-threading drove us to select Rust.

On the one hand, Rust compiles and produces optimized machine code and has little to no runtime; on the other, it offers features worthy of high-level languages like closures, generics, and iterators. It also has the advantage of having several collection types built into the standard library, such as hash maps and binary heaps. Moreover, safe Rust guarantees an absence of data races and other memory errors detected at compile time, which significantly helps

us build Coupe. Finally, we leverage Rust's ever-growing ecosystem of libraries seamlessly usable from the Cargo package manager, e.g., benchmark frameworks, property testing frameworks, parallel processing libraries, ...

4.2 Integration with other languages

While codes written in C can easily be called from other languages, it is not the case for Rust. For this reason, Coupe proposes a compatibility layer that exposes its features through the C ABI. This layer, called *FFI* (for Foreign Function Interface) or *C bindings*, is the baseline for bindings to other languages. The bindings target *C99* but should work with later versions of the language. The Listing 1 shows an example of the bindings where a mesh, with four cells of unitary weight, is partitioned into two parts using the RCB algorithm.

```c
#include <stdio.h>
#include <coupe.h>

int main() {
  uintptr_t partition[4];

  // Cells' center
  double center_array[4][2] = {{0.0,0.0}, {0.0,1.0},
    {1.0,0.0}, {1.0,1.0}};
  struct coupe_data *centers =
    coupe_data_array(4, COUPE_DOUBLE, center_array);

  // Cells' weight
  int one = 1;
  struct coupe_data *weights =
    coupe_data_constant(4, COUPE_INT, &one);

  // Geometric partitioning using the RCB algorithm
  uintptr_t iter = 1;
  double tolerance = 0.05;
  enum coupe_err err = coupe_rcb(
    partition, 2, centers, weights, iter, tolerance);

  if (err != COUPE_ERR_OK) {
    fprintf(stderr, "Error: %s\n", coupe_strerror(err));
    coupe_data_free(points);
    coupe_data_free(weights);
    return 1;
  }

  printf("With 1 iteration (2 parts), RCB returned:%s\n",
         coupe_strerror(err));
  return 0;
}
```

Listing 1: Coupe C binding in use.

In addition, to facilitate integration into existing source codes that use Metis, we are also working on a library that implements Metis 5.1 API using Coupe's algorithms. This library is a drop-in replacement for libmetis. It can be used by simply replacing *-lmetis* by *-lcoupemetis* in the linker flags.

4.3 The Coupe toolkit

As presented in the previous sections, Coupe can be used in both Rust (native) and C languages. Since we want to experiment with new partitioning approaches easily, multiple tools are provided to work with Coupe, and other partitioners, from the command line. It includes the following tools:

- *num-part*, a framework to evaluate the quality of number partitioning algorithms specifically. This program generates sets of random numbers that follow a given distribution, runs algorithms on these sets, then saves the results in a SQLite database. The supported distributions are:
  - the uniform distribution;
  - the normal/gaussian distribution;
  - the exponential distribution;
  - the pareto distribution;
  - the beta distribution.
- *weight-gen*, a tool used to generate a distribution of cell weights for a given MEDIT mesh. The associated values can either be integers or floating-points and can be laid out following multiple distributions such as:
  - constant: all weights are equal to a given value;
  - linear: weights follow a linear slope on the given axis;
  - spike: weights form spikes of given values at given locations.
- *mesh-part*, a tool to apply partitioning algorithm(s) onto a given mesh. When multiple algorithms are provided, they are chained together. Available algorithms belong to the following categories: number partitioning algorithms, number partition improving algorithms, geometric partitioning algorithms, graph partition improving algorithms, Metis partitioning algorithms, or Scotch partitioning algorithms.
- *part-bench*, a tool used to benchmark the speed of the given algorithms.
- *part-info*, a tool used to print information about the quality of a partition, e.g., the imbalance for each criterion, the edge cut, and the lambda cut.
- *mesh-dup* (resp. *mesh-refine*), a tool used to increase the size of a mesh by duplicating the vertices (resp. splitting its elements into smaller ones).
- *mesh-reorder*, a tool used to change the order of the vertices and the elements of a mesh.
- *apply-part* (resp. *apply-weight*), a tool that encodes a partition (resp. weight distribution) in a mesh file for visualization.

- *mesh-svg*, a tool outputting an SVG given a mesh file.
- a collection of shell scripts that aggregate results into visual reports.

All tools can read and write meshes in different formats via a common module called *mesh-io*. It currently supports reading from and writing to:

- MEDIT, both ASCII and binary variants;
- VTK legacy ASCII and binary.

The *mesh-io* module extracts node coordinates and basic element types (triangles, quads, tetrahedra and hexahedra) for use by other tools. Arbitrary polyhedra and other complex elements, as well as complex topology specifications are discarded, for now. Please note that this is not an issue for the Coupe partitioning library (Rust and C interfaces), which happily accepts any arbitrary graph, coordinate set, and numbers, whether they derive from mesh data or not.

The Listing 2 is a quick walk-through showing how easy it is to experiment with Coupe. We generate a weight distribution from a given mesh, chain several algorithms, analyze the quality of the resulting partition, and display the partitioned mesh.

```
# Cells' weight increase linearly according to its position
# on the X axis.
weight-gen --distribution linear,x,0,100 \
  <sample.mesh \
  >sample.linear.weights

# Partition the mesh and its weight distribution into 2
# parts using RCB and FM algorithms.
mesh-part --algorithm rcb,1 --algorithm fm \
  --mesh sample.mesh \
  --weights sample.linear.weights \
  >sample.linear.rcb-fm.part

# Analyse the partition
part-info --mesh sample.mesh \
  --weights sample.linear.weights \
  --partition sample.linear.rcb-fm.part

# Merge partition file into MEDIT mesh file and
# convert it to an .svg file.
apply-part \
  --mesh sample.mesh \
  --partition sample.linear.rcb-fm.part \
  | mesh-svg >sample.rcb-fm.svg
```

Listing 2: Example of use of Coupe commands.

Note that this sequence of instructions can be easily tested using the Docker image associated with the project.

5. EXPERIMENTS

In this section, we show measures performed on solutions returned by Scotch, Metis and Coupe. The solutions returned by Coupe are the result of different algorithm chains that we will detail. We used each partitioner to generate 100 solutions per instance. We configured Scotch and Metis to allow a maximum imbalance of 1%, i.e. with a tolerance $\varepsilon = 0.01$, and kept other standard parameters. We also used that value for the algorithms of Coupe which required a maximum imbalance value. The reason for such tolerance value is due to the fact that 1% is the default maximum imbalance for Scotch on version 6.1.3 (contrary to Metis which defaults at 3%).

The instances under consideration are obtained from one 2D mesh and two 3D meshes. The 2D mesh was generated from a geometric domain looking like a holed plate and is composed of $12e3$ cells. The two 3D meshes are extracted from the HexMe dataset [40] and can be found under the names `i08c_m8` and `i15u_s8`. They are both composed of tetrahedra and the first one represents a propeller composed of $475e3$ cells, while the second represents a joint composed of $130e3$ cells. From each of these meshes we have created two instances. Both were generated with the *weight-gen* tool. The first one uses a linear distribution where the cells' weight follow a linear slope on a selected axis and where the lowest coordinates on that axis are assigned a value of 1 while the one at the highest coordinates are assigned the value $1e3$. The second one is obtained by using the spike distribution on two different coordinates. The weights obtained form two spikes where weights are of the order of e^{-d} where d is the distance between a cell barycenter and the closest selected spike coordinate. The second type of instance is of great interest given that it is the most dreaded case when partitioning a mesh. Such distribution is illustrated in Figure 1 for the mesh `i08c_m8`. When representing the mesh with a topological model, we derived the weight of the edges from the vertices. Thus, the weight of an edge is equal to the weight of the vertices it connects.

Since we wanted to illustrate the composition of algorithms through Coupe, we have experimented using a couple of combinations that make sense. However, we have not tried all possible combinations because, at the time of writing, Coupe implements the following algorithms:

- Geometrical partitioning: RCB, Hilbert curves (2D only);
- Geometrical improvement: K-Means;
- Topological improvement: Fiduccia-Mattheyses, Arcswap;
- Numerical partitioning: Greedy, Karmarkar-Karp;
- Numerical improving: VNFirst, VNBest;
- Miscellaneous: random, default.

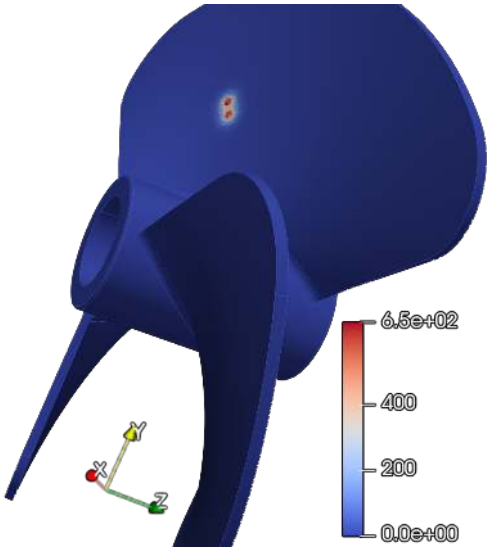

Figure 1: Paraview [41] screenshot of mesh `i08c_m8` where the cells' weights were obtained from two spikes.

As many algorithms consist in refining a solution, we will start every chain of algorithm with an algorithm that can create a partition from scratch. From our current experiments, number partitioning algorithms provide really poor communication costs. Therefore, we will start every composition of algorithms with a geometrical algorithm.

Now let us illustrate the kind of results we can obtain while chaining multiple algorithms. To do this, we focus on the instance with mesh `i08c_m8` and the spike distribution. In contrast to the experimental results presented in the following tables, the experiments were performed while configuring VNBest and Arcswap with a maximum imbalance of 5%.

Let us start off with the geometrical algorithm RCB. Figure 2, like all subsequent figures, is a Paraview screenshot. It illustrates the result obtained after using RCB to split the mesh into eight parts.

Listing 3 shows the associated Coupe command as well as the information associated to the quality of the solution. We can see that the imbalance if of five percents, while the edge cut and $\lambda - 1$ cut are around $60e3 - 70e3$. Note that we specified `rcb,3`, i.e. we asked to run three iterations of RCB, in order to obtain our eight parts.

Since the high weights are concentrated around small areas, we can use a refinement algorithm to improve the imbalance. Here, let us use the VNBest algorithm. As a greedy algorithm, it reduces the imbalance quickly with very few moves which, in our case, should happen close the cut set. Figure 3 shows the result obtained after refining the solution of RCB.

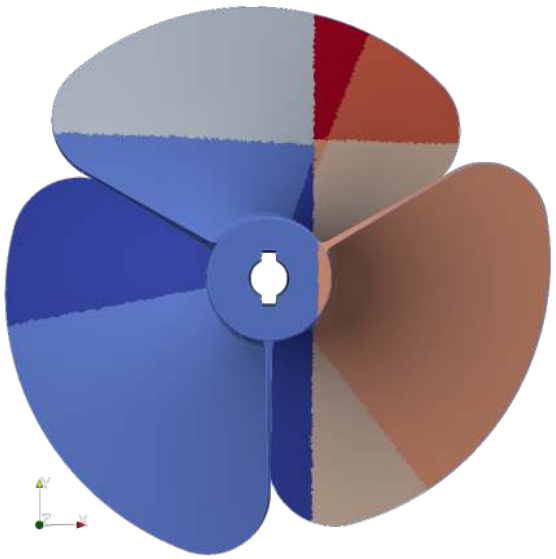

Figure 2: Partition obtained after using RCB to split, along each axis, the mesh into eight parts. Each color represents a part.

```
mesh-part -a rcb,3 -m $mesh -w $weights -E linear |
    part-info -m $mesh -w $weights -E linear -p /dev/stdin
imbalances: [0.05085917717083563]
edge cut: 74067
lambda cut: 60526
```

Listing 3: Coupe command to run RCB and quality information.

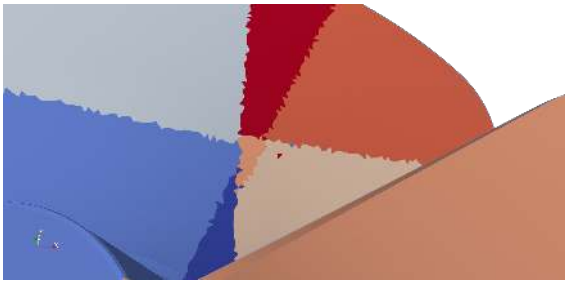

Figure 3: Partition obtained after refining the solution from RCB with VNBest. Because VNBest is unaware of topology, some parts have been disconnected.

You may notice the parts are not connected anymore. This happens frequently because VNBest neither work on geometry nor topology: cells are moved as long as their weight best reduce the imbalance.

In Listing 4, the Coupe command uses the verbose flag to obtain the number of moves made by VNBest. In this case, only fourteen moves have been performed. However, we can see that the imbalance has greatly decreased, while the edge cut and $\lambda - 1$ cut are of the

same order of magnitude.

```
mesh-part -a rcb,3 -a vn-best --verbose #...
vn-best: 14
imbalances: [2.803188159333215e-5]
edge cut: 78094
lambda cut: 64614
```

Listing 4: Coupe command to run RCB + VNBest with inner information.

Finally, as a last step, we try to counterbalance the loss of quality on the edge cut and $\lambda - 1$ cut. For this, we will use the Arcswap algorithm. Figure 4 illustrates the partition obtained after that the VNBest adjustments are corrected by Arcswap to reduce the edge cut. As the figure shows, most irregularities brought by VNBest have been corrected.

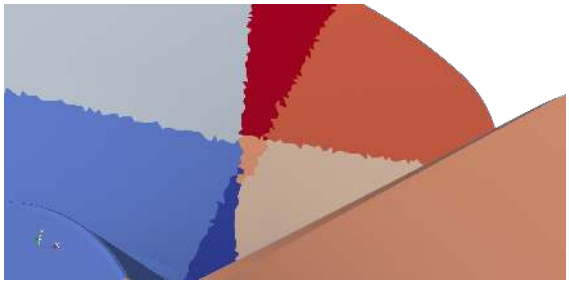

Figure 4: Partition obtained after using Arcswap on the solution from RCB + VNBest. Some adjustments from VNBest are cancelled to improve the edge cut while keeping imbalance below the 5% threshold.

Listing 4 illustrates the Coupe command used to partition the mesh with the chain of algorithms RCB + VNBest + Arcswap. We can see that the solution obtained has better edge cut and $\lambda - 1$ cut than the previous solution while having an imbalance lower than 5%.

```
mesh-part -a rcb,3 -a vn-best -a arcswap,0.05 #...
imbalances: [0.016473402416348194]
edge cut: 73563
lambda cut: 63433
```

Listing 5: Coupe command to run RCB + VNBest + Arcswap with quality information.

For reference, we run Metis' kway procedure. To do so, we have to accommodate with the input constraints Metis and Scotch have: weights are integers and must not be zero. *mesh-part* has to scale the weights to the whole integer type range, and shift them up by one, as done with existing simulation frameworks. Figure 5 shows the partition returned by Metis and listing 6 presents the associated quality information.

Let us now experiment on the 2D instances. Since Hilbert curves can be used for 2D meshes, we have

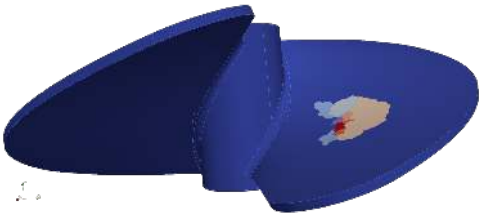

Figure 5: Partition obtained by Metis.

```
mesh-part -a metis:kway,8,0.05 # 8 parts, 5% tolerance
imbalances: [0.021706020313770193]
edge cut: 47706
lambda cut: 47202
```

Listing 6: Coupe command to run Metis with eight parts and 5% imbalance

tried two chains of algorithms: RCB + VNBest + Arcscap and Hilbert + VNBest. Table 1 presents the results of the mean imbalance, edge cut and $\lambda - 1$ cut for 100 runs on each instance. To simplify the writing of algorithm chains, the names of the algorithms are abbreviated as: R=RCB, H=Hilbert curves, V=VNBest, A=Arcswap. Note that the intermediate results are presented for each chain of algorithms.

In Tables 1 and 2,

- the imbalance is computed as the ratio between the weight of the heaviest part and the ideal weight per part, i.e. the weight of all cells divided by the number of parts;
- the edge cut is computed on the graph associated with the mesh, as the sum of the weights on edges that span two parts:

  Let $G = (V, E)$ be a graph, let $W(v)$ be the weight of the vertex v, and $\Pi(v)$ be the part assigned to v. Then the edge cut of Π on G is

  $$\sum_{(v_1,v_2)\in E, \Pi(v_1)\neq\Pi(v_2)} (W(v_1) + W(v_2))$$

- the $\lambda - 1$ cut is computed on the hypergraph associated with the mesh, like so:

  Le $H = (V, E)$ be the hypergraph associated with the mesh. For each hyperedge e in E, let $\lambda(e)$ be $|\{\Pi(v), v \in e\}|$, the number of parts on which e spans. Then the $\lambda - 1$ cut is

  $$\sum_{e\in E} (W(e) \cdot (\lambda(e) - 1))$$

From the results, we can see that using RCB algorithm can result in solutions whose cut quality may be better than that returned by Metis or Scotch. However,

this is at the expense of the load balancing. As for Hilbert, it seems to allow us to obtain better balanced solutions but for which the quality of the cut is not better than that of Metis. It should be noted that, for the spikes distribution, the result obtained is better than Scotch. As constated in the previous chaining walk through, using VNBest and Arcswap one after the other can improve the imbalance while avoiding too much degradation of the quality of the cut metrics. This is of particular interest as RCB + VNBest + Arcswap and Hilbert + VNBest manage to return solution with a mean imbalance that respect the imbalance constraint, i.e. have an imbalance of less than 1%. However, this is not the case for Scotch. Moreover, the average imbalance obtained can be better than that returned by Metis while having a mean cut quality close to that returned by Metis.

Partitioner	Imbalance	Edge cut	$\lambda - 1$ cut
Spike distribution			
R	7.3%	1.5e5	**1.5e5**
R-V	**0.078%**	4.0e5	3.7e5
R+V+A	0.47%	2.1e5	1.9e5
H	0.89%	2.1e5	1.8e5
H-V	0.29%	2.1e5	1.8e5
Metis	1.0%	**1.7e5**	**1.5e5**
Scotch	1.8%	3.1e5	3.0e5
Linear distribution			
R	7.4%	4.9e4	**3.8e4**
R+V	**0.009%**	7.7e4	6.2e4
R+V+A	0.47%	5.7e4	5.2e4
H	0.069%	7.6e4	6.1e4
H-V	0.022%	7.6e4	6.1e4
Metis	0.96%	**4.4e4**	**3.8e4**
Scotch	0.99%	4.5e4	4.4e4

Table 1: Measures of the mean imbalance, edge cut and $\lambda - 1$ cut for 100 runs on the two instance from 2D mesh. Best results for each metric are in bold. The names of the algorithms are abbreviated as: R=RCB, H=Hilbert curves, V=VNBest, A=Arcswap.

As an illustration of the results returned by Hilbert algorithm, Figure 6 presents the partition generated for the case where weights are distributed linearly.

Now, let us focus on the 3D instances. Since 3D Hilbert curve generation is yet to be implemented, we will focus on the chain RCB + VNBest + Arcswap. Contrary to the 2D instances, there is no 3D instance where the mean cut quality achieved with RCB is lower than Metis. However, it should be noted that this is not the case for Scotch. Moreover, we can see that there are instances where Scotch does not find a valid solution to the imbalance constraint, whereas this is not the case for our chain of algorithms. Finally, we can observe that the successive application of algorithms that do not optimise the same problems does not necessarily lead to antagonistic modifications that cannot be corrected. Indeed, for three instances out of four, we can observe that the average quality of the edge cut of the solutions returned by RCB is close or equal to that returned by RCB + VNBest + Arcswap no matter if the application of VNBest had greatly deteriorated the average quality.

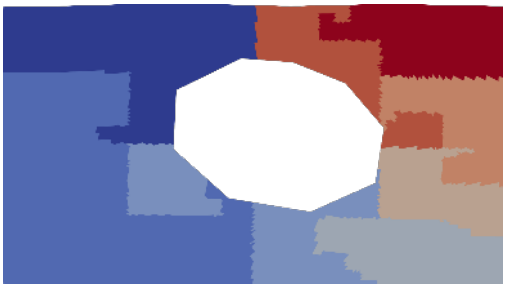

Figure 6: The 2D mesh partitionned using an Hilbert curve.

Partitioner	Imbalance	Edge cut	$\lambda - 1$ cut
`i08c_m8` Propeller - Spike distribution			
R	5.1%	7.4e4	6.1e4
R+V	**0.003%**	7.8e4	6.5e4
R+V+A	0.50%	7.7e4	6.4e4
Metis	0.86%	**4.9e4**	**4.6e4**
Scotch	1.8%	6.7e4	6.4e4
`i08c_m8` Propeller - Linear distribution			
R	6.7%	11e6	8.6e6
R+V	**0.001%**	18e6	14e6
R+V+A	0.99%	11e6	10e6
Metis	1.0%	**5.8e6**	**5.4e6**
Scotch	1.0%	21e6	21e6
`i15u_s8` Joint - Spike distribution			
R	7.4%	9.5e3	8.5e3
R+V	**0.060%**	11e3	9.5e3
R+V+A	0.13%	11e3	9.5e3
Metis	0.94%	**6.8e3**	**6.5e3**
Scotch	3.6%	11e3	11e3
`i15u_s8` Joint - Linear distribution			
R	2.1%	5.3e6	4.1e6
R+V	**0.004%**	6.7e6	5.2e6
R+V+A	0.96%	5.4e6	4.8e6
Metis	1.0%	**3.4e6**	**3.1e6**
Scotch	1.0%	8.5e6	8.1e6

Table 2: Measures of the mean imbalance, edge cut and $\lambda - 1$ cut for 100 runs on the two instance from 2D mesh. Best results for each metric are in bold. The names of the algorithms are abbreviated as: R=RCB, V=VNBest, A=Arcswap.

6. CONCLUSION AND PERSPECTIVES

This paper describes Coupe, a mesh partitioning platform that provides solutions to solve different variants of the mesh partitioning problem, mainly in the context of load-balancing parallel mesh-based applications. Its codebase is made modular, and multiple direct or refinement algorithm have already been implemented. These algorithms are geometrical, topological or number partitioning algorithms and can be easily composed together to allow users to fine-tune how they partition their mesh. Although it is still a work in progress, the project is very much usable, and algorithm chaining experiments can easily be performed and can be carried out on meshes of the order of a million cells.

Coupe can be used from Rust and C languages as well as command-line thanks to the Coupe toolkit. Among the proposed tools, some are dedicated to instance generation and can be used, for example, to generate weight cells according to several available distributions; others can be used to apply one or multiple partitioning algorithms on a mesh; and some are used to compute information on the quality of the generated solutions and aggregate results into visual reports.

All library and tooling code is open source and hosted on Github: `https://github.com/LIHPC-Computational-Geometry/coupe`. We also offer a Docker image for easier access and use.

From the experimental results obtained, we are convinced of the interest of geometric partitioning algorithms for mesh partitioning. Especially since the chaining of algorithms seems to be a sensible approach to compensate, if necessary, the solutions returned by geometric algorithms.

Soon, we intend to develop the library that implements Metis 5.1 API using Coupe's algorithms in order to facilitate integration into existing source codes that use Metis. We also want to integrate Coupe into the Arcane framework [42], a software development framework for 2D and 3D numerical simulation codes. This will set performance goals to attain and deliver profiling data. It is also planned to support multi-criteria runs, i.e. runs where cells have multiple associated computation costs. Finally, we also want Coupe to propose algorithms for the Memory-aware Mesh Partitioning Problem. A first step in this direction will be the implementation of [7].

References

[1] Leung J., Kelly L., Anderson J. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., 2004

[2] Boman E.G., Çatalyürek U.V., Chevalier C., Devine K.D. "The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering and Coloring." *Scientific Programming*, vol. 20, no. 2, 129–150, 2012

[3] Çatalyürek U.V., Aykanat C. "PaToH: Partitioning Tool for Hypergraphs.", Mar. 2011

[4] Karypis G., Kumar V. Metis *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN 55455, U.S.A., fourth edn., Sep. 1998

[5] Pellegrini F., Roman J. "Scotch: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs." H. Liddell, A. Colbrook, B. Hertzberger, P. Sloot, editors, *High-Performance Computing and Networking: International Conference and Exhibition HPCN EUROPE 1996 Brussels, Belgium, April 15–19, 1996 Proceedings*, pp. 493–498. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996

[6] Çatalyürek U., Boman E.G., Devine K.D., Bozdag D., Heaphy R., Riesen L.A. "Hypergraph-Based Dynamic Load Balancing for Adaptive Scientific Computations." *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–11. Mar. 2007

[7] Chevalier C., Ledoux F., Morais S. "A Multilevel Mesh Partitioning Algorithm Driven by Memory Constraints." *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing (CSC)*, Proceedings, pp. 85–95. Society for Industrial and Applied Mathematics, Jan. 2020

[8] Fischer E., Matsliah A., Shapira A. "Approximate Hypergraph Partitioning and Applications." *SIAM Journal on Computing*, vol. 39, no. 7, 3155–3185, Jan. 2010

[9] Chevalier C., Grospellier G., Ledoux F., Weill J. "Load Balancing for Mesh Based Multi-Physics Simulations in the Arcane Framework." *The Eighth International Conference on Engineering Computational Technology*, p. 4. Dubrovnik, Croatia

[10] Lachat C., Dobrzynski C., Pellegrini F. "Parallel Mesh Adaptation Using Parallel Graph Partitioning." *5th European Conference on Computational Mechanics (ECCM V)*, vol. 3, p. 2612. CIMNE -

International Center for Numerical Methods in Engineering, Jul. 2014

[11] Garey M.R., Johnson D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979

[12] Korf R.E. "From Approximate to Optimal Solutions: A Case Study of Number Partitioning." *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pp. 266–272. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995

[13] Dubins L.E., Spanier E.H. "How to Cut a Cake Fairly." *The American Mathematical Monthly*, vol. 68, no. 1P1, 1–17, Jan. 1961

[14] Peters J.V. "The Ham Sandwich Theorem and Some Related Results." *Rocky Mountain Journal of Mathematics*, vol. 11, no. 3, 473–482, Sep. 1981

[15] Valiant L.G. "A Bridging Model for Parallel Computation." *Communications of the ACM*, vol. 33, no. 8, 103–111, Aug. 1990

[16] Boman E., Catalyurek U., Chevalier C., Devine K., Safro I., Wolf M. "Advances in Parallel Partitioning, Load Balancing and Matrix Ordering for Scientific Computing." *Journal of Physics: Conference Series*, vol. 180. 2009

[17] Hendrickson B., Kolda T.G. "Graph Partitioning Models for Parallel Computing." *Parallel Computing*, vol. 26, no. 12, 1519–1534, Nov. 2000

[18] Sarkar V., Harrod W., Snavely A.E. "Software Challenges in Extreme Scale Systems." *Journal of Physics: Conference Series*, vol. 180, 012045, Jul. 2009

[19] Farhat C., Maman N., Brown G.W. "Mesh Partitioning for Implicit Computations via Iterative Domain Decomposition: Impact and Optimization of the Subdomain Aspect Ratio." *International Journal for Numerical Methods in Engineering*, vol. 38, no. 6, 989–1000, 1995

[20] Walshaw C., Cross M., Diekmann R., Schlimbach F. "Multilevel Mesh Partitioning for Optimising Aspect Ratio." *VECPAR*, pp. 285–300. 1998

[21] Diekmann R., Preis R., Schlimbach F., Walshaw C. "Shape-Optimized Mesh Partitioning and Load Balancing for Parallel Adaptive FEM." *Parallel Computing*, vol. 26, no. 12, 1555–1581, Nov. 2000

[22] Duff I.S., Meurant G.A. "The Effect of Ordering on Preconditioned Conjugate Gradients." *BIT Numerical Mathematics*, vol. 29, no. 4, 635–657, Dec. 1989

[23] Buluç A., Meyerhenke H., Safro I., Sanders P., Schulz C. "Recent Advances in Graph Partitioning." L. Kliemann, P. Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, Lecture Notes in Computer Science, pp. 117–158. Springer International Publishing, Cham, 2016

[24] Bichot C.E., Siarry P. *Graph Partitioning*, vol. 2011. 2011

[25] Karmarkar N., Karp R.M. "The Differencing Method of Set Partitioning." Tech. rep., University of California at Berkeley, Berkeley, CA, USA, 1983

[26] Boettcher S., Mertens S. "Analysis of the Karmarkar-Karp Differencing Algorithm." *European Physical Journal B*, vol. 65, no. 1, 131–140, 2008

[27] Berger, Bokhari. "A Partitioning Strategy for Nonuniform Problems on Multiprocessors." *IEEE Transactions on Computers*, vol. C-36, no. 5, 570–580, May 1987

[28] Deveci M., Rajamanickam S., Devine K.D., Çatalyürek U. "Multi-Jagged: A Scalable Parallel Spatial Partitioning Algorithm." *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, 803–817, Mar. 2016

[29] Pilkington J.R., Baden S.B. "Partitioning with Spacefilling Curves." Tech. rep., Dept. of Computer Science and Engineering, University of California, San Diego, 1994

[30] Bader M. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Publishing Company, Incorporated, 2012

[31] Sanders P., Schulz C. "Think Locally, Act Globally: Highly Balanced Graph Partitioning." V. Bonifaci, C. Demetrescu, A. Marchetti-Spaccamela, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pp. 164–175. Springer, Berlin, Heidelberg, 2013

[32] Barnard S.T., Simon H.D. "A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems." *Concurrency: Practice and Experience*, vol. 6, no. 2, 101–117, 1994

[33] Hendrickson B., Leland R. "A Multilevel Algorithm for Partitioning Graphs." *Proceedings of Supercomputing*. 1995

[34] Fiduccia C., Mattheyses R. "A Linear-Time Heuristic for Improving Network Partitions." *19th Design Automation Conference*, pp. 175–181. Jun. 1982

[35] Kernighan B.W., Lin S. "An Efficient Heuristic Procedure for Partitioning Graphs." *The Bell System Technical Journal*, vol. 49, no. 2, 291–307, Feb. 1970

[36] Pellegrini F. "A Parallelisable Multi-Level Banded Diffusion Scheme for Computing Balanced Partitions with Smooth Boundaries." T.P. A.-M. Kermarrec, L. Bougé, editor, *EuroPar*, vol. 4641 of *Lecture Notes in Computer Science*, pp. 195–204. Springer, Rennes, France, Aug. 2007

[37] Meyerhenke H., Monien B., Schamberger S. "Graph Partitioning and Disturbed Diffusion." *Parallel Computing*, vol. 35, no. 10, 544–569, Oct. 2009

[38] von Looz M., Tzovas C., Meyerhenke H. "Balanced K-Means for Parallel Geometric Partitioning." *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, pp. 1–10. Association for Computing Machinery, New York, NY, USA, Aug. 2018

[39] Matsakis N.D., Klock F.S. "The Rust Language." *ACM SIGAda Ada Letters*, vol. 34, no. 3, 103–104, Oct. 2014

[40] Beaufort P.A., Reberol M., Kalmykov D., Liu H., Ledoux F., Bommes D. "Hex Me If You Can." *Computer Graphics Forum*, 2022

[41] Schroeder W., Martin K.M., Lorensen W.E. *The visualization toolkit (2nd ed.): an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., USA, 1998

[42] Grospellier G., Lelandais B. "The Arcane Development Framework." *Proceedings of the 8th Workshop on Parallel/High-Performance Object-Oriented Scientific Computing*, POOSC '09. Association for Computing Machinery, New York, NY, USA, 2009. URL `https://doi.org/10.1145/1595655.1595659`