Two-constraint Domain Decomposition with Space Filling Curves

Christian Konrad
konrad@lri.fr

LRI, Univ. Paris-Sud, F-91405 Orsay, France

Abstract

In scientific computing, Space Filling Curves are a widely used tool for one-constraint domain decomposition. They provide a mechanism to sort multi-dimensional data in a locality preserving way, and, in this way, a (one dimensional) list of mesh elements is established which is subsequently split into partitions under consideration of the constraint. This procedure has a runtime of $O(N \log N)$ (N is the number of mesh elements) while nearly perfect load balancing can be established with reasonable partition surface sizes.

In this work, we discuss the extensibility of this procedure to two-constraint settings which is desirable, since the methodology is extremely fast. Here, the splitting operation is subject to two constraints, and, unlike to the one-constraint case, obtaining near perfect balancing is often hard to establish, and is, even more as in the one-constraint case, in conflict with the induced surface sizes (or edge-cuts). We discuss multiple strategies to tackle the splitting, and we present a fast, $O(N \log N)$ splitting heuristic algorithm which provides an integer σ that allows to trade off between balancing and surface sizes which results in a $O(N \log N)$ two-constraint decomposition method. Results are compared to the multi-constraint domain decomposition abilities implemented in the Metis software package, and show that the method produces higher surface sizes, but is orders of magnitudes faster which makes the method superior for certain applications.

1 Introduction

Combinatorial Scientific Computing (CSC) [8] comprises combinatorial problems arising in computational science. Especially in parallel settings, many problems, such as load balancing, optimizing data flows through interconnection networks, or the determination of independent data units, arise, and can be tackled by a combinatorial formulation, e.g. as graph partitioning problems, as flow networks or graph coloring problems. The quality of solutions to these problems impact significantly on the performance of the application [8]. *Hendrickson and Pothen* manifest this fact by entitling one of their papers with "Combinatorial Scientific Computing: The Enabling Power of Discrete Algorithms in Computational Science" [8].

Since many of the CSC problems are NP-complete, good heuristics are required, and, if the problem has to be solved during the execution of the parallel application, these heuristics should have low runtimes, which is in general in conflict with the quality of the solutions. These facts already announce that the selection of an appropriate method is highly problem dependent, and it is desirable that a variety of methods with different characteristics exist.

The present work considers domain decomposition problems, that is, given a computational mesh consisting of arbitrary elements, find a mapping that assigns a partition number to each element. Typically, there are at least two demands on the decomposition, that is, the *work load* should be distributed equally, and the *surfaces* between partitions should be as small as possible in order to minimize communication in case of data dependencies among adjacent elements across partition boundaries.

We assume work load to be distinguishable per mesh element. We model it as a mapping attaching a positive number to each mesh element which is proportional to the work load induced by this mesh element,

and we denote such mappings as *weights*. One-constraint (one weight) problems arise in many parallel mesh-based computations such as Finite Element (FE) type methods for wave propagation problems [17], or the Direct Simulation Monte Carlo (DSMC) method for gas flow problems [25].

In this work we focus on situations where two weights are present which is for instance the case for scientific applications with two independent computational phases. As shown in figure 1, the processes are synchronized not only at the beginning of each time step, but also between the independent computational phases of each time step.

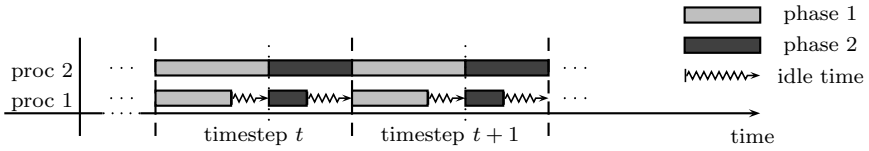

Figure 1: Two phase computation on two processors: each timestep consists of two independent computational phases. A load balancing strategy has to consider both phases.

Particle simulations such as the Particle-In-Cell (PIC) method [9] for plasma computations are a special type of two phase computation. At its origins, this work was inspired by a particular PIC application, however, we want to underline that we focus here on the general setting of two-constraint (two weight) problems. We refer to particle simulations to provide an illustrative example.

In the PIC method phase one corresponds to the solution of the Maxwell equations on a computational mesh, which imposes a first balancing constraint. In phase two, particles are moved. Since particles interact with their surrounding element, it is beneficial to place them onto the same processor as their governing elements. Hence, a domain decomposition has to balance both, the number of elements per partition, and the number of particles per partition.

Research on load balancing schemes for parallel PIC simulations is conducted since at least two decades, and a multitude of strategies different to what we discuss here were proposed, see for instance the general discussion by *Carmona and Chandler* [3] or a recent work by *Nakashima et al.* [18]. They, as well as many other works (e.g. [7]), follow a completely different approach which does not require that both, particles and mesh elements are balanced in each partition. They use a static grid decomposition independently of the particles. Their approach to handling the imbalance induced by the particles is a helping scheme where processors with few particles *help* processors that are overpopulated with particles.

Here, we do not wish to lead a discussion on load balancing schemes for PIC simulations, our goal is simply to provide the reader a descriptive two-constraint example. Figure 2 illustrates the two arrising constraints we aim to balance, that is the number of mesh elements and the number of particles per partition. The illustration on the right displays a solution balancing both constraints.

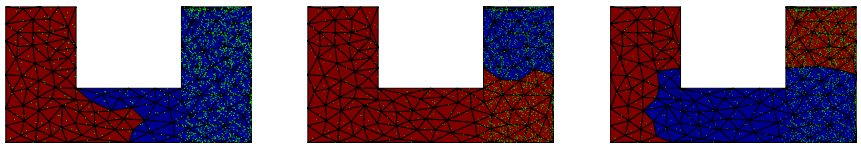

Figure 2: A two-constraint domain decomposition problem. Left: number of elements is balanced (one constraint), Center: number of particles is balanced (one constraint), Right: number of particles and number of elements is balanced (two constraints).

Furthermore, the example of a particle simulation also demonstrates the need for a fast domain decomposition method. Since particles move throughout the simulation, they may leave the domain of a processor's

partition and consequently migrate to another processor. However, in doing so, this induces imbalance for the second phase, and, if the imbalance grows too significantly, a domain redecomposition during the execution of the simulation is required. For instance *Wu and Tseng* report in [10] on a scientific application based on particles applying dynamic domain decomposition for the balancing of one weight.

In this paper we discuss domain decomposition for two-constraint problems based on Space Filling Curves (SFCs). The outline of this paper is as follows.

In section 2 we briefly state related works, section 3 introduces necessary notations. We discuss one-constraint domain decomposition with SFCs in section 4 to provide the necessary armamentarium for the extension of this method to two-constraints. This extension to two-constraints is the main contribution of this paper, and we discuss this in detail in section 5. In section 6 we demostrate the potential of this method with an example, and we conclude with section 7.

2 Related work

Domain decomposition with SFCs is matter of ongoing research. Publications range from theoretical discussions of the *locality* of SFCs, e.g. [4, 26, 5], which is directly related to the induced surfaces of a partitioning, to applications that profit of domain decompositions with SFCs e.g. [6]. However, to our best knowledge, there are no attempts to apply SFCs as a device for two-constraint domain decomposition.

Besides SFCs, domain decomposition problems are often tackled as graph partitioning problems. Robust graph partitioning software packages exist (such as Metis [22], Chaco [20], Jostle [21], Scotch [23]) which are convenient to apply, and result in good quality partitionings by reasonable runtimes. *Schamberger and Wierum* provide in [24] a comparison of Metis and SFC induced partitions aiming to produce partitions of equal sizes. It is difficult or impossible to derive absolute conclusions but, clearly, SFCs save a lot of time as well as memory resources by reasonably larger surfaces.

To our best knowledge, the only domain decomposition methods that are capable of balancing two-constraints are graph (and hypergraph) partitioners. Full multi-constraint graph partitioning [14] is implemented in the Metis package to which we compare our results.

3 Preliminaries

Throughout the paper, we consider an arbitrary triangulation $T = \{\tau_i | 1 \leq i \leq N\}$ into N arbitrary elements $(\tau_i)_{i=1\dots N}$, such that our geometry $\Omega = \bigcup_i \tau_i \subseteq [0,1]^d$, where $d \in \{2,3\}$ is the dimensionality. For the sake of simplicity, we restrict Ω to be a subset of the d-dimensional unit cube, since we define SFCs on this domain. $cg(\tau_i)$ denotes the *center of gravity* of an element τ_i. We assume $f : [0,1] \to [0,1]^d$ to be some continuous SFC and we assume the presence of some computation scheme that allows to approximate f^{-1} sufficiently exact.

A *domain decomposition* of T into p parts is a mapping $d : T \to [1,p]$, we write $[a,b] = \{a, a+1, \dots, b\}$ for integers a, b with $b \geq a$. We denote $P_i = \{j \,|\, d(\tau_j) = i\}$, $i = 1 \dots p$, the *partitions* induced by d. A *weight* is a mapping $\omega : T \to \mathbb{R}^+$, and we assume for simplicity that $\sum_{\tau \in T} \omega(\tau) = 1$. In the one-constraint situation (section 4), we assume the presence of one weight ω_1, in the two-constraint setting (section 5) we add a second weight ω_2.

The *load* of a partition P_i with respect to a weight ω is defined as $l(P_i, \omega) = \sum_{j \in P_i} \omega(\tau_j)$. We measure the *imbalance* of a weight ω as $\mu(\omega) = p \cdot \max\{l(P_i, \omega) \,|\, 1 \leq i \leq p\}$.

For a vector v we denote by $\max v$ the largest element of v, and by $\min v$ the smallest element of v.

4 One-constraint domain decomposition with Space Filling Curves

The usual procedure to decomposing T along the SFC f is as follows:

---

1. *Index computation:* $\forall i : h_i = f^{-1}(cg(\tau_i))$
2. *Sorting:* Find a permutation $\pi \in S_{[1,N]}$ such that $h_{\pi^{-1}(i)} < h_{\pi^{-1}(j)} \, \forall i < j$.
3. *Splitting:* Find a vector of separators $s \in \mathbb{N}^{p-1}$ with $0 < s_1 < \cdots < s_{p-1} < N$ (throughout the document we assume $s_0 = 0, s_p = N$) such that $\mu(\omega_1)$ with respect to the partitions $P_i = \{\tau_j \,|\, s_{i-1} < \pi(j) \leq s_i\}$ is minimized.

The domain decomposition d can be stated as: $d(\tau_i) = \min\{k \,|\, \pi(i) \leq s_k\}$.

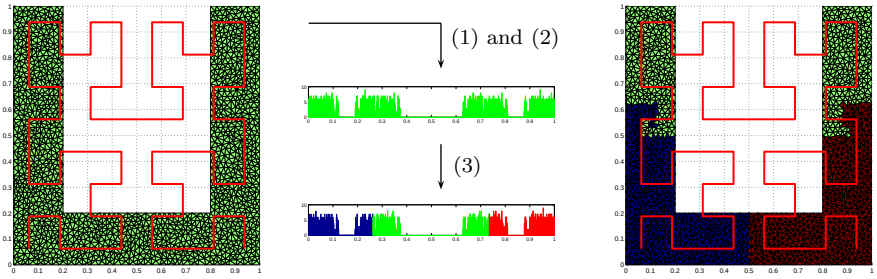

Figure 3: Domain decomposition of a U-shaped geometry into 3 partitions along the Hilbert curve: SFC indices are computed (1) and sorted (2). The histograms display the distribution of the Hilbert indices of the mesh elements. (3) indicates the splitting operation.

Figure 3 visualizes the one-constraint decomposition procedure. In this example, we apply a constant weight function $\omega_1 = 1/N$ which corresponds to the aim of balancing the number of mesh elements per partition.

The index computation depends on the SFC f, we assume that the computation of an index can be done in $O(\log N)$ [1] time, hence the index computation performs in $O(N \log N)$ time. With a standard sorting algorithm the sorting step equally performs in $O(N \log N)$.

The splitting problem is a well studied problem in the literature. We define it here precicely (decoupled from our application at hand) since two-constraint decomposition differs from one-constraint decomposition at this point.

Problem 4.1. *Let $N > p > 0$, and $W = (w_1, \ldots, w_N) \in (\mathbb{R}^+)^N$ with $w_i > 0$ and $\sum_i w_i = 1$. A solution to the **one-constraint splitting problem** is a vector $s \in \mathbb{N}^{p-1}$ with $0 < s_1 < s_2 < \cdots < s_{p-1} < N$ $(s_0 = 0, s_p = N)$, such that:*

$$I_1(s) = \max\{ \sum_{i=s_{i-1}+1}^{s_i} w_i \,|\, 0 < i \leq p\} \tag{1}$$

is minimal.

[1] Most computation schemes of SFC indices grow linearly with the desired precision, e.g. the Butz-algorithm [16, 2], or, more recently, a table driven framework [11]. We state logarithmic time since we require a precision of $O(\log N)$ in order to distinguish N different elements.

By setting $w_i = \omega_1(\tau_{\pi^{-1}(i)})$ we obtain the input to the one-constraint splitting problem.

To our best knowledge *Bokhari* [1] provided the first algorithm for this problem with a runtime of $O(N^3 p)$. After a series of improvements *Khana et al.* proposed in [15] an $O(N \log N)$ algorithm independent of p. Plugging this algorithm into this decomposition scheme allows us to compute a solution to the one-constraint domain decomposition problem in $O(N \log N)$ time.

We finish this section by showing that $I_1(s)$ can be bounded from above. We need this lemma in the analysis of our two-constraint splitting heuristic in section 5.3.6.

Lemma 4.2. *Let W, N, p be a one-constraint splitting problem instance with solution s. Furthermore, let $w_{max} = \max W$. Then:*

$$I_1(s) \leq \frac{1}{p} + w_{max}. \tag{2}$$

Proof: A splitting s' fulfilling equation 2 can be constructed by traversing W from left to right, accumulating the w_i and setting a separator when the accumulated value is about to exeed $\frac{1}{p} + w_{max}$. By definition $I_1(s) \leq I_1(s')$ which proofs the claim.

■

5 Two-constraint domain decomposition with Space Filling Curves

In this section, we assume the presence of a second weight $\omega_2 : T \to \mathbb{R}^+$. To compute a domain decomposition, we follow the same procedure *index computation, sorting, splitting* as in the one-constraint setting, however, we obtain a different setting for the splitting operation. While the one-constraint splitting problem corresponds to a vector splitting problem, an intuitive representation of the two-constraint splitting problem is the extension of the vector W to a $(\mathbb{R}^+)^{N\times 2}$ matrix (which we continue by denoting with W) - each row corresponds to one weight - which is to be split by vertical separators.

A straightforward way to split this $N \times 2$ matrix is to do it as in the one-constraint case: find $p-1$ separators such that a convex combination of the imbalances of the two weights is minimal. We define this formally:

Problem 5.1. *Let $N > p > 0$, $\alpha \in (0,1)$,*

$$W = \begin{pmatrix} w_1^1 & \dots & w_N^1 \\ w_1^2 & \dots & w_N^2 \end{pmatrix} \in (\mathbb{R}^+)^{N\times 2} \tag{3}$$

*with for $j \in \{1,2\} : \sum_i w_i^j = 1$. A solution to the **two-constraint splitting problem** is a vector $s \in \mathbb{N}^{p-1}$ with $0 < s_1 < s_2 < \dots < s_{p-1} < N$ $(s_0 = 0, s_p = N)$ such that:*

$$\begin{aligned} I_2(s) \quad = \quad & \alpha \cdot \max\{ \sum_{j=s_{i-1}+1}^{s_i} w_j^1 \,|\, 0 < i \leq p\} + \\ & (1-\alpha) \cdot \max\{ \sum_{j=s_{i-1}+1}^{s_i} w_j^2 \,|\, 0 < i \leq p\} \end{aligned} \tag{4}$$

is minimal.

This problem definition comes along with two major issues. Firstly, from a computational point of view the two-constraint splitting problem seems much more difficult than the one-constraint splitting problem. We are only able to come up with an $O(N^5 p^2)$ algorithm (see section 5.1) to this problem which is far from being pratically applicable. We aim to design a fast method preferably in $O(N \log N)$ time. There is surely

space for improvement, future research might bring down the complexity or lead to good approximation algorithms, however, so far we can not solve this problem sufficiently efficient.

Secondly, depending on the concrete decomposition instance at hand, even though we could solve this problem exactly in appropriate time, solutions might still result in huge imbalance. Consider the following example for $p = 2$:

$$w_i^1 = \frac{2}{N^2} i \tag{5}$$

$$w_i^2 = \frac{2}{N} - \frac{2}{N^2} i. \tag{6}$$

w^1 increases linearly while w^2 decreases linearly. Setting $\alpha = \frac{1}{2}$ in equation 4, the optimal solution is splitting the two partitions at $\frac{N}{2}$ which distributes the weights among the 2 partitions in a ratio of $3 : 1$. This is unsatifactory for a load balancing scheme. This example, though highly artificial, demonstrates the weakness of this approach. It is hard to balance two weights that are oppositionally distributed with respect to the linear ordering of the underlying SFC.

In this example, perfect load balancing can, however, be established by allowing a further separator. Forming partition 1 by the intervals $[1, 0.25N]$ and $[0.75N, N]$, and partition 2 by the interval $[0.25N, 0.75N]$ leads to perfect load balancing. Allowing more separators than partitions increases the edge cut of the resulting decomposition (one interval corresponds roughly to one connected component), allows, however, to obtain better load balancing. Based on this intuition we define what we call the *parametrized two-constraint splitting problem*:

Problem 5.2. *Let $N > q > p > 0$, $\alpha \in (0, 1)$,*

$$W = \begin{pmatrix} w_1^1 & \dots & w_N^1 \\ w_1^2 & \dots & w_N^2 \end{pmatrix} \in (\mathbb{R}^+)^{N \times 2} \tag{7}$$

*with for $j \in \{1, 2\} : \sum_i w_i^j = 1$. A solution to the **parametrized two-constraint splitting problem** is a vector $s \in \mathbb{N}^{q-1}$ with $0 < s_1 < s_2 < \cdots < s_{q-1} < N$ $(s_0 = 0, s_q = N)$ and a mapping $m : [1, q] \to [1, p]$ such that:*

$$\begin{aligned} I_2(s) = {} & \alpha \cdot \max\{ \sum_{t \in m^{-1}(i)} \sum_{j=s_{t-1}+1}^{s_t} w_j^1 \,|\, 0 < i \leq p\} + \\ & (1-\alpha) \cdot \max\{ \sum_{t \in m^{-1}(i)} \sum_{j=s_{t-1}+1}^{s_t} w_j^2 \,|\, 0 < i \leq p\} \end{aligned} \tag{8}$$

is minimal.

We will show in section 5.2 that this problem is NP-hard in parameter q and we do not try to solve this problem directly. The main contribution of this paper is a two-constraint decomposition heuristic that can be seen as a heuristic for the parametrized two-constraint splitting problem with parameter $q = \sigma p$ for some integer $\sigma > 1$. We discuss this method in section 5.3 in detail.

5.1 An algorithm for the two-constraint splitting problem

In this subsection we provide an algorithm for the two-constraint splitting problem. Equation 4 is a weighted sum of the maxima of two sets. A set comprises p sums, an ith sum corresponds to the weight that goes to partition P_i. The partition separators s_{i-1}, s_i uniquely define partition P_i, and hence they equally define the values of the corresponding sums of P_i.

Say we know that there is an optimal solution minimizing I_2 with partitions P_{m_1}, P_{m_2} $(1 \leq m_1, m_2 \leq p)$ posing the two maxima of each term in equation 4 (our algorithm iterates over all possible maxima

memberships). To uniquely define P_{m_1} and P_{m_2} we have to set up at most 4 partition separators (our algorithm will iterate over all possible setups). Now, given these partition separators defining unambiguously P_{m_1} and P_{m_2}, we have to check whether the remaining partition separators can be set up such that P_{m_1}, P_{m_2} remain the partitions that pose the maxima of equation 4. We will show that this constructive check can be done in $O(N)$ time.

Lemma 5.3. *Let $1 \leq m_1, m_2 \leq p$. W.l.o.g. we assume that $m_1 \leq m_2$. Furthermore, if $m_1 \neq m_2$ let $s \in \mathbb{N}^{p-1}$ with $m_1 - 1 \leq s_{m_1-1} < s_{m_1} \leq s_{m_2-1} < s_{m_2} \leq N - (p - m_2)$, otherwise let $s \in \mathbb{N}^{p-1}$ with $m_1 - 1 \leq s_{m_1-1} < s_{m_1} < N - (p - m_1)$. Then there is an $O(N)$ algorithm that checks in a constructive way whether the remaining partition separators s_i, $i \notin \{m_1 - 1, m_1, m_2 - 1, m_2\}$ can be set up such that the two maxima of equation 4 are given by P_{m_1} and P_{m_2}.*

Proof: Firstly, we compute $M_1 = \sum_{i=s_{m_1-1}}^{s_{m_1}} w_i^1$ and $M_2 = \sum_{i=s_{m_2-1}}^{s_{m_2}} w_i^2$ (clearly in $O(N)$). We split the remaining indices into three sets $\{s_1, \ldots, s_{m_1-2}\}$, $\{s_{m_1+1}, \ldots, s_{m_2-2}\}$, and $\{s_{m_2+1}, \ldots, s_{p-1}\}$ [2]. For each of these sets we check whether the partition separators can be set such that the weights of the induced partitions do not exceed M_1 and M_2. This can be done by a traversal of W from left to right: we accumulate the w_i^1 in acc_1 and the w_i^2 in acc_2, and we set a partition separator if either acc_1 is about to exceed M_1, or if acc_2 is about to exceed M_2. Once a partition separator is set, we reset the accumulators to 0 and we restart accumulating the weights. If this procedure succeeds for all sets, we return s, otherwise we return $false$.

■

Theorem 5.4. *There is an algorithm that computes an optimal splitting s' of W into p parts in $O(N^5 p^2)$.*

Proof:

Algorithm 1 Algorithm for the two-coinstraint splitting problem

```
1:  s' = ⊥, m = ∞
2:  for all (m_1, m_2) ∈ [1, p]^2 do
3:    for all interval boundary settings s defining intervals m_1, m_2 do
4:      if extend s to a well-formed solution according to lemma 5.3 then
5:        if I_2(s) < m' then
6:          m' = I_2(s), s' = s
7:        end if
8:      end if
9:    end for
10: end for
11: return s'
```

The correctness of algorithm 2 follows by construction. The outer loop is executed p^2 times, the inner loop is executed $O(N^4)$ times, since there are $O(N^4)$ possibilities to define 4 partition separators. The body of the loop, that is the extension of s according to lemma 5.3, and the computation of $I_2(s)$ requires $O(N)$ time, in total $O(N^5 p^2)$.

■

Remarks The main difficulty in designing an algorithm for this problem is that I_2 exhibits a multitude of local minima while this is not the case for I_1 in the one-contraint splitting problem. For this reason do not know how to adapt existing one-constraint splitting algorithms (see [19] for an overview, but as well [15]), to the two constraint setting.

Furthermore, we want to point out that our algorithm can be straightforwardly extended to tackle c-constraint splitting problems ($c \geq 2$) with a runtime of $O(p^c N^{2c+1} c)$.

[2] If the right delimiter of a set is larger than the its left then we consider the set to be empty.

5.2 The parametrized two-constraint splitting problem is NP-hard in q

We provide a reduction from the NP-complete problem PARTITION, see for instance [13].

Problem 5.5. *The partitioning problem (PARTITION) consists of, given a set of positive integers $M = \{m_1, \dots\}$, $|M| < \infty$, splitting M into two disjoint subsets $M_1, M \backslash M_1$ such that the discrepancy $|\sum_{m \in M_1} m - \sum_{m \in M \backslash M_1} m|$ is minimal.*

Theorem 5.6. *The parametrized two-constraint splitting problem is NP-hard.*

Proof: Given an instance of PARTITION (that is the set M), we show how to solve it with the help of an algorithm A for the parametrized two-constraint splitting problem. set $N = q = |M|, p = 2$,

$$W = \begin{pmatrix} m_1 & \dots & m_{|M|} \\ m_1 & \dots & m_{|M|} \end{pmatrix}$$

as the input to algorithm A. The two obtained partitions state as well the solution to the PARTITION problem.

■

Remarks The not here defined parametrized one-constraint splitting problem is already NP-hard. Furthermore, note that the reduction creates an instance s.t. $N = q$ showing the hardness for this specific setting. For demonstrating that the problem is hard for arbitrary N, there is a quite technical construction to split up the m_i appropriately to obtain in total N numbers such that a solution to this parametrized two-constraint problem still uniquely solves the underlying PARTITION instance.

5.3 Fast heuristic for two-constraint decomposition problems

This subsection presents our fast splitting heuristic for the two-constraint splitting problem which we consider the main contribution of this paper.

We consider the $N \times 2$ matrix W with non-negative entries as in problem 5.1 with the same notation for its entries which we repeat here for the sake of clarity:

$$W = \begin{pmatrix} w_1^1 & \dots & w_N^1 \\ w_1^2 & \dots & w_N^2 \end{pmatrix} \in (\mathbb{R}^+)^{N \times 2}. \tag{9}$$

Line one corresponds to a weight ω_1, line two corresponds to a weight ω_2. When speaking in the sequel about the balancing of ω_1 (ω_2), we refer to the balancing of line 1 (2) of matrix W. This matrix is to be split into p partitions. In the following, we deal with matrix subparts denoted by W_i and W_{ij} of W that are the result of a splitting of W by vertical separators such that:

$$W = (W_1 \,|\, W_2 \,|\, \dots), \tag{10}$$
$$W_i = (W_{i1} \,|\, W_{i2} \,|\, \dots). \tag{11}$$

For $c = 1, 2$ we introduce the abbreviations

$$l_i^c = \sum_{w_j^c \in W_i} w_j^c, \tag{12}$$
$$l_{ij}^c = \sum_{w_l^c \in W_{ij}} w_l^c, \tag{13}$$
$$w_{max}^c = \max\{w_i^c\}. \tag{14}$$

Furthermore, we assume an integer $\sigma \geq 2$. The algorithm consists of two phases which we denote as the *splitting phase*, and the *reunification phase*.

5.3.1 The splitting phase

In the splitting phase, firstly, matrix W is split into σ parts $W_1, \ldots, W_\sigma$ by applying the one-constraint splitting operation as described in section 4, balancing the first line of W (hence ω_1), and it holds by lemma 4.2 that:

$$l_i^1 \leq \frac{1}{\sigma} + w_{max}^1 \quad \forall i : 1 \leq i \leq \sigma, \tag{15}$$

where w_{max}^1 denotes the largest element of the first line of W. Secondly, each of these σ matrix parts is split into p parts balancing the second line by the application of a one-constraint splitting algorithm. Again by lemma 4.2 we obtain:

$$l_{ij}^2 \leq \frac{l_i^2}{p} + w_{max}^2 \quad \forall i, j : 1 \leq j \leq p, 1 \leq i \leq \sigma, \tag{16}$$

w_{max}^2 denotes the largest element of the second line of W. The splitting phase is visualized in figure 4.

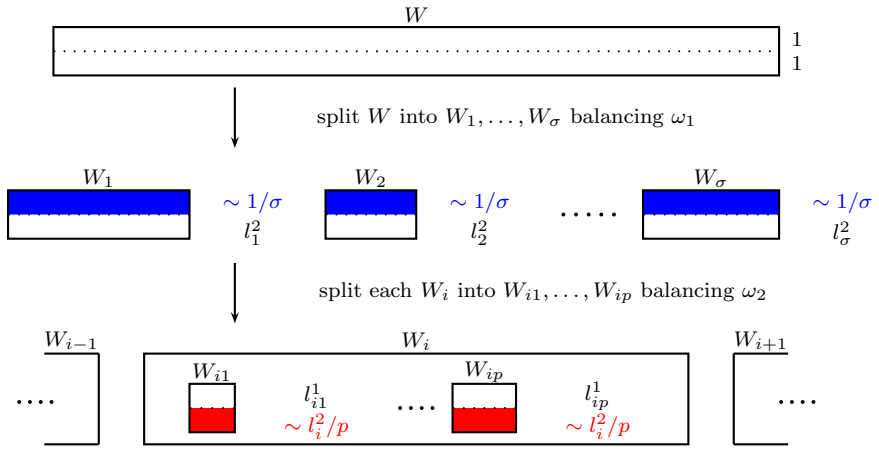

Figure 4: The splitting phase of our two-constraint algorithm. The blue and the red color indicate matrix parts consisting of the same weight.

5.3.2 The reunification phase

In the reunification phase, the partitions $P_1, \ldots, P_p$ are determined. A partition consists of σ matrix parts, whereas from each $(W_i)_{i=1,\ldots,\sigma}$ one $(W_{ij})_{j=1,\ldots,p}$ is taken. Figure 5 illustrates this procedure.

Concerning the weight balancing, note that any reunification automatically balances ω_2. Since ω_2 is balanced within each $(W_i)_{i=1,\ldots,\sigma}$, and the p parts $(W_{ij})_{j=1,\ldots,p}$ of each W_i are scattered to the p partitions, the ω_2 contribution of each W_i to the partitions is equal. For this reason, the reunification problem consists of forming partitions, such that the balancing of ω_1 is optimal.

We decouple the reunification problem from the matrix splitting problem, and discuss it independently.

Problem 5.7. *The reunification problem (REUNIFICATION) consists of, given $\sigma, p \geq 2$, the vectors $S^1, \ldots, S^\sigma \in (\mathbb{R}^+)^p$, find $Q^1, \ldots, Q^p \in [1, p]^\sigma$ with $Q_k^i = Q_k^j \Leftrightarrow i = j$ such that:*

$$\max\{\sum_{i=1}^{\sigma} S_{Q_i^k}^i \mid k = 1, \ldots p\} \to \min . \tag{17}$$

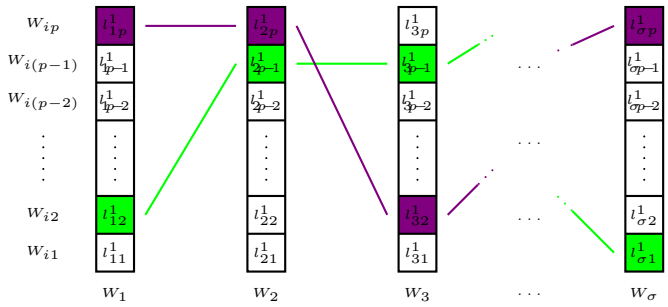

Figure 5: The reunification phase of our two-constraint algorithm. The green and the violet color represent final partitions. A final partition consists of σ matrix subparts taken from $W_1, \ldots, W_\sigma$. Each matrix subpart W_{ij} contains two weights l^1_{ij} and l^2_{ij}, however, since ω_2 is balanced automatically by the reunification, this weight is not illustrated here.

In our setting we have $S^i = (l^1_{i1}, l^1_{i2}, \ldots, l^1_{ip})$. The resulting Q_i indicate which matrix subpart W_{ij} goes to which partition P_i.

As we will show in the following subsection, REUNIFICATION is NP-hard. However, since the quality of the solution to the REUNIFICATION problem directly translates to the imbalance of ω_1 of the decomposition, a good heuristic is required. In subsection 5.3.4, we present a fast heuristic to the problem.

5.3.3 REUNIFICATION is NP-hard

Theorem 5.8. *REUNIFICATION is NP-hard.*

Proof: For showing that REUNIFICATION is NP-hard, we provide a reduction from PARTITION (problem 5.5) to REUNIFICATION. Let $M = \{m_1, \ldots, m_N\}$ be a set of N positive integers, which we aim to split into two disjoint subsets such that the discrepancy is minimal. We construct now a REUNIFICATION problem that solves a PARTITION instance. For this, we set $p = 2, \sigma = N$. Furthermore, we set $S^1 = (0, m_1), S^2 = (0, m_2), \ldots, S^\sigma = (0, m_N)$. An algorithm for REUNIFICATION provides partitions P_1, P_2 that state the solution to the PARTITION problem.

■

5.3.4 A heuristic for REUNIFICATION

The reduction of PARTITION to REUNIFICATION shows that these two problems are structurally similar. In this subsection, we propose a heuristic for REUNIFICATION that is a natural extension of the largest differencing method (LDM) of *Karmarkar and Karp* [12], the best known heuristic for PARTITION.

To solve PARTITION, LDM starts with a sorted list of N numbers. Each step, it decides that the two largest elements l_1, l_2 of this list go into different partitions. It removes l_1, l_2 from the list, computes their difference $|l_1 - l_2|$, and inserts this difference again into the list. $|l_1 - l_2|$ is the imbalance that is caused by the decision that l_1 and l_2 go into different partitions. Note, that it is not yet decided into which partition l_1 and l_2 go. The difference is reinserted into the list, since the algorithm aims to even out the imbalance. By keeping track of the procedure the two partitions can be determined bottom-up.

We construct a heuristic for REUNIFCATION that is structurally similar to LDM. The REUNIFICATION problem consists of σ vectors $S^1, \ldots, S^\sigma$. For these vectors S^i we define their *diameters* as $d(S^i) = \max S^i - \min S^i$. Analogous to the LDM, we group the sets $(S^i)_{i=1\ldots\sigma}$ in a data structure L where the ordering is with respect to their diameters ($d(L[1]) \geq d(L[2]) \geq \ldots$). We subsequently merge $L[1], L[2]$, the vectors with the largest diameter, via the merge operation shown in algorithm 2.

Algorithm 2 Merge operation

Require: S^j, S^k

$R \in (\mathbb{R}^+)^p$
sort S^j in ascending order
sort S^k in descecnding order
$\forall i : R[i] \leftarrow S^j[i] + S^k[i]$
return R

To highlight that the vectors S^j, S^k, R are realized with some data structures, brackets are used for accessing elements. $L[1], L[2]$ are removed from L, and merge$(L[1], L[2])$ is inserted to the list. While in the LDM, two numbers are substituted by their difference, here, two diameters $d(S^j), d(S^k)$ are replaced by a new diameter, and we will show that it holds that $|d(S^j) - d(S^k)| \leq d(merge(S^j, S^k)) \leq \max\{d(S^j), d(S^k)\}$. This procedure is repeated until there remains one final vector S'. The diameter of S' translates to the imbalance of the partitioning.

Algorithm 3 shows the entire algorithm.

Algorithm 3 Reunification heuristic

Require: L

while $|L| > 1$ **do**
    $R \leftarrow$ merge$(L[1], L[2])$
    L.remove$(L[1])$
    L.remove$(L[2])$
    L.insert(R)
end while

Table 1 shows an example of our reunification heuristic.

5.3.5 Runtime of the two-constraint splitting heuristic

During the splitting phase, a one-constraint splitting of W is performed into σ parts in $O(N \log N)$ time. Secondly, we split each $W_i, \dots W_\sigma$ into p parts. Their runtimes sum up to $O(N \log N)$ since the cardinalities of the W_i sum up to N. The splitting phase hence runs in $O(N \log N)$.

During the reunification phase we need $O(\sigma p + \sigma \log \sigma)$ time to set up the data structure L (diameter computation and sorting). We perform $\sigma - 1$ merge operations. The runtime of the *merge* operation is dominated by the sort operations, hence it runs in $O(p \log p)$. We account a factor $O(\log \sigma)$ for the insert operation of the data structure L. Hence, in total we obtain a runtime of $O(\sigma(p \log p + \log \sigma))$ for the reunification phase, and a runtime of the two-constraint splitting heuristic of $O(\sigma(p \log p + \log \sigma) + N \log N)$ which is $O(N \log N)$ under the reasonable assumption that $\sigma p = O(N)$.

5.3.6 Quality of the two-constraint splitting method

Firstly, ω_2, that is the second line of matrix W, is balanced automatically by the method. In the worst-case setting the imbalances of the σ one-constraint splittings accumulate in one final partition, hence the load of the partition is limited as follows:

$$l(P_i, \omega_2) \quad \leq \quad \frac{1}{p} + \sigma w_{max}^2 \, \forall i. \tag{18}$$

This leads to the imbalance:

$$\mu(\omega_2) \leq 1 + p\sigma w_{max}^2, \tag{19}$$

L	**Operation**
$\underbrace{\begin{pmatrix}2\\8\\12\end{pmatrix}}_{10}, \underbrace{\begin{pmatrix}9\\0\\7\end{pmatrix}}_{9}, \underbrace{\begin{pmatrix}11\\7\\13\end{pmatrix}}_{6}, \underbrace{\begin{pmatrix}1\\5\\6\end{pmatrix}}_{5}$	$\text{merge}(\begin{pmatrix}2\\8\\12\end{pmatrix}, \begin{pmatrix}9\\0\\7\end{pmatrix}) = \begin{pmatrix}12\\15\\11\end{pmatrix}$
$\underbrace{\begin{pmatrix}11\\7\\13\end{pmatrix}}_{6}, \underbrace{\begin{pmatrix}1\\5\\6\end{pmatrix}}_{5}, \underbrace{\begin{pmatrix}12\\15\\11\end{pmatrix}}_{4}$	$\text{merge}(\begin{pmatrix}11\\7\\13\end{pmatrix}, \begin{pmatrix}1\\5\\6\end{pmatrix}) = \begin{pmatrix}14\\16\\13\end{pmatrix}$
$\underbrace{\begin{pmatrix}12\\15\\11\end{pmatrix}}_{4}, \underbrace{\begin{pmatrix}14\\16\\13\end{pmatrix}}_{3}$	$\text{merge}(\begin{pmatrix}12\\15\\11\end{pmatrix}, \begin{pmatrix}14\\16\\13\end{pmatrix}) = \begin{pmatrix}28\\26\\27\end{pmatrix}$
$\underbrace{\begin{pmatrix}28\\26\\27\end{pmatrix}}_{2}$	Obtain the partitions: $\begin{pmatrix}28\\26\\27\end{pmatrix} = \begin{pmatrix}8+7+7+6\\12+0+13+1\\2+9+11+5\end{pmatrix}.$

Table 1: Reunification heuristic for an example with $\sigma = 4$ and $p = 3$. The numbers below the W_i denote their diameters. The heuristic finds a reunification with a discrepancy of 2. In this example, a best solution with an imbalance of 0 exists.

In order to analyze the balancing of ω_1, we require the following lemma:

Lemma 5.9. *Let $s = (s_1, s_2, \ldots, s_p) \in (\mathbb{R}^+)^p$ with $s_1 \geq s_2 \geq \cdots \geq s_p$, $t = (t_1, t_2, \ldots, t_p) \in (\mathbb{R}^+)^p$ with $t_1 \leq t_2 \leq \cdots \leq t_p$. W.l.o.g. we assume that $|s_1 - s_p| \geq |t_p - t_1|$. Then:*

$$(s_1 - s_p) - (t_p - t_1) \leq \max\{s_i + t_i \,|i\} - \min\{s_i + t_i \,|i\} \leq |s_1 - s_p|. \tag{20}$$

Proof: Let $1 \leq i \leq p$. We prove the first "$\leq$":

$$\begin{aligned} (s_1 - s_p) - (t_p - t_1) &= s_1 + t_1 - (s_p + t_p) && (21)\\ &\leq \max\{s_i + t_i \,|i\} - \min\{s_i + t_i \,|i\}. && (22) \end{aligned}$$

We prove the second "$\leq$": Let i_{max}, i_{min} be the indices responsible for the maxima and minima in equation 20.

$$\begin{aligned} \max\{s_i + t_i \,|i\} - \min\{s_i + t_i \,|i\} &= s_{i_{max}} + t_{i_{max}} - s_{i_{min}} - t_{i_{min}} && (23)\\ &= \underbrace{s_{i_{max}} - s_{i_{min}}}_{A} + \underbrace{t_{i_{max}} - t_{i_{min}}}_{B} && (24) \end{aligned}$$

$$\begin{aligned} \text{case 1: } i_{max} \leq i_{min} \quad &\Rightarrow \quad A \leq s_1 - s_p, B \leq 0 \Rightarrow && (25)\\ &\quad A + B \leq s_1 - s_p \quad \checkmark \\ \text{case 2: } i_{max} > i_{min} \quad &\Rightarrow \quad A \leq 0, B \leq t_p - t_1 \leq s_1 - s_p \Rightarrow && (26)\\ &\quad A + B \leq s_1 - s_p \quad \checkmark \end{aligned}$$

■

Corollary 5.10.

$$|d(S_1) - d(S_2)| \leq d(merge(S_1, S_2)) \leq \max(d(S_1), d(S_2)). \tag{27}$$

For ω_1 we can state the following worst-case scenario: due to equation 15, we have $l_{ij}^1 \leq l_i^1 \leq 1/\sigma + (\omega_1)_{max}$. Due to the upper bound of corollary 5.10, it holds for the maximal load difference of two partitions that $|l(P_i,\omega_1) - l(P_j,\omega_1)| \leq 1/\sigma + (\omega_1)_{max} \quad \forall i,j$. We obtain an upper bound by assuming that the partition with the maximal load $P_{i_{max}}$ has the mentioned load difference to all other partitions. In that case we obtain:

$$
\begin{aligned}
1 &= \sum_i l(P_i, w_1) && (28)\\
&= l(P_{i_{max}}, w_1) + \sum_{i \neq i_{max}} l(P_i, w_1) && (29)\\
&\geq l(P_{i_{max}}, w_1) + (p-1)(l(P_{i_{max}}, w_1) - 1/\sigma - (\omega_1)_{max}) && (30)\\
&\Rightarrow && (31)\\
\mu(\omega_1) &\leq 1 + \frac{p-1}{\sigma} + (p-1)(\omega_1)_{max}. && (32)
\end{aligned}
$$

This worst-case bound is quite unsatisfactory since it requires large σ values to guarantee an acceptable imbalance of ω_1. However, we experienced that for most practical problems the imbalance decreases expeditiously with the increase of σ. We verify this in the following test case.

6 Practical issues and comparison with Metis

To illustrate the capabilities of our two-constraint method, in this section we provide a two-constraint decomposition problem that arises in a parallel PIC simulation. The underlying mesh consists of $N = 3245588$ unstructured tetrahedra, and it is visualized in figure 6:

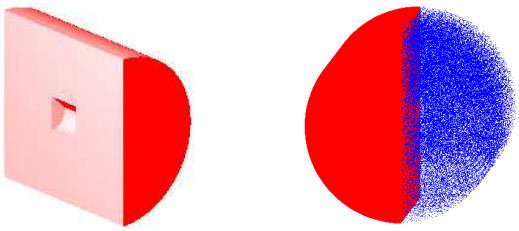

Figure 6: Left: cylindric mesh of 3245588 unstructured tetrahedra. The mesh is a cylinder with a small cylinder in its center cut out. For the visualization, the left half is cut away. Right: particles inside the mesh. For the visualization, the tetrahedra of the right half of the geometry are not displayed.

The two weights are defined as follows. ω_1 corresponds to the load induced for a first computational phase, ω_2 corresponds to the load induced for a second computational phase. We set $\omega_1(\tau_i) = \alpha$ if the x-coordinate of the center of gravity of an tetrahedron is smaller than 0.5, otherwise we set $\omega_1(\tau_i) = 5 \cdot \alpha$, for α correctly chosen such that $\sum_{\tau_i} \omega_1(\tau_i) = 1$. Hence, we assume that the load for the first computational phase induced by a tetrahedron that is located in the right half of the geometry is 5 times the load of a tetrahedron that is located in the left half.

The particles determine the load for the second computational phase. $\omega_2(\tau_i)$ is chosen to be the number of particles that reside in τ_i divided by the total number of particles for normalization. For this test case we put between 1 to 50 particles into a tetrahedron while this number grows linearly with y coordinate of the center of gravity of a tetrahedron. This choice is somewhat arbitrary, we try to simulate a realistic setting.

Before considering the decomposition quality (edge-cut and balancing), we discuss the runtime of the method in comparison to Metis. Even though we claim that the benefit of our method is clearly the runtime, we point out that this issue has to be carefully considered.

6.1 Runtime

When applying graph partitioning for decomposing a mesh, prior to the decomposition routine the dual graph of the mesh has to be computed. Subsequently, the decomposition routine computes a decomposition. If during a numerical simulation the weights change and a certain imbalance occurs, a redecomposition can be computed based on the prior computed dual graph. Hence, in this setting, the dual graph of a mesh just has to be computed once and for all.

When decomposing with SFCs, the setting is similar. For each mesh element, the center of gravity and a corresponding index with respect to some SFC (here we applied the Hilbert curve) has to be computed. The list of SFC indices has to be sorted, and, based on this sorting, a splitting algorithm states the domain decomposition. Here, it holds the same as for a decomposition by graph partitioning. When redecomposing, the sorted list of indices of the mesh elements can be reused, and only the splitting algorithm has to be rerun.

Computing the dual graph, as well as the computation of the centers of gravity, the computation of SFC indices, and the reordering are *precomputable* steps since they do not depend on the number of partitions into which we aim to decompose.

Table 2 shows execution times of the different phases of the decomposition process of Metis and our two-constraint algorithm for the cylindric geometry of figure 6 into 128 parts.

Metis		**SFCs**	
dual graph:	2.35 sec	centers of gravity:	1.3 sec
		SFC indices:	2.7 sec
		sorting:	0.66 sec
decomposition routine:	12.49 sec	two-constraint splitting:	0.1 sec

Table 2: Execution times of the neccessary steps when decomposition the mesh of figure 6 into 128 parts with Metis and our two-constraint SFC approach. The decompositions are done on a standard laptop.

In general we observed that the two-constraint splitting algorithm performs about 100-200 times faster than the two-constraint decomposition routine of Metis. Hence, especially for applications with redecompositions during the execution, the two-constraint decomposition algorithm is an interesting candidate.

6.2 Quality

We decompose the two-constraint problem into 2 to 512 parts. When applying Metis, we choose an imbalance bound of 1.03 for both weights. Our two-constraint method depends on the parameter σ, however, it is not evident how to select σ to obtain a meaningful comparison to Metis. We decided to experimentally determine the minimal σ such that ω_1 is also balanced within the bound of 1.03. The results are displayed in figure 7.

Figure 7 shows that to obtain similar balancings, in this example our SFC based approach produces edge-cuts between 2 and up to 3.5 times as large as those produced by Metis. This ratio depends on the number of partitions, and, as the right figure shows, with the increase of the number of partitions, σ as well has to be increased to obtain the same balancing. Equation 32 provides a faint hint for this behavior.

In figure 8, we display the results of a decomposition into 512 parts with varying σ values.

The bahavior of the imbalance of ω_1 (that is $\mu(\omega_1)$) with respect to σ is representative for the method. Increasing small σ values leads to a fast decrease of $\mu(\omega_1)$ while the increase of large σ values decreases the imbalance only slightly. In addition, the evolution of $\mu(\omega_1)$ with respect to σ is superposed with a fluctuation. This results from the fact that a different choice of σ poses differently natured reunification problems. Even though the initial diameters of the sets in the reunification phase decrease with the increase of σ, the problems might be harder to solve (or simply, there might be no better reunifications), and the final imbalance might be larger than those achieved with a smaller σ. However, if σ is increased sufficiently, $\mu(\omega_1)$ can be kept as small as it is desired in charge of an increased edge-cut.

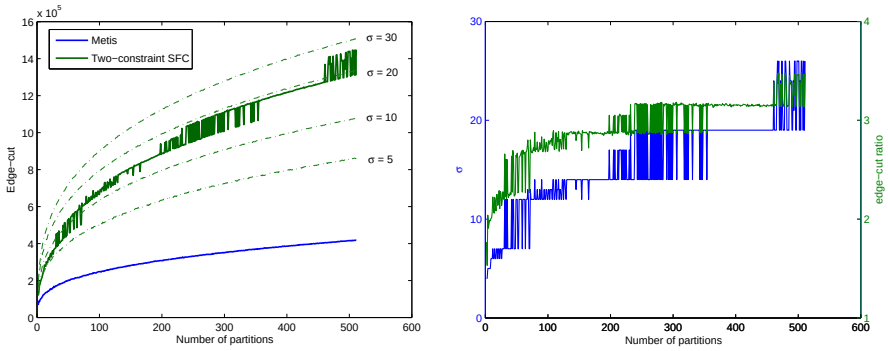

Figure 7: Left: the edge-cut of the decompositions provided by Metis and our two-constraint algorithm with a weight balancing constraint of 1.03. The dotted lines are the results of our two-constraint algorithm for certain fixed σ values. Right: the σ values that are required for obtaining a balancing of 1.03. The green figure shows the edge-cut ratio of our method to Metis.

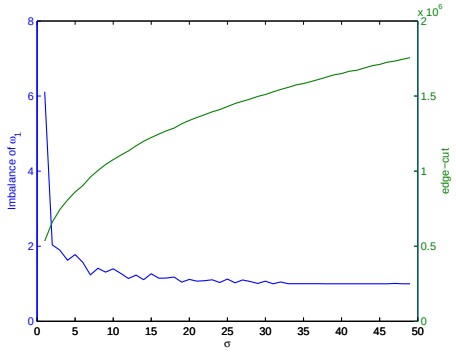

Figure 8: Influence of σ on the edge-cut and the balancing of ω_1. Here, we decompose into 512 parts.

6.3 Further remarks

Further experiments showed that to obtain similar balancings as Metis does the method produces edge-cuts between 1.5 up to 7 times worse than those produced by Metis, while we achieved these very bad ratios only by constructing very artificial test scenarios. So far, we do not want to claim that the method works for any two-constraint decomposition problem, however, we point out that there are problems where the method applies well.

Furthermore, in general 2-dimensional problems result in a higher edge-cut ratio than 3-dimensional problems which is a consequence of the fact that the surface of a SFC induced partition grows in $O(\sqrt[d]{\text{volume of partition}})$ (compare [26]).

7 Conclusion

In this paper, we presented a new two-constraint domain decomposition method based on SFCs. We provided a test case to experimentally demonstrate the capabilities of the method, and we pointed out that the method is fast by a reasonable loss of quality in terms of partition surface sizes with respect to two-constraint graph partitioning as implemented in the Metis software package.

The method requires to solve the REUNIFICATION problem. We showed that this problem is NP-hard, and we presented a heuristic algorithm for this problem that can be seen as a generalization of the LDM of Karmarkar and Karp for the interger partitioning problem.

Furthermore, many variants of the present two-constraint algorithm are possible, and one may fine tune the method to certain problem cases. Since the underlying SFC defines the sorting of the list of tetrahedra onto which the splitting algorithm acts, the applied SFC plays an important role in posing the splitting problem. The application of different curves may lead to different results. We also pursued the idea of constructing *problem-specific* versions of the Hilbert curve that take the defined weights into account such that the splitting algorithm performs better than based on the regular Hilbert curve. First results were promising (we could keep σ small for good balancings), however our computation scheme for these problem-specific curves is quite expensive and a full decomposition algorithm is very costly, and still produces higher edge-cuts than graph partitioners.

Moreover, we considered the subset of problems that exhibit a long series of tetrahedra $(\tau_i)_{j \le i \le k}$ in the list of tetrahedra with a weight $\omega_1(\tau_i) = 0$. Here, the decomposition problem can be split into a two-constraint problem (the τ_i with $i < j$ and $k < i$) and a one-constraint problem (τ_i with $j \le i \le k$). This setting allows, firstly, to solve the two-constraint problem, and, secondly, to solve the one-constraint problem under consideration of the resulting imbalance of the two-constraint splitting. We achieved satisfying results with this method.

We state some directions for further research:

- identification of areas of application for the method,
- efficient (approximation?) algorithms for the two-constraint problem,
- integrating randomization into the two-constraint heuristic as a protection against hard instances. An interesting approach is to perform a first splitting in the splitting phase that does not necessarily balance perfectly ω_1. This splitting could be such that the resulting reunification problem instances have better solutions.

References

[1] S. H. Bokhari. Partitioning problems in parallel, pipeline, and distributed computing. *IEEE Trans. Comput.*, 37(1):48–57, 1988.

[2] A.R. Butz. Alternative algorithm for Hilbert's space-filling curve. *IEEE Trans. Comput.*, 20(4):424–426, 1971.

[3] E. A. Carmona and L. J. Chandler. On parallel PIC versatility and the structure of parallel PIC approaches. *Concurrency - Practice and Experience*, 9(12):1377–1405, 1997.

[4] Bogdan S. Chlebus and Ludwik Czaja, editors. *Fundamentals of Computation Theory, 11th International Symposium, FCT '97, Kraków, Poland, September 1-3, 1997, Proceedings*, volume 1279 of *Lecture Notes in Computer Science*. Springer, 1997.

[5] C. Gotsman and M. Lindenbaum. The metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5(5):794–797, 1996.

[6] Michael Griebel and Gerhard Zumbusch. Parallel multigrid in an adaptive pde solver based on hashing and space-filling curves. *Parallel Comput.*, 25(7):827–843, 1999.

[7] R. Hatzky. Domain cloning for a particle-in-cell (pic) code on a cluster of symmetric-multiprocessor (smp) computers. *Parallel Comput.*, 32(4):325–330, 2006.

[8] B. Hendrickson and A. Pothen. Combinatorial scientific computing: The enabling power of discrete algorithms in computational science. In Michel J. Daydé, José M. L. M. Palma, Alvaro L. G. A. Coutinho, Esther Pacitti, and Joao Correia Lopes, editors, *VECPAR*, volume 4395 of *Lecture Notes in Computer Science*, pages 260–280. Springer, 2006.

[9] R. W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles*. 1981.

[10] K.-C. Tseng J.-S. Wu. Parallel dsmc method using dynamic domain decomposition. *International Journal for Numerical Methods in Engineering*, 63:37–76, 2005.

[11] G. Jin and J. Mellor-Crummey. Sfcgen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Trans. Math. Softw.*, 31(1):120–148, 2005.

[12] N. Karmarkar and R.M. Karp. The differencing method of set partitioning. Technical report, University of California, Berkeley, 1982.

[13] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[14] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical report, University of Minnesota, Department of Computer Science, 1998. `http://glaros.dtc.umn.edu/gkhome/node/90`.

[15] Sanjeev Khanna, S. Muthukrishnan, and Steven Skiena. Efficient array partitioning. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 616–626, London, UK, 1997. Springer-Verlag.

[16] J. Lawder. Calculation of mappings between one and n-dimensional values using the Hilbert space-filling curve, *Technical Report* jl1/00, 2000.

[17] S. Lanteri M. Bernacki, L. Fezoui and S. Piperno. Parallel discontinuous Galerkin unstructured mesh solvers for the calculation of three-dimensional wave propagation problems. *Applied Mathematical Modelling*, 30:744–763, 2006.

[18] Hiroshi Nakashima, Yohei Miyake, Hideyuki Usui, and Yoshiharu Omura. Ohhelp: a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations. In *ICS*, pages 90–99, 2009.

[19] Ali Pinar and Cevdet Aykanat. Fast optimal load balancing algorithms for 1d partitioning. *J. Parallel Distrib. Comput.*, 64(8):974–996, 2004.

[20] Chaco Project. http://www.sandia.gov/~bahendr/chaco.html.

[21] Jostle Project. http://staffweb.cms.gre.ac.uk/~wc06/jostle.

[22] Metis Project. http://glaros.dtc.umn.edu/gkhome/views/metis.

[23] Scotch Project. http://www.labri.fr/perso/pelegrin/scotch.

[24] S. Schamberger and J.-M. Wierum. Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In *PaCT*, pages 165–179, 2003.

[25] J.-S. Wu and K.-C. Tseng. Parallel DSMC method using dynamic domain decomposition. *International Journal for numerical methods in Engineering*, pages 37–76, 2005.

[26] Gerhard Zumbusch. On the quality of space-filling curve induced partitions. *Z. Angew. Math. Mech*, 81:25–28, 2000.