

Adaptive hierarchical subtensor partitioning for tensor compression

Virginie Ehrlacher, Laura Grigori, Damiano Lombardi, Hao Song

▶ **To cite this version:**

Virginie Ehrlacher, Laura Grigori, Damiano Lombardi, Hao Song. Adaptive hierarchical subtensor partitioning for tensor compression. SIAM Journal on Scientific Computing, 2021, 10.1137/19M128689X . hal-02284456

HAL Id: hal-02284456
https://inria.hal.science/hal-02284456

Submitted on 11 Sep 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ADAPTIVE HIERARCHICAL SUBTENSOR PARTITIONING FOR TENSOR COMPRESSION

VIRGINIE EHRLACHER *, LAURA GRIGORI†, DAMIANO LOMBARDI‡, AND HAO SONG §

Abstract. In this work a numerical method is proposed to compress a tensor by constructing a piece-wise tensor approximation. This is defined by partitioning a tensor into sub-tensors and by computing a low-rank tensor approximation (in a given format) in each sub-tensor. Neither the partition nor the ranks are fixed *a priori*, but, instead, are obtained in order to fulfill a prescribed accuracy and optimize, to some extent, the storage. The different steps of the method are detailed and some numerical experiments are proposed to assess its performances.

Key words. Tensors, Compression, CP and Tucker formats, HOSVD.

AMS subject classifications. 65F99, 65D15

1. Introduction. Tensor formats [12] have proved to be an effective and versatile tool to approximate high-dimensional functions or high-order tensors. A comprehensive overview is found in [14, 15, 10, 9]. Tensors can be used not only to approximate a given datum, but also to efficiently compute the solution of high dimensional problems [1, 2, 18, 16], specified by systems of equations and data. The method which is proposed in the present work is rather general, but was motivated by the computation of the solution of equations arising in Kinetic theory. In [7] the solution of the Vlasov-Poisson system was approximated by means of adaptive tensors. The solution of this system of partial differential equations at time $t > 0$ reads as a function for $x \in \mathbb{R}^d$ and $v \in \mathbb{R}^d$ where $d = 1, 2, 3$ denotes the space dimension. At each time $t > 0$, the function $f(t, x, v)$ was approximated by a low-rank function as follows

$$f(t, x, v) \approx \sum_{k=1}^{n_t} r_k(t, x) s_k(v, t),$$

where the rank n_t was adapted in order to control through the time evolution the error between the true function and its approximation. In several test cases, it was observed in [7] that the rank of the approximation of $f(t, x, v)$ grows linearly with the time t, which unfortunately makes the use of traditional tensor formats unfit for long time simulations. However, it was observed that the function $f(t, x, v)$ can be nevertheless very well approximated by low-rank approximations in *some large* regions of the phase space, and has to be approximated by full-order tensors in *some small* regions.

Henceforth, a parsimonious representation of the solution could be obtained by partitioning the domain in sub-regions and computing a piece-wise tensor approximation. In the present work, we start by considering the approximation of a given function (a compression problem), by taking care that the sub-regions and the rank of the tensor approximations are not fixed *a priori*. Instead, they are adapted automatically according to a prescribed accuracy. Similar ideas can be found in hierarchical matrices [3, 11]. In this work, we generalise this idea to tensors which *may not* be

*CERMICS, Ecole des Ponts Paristech, France (virginie.ehrlacher@enpc.fr)
†ALPINES, Inria Paris, France (laura.grigori@inria.fr)
‡COMMEDIA, Inria Paris, France (damiano.lombardi@inria.fr)
§The work of this author was performed as a member of ALPINES, Inria Paris

the discretisation of asymptotically smooth functions. The idea of hierarchical matrices was adapted to tensors in [13] in the context of the discretisation of Boltzmann equations. The method proposed hereafter is somehow similar in the spirit, but it is featured by automatic adaptation on the basis of an error criterion and storage optimisation. An adaptation principle along fibers for Tensor Train format is proposed in [8], in which the subdomains are splitted, with a top-down approach if an approximation criterion along each fiber is not satisfied. There are two main differences with respect to the strategy proposed in this work: we propose adaptation by splitting in sub-regions (and not along fibers), by using a bottom-up approach.

This work investigates an adaptive compression method for tensors based on local High-Order Singular Value Decomposition (HOSVD). Let us mention already here that we do not aim at approximating tensors of very high order, but only tensors of moderate order. Two main contributions are presented. The first one is a greedy method to automatically distribute the approximation error in the partitions, in which local HOSVD are computed. To the authors knowledge, no algorithmic procedure has been proposed so far to perform such a task. The second one is a merging strategy aimed at optimising the storage by fusing together sub-regions in which a low-rank (in HOSVD sense) decomposition performed on the union would be beneficial in terms of storage. The outcome of the method, that we call Hierarchical Partitioning Format (HPF) is a non-uniform adapted piece-wise tensor approximation, that guarantees a prescribed accuracy and provides a significant memory compression. This work can be seen as a first step to solve high-dimensional Partial Differential Equations.

The structure of the work is as follows: in Section 2 the notation and some elements of the theoretical analysis for continuous tensors are presented. The method proposed in the present work consists of two steps: the first one (a greedy strategy to approximate sub-tensors) is presented in Section 3, the second one, an adaptive merge to optimise the storage, is presented in Section 4. Then, some numerical experiments are detailed in Section 5.

Finally, we present the performance of our algorithm in terms of compression rates on several numerical tests, among which the compression of the solution of the Vlasov-Poisson system in a double-stream instability test case.

2. Partitioning for tensors: elements of theoretical analysis. The aim of this section is to motivate the interest of partitioning a tensor into subtensors with a view to represent them in an even sparser way with low-rank approximations.

To illustrate our point, we first consider in this section *continuous* tensors. Let $d \in \mathbb{N}^*$, $1 \leq q \leq +\infty$ and $m_1, \cdots, m_d \in \mathbb{N}^*$ and let $\Omega_1, \cdots, \Omega_d$ be open subsets of $\mathbb{R}^{m_1}, \cdots, \mathbb{R}^{m_d}$ respectively. We denote $\mathbf{\Omega} := \Omega_1 \times \cdots \times \Omega_d$.

A tensor $\mathcal{F}$ of order d defined on $\mathbf{\Omega}$ is a function $\mathcal{F} \in L^q(\mathbf{\Omega})$. The tensor $\mathcal{F}$ is said to be of canonical format with rank $R \in \mathbb{N}$ if

$$\mathcal{F}(x_1, \cdots, x_d) = \sum_{r=1}^{R} F_1^r(x_1) \cdots F_d^r(x_d), \quad \text{for} (x_1, \cdots, x_d) \in \mathbf{\Omega},$$

where for all $1 \leq j \leq d$ and $1 \leq r \leq R$, $F_j^r \in L^q(\Omega_j)$.

A domain partition $\{\mathbf{\Omega}^k\}_{1 \leq k \leq K}$ of $\mathbf{\Omega}$ is said to be *admissible* if it satisfies the following properties:

- for all $1 \leq k \leq K$, there exists $\Omega_1^k \subset \Omega_1, \cdots, \Omega_d^k \subset \Omega_d$ open subsets such that $\mathbf{\Omega}^k := \Omega_1^k \times \cdots \times \Omega_d^k$;
- for all $1 \leq k \neq k \leq K$, $\mathbf{\Omega}^k \cap \mathbf{\Omega}^l = \emptyset$;

- $\overline{\mathbf{\Omega}} = \bigcup_{k=1}^{K} \overline{\mathbf{\Omega}^k}$.

A particular case of admissible domain partition of $\mathbf{\Omega}$ can be for instance constructed as follows: for all $1 \leq j \leq d$, let $K_j \in \mathbb{N}^*$ and let us consider a collection of subsets $P_j := \left(\Omega_j^{k_j}\right)_{1 \leq k_j \leq K_j}$ of Ω_j such that

- $\overline{\Omega_j} := \bigcup_{1 \leq k_j \leq K_j} \overline{\Omega_j^{k_j}}$;
- $\Omega_j^{k_j} \cap \Omega_j^{l_j} = \emptyset$ for all $1 \leq k_j \neq l_j \leq K_j$.

Let us denote

$$\boldsymbol{P} := \left(\times_{j=1}^{d} \Omega_j^{k_j}\right)_{1 \leq k_1 \leq K_1, \cdots, 1 \leq k_d \leq K_d}.$$

The partition $\boldsymbol{P}$ then defines an admissible domain partition of $\mathbf{\Omega}$, and will be called hereafter the *tensorized domain partition* associated to the collection of domain partitions $(P_j)_{1 \leq j \leq d}$.

The heuristic of the approach proposed in this article is the following: consider a tensor $\mathcal{F} \in L^q(\mathbf{\Omega})$ which is a sufficiently regular function of $(x_1, \cdots, x_d) \in \mathbf{\Omega}$. It is of course not true, in general, that this function can be represented in a parsimonious way in a given tensor format (by exploiting separation of variable). However, under appropriate assumptions, it can be proved that there exists an admissible domain partition $\{\mathbf{\Omega}^k\}_{1 \leq k \leq K}$ of $\mathbf{\Omega}$ such that all the restrictions $\mathcal{F}^k := \mathcal{F}|_{\mathbf{\Omega}^k}$ can be represented in some tensor formats with low ranks.

The following result aims at making the above heuristics on the tensor approximation of functions precise, by providing a sufficient condition on which the above statement is true.

We introduce the following definition.

DEFINITION 2.1. *A tensor $\mathcal{F}$ defined on $\mathbf{\Omega}$ is said to be of Canonical Partitioning Format (CPF) if there exists an admissible partition $\{\mathbf{\Omega}^k\}_{1 \leq k \leq K}$ of $\mathbf{\Omega}$ such that for all $1 \leq k \leq K$, the subtensor $\mathcal{F}^k$ of $\mathcal{F}$ associated to $\mathbf{\Omega}^k$ is of canonical format on $\mathbf{\Omega}^k$. The tensor $\mathcal{F}$ is said to be of Canonical Partitioning Format (CPF) with rank $R \in \mathbb{N}^*$ if there exists an admissible domain partition $\boldsymbol{P} := \{\mathbf{\Omega}^k\}_{1 \leq k \leq K}$ of $\mathbf{\Omega}$ such that for all $1 \leq k \leq K$, the subtensor $\mathcal{F}^k$ of $\mathcal{F}$ associated to $\mathbf{\Omega}^k$ is of canonical format on $\mathbf{\Omega}^k$ with rank R.*

For all d multi-index $\alpha \in \mathbb{N}^d$, we denote $|\alpha| := \sum_{i=1}^{d} \alpha_i$, $x^\alpha =: x_1^{\alpha_1} \cdot \ldots \cdot x_d^{\alpha_d}$ where $x = (x_1, \cdots, x_d) \in \mathbf{\Omega}$, and $\alpha! := \prod_{i=1}^{d} \alpha_i$. The weak derivative of order α is denoted by $D^{(\alpha)}$.

The rationale behind the proposition proposed hereafter is the following: we show that there exists a sufficient condition on the function regularity such that, if the error is measured in a given norm, there exists an admissible domain partition such that a finite rank tensor approximation in each subdomain achieves a prescribed accuracy on the whole tensor.

PROPOSITION 2.2. *Let $\Omega_1 = \cdots = \Omega_d = (0,1)$ so that $\mathbf{\Omega} := (0,1)^d$. For all $M \in \mathbb{N}^*$, let us consider $P^M := \{\left(\frac{m-1}{M}, \frac{m}{M}\right)\}_{1 \leq m \leq M}$ be a collection of subsets of $(0,1)$ and let $\boldsymbol{P}^M$ be the tensorized domain partition of $\mathbf{\Omega}$ associated to the collection of domain partitions $(P_j)_{1 \leq j \leq d}$ where $P_j = P^M$ for all $1 \leq j \leq d$. Let $k \in \mathbb{N}^*$, $1 \leq p \leq q \leq \infty$ and $\varepsilon > 0$. We denote $\lambda := \frac{k}{d} - \frac{1}{p} + \frac{1}{q} > 0$. Let $\mathcal{F} \in W^{k,p}(\mathbf{\Omega})$ such that $\|\mathcal{F}\|_{W^{k,p}(\mathbf{\Omega})} \leq 1$. Then, there exists a constant $C > 0$ which depends only on k, p, d, q such that for all $M \in \mathbb{N}^*$ such that $\ln M \geq -\frac{1}{d\lambda} \ln\left(\frac{\varepsilon}{C}\right)$, there exists a tensor $\mathcal{F}^{CPF}$ of Canonical Partitioning Format with domain partition $\boldsymbol{P}^M$ and rank*

$R \leq \frac{(k-1+d)!}{(k-1)!d!}$ *such that*

$$\|\mathcal{F} - \mathcal{F}^{CPF}\|_{L^q(\mathbf{\Omega})} \leq \varepsilon. \tag{2.1}$$

Remark 2.3. Let us make a simple remark before giving the proof of Proposition 2.2. In the case where the tensor $\mathcal{F}$ belongs to $H^1(\mathbf{\Omega})$. Then, using the same notation as in Proposition 2.2, $k = 1$ and $p = 2$. Besides, choosing $q = 2$, since $\frac{(k-1+d)!}{(k-1)!d!} = 1$, there exists an admissible partition of $\mathbf{\Omega}$ into $\#\boldsymbol{P}^M = M^d = \mathcal{O}\left(\varepsilon^{-\frac{1}{\lambda}}\right)$ subdomains such that $\mathcal{F}$ is approximated in the $L^2(\mathbf{\Omega})$ with precision ε by a tensor in Canonical Partitioning Format with domain partition $\boldsymbol{P}^M$ with rank 1.

The proof is mainly based on arguments introduced in [6].

Proof. First, for all $\boldsymbol{m} := (m_1, \cdots, m_d) \in \{1, \cdots, M\}^d$, let us denote

$$\mathbf{\Omega}^{\boldsymbol{m}} := \times_{j=1}^{d} \left(\frac{m_j - 1}{M}, \frac{m_j}{M} \right).$$

For all $\boldsymbol{m} \in \{1, \cdots, M\}^d$,for all tensor $\mathcal{G} \in W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})$, we introduce a polynomial approximation of $\mathcal{G}$ on the domain $\mathbf{\Omega}^{\boldsymbol{m}}$, based on the Taylor kernel, which is defined as follows:

$$\Pi^{\boldsymbol{m}}\mathcal{G} = \sum_{|\alpha|<k} \int_{\mathbf{\Omega}^{\boldsymbol{m}}} \frac{(x-y)^\alpha}{\alpha!} D^{(\alpha)}\mathcal{G}(y)\, dy. \tag{2.2}$$

The projector $\Pi^{\boldsymbol{m}}\mathcal{G}$ defines a polynomial approximation of $\mathcal{G}$ on the subdomain $\mathbf{\Omega}^{\boldsymbol{m}}$ of polynomial degree $k-1$.

Denoting by $\mathcal{F}^{\boldsymbol{m}} := \mathcal{F}|_{\mathbf{\Omega}^{\boldsymbol{m}}}$, we then have

$$\|\mathcal{F}^{\boldsymbol{m}} - \Pi^{\boldsymbol{m}}\mathcal{F}^{\boldsymbol{m}}\|_{L^q(\mathbf{\Omega}^{\boldsymbol{m}})} \leq C\, |\mathbf{\Omega}^{\boldsymbol{m}}|^\lambda \, \|\mathcal{F}^{\boldsymbol{m}}\|_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})}, \tag{2.3}$$

where $C > 0$ is a constant which only depends on k, d, p and q (see [5][Lemma V.6.1,p.289] or [6][Lemma 1]). Let us denote $\mathcal{F}^{CPF} \in L^q(\mathbf{\Omega})$ defined by $\mathcal{F}^{CPF}|_{\mathbf{\Omega}^{\boldsymbol{m}}} = \Pi^{\boldsymbol{m}}\mathcal{F}^{\boldsymbol{m}}$. Then, we have:

$$\left\|\mathcal{F} - \mathcal{F}^{CPF}\right\|^q_{L^q(\mathbf{\Omega})} = \sum_{\boldsymbol{m}\in\{1,\cdots,M\}^d} \|\mathcal{F}^{\boldsymbol{m}} - \Pi^{\boldsymbol{m}}\mathcal{F}^{\boldsymbol{m}}\|^q_{L^q(\mathbf{\Omega}^{\boldsymbol{m}})} \leq \sum_{\boldsymbol{m}\in\{1,\cdots,M\}^d} C^q\, |\mathbf{\Omega}^{\boldsymbol{m}}|^{q\lambda}\, \|\mathcal{F}^{\boldsymbol{m}}\|^q_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})}. \tag{2.4}$$

Since $|\mathbf{\Omega}^{\boldsymbol{m}}| = \frac{1}{M^d}$ for all $\boldsymbol{m} \in \{1, \cdots, M\}^d$, we obtain

$$\left\|\mathcal{F} - \mathcal{F}^{CPF}\right\|^q_{L^q(\mathbf{\Omega})} \leq C^q \left(\frac{1}{M^d}\right)^{q\lambda} \sum_{\boldsymbol{m}\in\{1,\cdots,M\}^d} \|\mathcal{F}\|^q_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})}. \tag{2.5}$$

In order to bound the last term, let us consider the following inequality:

$$\sum_{\boldsymbol{m}\in\{1,\cdots,M\}^d} \|\mathcal{F}^{\boldsymbol{m}}\|^q_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})} = \sum_{\boldsymbol{m}\in\{1,\cdots,M\}^d} \left(\left\|\mathcal{F}^i\right\|^p_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})}\right)^{\frac{q}{p}} \leq \left(\sum_{\boldsymbol{m}\in\{1,\cdots,M\}^d} \|\mathcal{F}^{\boldsymbol{m}}\|^p_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})}\right)^{\frac{q}{p}} \leq 1, \tag{2.6}$$

which holds true because $\|\mathcal{F}^{\boldsymbol{m}}\|_{W^{k,p}(\mathbf{\Omega}^{\boldsymbol{m}})} \leq 1$ and $q \geq p$. Putting the estimates together, we obtain

$$\left\|\mathcal{F} - \mathcal{F}^{CPF}\right\|_{L^q(\mathbf{\Omega})} \leq C \left(\frac{1}{M^d}\right)^{\lambda}. \tag{2.7}$$

Since $\lambda > 0$, as soon as M is large enough so that $C\left(\frac{1}{M^d}\right)^\lambda \leq \varepsilon$, we obtain (2.1). To conclude the proof, let us observe that a multi-variate polynomial with degree lower than $k-1$ can be written as a tensor of order at most $R = \frac{(k-1+d)!}{(k-1)!d!}$. Hence the result. ∎

Some remarks are in order. Proposition 2.2 does not investigate the tensor approximation *per se*, but it shows a sufficient regularity condition under which an admissible domain partition of a function is such that a given piece-wise finite rank tensor approximation (with a rank that could depend upon the format and be smaller than the polynomial rank) can achieve a prescribed accuracy. The sufficient condition is essentially related to the compactness of the embedding $W^{k,p} \hookrightarrow L^q$ (Rellich-Kondrakov theorem). The fact that the admissible domain partition is performed by dividing the domain into 2^{dN} subdomains is a reminder that the method, when practically implemented, is suitable to approximate tensors of moderate dimension.

In the light of Proposition 2.2, for a given tensor $\mathcal{F} \in L^q(\mathbf{\Omega})$, the two following issues naturally arise:

- on the one hand, given a particular admissible domain partition $\{\mathbf{\Omega}^k\}_{1\leq k\leq K}$, one would like to look for an algorithm which can construct effective low-rank approximations in a given format for all subtensors $\mathcal{F}^k$ so that (i) the global error between the tensor $\mathcal{F}$ and the obtained approximation of the tensor is guaranteeed to be lower than an a priori chosen error criterion; (ii) the total memory storage of all the low-rank partitions of the tensor on each subdomain is minimal. This requires a careful strategy to distribute the error over all the different subsets $\mathbf{\Omega}^k$. The procedure we propose is described in details in Section 3;
- on the other hand, one would like to develop a numerical method to find an optimal of quasi-optimal admissible domain partition $\{\mathbf{\Omega}^k\}_{1\leq k\leq K}$, so that the total memory storage of the low-rank tensor approximations of each subtensor $\mathcal{F}^k$ to be minimal, provided that a global error criterion is satified for the whole tensor $\mathcal{F}$. The procedure we propose is described in details in Section 4.

The aim of the two following sections is to propose algorithms in order to address these issues from a numerical point of view. We make the choice to present the algorithms from now on using discrete tensors, but we stress on the fact that they can be easily generalized to deal with the approximation of continuous tensors in the case when $q = 2$.

3. Greedy-HOSVD for Tucker Partitioned Format (TPF). In this section an algorithm is presented, which constructs an approximation of a given tensor of order d, associated to an admissible partition of the set of indices of the tensor, where the tensor is approximated by a tensor in Tucker format on each indices subsets. The algorithm relies on a greedy procedure which enables to distribute the error among the subdomains in an optimal way.

3.1. Notation and definitions. We introduce some notation and definitions on discrete tensors and subtensors which are used in the sequel, and are very similar to those used in [13, 4].

We consider from now on and in all the rest of the paper the case of *discrete* tensors. Let $d \in \mathbb{N}^*$ and let $I_1, \cdots, I_d$ be finite discrete sets of indices. For all $1 \leq j \leq d$, we denote $n_j := \#I_j$ the cardinality of the set I_j. We also denote $\boldsymbol{I} := I_1 \times \cdots \times I_d$ and $n := n_1 \times \cdots \times n_d$.

A tensor $\mathcal{A}$ of order d defined on $\boldsymbol{I}$ is a collection of $n_1 \times \cdots \times n_d$ real numbers $\mathcal{A} := (a_{\boldsymbol{i}})_{\boldsymbol{i}\in\boldsymbol{I}} \in \mathbb{R}^{\boldsymbol{I}}$. Let $A_1 := (a^1_{i_1})_{1\leq i_1\leq n_1} \in \mathbb{R}^{n_1}, \cdots, A_d := (a^d_{i_d})_{1\leq i_d\leq n_d} \in \mathbb{R}^{n_d}$, the pure tensor product tensor $A_1 \otimes \cdots \otimes A_d = (a_{\boldsymbol{i}})_{\boldsymbol{i}\in\boldsymbol{I}} \in \mathbb{R}^{\boldsymbol{I}}$ is the tensor of order d defined such that for all $\boldsymbol{i} := (i_1, \cdots, i_d) \in \boldsymbol{I}$,

$$a_{\boldsymbol{i}} = \Pi_{j=1}^d a^j_{i_j}.$$

Given two tensors $\mathcal{A} = (a_{\boldsymbol{i}})_{\boldsymbol{i}\in\boldsymbol{I}}, \mathcal{B} = (b_{\boldsymbol{i}})_{\boldsymbol{i}\in\boldsymbol{I}} \in \mathbb{R}^{\boldsymbol{I}}$, the ℓ^2 scalar product between $\mathcal{A}$ and $\mathcal{B}$ in $\mathbb{R}^{\boldsymbol{I}}$ is denoted by

$$\langle \mathcal{A}, \mathcal{B}\rangle := \sum_{\boldsymbol{i}\in\boldsymbol{I}} a_{\boldsymbol{i}} b_{\boldsymbol{i}}.$$

The tensor $\mathcal{A}$ is said to be of canonical format with rank $R \in \mathbb{N}$ if

$$\mathcal{A} = \sum_{r=1}^{R} A^r_1 \otimes \cdots \otimes A^r_d,$$

where for all $1 \leq j \leq d$ and $1 \leq r \leq R$, $A^r_j \in \mathbb{R}^{I_j}$.

The tensor $\mathcal{A}$ is said to be of Tucker Format with rank $\boldsymbol{R} = (R_1, \cdots, R_d) \in (\mathbb{N})^d$ if

$$\mathcal{A} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} c_{r_1,\cdots,r_d} A^{r_1}_1 \otimes \cdots \otimes A^{r_d}_d, \tag{3.1}$$

where for all $1 \leq j \leq d$ and all $1 \leq r_j \leq R_j$, $A^{r_j}_j \in \mathbb{R}^{I_j}$ and $(c_{r_1,\cdots,r_d})_{1\leq r_1\leq R_1,\cdots,1\leq r_d\leq R_d} \in \mathbb{R}^{R_1\times\cdots\times R_d}$.

It is clear from expression (3.1) that the memory needed to store a tensor defined on a set of indices $\boldsymbol{I} = I_1 \times \cdots \times I_d$ with ranks $\boldsymbol{R} := (R_1, \cdots, R_d) \in \mathbb{N}^d$ is equal to

$$M_{TF}(\boldsymbol{I}, \boldsymbol{R}) := \begin{cases} \Pi_{j=1}^d R_j + \sum_{j=1}^d R_j |I_j| & \text{if } R_j > 0 \text{ for all } 1 \leq j \leq d. \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Several other tensor formats can be found in the litterature. We refer the reader for instance to [10, 9, 14, 17] for the precise definitions of the Tensor Train and the Hierarchical Tree Tensor formats. For the sake of simplicity, we do not give their definition in full details here.

Let now $J_1 \subset I_1$, ..., $J_d \subset I_d$ be non-empty subsets of indices and $\boldsymbol{J} := J_1 \times \cdots \times J_d$. The subtensor of $\mathcal{A}$ associated to $\boldsymbol{J}$ is the tensor $\mathcal{A}^{\boldsymbol{J}}$ defined as $\mathcal{A}^{\boldsymbol{J}} := (a_{\boldsymbol{i}\in\boldsymbol{J}}) \in \mathbb{R}^{\boldsymbol{J}}$.

Let us now consider a partition $\mathcal{P}$ of $\boldsymbol{I}$ such that for all $\boldsymbol{J} \in \mathcal{P}$, there exists $J_1 \subset I_1, \cdots, J_d \subset I_d$ such that $\boldsymbol{J} := J_1 \times \cdots \times J_d$. Recall that the fact that $\mathcal{P}$ is a partition of $\boldsymbol{I}$ implies that $\boldsymbol{I} = \bigcup_{\boldsymbol{J}\in\mathcal{P}} \boldsymbol{J}$ and that for all $\boldsymbol{J}_1, \boldsymbol{J}_2 \in \mathcal{P}$ such that $\boldsymbol{J}_1 \neq \boldsymbol{J}_2$, then $\boldsymbol{J}_1 \cap \boldsymbol{J}_2 = \emptyset$. Following a denomination already introduced in [13], such a partition will be called hereafter an *admissible* partition of $\boldsymbol{I}$.

Then, the collection of subtensors of $\mathcal{A}$ associated to the partition $\mathcal{P}$ is defined as the set of subtensors $(\mathcal{A}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}$.

A particular case of admissible partition of $\boldsymbol{I}$ can be for instance constructed as follows: for all $1 \leq j \leq d$, let $K_j \in \mathbb{N}^*$ and let us consider a partition $\mathcal{P}_j :=$

$\left\{I_j^{k_j}\right\}_{1\leq k_j\leq K_j}$ of I_j such that $I_j := \bigcup_{1\leq k_j\leq K_j} I_j^{k_j}$ and $I_j^{k_j}\cap I_j^{l_j}=\emptyset$ for all $1\leq k_j\neq l_j\leq K_j$. Let us denote

$$\mathcal{P} := \left(\times_{j=1}^{d} I_j^{k_j}\right)_{1\leq k_1\leq K_1,\cdots,1\leq k_d\leq K_d}.$$

The partition $\mathcal{P}$ then defines an admissible partition of $\boldsymbol{I}$, and will be called hereafter the *tensorized partition* associated to the collection of partitions $(\mathcal{P}_j)_{1\leq j\leq d}$.

The following definitions are introduced:

Definition 3.1.

- *A tensor $\mathcal{A}$ defined on $\boldsymbol{I}$ is said to be of Canonical Partitioning Format (CPF) if there exists an admissible partition $\mathcal{P}$ of $\boldsymbol{I}$ such that for all $\boldsymbol{J}\in\mathcal{P}$, the subtensor $\mathcal{A}^{\boldsymbol{J}}$ of $\mathcal{A}$ associated to $\boldsymbol{J}$ is of canonical format on $\boldsymbol{J}$.*
- *A tensor $\mathcal{A}$ defined on $\boldsymbol{I}$ is said to be of Tucker Partitioning Format (TPF) if there exists an admissible partition $\mathcal{P}$ of $\boldsymbol{I}$ such that for all $\boldsymbol{J}\in\mathcal{P}$, the subtensor $\mathcal{A}^{\boldsymbol{J}}$ of $\mathcal{A}$ associated to $\boldsymbol{J}$ is of Tucker format on $\boldsymbol{J}$.*

3.2. Greedy-HOSVD for Tucker format. Let $\mathcal{A} := (a_{\boldsymbol{i}})_{\boldsymbol{i}\in\boldsymbol{I}}\in\mathbb{R}^{\boldsymbol{I}}$ be a discrete tensor of order d. The aim of the two following sections is to present an algorithm which, given a particular admissible partition of the set of indices, provides effective low-rank approximations in Tucker format for all subtensors of $\mathcal{A}$. The algorithm presented hereafter guarantees that the global l^2 error between the tensor $\mathcal{A}$ and the obtained approximation of the tensor is lower than an a priori chosen error criterion. The main novelty of this algorithm consists in using a greedy algorithm in conjunction with the well-known HOSVD procedure which enables to distribute the error in a non-uniform adapted way among the different unfoldings of the tensor $\mathcal{A}$. This Greedy-HOSVD procedure is the starting point of the Hierarchical Merge algorithm which we present in Section 4.

We recall here some well-known definitions and introduce some notation about unfoldings and singular value decomposition. For all $1\leq j\leq d$, let $\widehat{n}_j := \Pi_{j'\neq j} n_{j'}$ and $\widehat{\boldsymbol{I}}_j := I_1\times\cdots\times I_{j-1}\times I_{j+1}\times\cdots\times I_d$. For all $\boldsymbol{i}=(i_1,\cdots,i_d)\in\boldsymbol{I}$, let $\widehat{\boldsymbol{i}}_j := (i_1,\cdots,i_{j-1},i_{j+1},\cdots,i_d)\in\widehat{\boldsymbol{I}}_j$.

The j^{th} unfolding associated to the tensor $\mathcal{A}$ is the matrix $\mathcal{A}_j\in\mathbb{R}^{I_j\times\widehat{\boldsymbol{I}}_j}$ which is defined such that

$$\forall \boldsymbol{i} := (i_1,\cdots,i_d)\in\boldsymbol{I},\quad (\mathcal{A}_j)_{i_j,\widehat{\boldsymbol{i}}_j} = a_{\boldsymbol{i}}.$$

The singular values of $\mathcal{A}_j$ (ranged in decreasing order) are then defined by $\sigma_j^1(\mathcal{A})\geq\sigma_j^2(\mathcal{A})\geq\cdots\geq\sigma_j^{p_j(\mathcal{A})}(\mathcal{A})$, where $p_j(\mathcal{A}) := \min(n_j,\widehat{n}_j)$. For all $1\leq q\leq p_j(\mathcal{A})$, we denote $U_j^q(\mathcal{A})\in\mathbb{R}^{n_j}$ a left-hand side singular mode of $\mathcal{A}_j$ associated to the singular value $\sigma_j^q(\mathcal{A})$ so that $(U_j^1(\mathcal{A}),U_j^2(\mathcal{A}),\cdots,U_j^{p_j(\mathcal{A})}(\mathcal{A}))$ is an orthonormal family of $\mathbb{R}^{n_j}$.

For all $1\leq r_1\leq p_1(\mathcal{A})$, ..., $1\leq r_d\leq p_d(\mathcal{A})$, let us define

$$c^{\mathcal{A}}_{r_1,\cdots,r_d} := \langle \mathcal{A}, U_1^{r_1}(\mathcal{A})\otimes\cdots\otimes U_d^{r_d}(\mathcal{A})\rangle.$$

Let $\boldsymbol{R} := (R_1,\cdots,R_d)\in(\mathbb{N}^*)^d$ such that for all $1\leq j\leq d$, $1\leq R_j\leq p_j(\mathcal{A})$ and let us define

$$\mathcal{A}^{TF,\boldsymbol{R}} := \sum_{1\leq r_1\leq R_1}\cdots\sum_{1\leq r_d\leq R_d} c^{\mathcal{A}}_{r_1,\cdots,r_d} U_1^{r_1}(\mathcal{A})\otimes\cdots\otimes U_d^{r_d}(\mathcal{A}).$$

Then, the following inequality holds [10]

$$\left\|\mathcal{A}-\mathcal{A}^{TF,\boldsymbol{R}}\right\| \leq \sqrt{\sum_{1\leq j\leq d}\sum_{R_j+1\leq q_j\leq p_j(\mathcal{A})}\left|\sigma_j^{q_j}(\mathcal{A})\right|^2}. \tag{3.3}$$

Note that the tensor $\mathcal{A}^{TF,\boldsymbol{R}}$ is a tensor of Tucker format with rank $\boldsymbol{R}$. A natural question is then the following: given an *a priori* chosen error tolerance $\epsilon > 0$, how should one choose the rank $\boldsymbol{R}$ to ensure that

$$\left\|\mathcal{A}-\mathcal{A}^{TF,\boldsymbol{R}}\right\| \leq \epsilon. \tag{3.4}$$

The most commonly used strategy to choose $\boldsymbol{R}$ in order to guarantee (3.4) is the following [10]. For all $1 \leq j \leq d$, the integer R_j is chosen so that

$$R_j := \min\left\{1 \leq R \leq p_j(\mathcal{A}), \sum_{p_j(\mathcal{A})\geq q\geq R+1}|\sigma_j^q(\mathcal{A})|^2 \leq \epsilon^2/d\right\}.$$

By construction, using (3.3), it holds that $\mathcal{A}^{TF,\boldsymbol{R}}$ obviously satisfies (3.4). Such a choice implies that the squared error tolerance ϵ^2 is uniformly distributed with respect to each value of $1 \leq j \leq d$.

In the present work, it appeared that distributing the error in an appropriate manner with respect to j is a crucial feature for the proposed algorithms (based on partitioning) to be efficient in terms of memory compression. The first contribution of this paper consists in suggesting an alternative numerical strategy to choose the rank $\boldsymbol{R}$ so that $\mathcal{A}^{TF,\boldsymbol{R}}$ satisfies (3.4), which appears to yield sparser approximations of the tensor $\mathcal{A}$ while maintaining the same level of accuracy than the strategy presented above. The method is based on a greedy algorithm, which is an iterative procedure, whose aim is to compute a set of ranks $\boldsymbol{R} \in (\mathbb{N}^*)^d$ such that

$$\left\|\mathcal{A}-\mathcal{A}^{TF,\boldsymbol{R}}\right\| \leq \varepsilon,$$

where ε is a desired error tolerance, and the principle of the algorithm is the following. Assume that we have already an approximation of $\mathcal{A}$ at hand, given by $\mathcal{A}^{TF,\widetilde{\boldsymbol{R}}}$ for some $\widetilde{\boldsymbol{R}} := (\widetilde{R}_1, \cdots, \widetilde{R}_d) \in (\mathbb{N}^*)^d$. The backbone of the greedy algorithm is to increase the rank corresponding to the variable $1 \leq j_0 \leq d$ which has the greatest contribution to the error

$$\sqrt{\sum_{1\leq j\leq d}\sum_{\widetilde{R}_j+1\leq q_j\leq p_j(\mathcal{A})}\left|\sigma_j^{q_j}(\mathcal{A})\right|^2}.$$

More precisely, we select the integer $1 \leq j_0 \leq d$ such that $j_0 \in \underset{1\leq j\leq d}{\operatorname{argmax}}\,\sigma_j^{\widetilde{R}_j+1}(\mathcal{A})$ and increase the j_0^{th} rank by one. This procedure is repeated until we obtain a set of ranks such that the desired error tolerance is reached. Algorithm 3.1 summarizes the Greedy-HOSVD procedure.

In view of (3.3), it can be obviously seen that the rank $\boldsymbol{R}$ computed by the Greedy-HOSVD procedure described in Algorithm 3.1 necessarily implies that $\mathcal{A}^{TF,\boldsymbol{R}}$ satisfies (3.4).

Algorithm 3.1 Greedy-HOSVD

Input:
$\mathcal{A} \in \mathbb{R}^{\boldsymbol{I}} \leftarrow$ a tensor of order d
$\varepsilon > 0 \leftarrow$ error tolerance criterion

Output:
Rank $\boldsymbol{R} := (R_1, \cdots, R_d) \in \mathbb{N}^d$

Begin:
Set $\boldsymbol{R} := (0, \cdots, 0)$
while $\displaystyle\sum_{1 \leq j \leq d} \sum_{R_j + 1 \leq q_j \leq p_j(\mathcal{A})} \left|\sigma_j^{q_j}(\mathcal{A})\right|^2 > \varepsilon^2$ **do**
    Select $1 \leq j_0 \leq d$ such that

$$j_0 = \underset{1 \leq j \leq d}{\operatorname{argmax}}\, \sigma_j^{R_j+1}(\mathcal{A}).$$

    if $\boldsymbol{R} = (0, \cdots, 0)$ **then**
        Set $\boldsymbol{R} := (1, \cdots, 1)$
    else
        $R_{j_0} \leftarrow R_{j_0} + 1.$
    return $\boldsymbol{R}$

3.3. PF-Greedy-HOSVD for Partitioned Tucker format. We now present a direct generalization of the Greedy-HOSVD procedure described in Algorithm 3.1 in order to construct an approximation of the tensor $\mathcal{A}$ in a Partitioned Tucker format associated to an a priori fixed admissible partition $\mathcal{P}$ of $\boldsymbol{I}$.

For all $\boldsymbol{J} \in \mathcal{P}$, let $\boldsymbol{R}^{\boldsymbol{J}} := (R_1^{\boldsymbol{J}}, \cdots, R_d^{\boldsymbol{J}}) \in \mathbb{N}^d$ be a set of ranks. We define an approximation $\mathcal{A}^{PTF,(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}}$ of the tensor $\mathcal{A}$ in Partitioned Tucker Format (PTF) as follows: for all $\boldsymbol{J}_0 \in \mathcal{P}$,

$$\left(\mathcal{A}^{PTF,(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}}\right)^{\boldsymbol{J}_0} = \left(\mathcal{A}^{\boldsymbol{J}_0}\right)^{TF,\boldsymbol{R}^{\boldsymbol{J}_0}}.$$

It then naturally holds, using (3.3), that

$$\left\|\mathcal{A} - \mathcal{A}^{PTF,(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}}\right\| \leq \sqrt{\sum_{\boldsymbol{J}\in\mathcal{P}} \sum_{1 \leq j \leq d} \sum_{R_j^k + 1 \leq q_j \leq p_j(\mathcal{A}^{\boldsymbol{J}})} \left|\sigma_j^{q_j}(\mathcal{A}^{\boldsymbol{J}})\right|^2}.$$

For a given error tolerance $\varepsilon > 0$, the procedure described in Algorithm 3.2 then naturally produces a set of ranks $(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}$ such that

$$\left\|\mathcal{A} - \mathcal{A}^{PTF,(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}}\right\| \leq \varepsilon,$$

and is also a greedy algorithm.

4. Hierarchical low rank tensor approximation. In this section we describe the main contribution of the present paper, which is a hierarchical low rank tensor approximation procedure designed to compress tensors that have overall high ranks, but are formed by many subtensors of low ranks.

Algorithm 3.2 PF-Greedy-HOSVD

Input:
$\mathcal{A} \in \mathbb{R}^{\boldsymbol{I}} \leftarrow$ a tensor of order d
$\mathcal{P} \leftarrow$ an admissible partition of $\boldsymbol{I}$
$\varepsilon > 0 \leftarrow$ error tolerance criterion

Output:
Set of Ranks $(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}} \subset \mathbb{N}^d$
Set of local errors $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}$ satisfying $\sum_{\boldsymbol{J}\in\mathcal{P}} |\varepsilon^{\boldsymbol{J}}|^2 < |\varepsilon|^2$.

Begin:
Set $\boldsymbol{R}^{\boldsymbol{J}} := (0, \cdots, 0)$ for all $\boldsymbol{J} \in \mathcal{P}$
while $\displaystyle\sum_{\boldsymbol{J}\in\mathcal{P}} \sum_{1\le j\le d} \sum_{R_j^{\boldsymbol{J}}+1\le q_j\le p_j(\mathcal{A}^{\boldsymbol{J}})} \left|\sigma_j^{q_j}(\mathcal{A}^{\boldsymbol{J}})\right|^2 \ge \varepsilon^2$ **do**
  Select $1 \le j_0 \le d$ and $\boldsymbol{J}_0 \in \mathcal{P}$ such that

$$(j_0, \boldsymbol{J}_0) = \operatorname*{argmax}_{1\le j\le d,\ \boldsymbol{J}\in\mathcal{P}} \sigma_j^{R_j^{\boldsymbol{J}}+1}(\mathcal{A}^{\boldsymbol{J}}).$$

  if $\boldsymbol{R}^{\boldsymbol{J}_0} = (0, \cdots, 0)$ **then**
    Set $\boldsymbol{R}^{\boldsymbol{J}_0} := (1, \cdots, 1)$
  else
    Assume that $\boldsymbol{R}^{\boldsymbol{J}_0} = (R_1^{\boldsymbol{J}_0}, \cdots, R_d^{\boldsymbol{J}_0})$.
    $R_{j_0}^{\boldsymbol{J}_0} \leftarrow R_{j_0}^{\boldsymbol{J}_0} + 1$.
Define $\varepsilon^{\boldsymbol{J}} := \sqrt{\sum_{1\le j\le d} \sum_{R_j^{\boldsymbol{J}}+1\le q_j\le p_j(\mathcal{A}^{\boldsymbol{J}})} \left|\sigma_j^{q_j}(\mathcal{A}^{\boldsymbol{J}})\right|^2}$ for all $\boldsymbol{J} \in \mathcal{P}$.
return $\left(\boldsymbol{R}^{\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{P}}$ and $\left(\varepsilon^{\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{P}}$

The PF-Greedy-HOSVD Algorithm 3.2 presented in Section 3 is based on the assumption that the partitioning of the tensor format is fixed. In the following, we introduce an algorithm that allows to identify the low-rank-ness of some potentially large parts of a tensor, for which a different representation format than the one prescribed by PF-Greedy-HOSDV Algorithm 3.2 is used to obtain a better compression.

4.1. Partition tree. To present the algorithm, we first need to introduce the notion of *partition tree*. A partition tree may be seen as a generalization of *cluster tree* as defined in [3] for hierarchical matrices.

Let $T_{\boldsymbol{I}}$ be a tree with vertices (or nodes) $\mathcal{V}(T_{\boldsymbol{I}})$ and edges $\mathcal{E}(T_{\boldsymbol{I}})$. For all $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}})$, we denote

$$\mathcal{S}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) := \left\{\boldsymbol{J}' \in \mathcal{V}(T_{\boldsymbol{I}}),\ (\boldsymbol{J}, \boldsymbol{J}') \in \mathcal{E}(T_{\boldsymbol{I}})\right\}$$

the set of sons of the vertex $\boldsymbol{J}$. By induction, we define the set of sons of $\boldsymbol{J}$ of the k^{th} generation, denoted by $\mathcal{S}_{\boldsymbol{J}}^k(T_{\boldsymbol{I}})$ with $k \in \mathbb{N}^*$ as follows

$$\mathcal{S}_{\boldsymbol{J}}^1(T_{\boldsymbol{I}}) = \mathcal{S}_{\boldsymbol{J}}(T_{\boldsymbol{I}}), \quad \mathcal{S}_{\boldsymbol{J}}^k(T_{\boldsymbol{I}}) = \left\{\boldsymbol{J}'' \in \mathcal{V}(T_{\boldsymbol{I}}),\ \exists \boldsymbol{J}' \in \mathcal{S}_{\boldsymbol{J}}^{k-1}(T_{\boldsymbol{I}}),\ (\boldsymbol{J}', \boldsymbol{J}'') \in \mathcal{E}(T_{\boldsymbol{I}})\right\}.$$

The set of leaves of $T_{\boldsymbol{I}}$ is defined as

$$\mathcal{L}(T_{\boldsymbol{I}}) := \left\{\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}}), \mathcal{S}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) = \emptyset\right\}.$$

The set of parents of leaves of a tree $T_{\boldsymbol{I}}$ is defined as the set $\mathcal{L}^p(T_{\boldsymbol{I}}) \subset \mathcal{V}(T_{\boldsymbol{I}})$ of vertices

of $T_{\boldsymbol{I}}$ which have at least one son which is a leaf of $T_{\boldsymbol{I}}$, i.e.

$$\mathcal{L}^p(T_{\boldsymbol{I}}) := \{\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}}),\ \mathcal{S}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) \cap \mathcal{L}(T_{\boldsymbol{I}}) \neq \emptyset\}.$$

For any $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}})$, the set of descendants of $\boldsymbol{J}$ in $T_{\boldsymbol{I}}$ is defined as

$$\mathcal{D}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) := \left\{\boldsymbol{J}' \in \mathcal{V}(T_{\boldsymbol{I}}),\ \exists k \in \mathbb{N}^*,\ \boldsymbol{J}' \in \mathcal{S}^k_{\boldsymbol{J}}(T_{\boldsymbol{I}})\right\}.$$

We are now in position to state the definition of a partition tree.

DEFINITION 4.1. *A tree $T_{\boldsymbol{I}}$ is called a partition tree for the set $\boldsymbol{I}$ if the following conditions hold:*

- *$\boldsymbol{I}$ is the root of $T_{\boldsymbol{I}}$;*
- *For all $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{L}(T_{\boldsymbol{I}})$, $\mathcal{S}_{\boldsymbol{J}}(T_{\boldsymbol{I}})$ is an admissible partition of $\boldsymbol{J}$ and $|\mathcal{S}_{\boldsymbol{J}}(T_{\boldsymbol{I}})| \geq 2$;*
- *For all $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}})$, $\boldsymbol{J} \neq \emptyset$.*

The goal of a partition tree $T_{\boldsymbol{I}}$, with respect to the adaptivity of the partitioning of a given tensor $\mathcal{A}$, is twofold:

- The current admissible partition of a tensor $\mathcal{A}$ will be given by the set of leaves of the tree $\mathcal{L}(T_{\boldsymbol{I}})$;
- The different merging scenarii to be tested will be encoded through the different vertices of the tree that are not leaves $\mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{L}(T_{\boldsymbol{I}})$.

We also introduce here the definition of the merged tree of a partition tree $T_{\boldsymbol{I}}$ associated to a vertex $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{L}(T_{\boldsymbol{I}})$.

DEFINITION 4.2. *Let $T_{\boldsymbol{I}}$ be a partition tree for $\boldsymbol{I}$ with vertices (or nodes) $\mathcal{V}(T_{\boldsymbol{I}})$ and edges $\mathcal{E}(T_{\boldsymbol{I}})$. Let $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{L}(T_{\boldsymbol{I}})$. The merged tree of $T_{\boldsymbol{I}}$ associated to the vertex $\boldsymbol{J}$ is the tree denoted by $T^m_{\boldsymbol{I}}(\boldsymbol{J})$ with root $\boldsymbol{I}$, vertices $\mathcal{V}(T^m_{\boldsymbol{I}}(\boldsymbol{J})) := \mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{D}_{\boldsymbol{J}}(T_{\boldsymbol{I}})$ and edges*

$$\begin{aligned}\mathcal{E}\left(T^m_{\boldsymbol{I}}(\boldsymbol{J})\right) &:= \mathcal{E}(T_{\boldsymbol{I}}) \cap \left(\mathcal{V}(T^m_{\boldsymbol{I}}(\boldsymbol{J})) \times \mathcal{V}(T^m_{\boldsymbol{I}}(\boldsymbol{J}))\right) \\ &= \mathcal{E}(T_{\boldsymbol{I}}) \setminus \left(\mathcal{V}(T_{\boldsymbol{I}}) \times \mathcal{D}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) \cup \mathcal{D}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) \times \mathcal{V}(T_{\boldsymbol{I}})\right).\end{aligned}$$

The merged tree $T^m_{\boldsymbol{I}}(\boldsymbol{J})$ is the partition tree which will be associated to a tensor if it is decided, through the merging algorithm, that it is more favorable to merge all the indices subsets of the present partition included in $\boldsymbol{J}$ into a single indices subset $\boldsymbol{J}$. Indeed, the set of leaves of the merged tree of $T_{\boldsymbol{I}}$ associated to the vertex $\boldsymbol{J}$ can be characterized as

$$\mathcal{L}(T^m_{\boldsymbol{I}}(\boldsymbol{J})) = \boldsymbol{J} \cup \bigcup_{\substack{\boldsymbol{J}' \in \mathcal{L}(T_{\boldsymbol{I}}) \\ \boldsymbol{J}' \cap \boldsymbol{J} = \emptyset}} \{\boldsymbol{J}'\}.$$

We collect in the following lemma a few useful results that can be easily proved by a recursive argument.

LEMMA 4.3. *Let $T_{\boldsymbol{I}}$ be a partition tree for $\boldsymbol{I}$. Let $\boldsymbol{J} \in \mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{L}(T_{\boldsymbol{I}})$.*

(i) For all $\boldsymbol{J}' \in \mathcal{V}(T_{\boldsymbol{I}})$, $\boldsymbol{J}' \subset \boldsymbol{I}$.

(ii) The set of leaves $\mathcal{L}(T_{\boldsymbol{I}})$ is an admissible partition of the set $\boldsymbol{I}$.

(iii) The set $\mathcal{D}_{\boldsymbol{J}}(T_{\boldsymbol{I}}) \cap \mathcal{L}(T_{\boldsymbol{I}})$ forms an admissible partition of the set $\boldsymbol{J}$.

(iv) The merged tree $T^m_{\boldsymbol{I}}(\boldsymbol{J})$ is a partition tree for $\boldsymbol{I}$.

(v) The set of leaves of $T^m_{\boldsymbol{I}}(\boldsymbol{J})$ is $\mathcal{L}\left(T^m_{\boldsymbol{I}}(\boldsymbol{J})\right) = \{\boldsymbol{J}\} \cup \left(\mathcal{L}(T_{\boldsymbol{I}}) \setminus \mathcal{D}_{\boldsymbol{J}}(T_{\boldsymbol{I}})\right)$.

4.1.1. Example: dyadic partition tree. We give here an example of partition tree in the particular case when $\boldsymbol{I} = I_1 \times \cdots \times I_d$ with $I_1 = \cdots = I_d =: I$. Let $\ell \in \mathbb{N}^*$ and assume that there exists a partition $\{I^{\ell,k}\}_{1\leq k\leq 2^\ell}$, $l \geq d$, of the set of indices I such that for all $1 \leq k \leq 2^\ell$, $I^{\ell,k} \neq \emptyset$. For all $1 \leq k \leq 2^{l-1}$, let $I^{l-1,k} := \bigcup_{(k-1)\cdot 2^d+1\leq j\leq k\cdot 2^d} I^{l,j}$. A merged dyadic partition tree $T_{\boldsymbol{I}}$ is obtained by allowing, for each vertex $I^{l-1,k}$, the merging of 2^d indices subsets $\{I^{\ell,k}\}_{(k-1)\cdot 2^d+1\leq j\leq k\cdot 2^d}$ into a single domaine $I^{l-1,k}$. The merge can continue recursively until depth ℓ. The merged tree $T_{\boldsymbol{I}}$ is a full 2^d-ary tree, that is a tree in which every vertex has either 0 or 2^d children and its height is at most ℓ.

4.2. PF-MERGE procedure. The proposed hierarchical tensor approximation is presented in Algorithm 4.1. It takes as input the tensor $\mathcal{A} \in \mathbb{R}^{\boldsymbol{I}}$ of order d, an initial partition tree $T_{\boldsymbol{I}}^{\text{init}}$, and the error ε that will be satisfied by the approximation. The partition tree $T_{\boldsymbol{I}}^{\text{init}}$ provides an initial hierarchical partitioning of the tensor $\mathcal{A}$ into subtensors, where the root of the tree is associated with the original tensor $\mathcal{A}$, and every vertex of the tree is associated with a subtensor.

The PF-MERGE procedure computes an approximation of $\mathcal{A}$ by traversing the hierarchy of subtensors in a bottom-up approach and adapting throughout the iterations the initial partition tree while ensuring that the error of the approximation remains smaller than ε. It provides as output the final partition tree $T_{\boldsymbol{I}}$, the errors and the ranks of the approximation in Tucker format of the subtensors corresponding to the leaves of $T_{\boldsymbol{I}}$, $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$ and $\left(\boldsymbol{R}^{\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})} \subset \mathbb{N}^d$ respectively, and the approximation $\mathcal{A}^{PTF,(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}}$ of the tensor $\mathcal{A}$ in Partitioned Tucker Format. The algorithm ensures that $\left\|\mathcal{A} - \mathcal{A}^{PTF,(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{P}}}\right\| \leq \varepsilon$.

The algorithm starts by compressing the subtensors associated with the leaves of $T_{\boldsymbol{I}}^{\text{init}}$, using the PF-Greedy-HOSVD Algorithm 3.2. This leads to an approximation of $\mathcal{A}$ in a partitioned Tucker format with a greedy distribution of the error ε among subtensors. Then the partition tree and the associated hierarchy of subtensors are traversed in a bottom-up approach. For this, the algorithm uses two sets of vertices, a set of vertices that are considered for merging $\mathcal{N}_{\text{totest}}$, which is initialized with $\mathcal{L}^p(T_{\boldsymbol{I}})$, and a complementary set of vertices for which no merging is attempted, $\mathcal{N}_{\text{nomerge}}$, initialized with the empty set.

At each iteration of the algorithm, a vertex $\boldsymbol{J}_0 \in \mathcal{N}_{\text{totest}}$ is chosen and the MERGE procedure determines whether it is more favorable, in terms of memory consumption, to merge the subtensors corresponding to the sons of $\boldsymbol{J}_0$ into a single subtensor and approximate it in Tucker format, or to keep them splitted. The MERGE procedure is presented in Algorithm 4.2 and its description is postponed to the end of this section. If the merge is more favorable, then a new partition tree $T_{\boldsymbol{I}}$ that reflects the merging is defined. If the parent of $\boldsymbol{J}_0$ is not already a vertex in $\mathcal{N}_{\text{nomerge}}$, then it is added to $\mathcal{N}_{\text{totest}}$, and the errors $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$ and the ranks $(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$ of the leaves of $T_{\boldsymbol{I}}$ are updated. Otherwise, the vertex $\boldsymbol{J}_0$ is added to the set $\mathcal{N}_{\text{nomerge}}$ and removed from the set $\mathcal{N}_{\text{totest}}$. The algorithm continues until the set $\mathcal{N}_{\text{totest}}$ becomes empty.

The MERGE procedure is described in Algorithm 4.2. Given a vertex $\boldsymbol{J}_0 \in \mathcal{N}_{\text{totest}}$ and a partition tree $T_{\boldsymbol{I}}$, the algorithm computes M_{nomerge}, the memory needed for storing in Tucker format the approximation of the subtensors corresponding to the sons of $\boldsymbol{J}_0$ in the partition tree, and $\eta := \sqrt{\sum_{\boldsymbol{J}\in\mathcal{P}_{\boldsymbol{J}_0}} |\varepsilon^{\boldsymbol{J}}|^2}$, the contribution of the errors of those approximations to the total approximation error. Then it calls the Greedy-HOSVD Algorithm 3.1 to compute an approximation in Tucker format of

Algorithm 4.1 PF-MERGE

Input:
$\mathcal{A} \in \mathbb{R}^{\boldsymbol{I}} \leftarrow$ a tensor of order d
$T_{\boldsymbol{I}}^{\text{init}}$ an initial partition tree of $\boldsymbol{I}$
$\varepsilon > 0 \leftarrow$ error tolerance criterion

Output:
$T_{\boldsymbol{I}}$ a final partition tree of $\boldsymbol{I}$
A set of leaf errors $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$
A set of leaf ranks $\left(\boldsymbol{R}^{\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})} \subset \mathbb{N}^d$

Begin:
Set $T_{\boldsymbol{I}} = T_{\boldsymbol{I}}^{\text{init}}$.
Compute $((\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}, (\overline{\varepsilon}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}) = \text{PF-Greedy-HOSVD}(\mathcal{A}, \mathcal{L}(T_{\boldsymbol{I}}), \varepsilon)$
Compute $\eta^2 := \varepsilon^2 - \sum_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})} |\overline{\varepsilon}^{\boldsymbol{J}}|^2$.
For all $\boldsymbol{J} \in \mathcal{L}(T_{\boldsymbol{I}})$, define $\varepsilon^{\boldsymbol{J}} := \sqrt{|\overline{\varepsilon}^{\boldsymbol{J}}|^2 + \frac{|\boldsymbol{J}|}{|\boldsymbol{I}|}\eta^2}$.
Set $\mathcal{N}_{\text{totest}} = \mathcal{L}^p(T_{\boldsymbol{I}})$ and $\mathcal{N}_{\text{nomerge}} = \emptyset$.
while $\mathcal{N}_{\text{totest}} \neq \emptyset$ **do**
    Choose $\boldsymbol{J}_0 \in \mathcal{N}_{\text{totest}}$.
    $(T_{\boldsymbol{I}}^{\text{fin}}, merge, (\varepsilon^{\text{fin},\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}, \left(\boldsymbol{R}^{\text{fin},\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}) =$ MERGE$(\mathcal{A}, T_{\boldsymbol{I}}, (\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}, (\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}, \boldsymbol{J}_0)$
    if $merge =$ true **then**
        $T_{\boldsymbol{I}} = T_{\boldsymbol{I}}^{\text{fin}}$
        $\mathcal{N}_{\text{totest}} = \mathcal{L}^p(T_{\boldsymbol{I}}^{\text{fin}}) \setminus \mathcal{N}_{\text{nomerge}}$.
        $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})} = (\varepsilon^{\text{fin},\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}$, $(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})} = \left(\boldsymbol{R}^{\text{fin},\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}$
    else
        $\mathcal{N}_{\text{nomerge}} = \mathcal{N}_{\text{nomerge}} \cup \{\boldsymbol{J}_0\}$; $\mathcal{N}_{\text{totest}} = \mathcal{N}_{\text{totest}} \setminus \{\boldsymbol{J}_0\}$.
    return $T_{\boldsymbol{I}}$, $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$, $(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$

$\mathcal{A}^{\boldsymbol{J}_0}$, the subtensor associated with the vertex $\boldsymbol{J}_0$, that satisfies the error tolerance η. This ensures that the contribution to the total approximation error is preserved if the merge is performed. If M_{merge}, the memory needed to store the approximation of the subtensor associated to the vertex $\boldsymbol{J}_0$, is smaller than M_{nomerge}, then the merge is performed. Otherwise, it is not.

For the extreme case where the whole original tensor is of very low rank, the merge stage will eventually choose to merge all the subtensors, meaning that the final result of the merge stage will be a trivial approximation in Tucker format of the original very low rank tensor, which is the desired result. For a typical practical case, the merge stage will lead to a hierarchical representation of the original tensor, where the higher rank subtensors have more storage assigned for their complicated subtree structure, and the low rank subtensors have relatively simple Tucker format approximations corresponding to the leaves in a tree.

As described in Algorithm 4.1, once a merge step rejects the merge and keeps the partitioning in a group of subtensors, no further merge will be attempted for a subdomain containing this group of subtensors. The advantage of this is that the number of merge steps is reduced and the merge tends to be performed on relatively

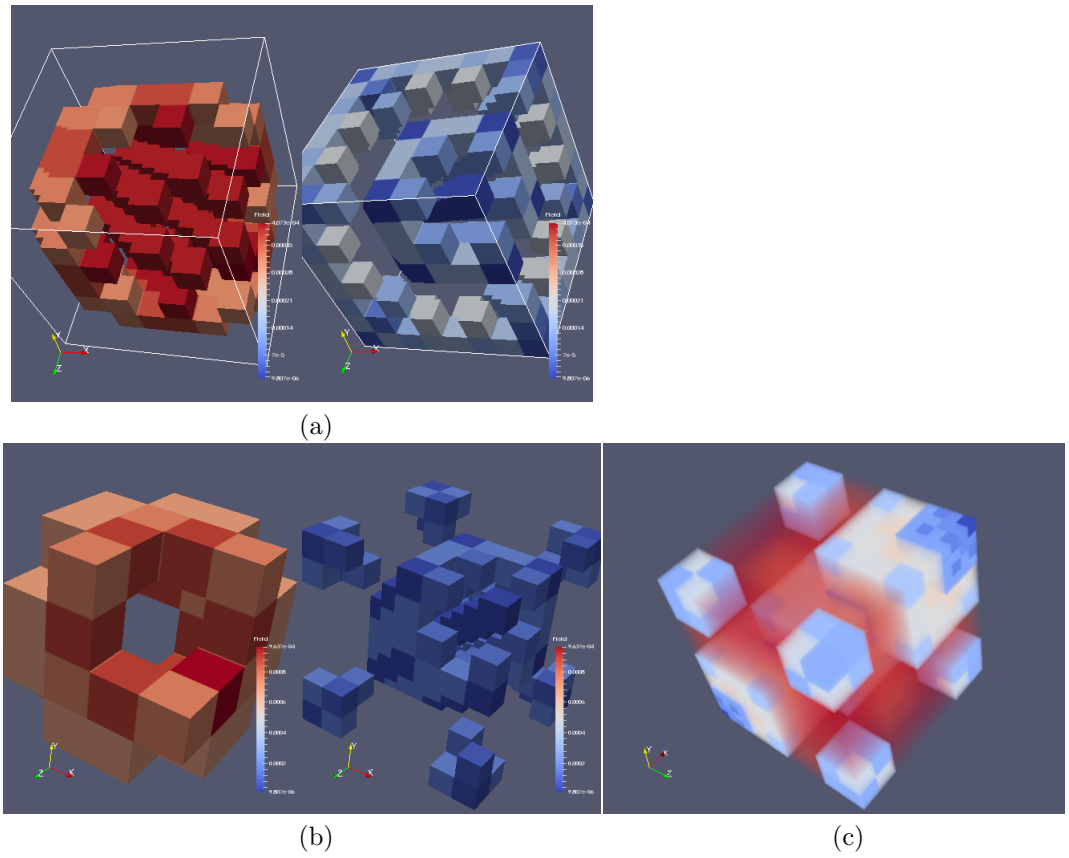


FIG. 1. *Coulomb 3D case: a) error distribution per subtensor after the greedy step (512 subtensors); b) error distribution per subtensor after the first merge step; c) error distribution in the final tensor in partitioning format.*

small tensors, since it tends to stop when it encounters a high rank subtensor inside a domain. Considering that a HOSVD should be computed for every merge step, decreasing the number of merge steps reduces the computational cost tremendously. However, it could also produce suboptimal results due to the premature rejection of a merge. A merge several steps afterwards might outperform the current storage, though in the current step not merging might be better.

At this point, we would like to stress on the fact that this problem could be overcome in principle by using an algorithm which could rapidly compute the HOSVD decomposition of a tensor, knowing the HOSVD decomposition of its subtensors. For the sake of conciseness, we leave this question for future work and do not adress this issue here.

We illustrate in Figure 1 the compression of the 3D Coulomb potential obtained by using Algorithm 4.2. This function is described in more details in section 5.1. The partition tree is an octree, each vertex is associated with a tensor, which is recursively divided into eight subtensors. In this example the recursion stops at depth 2, and the partition tree has 512 leaves and associated subtensors. Figure 1 (a) displays the distribution of the error among subtensors obtained by using PF-Greedy-HOSVD

algorithm, where the subtensors with higher errors are displayed on the left. The result of the first merge step is displayed in Figure 1 (b), while the error distribution in the final tensor in partitioned Tucker format is displayed in Figure 1 (b). As expected in this case, the subtensors along the superdiagonal are not merged since they have higher ranks, while subtensors further away from the superdiagonal are merged into larger subtensors since they have smaller ranks.

Algorithm 4.2 MERGE

Input:
$\mathcal{A} \in \mathbb{R}^{\boldsymbol{I}} \leftarrow$ a tensor of order d
$T_{\boldsymbol{I}}$ an initial partition tree of $\boldsymbol{I}$
A set of leaf errors $(\varepsilon^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})}$
A set of leaf ranks $(\boldsymbol{R}^{\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}})} \subset \mathbb{N}^d$
$\boldsymbol{J}_0 \in \mathcal{V}(T_{\boldsymbol{I}}) \setminus \mathcal{L}(T_{\boldsymbol{I}})$.
Output:
$T_{\boldsymbol{I}}^{\text{fin}}$ a final partition tree of $\boldsymbol{I}$
merge a boolean indicating if the tree has been merged or not
A set of leaf errors $(\varepsilon^{\text{fin},\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}$
A set of leaf ranks $\left(\boldsymbol{R}^{\text{fin},\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})} \subset \mathbb{N}^d$
Begin:
Set $\mathcal{P}_{\boldsymbol{J}_0} := \mathcal{D}_{\boldsymbol{J}_0}(T_{\boldsymbol{I}}) \cap \mathcal{L}(T_{\boldsymbol{I}})$. From Lemma 4.3 (iii), $\mathcal{P}_{\boldsymbol{J}_0}$ is an admissible partition of the set $\boldsymbol{J}_0$.
Set $M_{\text{nomerge}} := \sum_{\boldsymbol{J}\in\mathcal{P}_{\boldsymbol{J}_0}} M_{\text{TF}}\left(\boldsymbol{J}, \boldsymbol{R}^{\boldsymbol{J}}\right)$
Set $\eta := \sqrt{\sum_{\boldsymbol{J}\in\mathcal{P}_{\boldsymbol{J}_0}} |\varepsilon^{\boldsymbol{J}}|^2}$.
Compute $\boldsymbol{R} =$ Greedy-HOSVD($\mathcal{A}^{\boldsymbol{J}_0}$, η)
Compute $M_{\text{merge}} := M_{\text{TF}}\left(\boldsymbol{J}_0, \boldsymbol{R}\right)$.
if $M_{\text{merge}} < M_{\text{nomerge}}$ **then**
    Set *merge* = true, $T_{\boldsymbol{I}}^{\text{fin}} = T_{\boldsymbol{I}}^{m}(\boldsymbol{J}_0)$.
    for $\boldsymbol{J} \in \mathcal{L}\left(T_{\boldsymbol{I}}^{\text{fin}}\right)$ **do**
        if $\boldsymbol{J} = \boldsymbol{J}_0$ **then**
            Set $\boldsymbol{R}^{\text{fin},\boldsymbol{J}_0} := \boldsymbol{R}$ and $\varepsilon^{\text{fin},\boldsymbol{J}_0} := \eta$.
        else
            From Lemma 4.3 (v), necessarily $\boldsymbol{J} \in \mathcal{L}(T_{\boldsymbol{I}})$.
            Set $\boldsymbol{R}^{\text{fin},\boldsymbol{J}} := \boldsymbol{R}^{\boldsymbol{J}}$ and $\varepsilon^{\text{fin},\boldsymbol{J}} := \varepsilon^{\boldsymbol{J}}$.
else
    Set *merge* = false, $T_{\boldsymbol{I}}^{\text{fin}} = T_{\boldsymbol{I}}$.
    for $\boldsymbol{J} \in \mathcal{L}\left(T_{\boldsymbol{I}}^{\text{fin}}\right) = \mathcal{L}(T_{\boldsymbol{I}})$ **do**
        Set $\boldsymbol{R}^{\text{fin},\boldsymbol{J}} := \boldsymbol{R}^{\boldsymbol{J}}$ and $\varepsilon^{\text{fin},\boldsymbol{J}} := \varepsilon^{\boldsymbol{J}}$.
    return $T_{\boldsymbol{I}}^{\text{fin}}$, *merge*, $(\varepsilon^{\text{fin},\boldsymbol{J}})_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}$, $\left(\boldsymbol{R}^{\text{fin},\boldsymbol{J}}\right)_{\boldsymbol{J}\in\mathcal{L}(T_{\boldsymbol{I}}^{\text{fin}})}$

5. Numerical experiments. In this section, some numerical experiments are proposed to assess the properties of the algorithms presented above in terms of memory compression of a given tensor. Three different tests are presented: the first and the second examples are potentials whose expression is known in analytic form, the Coulomb and the Gibbs potential, for which we will present tests in $d = 2, 3$.

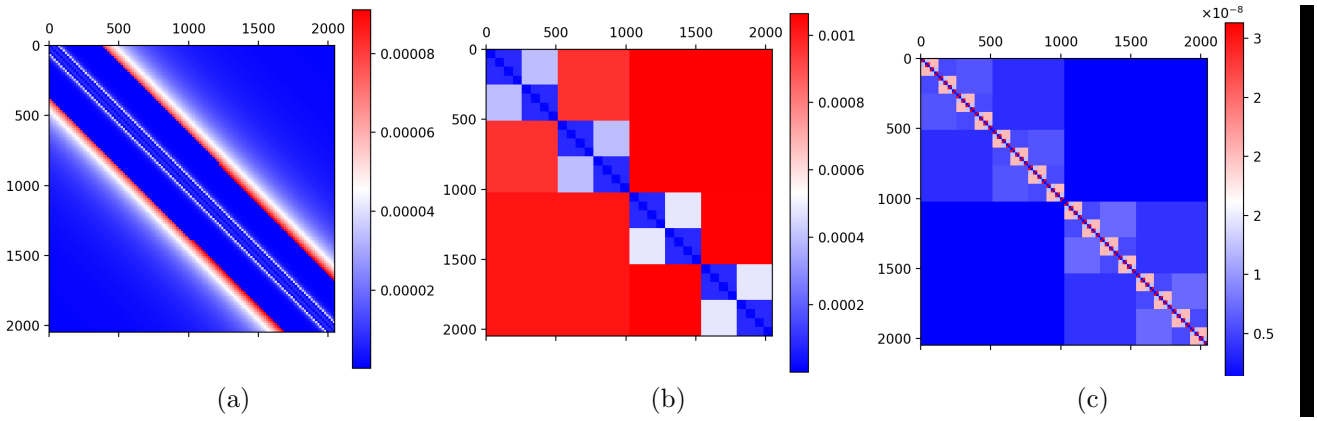


FIG. 2. *Coulomb 2D case, section 5.2: a) error distribution per subtensor after the greedy step (16384 subtensors); b) error per subdomain after the optimisation; c) error normalised with the subtensor size.*

These are two examples of multi-variate functions that can hardly be represented in separated form.

The last test case is a perspective on the use of the proposed method to compress solutions of high-dimensional Partial Differential Equation: a Vlasov-Poisson solution is presented.

For all the figures presented below, let $\mathcal{F}^*$ denote the function to be approximated and $\mathcal{F}$ its compression, the errors shown are defined as:

$$e_i = \left(\int_{\mathbf{\Omega}^i} (\mathcal{F}^* - \mathcal{F})^2 \, dx \right)^{1/2}, \tag{5.1}$$

$$\bar{e}_i = \frac{e_i}{\mu(\mathbf{\Omega}_i)}, \tag{5.2}$$

so that the error plotted in a subdomain is constant over the subdomain and it represents the total $L^2(\mathbf{\Omega}^i)$ error achieved by the tensor approximation. When we plot the error relative to the volume (denoted by *Rel. error* in the fire for the $d = 3$ test cases) we show the quantity $\bar{e}_i$ which is the error in the subdomain renormalised with the volume of the subdomain.

5.1. Coulomb potential. The Coulomb potential is a function $V : \mathbb{R}^d \to \mathbb{R}^+$ which has the following expression:

$$V(x_1, \dots, x_d) = \sum_{1 \le i < j \le d} \frac{1}{|x_i - x_j|}. \tag{5.3}$$

This can be interpreted, from a physical standpoint, as the electrostatic potential generated by a number of fix charges.

5.1.1. 2D cases. The tests on a 2D Coulomb potential are presented.

The results in terms of memory needed in order to store the potential are presented in Table 1, for different values of the accuracy ε and of tree depth ℓ. The first set of tests was performed with a discretisation of the Coulomb potential with $n_i = 2^8$, $i = 1, 2$ degrees of freedom per direction. The total storage (denoted by *Full* in Table1) is 2^{16} doubles. The Coulomb potential is a function for which the classical HOSVD

ε	ℓ	Full	HOSVD	Greedy	Hierarchical
10^{-1}	2	$6.55 \cdot 10^4$	$--$	$3.03 \cdot 10^4$	$3.03 \cdot 10^4$
	3	$6.55 \cdot 10^4$	$--$	$1.73 \cdot 10^4$	$1.69 \cdot 10^4$
	4	$6.55 \cdot 10^4$	$--$	$1.14 \cdot 10^4$	$1.05 \cdot 10^4$
10^{-2}	2	$6.55 \cdot 10^4$	$--$	$3.52 \cdot 10^4$	$3.52 \cdot 10^4$
	3	$6.55 \cdot 10^4$	$--$	$2.09 \cdot 10^4$	$2.01 \cdot 10^4$
	4	$6.55 \cdot 10^4$	$--$	$1.67 \cdot 10^4$	$1.36 \cdot 10^4$
10^{-3}	2	$6.55 \cdot 10^4$	$--$	$3.69 \cdot 10^4$	$3.69 \cdot 10^4$
	3	$6.55 \cdot 10^4$	$--$	$2.29 \cdot 10^4$	$2.28 \cdot 10^4$
	4	$6.55 \cdot 10^4$	$--$	$1.87 \cdot 10^4$	$1.64 \cdot 10^4$
10^{-5}	7	$4.19 \cdot 10^6$	$--$	$7.85 \cdot 10^5$	$2.42 \cdot 10^5$

TABLE 1

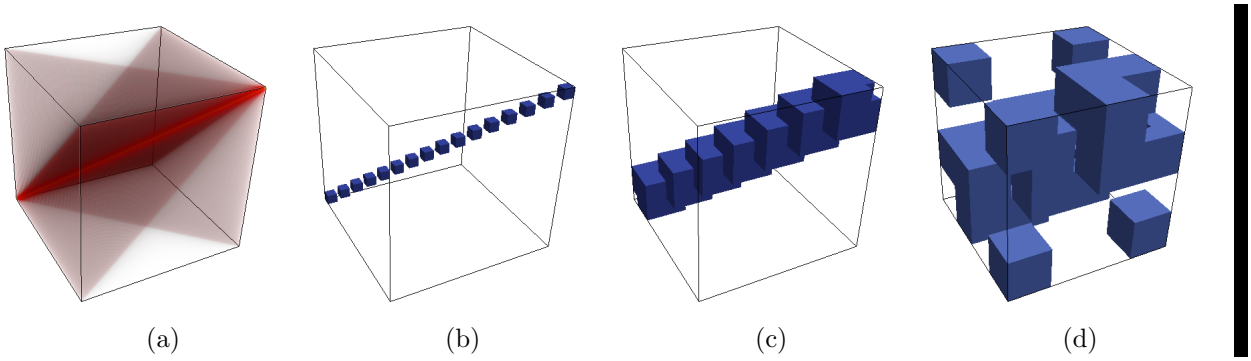
2D Coulomb testcase, Section 5.1: the table presents the memory storages of the full tensor, the one achieved by classical HOSVD, by the first step of the proposed method (Greedy) and by the optimised hierarchical construction.

algorithm (that reduces to classical SVD for $d = 2$) does not make it possible to have a storage smaller than the full tensor. On the contrary, the proposed strategy is quite effective, as it can be seen in the last two columns of the table. As expected, when the required accuracy is increased, the memory needed increases too, at constant tree depth. When the tree depth is increased, the compression rate is improved, and this is related to the fact that the representation adapts better to the function at hand.

Another test is performed (reported in the last line of the table), with an error threshold of $\varepsilon = 10^{-5}$ and a tree depth of $\ell = 7$. The number of degrees of freedom per direction is $n_i = 2^{11},\ i = 1, 2$. The total storage is henceforth of 2^{22} doubles. The compression achieved by the hierarchical method is of about 5% of the total storage, whereas the classical HOSVD is at 100%. The results for this test are represented in Fig.2. At the left, the distribution of the errors after the greedy phase, in which all the subtensors are of equal size. At the center and on the left, the error after the optimisation of the subtensors. The total errors are smaller in the small subdomains. If we look at the errors renormalised with respect to the subtensor size, we can see that the error is still higher where the function is more difficult to be represented well in separated form up to a threshold of ε, but in general it can be stated that the distribution of the renormalised errors is more uniform. This is also reflected in the ranks of the approximation with respect to the total number of elements inside the subtensor: after the optimisation it tends to be more uniform (and as low as possible, hence optimising the storage).

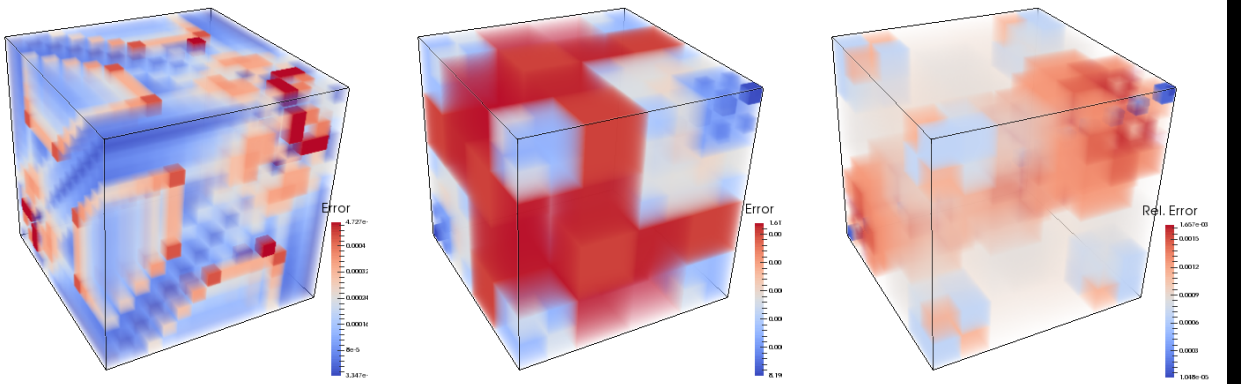
5.1.2. 3D cases. Some tests in $d = 3$ are presented.

The resolution of the tensor considered is $n_i = 2^8,\ i = 1, 2, 3$ degrees of freedom per direction. The maximal tree depth is chosen to be $\ell = 5$ that corresponds to subdivide the tensor into 2^{15} subtensors. As for the case $d = 2$ presented above, the classical HOSVD cannot achieve a compression for such a function. After the greedy algorithm in the first phase, the compression rate is of about 18% and after optimisation, the memory required to guarantee $\varepsilon = 10^{-5}$ is $\sim 7\%$ of the full tensor storage. In Fig.3.a) the tensor entries are represented. In the same figure, the subtensors of small, medium and large size are represented in Fig.3.b-c-d) respectively. As it can be seen, the subtensors size chosen automatically by the method follows, in some sense, the tensor entries structure. The smaller in size subtensors are located along the



(a) (b) (c) (d)

FIG. 3. *Coulomb potential, section 5.1: (a) the tensor entries, in red the largest entries; (b) the small size subtensors, (c) and (d) the mid size and the larger size subtensors. The largest subtensors are in the complement of the cube.*



(a) (b) (c)

FIG. 4. *Coulomb 3D case, section 5.1:*

principal diagonal of the tensor, and the size is increased as we moved away from the diagonals. The errors distribution is represented in Fig.4. The observed behaviour is the same commnted for the $d = 2$ test case presented above.

5.2. Gibbs potential. The Gibbs potential is a function $G : \mathbb{R}^d \to \mathbb{R}^+$ that has the following expression:

$$G(x_1, \dots, x_d) = \exp(-\beta V(x_1, \dots, x_d)), \tag{5.4}$$

$$V(x_1, \dots, x_d) = \sum_{1 \le i < j \le d} V_{ij}(|x_i - x_j|), \tag{5.5}$$

$$V_{ij}(r) = \frac{a_{ij}}{r^6} - \frac{b_{ij}}{r^{12}}, \ \forall 1 \le i < j \le d. \tag{5.6}$$

5.2.1. 2D cases. The tests detailed in the present section are performed in the following 2D configuration: let $\Omega = [-2, 2]^2$, for nitrogen and oxygen atoms.

The results of the tests are reported in Table 2, for different values of the accuracy and tree depth. The observed trend is similar to the one previously commented for

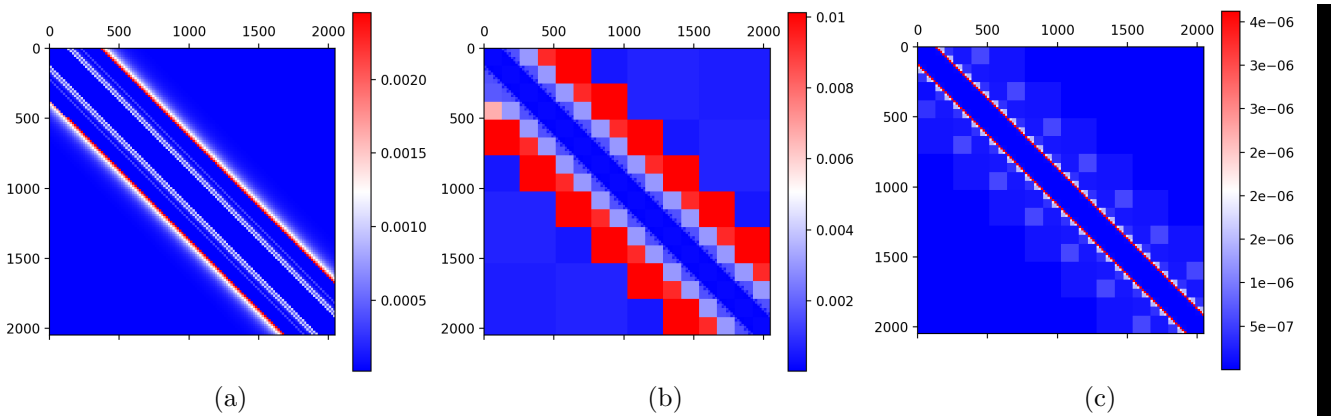


FIG. 5. *Gibbs 2D case, section 5.2:*

ε	ℓ	Full	HOSVD	Greedy	Hierarchical
10^{-1}	2	$6.55 \cdot 10^4$	−−	$4.10 \cdot 10^3$	$5.12 \cdot 10^2$
	3	$6.55 \cdot 10^4$	−−	$5.31 \cdot 10^3$	$5.12 \cdot 10^2$
	4	$6.55 \cdot 10^4$	−−	$8.20 \cdot 10^3$	$5.12 \cdot 10^2$
10^{-2}	2	$6.55 \cdot 10^4$	−−	$1.82 \cdot 10^4$	$1.82 \cdot 10^4$
	3	$6.55 \cdot 10^4$	−−	$1.62 \cdot 10^4$	$1.54 \cdot 10^4$
	4	$6.55 \cdot 10^4$	−−	$1.48 \cdot 10^4$	$1.21 \cdot 10^4$
10^{-3}	2	$6.55 \cdot 10^4$	−−	$3.29 \cdot 10^4$	$3.29 \cdot 10^4$
	3	$6.55 \cdot 10^4$	−−	$2.82 \cdot 10^4$	$2.75 \cdot 10^4$
	4	$6.55 \cdot 10^4$	−−	$2.14 \cdot 10^4$	$1.84 \cdot 10^4$
10^{-5}	7	$4.19 \cdot 10^6$	−−	$6.93 \cdot 10^5$	$2.38 \cdot 10^5$

TABLE 2

2D Gibbs testcase, Section 5.2: the table presents the memory storages of the full tensor, the one achieved by classical HOSVD, by the first step of the proposed method (Greedy) and by the optimised hierarchical construction.

the 2D Coulomb test case. Concerning the HOSVD, no gain in memory was possible with respect to the full tensor, and this is due, as for the Coulomb potential, to the structure of the larger entries, which follow a pattern aligned with the diagonals of the different directions. There are some differences with respect to the Coulomb test case: when an accuracy $\varepsilon = 10^{-1}$ is considered, subdividing more in the greedy phase of the algorithm is not effective and it results in an increased memory, as it can be seen in the lines $\varepsilon = 10^{-1}$. This is because the fine structures characterising the solution have a norm which is less than the error threshold, or, otherwise stated, we are looking for a (too) coarse approximation. When the accuracy is increased to $\varepsilon = 10^{-2}$ and beyond, we recover the expected behaviour as function of ε, ℓ. When $n_i = 2^{11}$, $i = 1, 2$, and $\ell = 7$ levels are used, the optimisation of the hierarchical structure improves of a factor ~ 3 the storage achieved by the greedy phase of the method, and the storage needed to guarantee a precision of $\varepsilon = 10^{-5}$ is of about 5% of the storage of the full tensor.

5.2.2. 3D cases. The case $d = 3$ shown is performed by considering $n_i = 2^8$, $i = 1, 2, 3$, which corresponds to a full storage of 2^{24} doubles. The tree considered has a maximal depth of $\ell = 5$, that corresponds to an initial partition of the tensor into 2^{15}

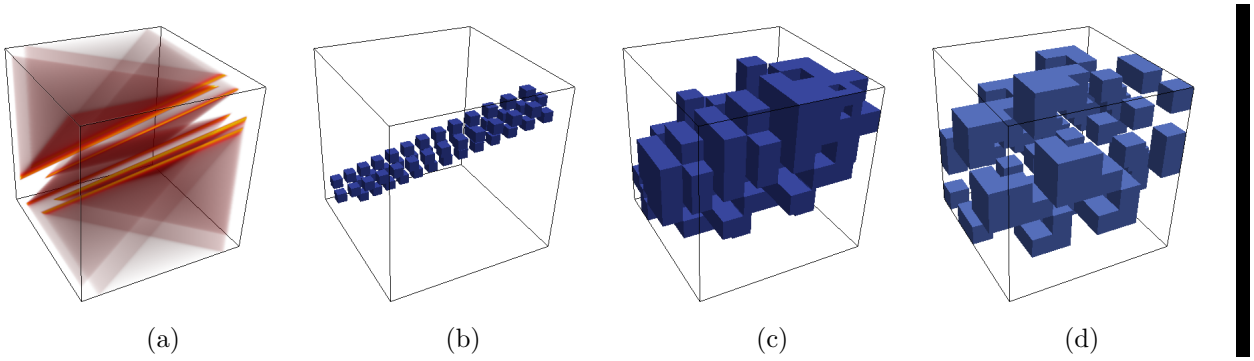


FIG. 6. *Gibbs potential, section 5.2: (a) the tensor entries, in red the largest entries; (b) the small size subtensors, (c) and (d) the mid size and the larger size subtensors. The largest subtensors are in the complement of the cube.*

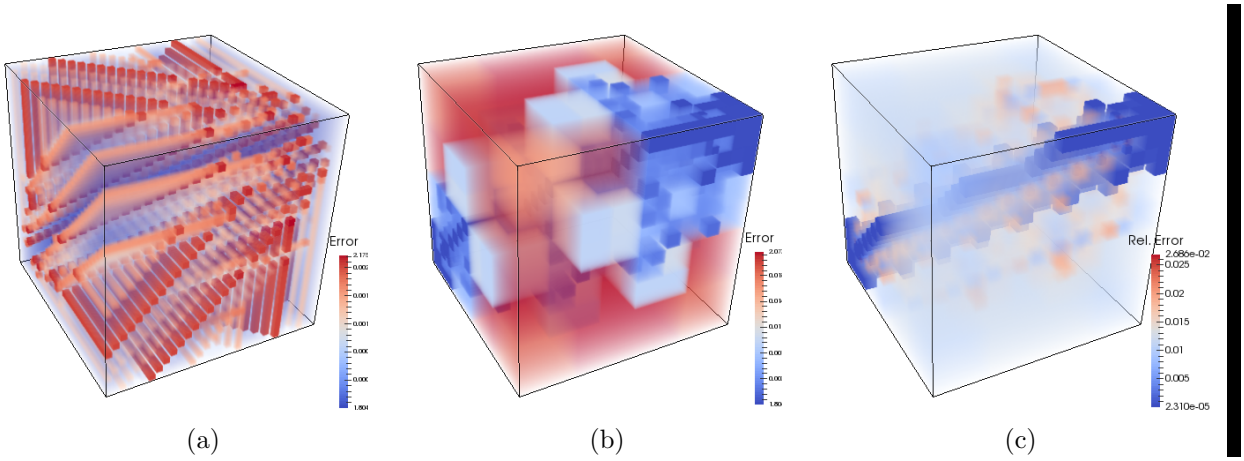


FIG. 7. *Gibbs 3D case, section 5.2:*

subtensors. To get an approximation with an accuracy of $\varepsilon = 10^{-5}$, the compression rate after the greedy phase is of 16%, which is improved by the optimisation of the subtensors size, to achieve a final memory of $\sim 8\%$ of the full tensor storage.

5.3. Vlasov-Poisson solution. The Vlasov-Poisson equation describes the probability density of finding a particle in a given position-momentum of the phase space, at a certain time, and it is used as a model, in kinetic theory, to describe collisionless plasmas. Models in kinetic theory are a class of high-dimensional problems for which the present approach could be of interest in terms of compression of a given simulation.

As for the other tests presented, the method follows the structure of the solution in order to adapt the subtensor sizes. This has the effect of redistributing the errors and hence to achieve a better compression rate.

6. Conclusions and perspectives. A method is proposed to construct a piecewise tensor adaptive compression. The partition in subtensors is not fixed *a priori*, but it is, instead, a result of the proposed method. In this work two contributions are described: the first one consists in a greedy method that, given a partition, constructs

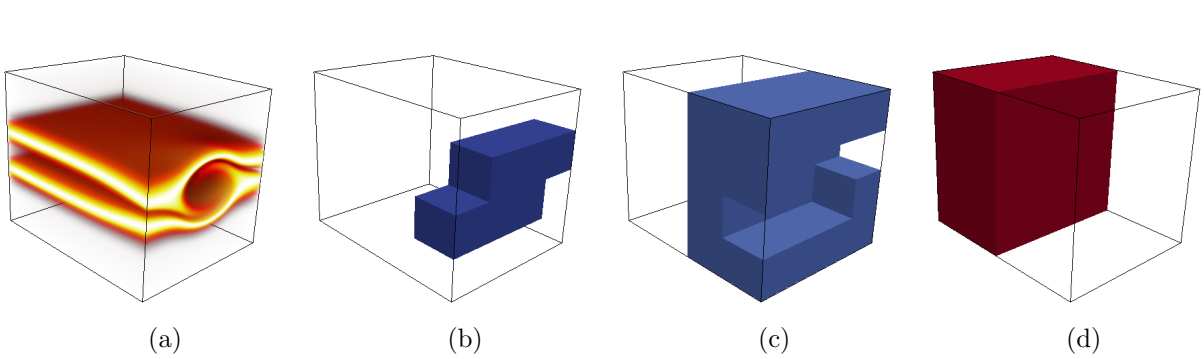


FIG. 8. *Vlasov-Poisson solution, section 5.3: (a) the tensor entries, in red the largest entries; (b) the small size subtensors, (c) and (d) the mid size and the larger size subtensors. The largest subtensors are in the complement of the cube.*

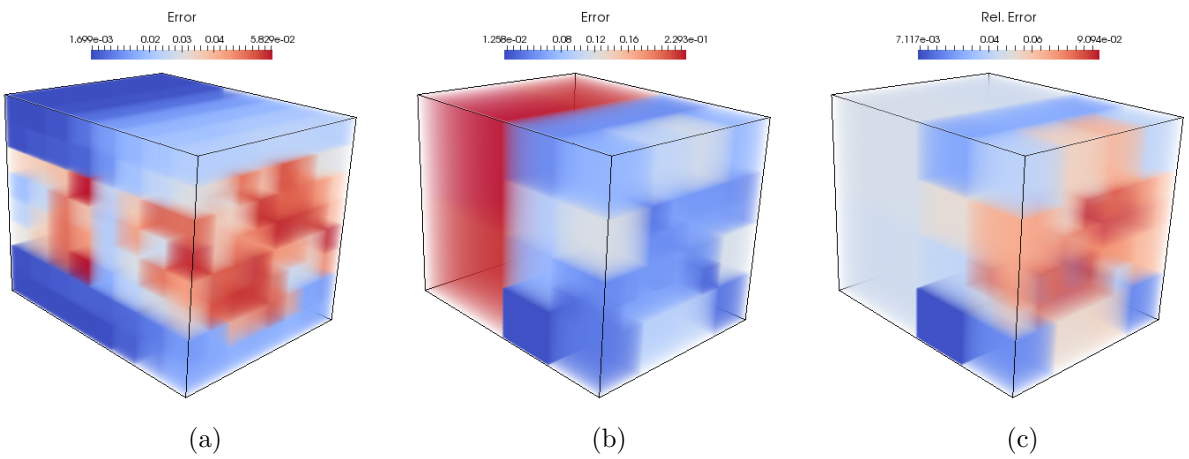


FIG. 9. *Vlasov-Poisson case, section 5.3:*

a parsimonious piece-wise tensor approximation such that a prescribed accuracy on the approximation of the whole tensor is fulfilled; the second one consists in an algorithm that defines a partition tree, to adapt the subtensor partition and improve the storage. Several numerical experiments are proposed to assess the performances of the method, which is suitable, at present, to moderate order tensors. The main perspectives are the improvement of the efficiency of the partition tree construction and the extension of the piece-wise tensor approximation to large order tensors.

REFERENCES

[1] M. BACHMAYR, R. SCHNEIDER, AND A. USCHMAJEW, *Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations*, Foundations of Computational Mathematics, 16 (2016), pp. 1423–1472.
[2] J. BALLANI AND L. GRASEDYCK, *Hierarchical tensor approximation of output quantities of parameter-dependent PDEs*, SIAM/ASA Journal on Uncertainty Quantification, 3 (2015), pp. 852–872.
[3] M. BEBENDORF, *Hierarchical matrices*, Springer, 2008.
[4] M. BEBENDORF, *Hierarchical matrices*, Springer, 2008.

[5] D. Edmunds and W. D. Evans, *Spectral theory and differential operators*, Oxford University Press, 2018.
[6] D. Edmunds and J. SUN, *Approximation and entropy numbers of sobolev embeddings over domains with finite measure*, The Quarterly Journal of Mathematics, 41 (1990), pp. 385–394.
[7] V. Ehrlacher and D. Lombardi, *A dynamical adaptive tensor method for the vlasov–poisson system*, JCP, 339 (2017), pp. 285–306.
[8] A. Gorodetsky, S. Karaman, and Y. Marzouk, *A continuous analogue of the tensor-train decomposition*, Computer Methods in Applied Mechanics and Engineering, 347 (2019), pp. 59–84.
[9] L. Grasedyck, D. Kressner, and C. Tobler, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitteilungen, 36 (2013), pp. 53–78.
[10] W. Hackbusch, *Tensor spaces and numerical tensor calculus*, vol. 42, Springer Science & Business Media, 2012.
[11] W. Hackbusch, *Hierarchical matrices: algorithms and analysis*, vol. 49, Springer, 2015.
[12] W. Hackbusch and B. N. Khoromskij, *Tensor-product approximation to operators and functions in high dimensions*, Journal of Complexity, 23 (2007), pp. 697–714.
[13] B. Khoromskij, *Structured data-sparse approximation to high order tensors arising from the deterministic boltzmann equation*, Mathematics of computation, 76 (2007), pp. 1291–1315.
[14] B. N. Khoromskij, *Tensors-structured numerical methods in scientific computing: Survey on recent advances*, Chemometrics and Intelligent Laboratory Systems, 110 (2012), pp. 1–19.
[15] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM review, 51 (2009), pp. 455–500.
[16] D. Kressner and C. Tobler, *Low-rank tensor krylov subspace methods for parametrized linear systems*, SIAM Journal on Matrix Analysis and Applications, 32 (2011), pp. 1288–1316.
[17] I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.
[18] C. Schwab and C. J. Gittelson, *Sparse tensor discretizations of high-dimensional parametric and stochastic PDEs*, Acta Numerica, 20 (2011), pp. 291–467.