Coupling-Aware Graph Partitioning Algorithms: Preliminary Study

Maria Predari, Aurélien Esnard

Coupling-Aware Graph Partitioning Algorithms: Preliminary Study

Maria Predari and Aurélien Esnard
Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France.
CNRS, LaBRI, UMR 5800, F-33400 Talence, France.
HiePACS Project, INRIA, F-33400 Talence, France.

***Abstract*—In the field of scientific computing, load balancing is a major issue that determines the performance of parallel applications. Nowadays, simulations of real-life problems are becoming more and more complex, involving numerous coupled codes, representing different models. In this context, reaching high performance can be a great challenge. In this paper, we present graph partitioning techniques, called *co-partitioning*, that address the problem of load balancing for two coupled codes: the key idea is to perform a "coupling-aware" partitioning, instead of partitioning these codes independently, as it is usually done. Finally, we present a preliminary experimental study which compares our methods against the usual approach.**

I. Introduction

One of the key issues in the field of scientific computing is the load balancing for distributed computing, which highly determines the performance of parallel programs. A general strategy is to apply a static balancing algorithm before running the parallel program, that equilibrates the computational load among the available processors.

Graph theory can be used to solve the problem of load balancing, with graph partitioning [1]. More precisely, each vertex of the graph represents a basic computational task of the program (related to a mesh element) and has a weight proportional to the task's cost. Besides, each edge represents a dependency in calculations between two tasks assigned on different processors and has a weight proportional to the communication costs between the processors.

In order to balance the load among N processors, one performs a *graph partitioning* in N parts, each part being assigned to a processor. The main objective of graph partitioning is to divide the given graph into N smaller parts (vertex subsets), such that they have roughly equal computational loads, and a minimal number of edges cut between them. Thus, graph partitioning appears as a fundamental technique for parallelization, that offers high performance by substantially minimizing the total execution time. Unfortunately, graph partitioning is known to be NP-hard [2], so finding a partition of a given graph is usually based on heuristic techniques or approximating algorithms. Integrated software for graph partitioning exist, such as Metis [3], Jostle [4], Kahip [5] or Scotch [6].

Another crucial issue rising in scientific computing nowadays is that numerical simulations are mixing several models that represent different physics or scales (multiphysics and multiscale models). Here, the key idea is to reuse available legacy codes through a coupling framework instead of merging them into a standalone application. For instance, the simulation of the earth's climate system typically includes coupled interactions between four geophysical components (atmosphere, ocean, land, and sea-ice) [7]. Other examples of applications emerge in the field of aeronautic propulsion where the behavior of hot components is impacted by complex interactions between different physics such as turbulent combustion, radiation and heat conduction. To predict such thermal environments, coupling of these different heat transfer models (combustion, radiation and conduction) is necessary [8]. Combining such different models in massively parallel computations, is still a challenge to reach high performance and scalability. In this context, the load balancing of the overall coupled simulation remains an open question.

In order to solve the load balancing problem for complex, multiphysics or multiscale applications, we use the *coupled model* definition proposed in [9], that describes the general execution of coupled codes. A coupled model consists of a number of component models (or components), that may interact periodically with each other, throughout the coupled simulation, using a coupling framework like for instance CCSM/CESM [7], OASIS [10] or OpenPALM [11]. A component A represents a model that solves an individual system defined in a computational domain Γ_A. Two components A, B interact, and thus are coupled when parts or their entire domains overlap, imposing data dependencies on their models. The overlapping creates a common *coupling interface* $\Omega_{AB} = \Gamma_A \cap \Gamma_B$. An example of a 3D coupling interface is given in Figure 1.

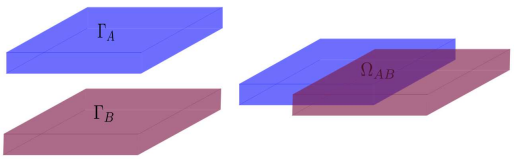


Fig. 1: Example of coupling interface for two rectangular computational domains.

The general model of a coupled simulation with two components is shown in Figure 2. As one can see, a coupled model evolves iteratively in time, entering a sequence of *regular* and *coupling* phases. A regular phase occurs when each component

computes solutions independently, on its own computational domain. The number of processors assigned to A and B in the regular phase is N_A and N_B respectively. During a regular phase, each component model uses a different time and space discretization, so the number of iterations performed by each solver may vary, that is $k_A \neq k_B$. That usually leads to differences in execution time between components at the end of the regular phase, noted as T_{idle} in the figure. It indicates the time the fastest component has to wait until it enters a coupling phase. On the other hand, a coupling phase occurs when components interact with each other, exchanging data on the coupling interface Ω_{AB}. Additionally, during this phase the coupling framework drives the overall application, ensuring synchronization among the components, locating data through partitioned geometries and performing the interpolation step between different meshes. Note that the number of active processors for each component during the coupling phase is often lower, namely $N_A^{cpl} \leq N_A$ and $N_B^{cpl} \leq N_B$. These are the processors of each component that own data on Ω_{AB}.

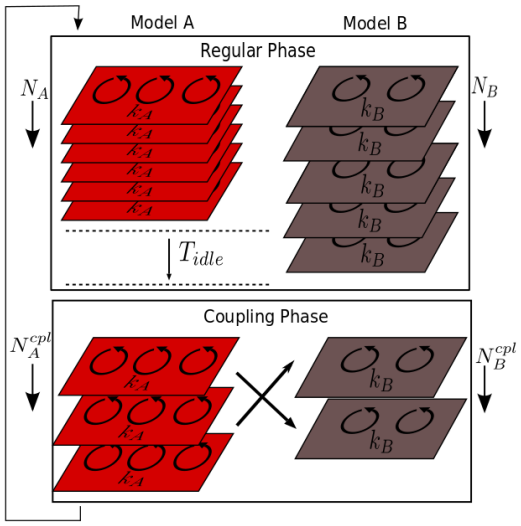


Fig. 2: General model of a complex simulation with two coupled components A and B.

In order to minimize the execution time of a coupled application, we identify two important subproblems:

i) *The resource distribution problem:* Here, the problem is to find a good distribution of the total number of available processors among the components, in order to minimize T_{idle}. The main idea is to assign less processor resources to the fastest component following an empiric approach based on performance studies [12]. That way, the time the fastest component waits before entering a coupling phase is minimized.

ii) *The data distribution problem:* Assuming the number of processors have been selected for each component (by solving the previous problem), we aim to find a good data distribution in order to minimize the total execution time, both in the regular phase and in the coupling phase.

In this paper, we mainly focus on the data distribution problem, assuming the number of processors N_A and N_B for each component is given. Thus, the motivation behind this paper is to solve the load balancing problem using graph partitioning techniques for the whole coupled application and not only for each standalone model. Therefore we propose graph partitioning algorithms that take into consideration the coupling phase and perform a *coupling-aware* partitioning of the whole application; we call them *co-partitioning*.

In section II we present related work on the load balancing problem of coupled applications. Then, in section III we describe the co-partitioning algorithms and in section IV we present some experimental results, comparing our methods with the state-of-the-art approach. Finally, in section V we state the conclusion of our research and our future work.

II. Related Work

The static load balancing problem of a meshed-based application involves dividing the mesh into subdomains at the beginning of the execution and assign them over a set of processors in a distributed environment, so that each processor has about the same amount of computation and the communication costs are minimized.

A great variety of methods that address the static load balancing problem exists in literature, mainly based on graph model. A popular graph partitioning algorithm is the multilevel K-way graph partitioning [13], [14]. The multilevel paradigm consists of three phases: i) coarsening phase, ii) initial partitioning phase, iii) uncoarsening phase. During the coarsening phase, a sequence of successively coarser graphs is constructed from the original one, collapsing vertices using edge matching. In the initial partitioning phase, a direct K-way partition of the coarsest graph is computed, using a conventional partitioning algorithm, like recursive bisection (RB) [15], or K-way greedy graph growing (KGGGP) [16]. Finally, during the uncoarsening phase, the partition of the coarsest graph is projected back and refined using FM-like algorithms [17], until the original graph.

Although the classic load-balancing problem of a standalone simulation is well studied, as far as we know, there are little work that attempt to solve the load-balancing problem on coupled applications. Nevertheless, some interesting, related studies exist and we briefly review them here.

A similar problem to the one we study is the multi-phase mesh partitioning problem. Multi-phase computations consist of several distinct computational phases, each separated by an explicit synchronization step. In this context, one must partition the mesh such that the computation in each phase is balanced, and the communication cost among processors in each phase is minimized. Methods that solve this problem use the multi-constraint partitioning paradigm [18], [19]. At first glance, one could consider the co-partitioning problem as several multi-phase problems, each one representing a different component. However this simple representation is not correct, since it does not include the coupling communications between components which indicates dependencies among the multi-phase problems. Moreover, this approach imposes the use

of all available processors in the coupling phase, that could degrade the global edgecut especially if the coupling interface is small compared to the whole computation domain.

An example of a multiphysics problem is the optimization of gas turbines, as studied in [12]. To improve the results of the standalone simulations that are currently used to optimize gas turbines, they introduce a scalable conjugate heat transfer simulation by coupling a computational fluid dynamics (CFD) code with a thermal conduction simulation. Following a case study, they accelerate the thermal solver by assigning more processors to it, compared to the fluid one. Moreover, they propose a fully distributed coupling methodology in order to manage data transfer, data interpolation and synchronization between different components. Indeed, their results indicate that massively parallel simulations (i.e., until 100000 cores) can be coupled through this framework, without severely damaging the performance of the models.

As we mentioned before, climate modeling is another typical example of coupled simulations which involves several interacting components for atmosphere, oceans, land, and sea-ice. An example of a flexible coupling strategy for the climate model, along with a new coupler (CCSM4) is presented in [7], [20]. The coupler contains a top level driver that calls model components and coordinates their time sequence to provide greater flexibility in processor layouts. More precisely, the coupler dynamically adapts the number of processors that will be assigned to each component, by iteratively improve the processors' layout, until the time execution of all components is almost equal. Typically, the tunning step only needs a few runs to produce a good configuration with relative small idle time.

Despite the performance gains obtained by the above studies, it mainly addresses the resource distribution problem and not the data distribution problem we focus on. In this paper, we investigate new techniques to overcome the load balancing issues of coupled simulations. Thus, we present new graph partitioning algorithms that are aware of the coupling process, in order to minimize the overall execution time.

III. Algorithm

In this section, we first give some background definitions and a formal statement of the co-partitioning problem. Then we introduce some graph operators based upon which we describe our co-partitioning algorithms, called *AWARE* and *PROJREPART*, and the *NAIVE* method used as the state-of-the-art solution.

A. Background definitions

Consider a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. Each vertex $u \in V$ has a weight $w(u)$ representing the computational load at each processor, while each edge $e \in E$ has a weight $w(e)$ representing the communication cost between different processors. $P = (V_1, V_2, \ldots, V_K)$ is a K-way partition of G if the following conditions hold: each part $V_k, 1 \leq k \leq K$ is a non empty subset of V, parts are pairwise disjoint ($V_k \cap V_l = \emptyset$ for all $1 \leq k, l \leq K$) and union of K parts is equal to V. Given a vertex v mapped to the part V_k, one notes $P[v] = k$ its part number. A partition is said to be balanced if each part V_k satisfies *the balance criterion*:

$$W_k \leq W_{avg}(1 + \epsilon) \text{ for } k = 1, \ldots, K. \quad (1)$$

In 1, weight W_k of part V_k is defined as the sum of the weights of vertices in that part. $W_{avg} = \sum_{u_i \in V} w(u_i)/K$ represents the weight of each part in G under the perfect load balance, and ϵ denotes the maximum imbalance tolerance allowed. A typical imbalance tolerance is $\epsilon = 5\%$ of the ideal weight. The *edgecut* of a partition is said to be the weight of all edges whose pair of vertices belong to two different parts. The edgecut definition for representing the cost of a partition is $\sum_{e \in \mathcal{F}}(w(e))$, where $\mathcal{F} = \{(a, b) \in E, a \in V_i \wedge b \in V_j \wedge i \neq j\}$. This classical edgecut metric is known to approximate the total communication volume [21].

Hence, the graph partitioning problem can be defined as the task of dividing a graph into a number of parts in a way that the edgecut is minimized while the balance criterion 1 for all parts is maintained.

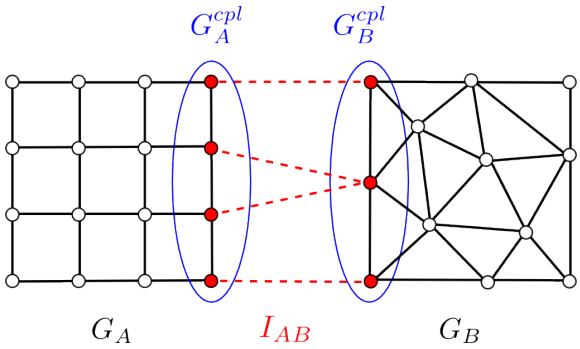


Fig. 3: Example of a global graph G_{AB} based on two graphs G_A and G_B and interedges I_{AB} (in dashed red), showing the coupled subgraphs G_A^{cpl} and G_B^{cpl} (circled in blue).

B. The co-partitioning problem

In the case of coupled models, where more than one component model is involved, we propose to enrich the classic graph model to take into account the coupling phase of the simulation explicitly. For reasons of simplicity in the remainder of the paper, we assume that only two models are involved in the coupled simulation, though this can clearly be extended to a larger number of components. Subscript letters A, B represent component models, while the exponent notation cpl denotes the coupling phase.

Let us consider that each individual component involved in the simulation, is represented by a graph, that is $G_A = (V_A, E_A)$ for model A and $G_B = (V_B, E_B)$ for B. As shown in Figure 3, we represent the coupled models with the *global graph*, $G_{AB} = (V_A \cup V_B, E_A \cup E_B \cup I_{AB})$, where I_{AB} is the set of weighted edges that interconnect some vertices of V_A and V_B. We call those edges *interedges*[1] (as they connect vertices

[1]In other words, the interedges I_{AB} just represent a binary relation from V_A to V_B.

from different graphs), and they represent the communication costs of exchanging data that belong to different components during the coupling phase. In practice, these interedges are located thanks to geometric intersection between mesh cells of A and B.

Moreover, we can define the *coupled graph* G_{AB}^{cpl} as the subgraph of G_{AB} whose vertex set includes only vertices that are interedge endpoints. Likewise we can obtain the *coupled subgraphs* G_A^{cpl} and G_B^{cpl} from G_A and G_B respectively.

In this context, we aim to find a partitioning of the global graph that is aware of the coupling subgraph. We call this problem *the co-partitioning problem*. More precisely, a K-way partition P_{AB} of the global graph G_{AB} is said to be a $(N_A, N_B, N_A^{cpl}, N_B^{cpl})$-way *co-partition*, if the following conditions hold:

- P_A is a N_A-way balanced partition of G_A,
- P_B is a N_B-way balanced partition of G_B,
- P_A^{cpl} is a N_A^{cpl}-way balanced partition of G_A^{cpl},
- P_B^{cpl} is a N_B^{cpl}-way balanced partition of G_B^{cpl},
- $P_{AB} = P_A \cup P_B$,
- $K = N_A + N_B$,
- $N_A^{cpl} \leq N_A$ and $N_B^{cpl} \leq N_B$.

Following the above definition, we expect that such a co-partition will provide a good load balancing for both the regular phase and the coupling phase, and for both components A and B, since it explicitly finds a partition for the graphs that participate in the regular phase (G_A and G_B) and their subgraphs in coupled phase (G_A^{cpl} and G_B^{cpl}). As an additionnal criterion, we aim to minimize the inter-component communication cost (represented by the weighted interedges), which implies to minimize the total number of messages ($totZ$) and the total volume of data exchanged ($totV$) during the coupling phase.

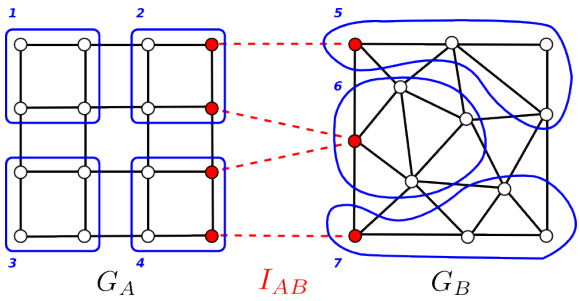


Fig. 4: Example of $(4, 3, 2, 3)$-way co-partition of G_{AB}.

Figure 4 depicts a $(4, 3, 2, 3)$-way co-partition. Assuming all vertex/edge weights are 1, we see that the balance criterion is perfectly respected in the regular phase for both P_A and P_B ($N_A = 4$ and $N_B = 3$), with an edgecut respectively equal to 8 and 12. As for the coupling phase (red vertices), P_A^{cpl} and P_B^{cpl} are perfectly balanced as well ($N_A^{cpl} = 2$ and $N_B^{cpl} = 3$), with an edgecut equal to 1 and 2 respectively. Considering the coupling communication, $V_{tot} = 4$ and $Z_{tot} = 4$, such that we have the following processor pairs implied: $(2, 5), (2, 6), (4, 6), (4, 7)$.

C. Graph operators

Subsequently, we describe in details the basic steps of the proposed algorithms. To facilitate this description, we introduce some graph operators that produce new graph structures or partitions. Namely, we use the following operators: *partition, repartition, restriction, extension* and *projection*.

a) Partition: First, we define the partition operator as $Part(G, K) \rightarrow P$ which simply returns a K-way balanced partition of the graph G (with respect to an imbalance tolerance ϵ).

b) Repartition: Then, the repartition operator is defined as $Repart(G, P, M, N) \rightarrow P'$. This operator computes a new N-way balanced partition of a graph G using a former (possibly unbalanced) M-way partition of the same graph, such that the migration volume is minimized as an additional criterion. This repartition operator uses the $M \times N$ repartitioning algorithm presented in [22], that contrarily to classic repartitioning algorithms (scratch-remap [23], diffusion [24], [25] or biased partitioning [26], [27]) enables to change the target number of parts (i.e., $N \neq M$). The $M \times N$ repartitioning algorithm constructs, with a greedy strategy, a good migration matrix[2] C, that minimizes both the total migration volume ($totV$) and the total number of messages exchanged during migration ($totZ$). Then it performs a biased partitioning of the graph G, enriched with N additional fixed vertices[3] connected to the M former parts based on the migration matrix C (Fig. 5). More details on this operator and its implementation can be found in [28], [22].

c) Restriction: The restriction operator can be expressed as $Rest(G, V') \rightarrow G'$, and returns a subgraph G', that is the restriction of $G = (V, E)$ to vertex set $V' \subset V$. Given the interedges I_{AB} and the graph G_A, we will use this operator to compute the coupled subgraph G_A^{cpl} as $Rest(G_A, Dom(I_{AB}))$ where $Dom(I_{AB}) = \{u_a \in V_A, (u_a, u_b) \in I_{AB}\}$ is the departure domain of I_{AB}. The Figure 6 illustrates this operator on two cubic meshes with a surface coupling.

d) Extension: Next, we define the extension operator $Ext(G, G', P', K, L) \rightarrow P$, which returns a L-way partition of a graph G using a given K-way partition P' of a subgraph $G' \subset G$, where $L \geq K$. The idea behind this operator is to extend a given partition P_A^{cpl} of the coupled subgraph G_A^{cpl} to the graph G_A, such that vertices already assigned to a part in P_A^{cpl} remain fixed in the new partition P_A, as shown in Figure 7. This method is trivially implemented by using a partitioning routine that handle fixed vertices.

e) Projection: Finally, we define the projection operator, $Proj(G, G', I, P, K) \rightarrow P'$, that finds a K-way partition of a graph G', using a given K-way partition of a graph G and the interedges $I \subset V \times V'$, with $G = (V, E)$ and $G' = (V', E')$. The key idea of the projection operator is to compute a *similar* partition of G on G' using the relation provided by

[2]A migration matrix $C = (C_{i,j})$ of dimension $M \times N$ represents the amount of data that migrates from a *former* part i to a *newer* part j (if $i \neq j$) or the amount of data that remains in place (if $i = j$).

[3]When computing a partition, fixed vertices are those previously assigned to a given part, while regular vertices are free to be assigned in any parts.

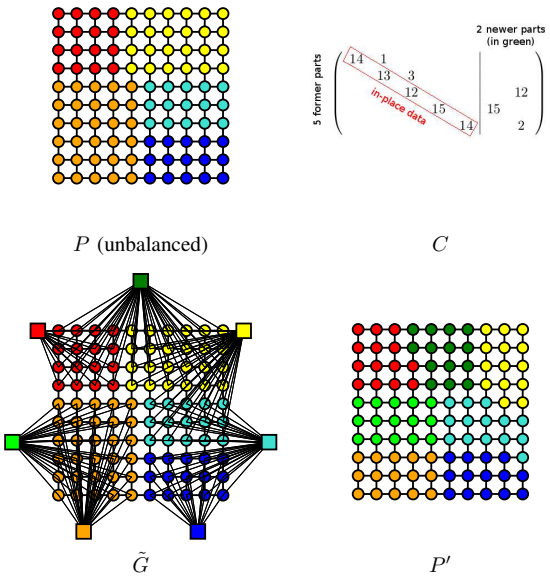


Fig. 5: Sample of a 5×7 repartitioning of a 2D grid of dimensions 10×10. The migration matrix C explains how vertices will migrate from the 5 former parts to the 7 newer. It is chosen to minimize both $totZ$ and $totV$. On the enriched graph $\tilde{G}$, the square vertices represents the N fixed vertices, connected to former parts according to C. The final partition obtained is well balanced and respects the communication scheme imposed by C.

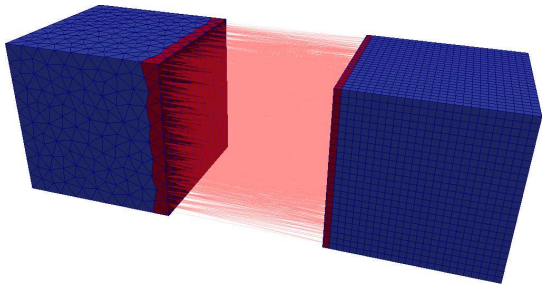


Fig. 6: Overview of the surface coupling between two mesh domains A (left) and B (right) showing the restriction to the coupled subdomains (in red) and interedges between cells. It corresponds to the `exp3` test case described in section IV.

interedges I, that map vertices from V to V'. More precisely, lets consider a vertex $u' \in V'$ and the vertex set $S(u') = \{u \in V, (u, u') \in I\}$. In the case where all the vertices of $S(u')$ are mapped to the same part p in P, then u' is trivially assigned to this part p in P'. In the case where the vertices of $S(u')$ are mapped to different parts, the situation is ambiguous, and one must select a part for u' according to a second criterion, like the edgecut optimization. In practice, this operator is used to compute a similar partition between the two coupled subgraphs G_A^{cpl} and G_B^{cpl}, connected through interedges I_{AB}, as shown in Figure 8. In this example, the projection is trivial for interedges (a, e) and (d, g), that are respectively mapped to part 1 and 2. But, it is clearly ambiguous for vertex f, that is shared by

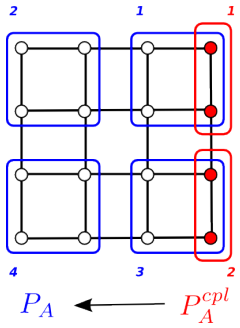


Fig. 7: Given the N_A^{cpl}-way partition P_A^{cpl} of G_A^{cpl} (in red), the extension operator computes a N_A-way partition P_A of G_A (in blue), with $N_A^{cpl} = 2$ and $N_A = 4$.

interedges (b, f) and (c, f), since b and c are already mapped to different parts. As the edgecut criterion in G_{AB}^{cpl} gives the same result for this vertex, one chooses randomly to assign f in part 1.

To implement this operation, we just perform a graph partitioning of G_{AB}^{cpl} in K parts, such that all vertex weights are set to zero (i.e., no balance constraint) and such that all vertices of G_A^{cpl} are fixed to their own part according to P_A^{cpl}. As an optimization, one collapses all vertices of G_A^{cpl} into K fixed super-vertices that represent each part. By minimizing the edgecut of such an enriched graph, the partitoning routine will compute the desired projection (possibly unbalanced).

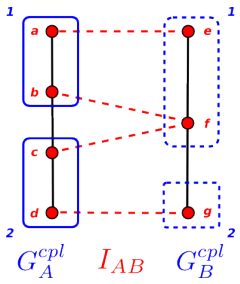


Fig. 8: Illustration of the projection operator on the graph G_{AB}^{cpl}. Given the N_A^{cpl}-way partition P_A^{cpl} (in blue), one aims to find a N_A^{cpl}-way partition of G_B^{cpl} (in dashed blue), with $K = N_A^{cpl} = 2$.

D. Co-partitioning algorithms

We will now present our co-partitioning algorithms, called *AWARE* and *PROJREPART*, and the *NAIVE* method used as the state-of-the-art solution. All these algorithms are precisely described as a sequence of the graph operators previously defined.

For these algorithms, we use as an input the graphs G_A and G_B, the interedges I_{AB} (or $I_{AB}^{-1} = I_{BA}$) and the number of processors N_A and N_B used for both components. Besides, as we do not address the problem of resource distribution (introduced in section I), we assume that the number of processors used in the coupling phase for both components (i.e., N_A^{cpl} and N_B^{cpl}) are also provided as an input). Nevertheless, in section IV, we will give a simple heuristic to compute these

inputs. All our algorithms solve the co-partitioning problem (defined in Sec. III-B) and compute the output partitions P_A and P_B, such that $P_{AB} = P_A \cup P_B$.

Let us now give some explanations about our three co-partitioning algorithms:

a) NAIVE: As described in Figure 10, the *NAIVE* method just computes a partition for each graph (G_A and G_B) independently, without taking into account the interedges, as it is currently done in complex coupled simulations. This simple algorithm is clearly not aware of the coupling phase. It will be used as a standard to compare against other methods in the experimental study (Sec. IV). In this method, the parameters N_A^{cpl} and N_B^{cpl} are not managed, and the coupled partitions P_A^{cpl} and P_B^{cpl} are not controlled.

b) AWARE: This method is divided in 3 main steps, symmetrically applied to graphs G_A and G_B (Fig. 11). It starts by computing the coupled subgraphs (G_A^{cpl} and G_B^{cpl}) based on the interedges, using the restriction operator. Then, one partitions each subgraph independently into N_A^{cpl} and N_B^{cpl} parts respectively. Finally, the obtained partitions are stretched to the global graphs using the extension operator.

c) PROJREPART: This method works exactly as the *AWARE* method, except it replaces the "naive" partition of G_B^{cpl} by the projection & repartition steps (Fig. 12). Here, the key idea is to improve the communication scheme during the coupling phase, by minimizing both the amount of data exchanged ($totV$) and the number of messages ($totZ$). First, the projection operator tries to keep the parts of P_A^{cpl} face-to-face with those of $\tilde{P}_B^{cpl}$ (Fig. 9a and 9b). Then, the repartitioning operator computes the partition P_B^{cpl} in a way to maintain $totV$ and $totZ$ quite low (Fig. 9c). In practice, it injects the newer parts (in the case where $N_A^{cpl} < N_B^{cpl}$) and tries to keep the former ones in place as far as it is possible. Finally, we extend the partitions obtained for G_A^{cpl} and G_B^{cpl} to the global graph as we do also in *AWARE*. Note that in the case where $N_A^{cpl} = N_B^{cpl}$, the repartition step is still required, because the projection operator possibly gives an unbalanced partition.

Our algorithms can be implemented with any graph partitioning software that supports fixed vertices. Nevertheless, and as far as we know, this feature is not available in classic tools like Metis or Jostle, but only in Scotch[4]. As explained in [29], it is better to use multilevel direct K-way (MLKW) partitioning instead of multilevel recursive bisection (MLRB) to manage fixed vertices. Moreover, for similar reasons, the initial partitioning of the coarsest graph should not use the recursive bisection paradigm (see Sec. 4.1.2 and Sec. 4.2.5 in [30]). As a consequence, we decide to develop our own paritioning routine called KGGGPML (multilevel K-way greedy graph growing partitioning) method, that extends some existing work [16]. As a reminder, the complexity of our algorithms is dominated by the partitioning routine, known to be $\mathcal{O}(\log k.|E|)$ for MLRB and $\mathcal{O}(k.|E|)$ for MLKW, improved to $\mathcal{O}(|E|)$ in [31].

[4]Note that some hypergraph partitioning software like Patoh, HMetis or Zoltan handle fixed vertices as it is useful for classic VLSI applications.

Inputs: G_A, G_B, N_A, N_B
Outputs: P_A, P_B

i) $Part(G_A, N_A) \rightarrow P_A$
ii) $Part(G_B, N_B) \rightarrow P_B$

Fig. 10: Description of the *NAIVE* co-partitioning algorithm.

Inputs: $G_A, G_B, I_{AB}, N_A, N_B, N_A^{cpl}, N_B^{cpl}$
Outputs: P_A, P_B

i) $Rest(G_A, Dom(I_{AB})) \rightarrow G_A^{cpl}$
ii) $Rest(G_B, Dom(I_{AB}^{-1})) \rightarrow G_B^{cpl}$
iii) $Part(G_A^{cpl}, N_A^{cpl}) \rightarrow P_A^{cpl}$
iv) $Part(G_B^{cpl}, N_B^{cpl}) \rightarrow P_B^{cpl}$
v) $Ext(G_A, G_A^{cpl}, P_A^{cpl}, N_A^{cpl}, N_A) \rightarrow P_A$
vi) $Ext(G_B, G_B^{cpl}, P_B^{cpl}, N_B^{cpl}, N_B) \rightarrow P_B$

Fig. 11: Description of the *AWARE* co-partitioning algorithm.

Inputs: $G_A, G_B, I_{AB}, N_A, N_B, N_A^{cpl}, N_B^{cpl}$
Outputs: P_A, P_B

i) $Rest(G_A, Dom(I_{AB})) \rightarrow G_A^{cpl}$
ii) $Rest(G_B, Dom(I_{AB}^{-1})) \rightarrow G_B^{cpl}$
iii) $Part(G_A^{cpl}, N_A^{cpl}) \rightarrow P_A^{cpl}$
iv) $Proj(G_A^{cpl}, G_B^{cpl}, I_{AB}, P_A^{cpl}, N_A^{cpl}) \rightarrow \tilde{P}_B^{cpl}$
v) $Repart(G_B^{cpl}, \tilde{P}_B^{cpl}, N_A^{cpl}, N_B^{cpl}) \rightarrow P_B^{cpl}$
vi) $Ext(G_A, G_A^{cpl}, P_A^{cpl}, N_A^{cpl}, N_A) \rightarrow P_A$
vii) $Ext(G_B, G_B^{cpl}, P_B^{cpl}, N_B^{cpl}, N_B) \rightarrow P_B$

Fig. 12: Description of the *PROJREPART* co-partitioning algorithm.

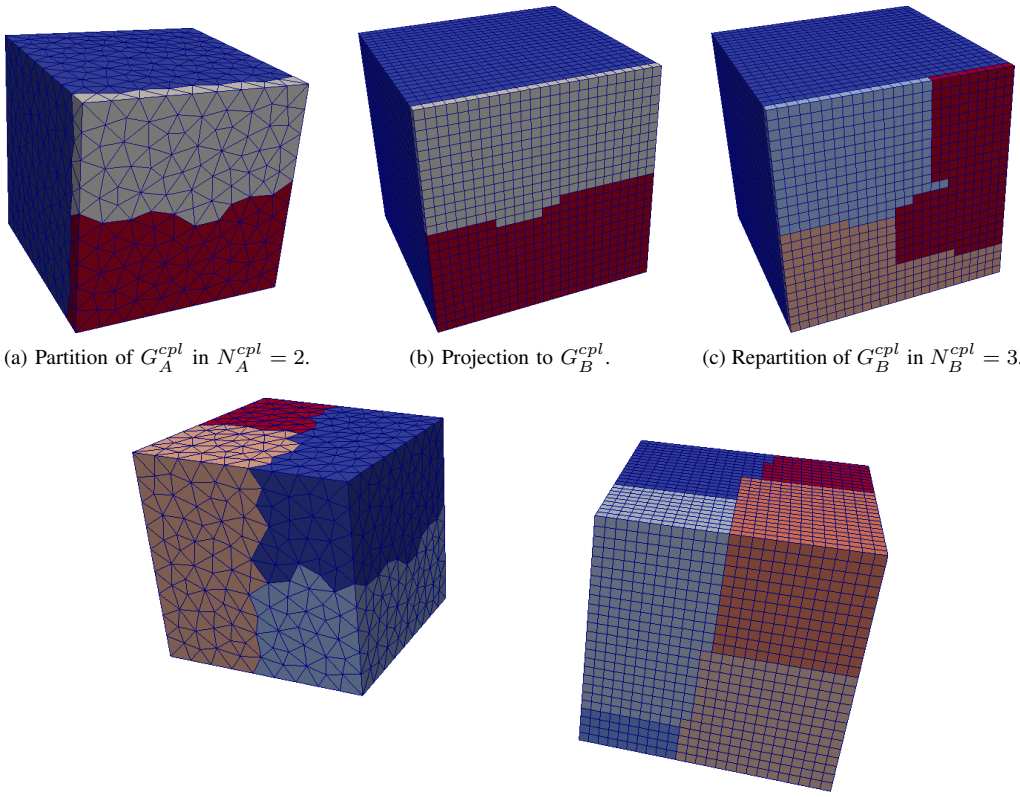
IV. Experimental Results

In this section, we present some experimental results on synthetically generated problems for the co-partitioning algorithms presented above. Note that we consider the *NAIVE* algorithm as the usual method to partition coupled applications, so we use it as a reference, in order to evaluate the quality of *AWARE* and *PROJREPART* algorithms.

In the following experiments, all algorithms are implemented thanks to the KGGGPML partitioning routine[5] as explained before. In all experiments, one uses exactly the same partitioning parameters (coarsening, refinement, *etc.*) with an imbalance factor set to 5%.

In the experiments we use synthetically generated graphs, whose characteristics are described in Table I. The coupling that we perform in these experiments is a simple surface coupling between two cubic domains, using hexaedral or tetrahedral mesh discretization as shown in Figure 6. Besides, one assumes that all graphs have vertex/edge weigth of 1. In the remainder of this section, we present the results of

[5]The code is publicly available in the LBC2 library on http://gforge.inria.fr/projects/mpicpl.



(a) Partition of G_A^{cpl} in $N_A^{cpl} = 2$.

(b) Projection to G_B^{cpl}.

(c) Repartition of G_B^{cpl} in $N_B^{cpl} = 3$.

(d) Overview of partitions for G_A (left) and G_B (right) after extension ($N_A = 4$ and $N_B = 6$).

Fig. 9: Example of the *PROJREPART* co-partitioning for a test case similar to `exp3` (see Sec. IV).

four experiments, named `exp1`, `exp2`, `exp3` and `exp4`, as illustrated in Table II. We repeat each experiment 5 times in order to compute average values for all our metrics. In all our experiments we keep the number of processors for A fixed to 16, while the number of processors for B takes values from the range $[16 - 128]$.

Due to the difference of mesh discretization between A and B, the problem of minimizing the communication costs in the coupling interface is more complex for `exp3` and `exp4` than for `exp1` or `exp2`. Indeed, for `exp1` and `exp2`, we use similar mesh discretization (with hexaedral elements well aligned), unlike for `exp3` or `exp4` where we use misaligned elements (hexaedral or tetraedral). In Figure 13, we illustrate two examples of mesh alignment in the coupling interface and we show the number of messages that need to be exchanged in each case. In 13a, there is an exact correspondence of one element in mesh A to several elements in mesh B, like for `exp1`. On the contrary, in 13b, a bad alignment between the two meshes creates possible additional small messages, denoted with the dashed lines, like for `exp3` or `exp4`.

Besides, we should mention that *NAIVE* algorithm does not control the values of N_A^{cpl} and N_B^{cpl}, since it is not aware of the coupling phase at all. This is why these values are not integer in the different tables. On the other hand, *AWARE* and *PROJREPART* algorithms use a simple heuristic to calculate N_A^{cpl} and N_B^{cpl}. The idea behind that is to find a proportion of the initial number of processors and assign it to the coupled subgraph, equal to the geometrical ratio between the coupling surface and the whole cubic domain, i.e., $N_X^{cpl} = \left\lfloor {N_X}^{2/3} \right\rceil$. However, the values of N_A^{cpl}, N_B^{cpl} computed for *AWARE* and *PROJREPART* algorithms in our experiments are quite similar to the ones obtained by *NAIVE* methods, allowing us to fairly compare the three algorithms on the partitioning quality.

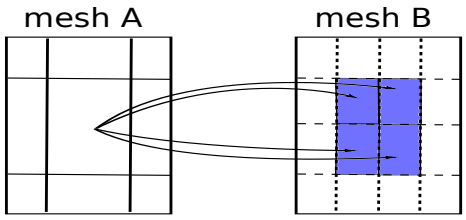
TABLE I: Description of the graphs used in the experiments.

Graph	Elements	$\|V\|$	$\|E\|$
cube-hexa-25x25x25	hexa	15 625	45 000
cube-hexa-70x70x70	hexa	343 000	1 014 300
cube-hexa-100x100x100	hexa	1 000 000	2 970 000
cube-tetra-40630	tetra	40 630	78 131
cube-tetra-486719	tetra	486 719	953 488

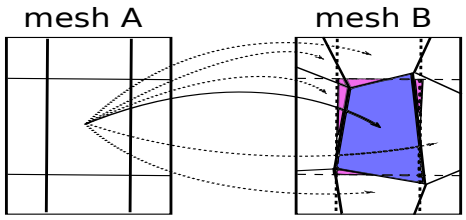
TABLE II: Description of the experiments.

Exp.	Graph A	Graph B
exp1	cube-hexa-25x25x25	cube-hexa-100x100x100
exp2	cube-hexa-25x25x25	cube-hexa-70x70x70
exp3	cube-tetra-40630	cube-hexa-100x100x100
exp4	cube-tetra-40630	cube-tetra-486719

To measure the overall partitioning quality, we initially compare our algorithms against the main objectives of partitioning, that are the global edgecut and partition imbalance for graphs G_A and G_B. In our results, they are denoted as cut_A, cut_B and



(a) Good mesh alignment



(b) Bad mesh alignment

Fig. 13: Examples of different mesh alignment in the coupling interface.

ub_A, ub_B respectively. In order to measure the co-partitioning quality, we introduce here some additional metrics that give us an insight on the quality of the partitioning during the coupling phase. Thus, two obvious yet important metrics in this context are the edgecut and the partition imbalance of the coupled subgraphs G_A^{cpl} and G_B^{cpl}, denoted as cut_A^{cpl}, cut_B^{cpl} and ub_A^{cpl}, ub_B^{cpl} respectively. Finally we are interested in measuring the communication costs between the two graphs during the coupling phase, so we also compare our results against $totV$ (total volume of data exchanged) and $totZ$ (total number of messages).

Following, we make some comments on the results of all four experiments, depicted in Tables III, IV, V and VI. A quite obvious but still important remark is that all algorithms respect the global balance constraint for both meshes, ub_A, ub_B. On the other hand as expected, *NAIVE* algorithm fails to balance the computational load during the coupling phase whereas *AWARE* and *PROJREPART* give a good load balance for both subgraphs, (ub_A^{cpl}, ub_B^{cpl}). Remember that in the experiments N_A is fixed, and since the steps performed on graph A are the same for *AWARE* and *PROJREPART*, we do not expect any differences in their results for cut_A, cut_A^{cpl}, ub_A and ub_A^{cpl}. Obviously, as illustrated in the tables, their results on the above metrics are the same. Note also that we do not compute the values of cut_A^{cpl}, cut_B^{cpl} for the *NAIVE* algorithm since there is no coupled subgraph involved in that method. In that case, we use the *AWARE* algorithm as reference, because it performs a simple partitioning of the coupled subgraph without other constraints, and minimize this edgecut as a primary objective. For *PROJREPART*, cut_B^{cpl} reaches an acceptable overhead of 10-15% in most cases compared to *AWARE*. Finally, one can see that in all experiments, the *AWARE* and *PROJREPART* algorithms manage to maintain the edgecut of graph B, cut_B, low relatively to the *NAIVE* algorithm, despite of the additional constraints imposed by these algorithms (minimizing load balance and communication costs during the coupling phase).

To draw conclusions on the results of $totV$ and $totZ$, we address each experiment separately. In Tables III and IV, the results of `exp1` and `exp2` indicate that the *PROJREPART* algorithm has always better results compared to *AWARE* and *NAIVE* as it succeeds to minimize both $totZ$ and $totV$ efficiently, due to good mesh alignment. In Tables V and VI, we illustrate the results of `exp3` and `exp4` where the two meshes A and B are strongly misaligned. Subsequently, the *PROJREPART* algorithm fails to minimize $totZ$ or $totV$, since the projection step is much more complex. In this case, the *AWARE* algorithm is clearly a better choice among all proposed algorithms.

Finally, we present results on execution time (given in *ms* in the last column of tables). In most cases, one can see that the *PROJREPART* method is less than 5% slower compared to *NAIVE*, while the *AWARE* method is roughly as fast as *NAIVE*. We evaluate that our partitioning routine is about 10 times slower than *kMetis* and 2 times slower than *Scotch* on the variety of graphs presented here, for a 128-way partition of same quality. It is mainly due to the lack of optimization of our K-way refinement routine, but it correctly handles fixed vertices as required by our algorithm implementation.

V. Conclusion and Future Work

In this paper, we initially give some formal definitions on the co-partitioning problem, i.e., the load balancing problem for coupled applications. We propose to enrich the classic graph model with interedges, that represent the coupling phase of component model interactions. Finally we present two new algorithms, called *AWARE* and PROJREPART, and compare them to the currently used approach (i.e., *NAIVE*). Both *AWARE* and *PROJREPART* algorithms succeed to balance the computational load in the coupling phase and in some cases they succeed to reduce the coupling communications costs. Surprisingly we notice that our algorithms do not degrade the global graph edgecut, despite of the additional constraints that they impose.

In future work, we aim to validate our results on real-life cases in the field of aeronautic propulsion. In order to achieve that, we plan to integrate our algorithms within the Scotch framework. Finally, our algorithms should be implemented in parallel and should be extended in order to manage more complex applications with more than two interacting models.

References

[1] J. D. Teresco, K. D. Devine, and J. E. Flaherty, "Partitioning and dynamic load balancing for the numerical solution of partial differential equations," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, ser. Lecture Notes in Computational Science and Engineering, T. J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, T. Schlick, A. M. Bruaset, and A. Tveito, Eds. Springer Berlin Heidelberg, 2006, vol. 51, pp. 55–88.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[3] G. Karypis, "METIS," http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

TABLE III: Experiment 1: results for mesh A (cube-hexa-25x25x25) and mesh B (cube-hexa-100x100x100).

EXP1	N_A	N_B	N_A^{cpl}	N_B^{cpl}	ub_A	ub_B	ub_A^{cpl}	ub_B^{cpl}	cut_A	cut_B	cut_A^{cpl}	cut_B^{cpl}	tot_V	tot_Z	$time$
NAIVE	16	16	6.80	7.80	4.71%	4.97%	37.95%	49.87%	3429	57078	–	–	706	21.00	3903
AWARE	16	16	6	6	4.86%	5.00%	0.80%	4.46%	3454	55070	81	317	699	18.00	3910
PROJREPART	16	16	6	6	4.86%	4.82%	0.80%	0.80%	3454	56188	81	324	625	6.00	3799
NAIVE	16	24	6.80	9.40	4.71%	4.95%	37.95%	48.15%	3429	71292	–	–	716	23.60	4559
AWARE	16	24	6	8	4.86%	5.00%	0.80%	4.96%	3454	71955	81	481	702	22.00	4479
PROJREPART	16	24	6	8	4.86%	4.99%	0.80%	4.70%	3454	67928	81	441	671	12.00	4588
NAIVE	16	32	6.80	11.00	4.71%	5.00%	37.95%	45.76%	3429	80198	–	–	723	26.80	5134
AWARE	16	32	6	10	4.86%	5.00%	0.80%	5.00%	3454	78915	81	527	697	24.00	4615
PROJREPART	16	32	6	10	4.86%	5.00%	0.80%	3.72%	3454	80982	81	502	696	14.00	4924
NAIVE	16	48	6.80	15.20	4.71%	5.00%	37.95%	39.21%	3429	96605	–	–	748	32.20	5520
AWARE	16	48	6	13	4.86%	5.00%	0.80%	4.91%	3454	97017	81	598	735	27.00	5430
PROJREPART	16	48	6	13	4.86%	4.99%	0.80%	4.78%	3454	97956	81	617	717	18.00	5545
NAIVE	16	64	6.80	17.60	4.71%	5.00%	37.95%	59.38%	3429	110148	–	–	763	36.60	6369
AWARE	16	64	6	16	4.86%	5.00%	0.80%	4.96%	3454	109251	81	653	766	33.00	5782
PROJREPART	16	64	6	16	4.86%	5.00%	0.80%	4.93%	3454	110361	81	740	738	20.00	6289
NAIVE	16	96	6.80	23.80	4.71%	4.99%	37.95%	46.58%	3429	132728	–	–	792	43.00	6925
AWARE	16	96	6	20	4.86%	4.99%	0.80%	5.00%	3454	133210	81	847	805	39.00	7384
PROJREPART	16	96	6	20	4.86%	4.99%	0.80%	5.00%	3454	132593	81	892	765	24.00	7647
NAIVE	16	128	6.80	29.80	4.71%	4.99%	37.95%	73.83%	3429	149464	–	–	829	52.00	9040
AWARE	16	128	6	25	4.86%	4.99%	0.80%	5.00%	3454	147484	81	876	814	46.00	8298
PROJREPART	16	128	6	25	4.86%	4.99%	0.80%	5.00%	3454	150489	81	1077	793	30.00	9020

TABLE IV: Experiment 2: results for mesh A (cube-hexa-25x25x25) and mesh B (cube-hexa-70x70x70).

EXP2	N_A	N_B	N_A^{cpl}	N_B^{cpl}	ub_A	ub_B	ub_A^{cpl}	ub_B^{cpl}	cut_A	cut_B	cut_A^{cpl}	cut_B^{cpl}	tot_V	tot_Z	$time$
NAIVE	16	16	6.80	8.00	4.71%	4.96%	37.95%	46.67%	3429	27398	–	–	723	21.60	1706
AWARE	16	16	6	6	4.86%	4.92%	0.80%	4.08%	3454	27212	81	237	712	18.00	1792
PROJREPART	16	16	6	6	4.86%	4.99%	0.80%	2.69%	3454	27381	81	226	699	18.80	1752
NAIVE	16	24	6.80	9.00	4.71%	4.99%	37.95%	38.21%	3429	33144	–	–	728	24.00	1960
AWARE	16	24	6	8	4.86%	4.96%	0.80%	4.82%	3454	33633	81	302	734	21.00	2067
PROJREPART	16	24	6	8	4.86%	4.99%	0.80%	4.29%	3454	33257	81	303	727	21.60	2187
NAIVE	16	32	6.80	12.60	4.71%	4.98%	37.95%	48.69%	3429	38991	–	–	754	28.60	2240
AWARE	16	32	6	10	4.86%	4.97%	0.80%	3.06%	3454	38723	81	351	750	23.00	2470
PROJREPART	16	32	6	10	4.86%	4.98%	0.80%	3.43%	3454	38738	81	344	741	23.00	2551
NAIVE	16	48	6.80	15.00	4.71%	4.98%	37.95%	56.76%	3429	47385	–	–	782	33.20	2877
AWARE	16	48	6	13	4.86%	4.98%	0.80%	4.53%	3454	47358	81	422	775	29.00	3032
PROJREPART	16	48	6	13	4.86%	4.98%	0.80%	4.42%	3454	47727	81	438	775	27.00	2814
NAIVE	16	64	6.80	17.60	4.71%	4.97%	37.95%	46.97%	3429	53314	–	–	801	37.20	3256
AWARE	16	64	6	16	4.86%	4.97%	0.80%	4.82%	3454	53756	81	479	799	34.00	3365
PROJREPART	16	64	6	16	4.86%	4.97%	0.80%	4.82%	3454	54973	81	533	815	32.00	3156
NAIVE	16	96	6.80	22.60	4.71%	4.96%	37.95%	50.20%	3429	64448	–	–	837	42.20	3693
AWARE	16	96	6	20	4.86%	4.96%	0.80%	4.90%	3454	64088	81	569	836	42.00	3817
PROJREPART	16	96	6	20	4.86%	4.96%	0.80%	4.90%	3454	64684	81	634	845	40.00	4328
NAIVE	16	128	6.80	29.00	4.71%	4.94%	37.95%	57.71%	3429	71832	–	–	861	51.40	4742
AWARE	16	128	6	25	4.86%	4.94%	0.80%	4.59%	3454	72011	81	637	868	49.00	4507
PROJREPART	16	128	6	25	4.86%	4.94%	0.80%	4.59%	3454	72500	81	749	892	44.60	5012

TABLE V: Experiment 3: results for mesh A (cube-tetra-40630) and mesh B (cube-hexa-100x100x100).

EXP3	N_A	N_B	N_A^{cpl}	N_B^{cpl}	ub_A	ub_B	ub_A^{cpl}	ub_B^{cpl}	cut_A	cut_B	cut_A^{cpl}	cut_B^{cpl}	tot_V	tot_Z	$time$
NAIVE	16	16	6.40	7.80	4.91%	4.86%	50.47%	41.03%	3031	56312	–	–	3276	22.60	3972
AWARE	16	16	6	6	4.87%	4.86%	0.17%	4.70%	3109	57464	112	324	3198	22.00	3978
PROJREPART	16	16	6	6	4.87%	4.99%	0.17%	3.54%	3109	57698	112	418	3218	25.00	4163
NAIVE	16	24	6.40	9.40	4.91%	5.00%	50.47%	48.87%	3031	70745	–	–	3305	23.80	4790
AWARE	16	24	6	8	4.87%	5.00%	0.17%	4.96%	3109	70099	112	440	3307	24.00	4506
PROJREPART	16	24	6	8	4.87%	5.00%	0.17%	3.81%	3109	71482	112	472	3300	26.00	4746
NAIVE	16	32	6.40	11.20	4.91%	5.00%	50.47%	54.01%	3031	81100	–	–	3345	26.40	4984
AWARE	16	32	6	10	4.87%	5.00%	0.17%	5.00%	3109	80839	112	543	3348	31.00	4647
PROJREPART	16	32	6	10	4.87%	5.00%	0.17%	4.92%	3109	81404	112	561	3343	31.20	5136
NAIVE	16	48	6.40	14.60	4.91%	5.00%	50.47%	44.14%	3031	97591	–	–	3413	33.20	5510
AWARE	16	48	6	13	4.87%	5.00%	0.17%	4.91%	3109	96063	112	578	3387	34.00	5805
PROJREPART	16	48	6	13	4.87%	5.00%	0.17%	4.55%	3109	99427	112	676	3429	35.00	5672
NAIVE	16	64	6.40	19.80	4.91%	5.00%	50.47%	64.04%	3031	110233	–	–	3510	41.80	6036
AWARE	16	64	6	16	4.87%	5.00%	0.17%	4.96%	3109	111377	112	687	3457	42.00	6028
PROJREPART	16	64	6	16	4.87%	5.00%	0.17%	4.96%	3109	112076	112	784	3495	43.40	6491
NAIVE	16	96	6.40	24.40	4.91%	4.99%	50.47%	68.45%	3031	132408	–	–	3575	46.20	7302
AWARE	16	96	6	20	4.87%	4.99%	0.17%	5.00%	3109	132712	112	830	3566	46.00	6871
PROJREPART	16	96	6	20	4.87%	4.99%	0.17%	5.00%	3109	133798	112	974	3614	51.20	7407
NAIVE	16	128	6.40	28.80	4.91%	4.99%	50.47%	59.99%	3031	147993	–	–	3645	54.40	9184
AWARE	16	128	6	25	4.87%	4.99%	0.17%	5.00%	3109	147802	112	912	3591	52.00	8710
PROJREPART	16	128	6	25	4.87%	4.99%	0.17%	5.00%	3109	151229	112	1125	3727	59.20	8971

TABLE VI: Experiment 4: results for mesh A (cube-tetra-40630) and mesh B (cube-tetra-486719).

EXP4	N_A	N_B	N_A^{cpl}	N_B^{cpl}	ub_A	ub_B	ub_A^{cpl}	ub_B^{cpl}	cut_A	cut_B	cut_A^{cpl}	cut_B^{cpl}	tot_V	tot_Z	$time$
NAIVE	16	16	6.40	7.80	4.91%	4.93%	50.14%	78.20%	3031	17045	–	–	3549	21.40	1803
AWARE	16	16	6	6	4.95%	4.77%	0.25%	4.68%	3035	17050	142	349	3549	20.00	1725
PROJREPART	16	16	6	6	4.95%	4.95%	0.25%	3.12%	3035	17301	142	361	3564	23.60	2719
NAIVE	16	24	6.40	8.00	4.91%	4.98%	50.14%	66.61%	3031	21106	–	–	3565	23.40	2007
AWARE	16	24	6	8	4.95%	4.97%	0.25%	4.82%	3035	21192	142	451	3612	25.00	2021
PROJREPART	16	24	6	8	4.95%	4.96%	0.25%	4.46%	3035	21562	142	520	3722	26.80	2609
NAIVE	16	32	6.40	9.80	4.91%	4.96%	50.14%	55.79%	3031	23970	–	–	3625	25.80	2114
AWARE	16	32	6	10	4.95%	4.97%	0.25%	4.80%	3035	24454	142	537	3723	26.00	2176
PROJREPART	16	32	6	10	4.95%	4.97%	0.25%	4.63%	3035	24636	142	622	3822	30.60	2565
NAIVE	16	48	6.40	12.20	4.91%	4.98%	50.14%	56.65%	3031	28743	–	–	3727	30.00	2538
AWARE	16	48	6	13	4.95%	4.98%	0.25%	4.93%	3035	29451	142	680	3851	35.00	2328
PROJREPART	16	48	6	13	4.95%	4.96%	0.25%	4.92%	3035	29741	142	801	3995	34.60	3013
NAIVE	16	64	6.40	16.20	4.91%	4.98%	50.14%	77.91%	3031	32534	–	–	3801	34.60	3208
AWARE	16	64	6	16	4.95%	4.97%	0.25%	4.51%	3035	32532	142	678	3862	37.00	3005
PROJREPART	16	64	6	16	4.95%	4.98%	0.25%	4.58%	3035	33816	142	921	4123	40.20	3382
NAIVE	16	96	6.40	20.20	4.91%	4.97%	50.14%	65.67%	3031	38315	–	–	3948	41.40	3896
AWARE	16	96	6	20	4.95%	4.97%	0.25%	4.91%	3035	39112	142	856	4044	46.00	3852
PROJREPART	16	96	6	20	4.95%	4.97%	0.25%	4.85%	3035	39758	142	1069	4289	50.40	4158
NAIVE	16	128	6.40	24.60	4.91%	4.98%	50.14%	65.05%	3031	43167	–	–	4025	48.40	4483
AWARE	16	128	6	25	4.95%	4.98%	0.25%	4.49%	3035	43429	142	975	4167	55.00	4075
PROJREPART	16	128	6	25	4.95%	4.98%	0.25%	4.83%	3035	44785	142	1339	4567	57.60	5102

[4] C. Walshaw, "JOSTLE," http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/.

[5] P. Sanders and C. Schulz, "Think Locally, Act Globally: Highly Balanced Graph Partitioning," in *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, ser. LNCS, vol. 7933. Springer, 2013, pp. 164–175.

[6] F. Pellegrini, "SCOTCH," http://www.labri.fr/perso/pelegrin/scotch/.

[7] A. P. Craig, M. Vertenstein, and R. Jacob, "A new flexible coupler for earth system modeling developed for ccsm4 and cesm1," *International Journal of High Performance Computing Applications*, vol. 26, no. 1, pp. 31–42, 2012.

[8] J. Amaya, E. Collado, B. Cuenot, and T. Poinsot, "Coupling LES, radiation and structure in gas turbine simulations," in *Proceedings of the Summer Program*, Center for Turbulence Research, NASA AMES, Stanford University, USA, 2010.

[9] J. W. Larson, "Ten organising principles for coupling in multiphysics and multiscale models," *ANZIAM Journal*, vol. 48, pp. C1090–C1111, 2009.

[10] S. Valcke, "The oasis3 coupler: a european climate modelling community software," *Journal of Geosci. Model Dev.*, pp. 373–388, 2013.

[11] A. T. A. Piacentini, T. Morel and F. Duchaine, "Open-palm: an open source dynamic parallel coupler." in *In IV International Conference on Computational Methods for Coupled Problems in Science and Engineering*, 2011.

[12] S. Jauré, F. Duchaine, and L. Gicquel, "Comparisons of coupling strategies for massively parallel conjugate heat transfer with large eddy simulation," in *In IV International Conference on Computational Methods for Coupled Problems in Science and Engineering*, Kos Island, Greece, 2011.

[13] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," in *Proceedings of 1995 ACM/IEEE conference on Supercomputing*, 1995.

[14] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Dec. 1998.

[15] R. Van Driessche and D. Roose, "Dynamic load balancing with a spectral bisection algorithm for the constrained graph partitioning problem," in *High-Performance Computing and Networking*, ser. Lecture Notes in Computer Science, B. Hertzberger and G. Serazzi, Eds. Springer Berlin / Heidelberg, 1995, vol. 919, pp. 392–397.

[16] S. Jain, C. Swamy, and K. Balaji, "Greedy algorithms for k-way graph partitioning," in *the 6th international conference on advanced computing*, 1998.

[17] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," *19th Design Automation Conference*, pp. 175–181, 1982.

[18] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, ser. SC '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–13.

[19] C. Walshaw, M. Cross, and K. McManus, "Multiphase mesh partitioning," *Applied Mathematical Modelling*, vol. 25, no. 2, pp. 123 – 140, 2000, dynamic load balancing of mesh-based applications on parallel.

[20] J. M. Dennis, M. Vertenstein, P. H. Worley, A. A. Mirin, A. P. Craig, R. L. Jacob, and S. A. Mickelson, "Computational performance of ultra-high-resolution capability in the community earth system model." *IJHPCA*, vol. 26, no. 1, pp. 5–16, 2012.

[21] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, Nov. 2000.

[22] C. Vuchener and A. Esnard, "Graph Repartitioning with both Dynamic Load and Dynamic Processor Allocation," in *International Conference on Parallel Computing - ParCo2013*, ser. Advances of Parallel Computing, München, Allemagne, 2013, pp. 243–252.

[23] L. Oliker and R. Biswas, "Plum: parallel load balancing for adaptive unstructured meshes," *J. Parallel Distrib. Comput.*, vol. 52, pp. 150–177, August 1998.

[24] K. Schloegel, G. Karypis, and V. Kumar, "Multilevel diffusion schemes for repartitioning of adaptive meshes," *Journal of Parallel and Distributed Computing*, vol. 47, no. 2, pp. 109 – 124, 1997.

[25] S. Kirk, K. George, and K. Vipin, "Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 5, pp. 451–466, May 2001.

[26] C. Aykanat, B. B. Cambazoglu, F. Findik, and T. Kurc, "Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids," *J. Parallel Distrib. Comput.*, vol. 67, pp. 77–99, January 2007.

[27] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. A. Riesen, "A repartitioning hypergraph model for dynamic load balancing," *J. Parallel Distrib. Comput.*, vol. 69, no. 8, pp. 711–724, 2009.

[28] C. Vuchener and A. Esnard, "Dynamic Load-Balancing with Variable Number of Processors based on Graph Repartitioning," in *Proceedings of High Performance Computing (HiPC 2012)*, Pune, Inde, 2012, 9 pages.

[29] C. Aykanat, B. B. Cambazoglu, and B. Uçar, "Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices," *J. Parallel Distrib. Comput.*, vol. 68, pp. 609–625, May 2008.

[30] C. Vuchener, "Équilibrage de charge dynamique avec un nombre variable de processeurs basé sur des méthodes de partitionnement de graphe," Ph.D. dissertation, Université de Bordeaux, Feb. 2014.

[31] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, pp. 96–129, 1998.