Dynamic Load-Balancing with Variable Number of Processors based on Graph Repartitioning

Clément Vuchener, Aurélien Esnard

Dynamic Load-Balancing with Variable Number of Processors based on Graph Repartitioning

Clément Vuchener 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*—Dynamic load balancing is an important step conditioning the performance of parallel adaptive codes whose load evolution is difficult to predict. Most of the studies which answer this problem perform well, but are limited to an initially fixed number of processors which is not modified at runtime. These approaches can be very inefficient, especially in terms of resource consumption. In this paper, we present a new graph repartitioning algorithm which accepts to dynamically change the number of processors, assuming the load is already balanced. Our algorithm minimizes both data communication and data migration overheads, while maintaining the computational load balanced. This algorithm is based on a theoretical result, that constructs optimal communication patterns with both a minimum migration volume and a minimum number of communications. An experimental study which compares our work against state-of-the-art approaches is presented.**

I. Introduction

In the field of scientific computing, the load-balancing is a crucial issue, which determines the performance of parallel programs. As a general rule, one applies a static balancing algorithm, which equilibrates the computational load between processors before running the parallel program. For some scientific applications, such as adaptive codes (e.g., adaptive mesh refinement), the evolution of the load is unpredictable. Therefore, it is required to periodically compute a new balancing at runtime, using a dynamic load-balancing algorithm. As this step may be performed frequently, it must use a fast and incremental algorithm with a quality trade-off. As computation progresses, the global workload may increase drastically, exceeding memory limit for instance. In such a case, we argue it should be relevant to adjust the number of processors while maintaining the load balanced. However, this is still an open question that we investigate in this paper.

A very common approach to solve the load-balancing problem (static or dynamic) is based on graph (or hypergraph) model [1]. Each vertex of the graph represents a basic computational task and can have a weight proportional to the task duration. Each edge represents a dependency in the calculation between two tasks and can have a weight proportional to the size of the communication needed if the two tasks are on different processors. To equilibrate the load between M processors, one performs a *graph partitioning* in M parts, each part being assigned to a given processor. More precisely, the objective consists of dividing the graph into M parts (or vertex subsets), such that the parts are disjoint and have equal size, and there are few edges cut between the parts. Here are the classical partitioning criteria:

- minimize the computation time (T_{comp}), which consists of dividing the graph in parts of equal weight, up to an unbalance factor;
- minimize the communication time (T_{comm}), which consists of minimizing the cut size of the graph induced by the new partition.

The weight of a part is simply the sum of the weights of all the vertices that are assigned to this part; and the cut size (or edge-cut) is the sum of the weights of edges whose ends belong to two different parts.

If the load changes at runtime, the current partition becomes unbalanced and it is required to perform a *graph repartitioning*. In addition to the classical partitioning criteria, the problem of repartitioning optimizes the following criteria [2]:

- minimize the migration time (T_{mig}), which consists of minimizing the vertex weight moving from the former partition to the new one;
- minimize the repartitioning time (T_{repart}).

It should be noticed that the repartitioning and migration steps are not performed at each iteration in the application, but periodically (e.g., every α iterations). As a consequence, the total time period of the code is written: $T_{total} = \alpha.(T_{comp} + T_{comm}) + T_{mig} + T_{repart}$. Assuming T_{repart} is negligible compared to the other terms, and if we consider that T_{comp} is implicitly minimized by balancing the parts, it follows that to minimize T_{total}, one must minimize $\alpha.T_{comm} + T_{mig}$. Finally, it clearly shows there is a trade-off between the optimization of the communication time (T_{comm}) and optimization of the migration time (T_{mig}). This compromise is controlled by the parameter α, which depends on the target application.

As we will see in the following section, there are many studies around the dynamic load-balancing and graph repartitioning. However, all these studies are limited—as far as we know—to the case where the number of processors is initially fixed and will not be modified at runtime. This can be very inefficient, especially in terms of resource consumption as demonstrated by Iqbal *et al.* [3], [4]. To overcome this issue, we propose in section III a new graph repartitioning algorithm, which accepts a variable number of processors, assuming the load is already balanced. We call this problem the $M \times N$ graph repartitioning problem, with M the

number of former parts and N the number of newer parts. Our algorithm minimizes both data communication (i.e., cut size) and data migration overheads, while maintaining the computational load balance in parallel. This algorithm is based on a theoretical result, that constructs optimal communication matrices with both a minimum migration volume and a minimum number of communications (see Sec. III-B). Moreover, it uses recent graph partitioning technique with fixed vertices to take into account migration constraints. Finally, we validate this algorithm in section IV with some experimental results, that compare our approach with state-of-the-art partitioning softwares.

II. Related Work

There are many work in the field of dynamic load-balancing [5], [6]. We briefly review the most popular methods based on graph (or hypergraph) repartitioning techniques.

The simplest approach is certainly the *Scratch-Remap* scheme [7], which calculates a new partitioning from scratch, that is to say, without taking into account the former partition. This technique obviously minimizes the cut, but does not control the migration at all. To reduce this latter cost, an additional step of *remapping* attempts to renumber the new parts in order to maximize data remaining in place.

Another approach are the diffusive methods. In their simplest form, they are based on the heat equation to dynamically equilibrate the load [8]. It is an iterative algorithm, where two neighboring processors exchange at each step an amount of data proportional to their load difference. After several steps, the convergence of the diffusion scheme reaches a new load balancing, that defines a new partitioning.

A more recent approach consists in repartitioning graph (or hypergraph) by minimizing both the cut size and the data movement due to migration (*RM-Metis* [9] and *Zoltan* [2]). For each part, a *fixed vertex* of zero weight is added. This particular vertex is connected by new edges—called *migration edges*—to all regular vertices that corresponds to this part. Then, one performs a partitioning of this enriched graph, with the constraint that fixed vertices are required to be assigned to their respective part in the final solution. Other vertices are free to move. Thus, if a regular vertex changes its part, this involves to cut a migration edge and to pay for an additional migration cost associated with this edge. As a partitioner attempts to minimize the cut size, it will also minimize the data movement due to migration. *Scotch* has recently added a similar graph repartitioning method based on fixed vertices, using a local diffusive refinement [10].

One can find in the literature many other work on dynamic load-balancing, including geometric methods like *Recursive Coordinate Bisection* (RCB) [5] or *Space-Filling Curve* (SFC) [11], spectral methods [12], or still more exotic approaches such as *skewed graph* partitioning [13].

All these studies are very interesting, but are limited to the case where the number of processors is initially fixed and is not modified at runtime. In our knowledge, there is no research that investigates the problem of graph (or hypergraph) partitioning with a variable number of processors. However, some recent studies have shown the interest of such an approach, by dynamically adjusting the number of processors in an adaptive code (AMR) to optimize both the parallel runtime and resource consumption [3], [4].

III. MxN Repartitioning Algorithm

We present in this section our $M \times N$ graph repartitioning algorithm which computes a newer partition in N from a former balanced partition in M. It is based on a theoretical result on *optimal* communication matrices, that minimizes both the data volume and the number of communications during the migration phase. These matrices are conveniently represented by a *repartitioning hypergraph*, that captures the optimal communication scheme we will impose. Then, the initial graph is enriched with fixed vertices, that models our additional migration constraints in a similar way to *Zoltan* [2] or *RM-Metis* [9]. Thus, the partitioning of this graph will minimize both the regular cut size and the data movement due to migration, while respecting the optimal communication scheme.

A. Communication Matrix and Repartitioning Hypergraph

Let consider a graph $G = (V, E)$, where V is the set of vertices, and E is the set of edges. Let w be the weight function that maps to a vertex subset of G its weight. We notice $W = w(V)$ the weight of the whole graph. Let $P = (V_1, V_2, \ldots, V_M)$ be the initial partition of V into M parts and $P' = (V'_1, V'_2, \ldots, V'_N)$ the final partition into N parts.

Let $C = (C_{i,j})$ be the $M \times N$ communication matrix associated with the repartitioning of G from P to P'. The element $C_{i,j}$ is the amount of data sent by the processor i to the processor j. According to the graph model, $C_{i,j}$ is equal to $w(V_i \cap V'_j)$. In this paper, we focus on *perfect* communication matrix, which results from two perfectly balanced partitions, P and P'. Such matrices satisfy the following constraints: for each row i, $w(V_i) = \sum_{1 \leq j \leq N} C_{i,j} = W/M$ (row constraint) and for each column j, $w(V'_j) = \sum_{1 \leq i \leq M} C_{i,j} = W/N$ (column constraint). As a consequence, W must be a multiple of both M and N.

We define the *number of communications*, $Z(C)$, as the number of non-zero terms in C. It represents the number of messages exchanged between former and newer parts, including "in-place" communications from a processor to itself. In the case of perfect communication matrix, we will demonstrate in the following section that this number is minimum for $M+N-GCD(M,N)$ and obviously maximum for $M.N$. Then, we define the *migration volume*, $Mig(C)$, as the amount of data being sent to a different processor, i.e., $Mig(C) = \sum_{i \neq j} C_{i,j}$.

The matrix C can be interpreted as an hypergraph H, called *repartitioning hypergraph*. This hypergraph is composed of M vertices representing the initial parts and N hyperedges representing the new parts obtained after the repartitioning step. A vertex i of H belongs to an hyperedge j if data are

exchanged between the former part i and the new part j during the migration. The repartitioning hypergraph allows to model the *communication scheme* without detailing the volume of data exchanged as the communication matrix does. We will see how this hypergraph representation makes easier to solve the correspondence problem we have in section III-C.

B. Optimal Communication Matrices

Our goal in this section is to seek communication matrices with good properties to perform efficiently the migration step. To simplify our discussion, we will assume in all this section that the communication matrix C of dimension $M \times N$ is perfect with $W = M.N$. As the initial and final partition are perfectly balanced, a source processor sends a data volume of N and a target processor receives a data volume of M (including "in-place" communications).

Definition 1: In this paper, a perfect communication matrix C is said to be optimal if it minimizes both the migration volume $Mig(C)$ and the number of communications $Z(C)$.

Theorem 1: Let C be a perfect communication matrix of dimension $M \times N$. The minimum number of communications is $Z_{opt} = M + N - GCD(M, N)$.

Proof: Let $\mathcal{G} = ((A, B), E)$ be the bipartite graph that represents the communication of matrix C from $M = |A|$ processors to $N = |B|$ processors. Let K be the number of connected components of $\mathcal{G}$, noted $\mathcal{G}_i = ((A_i, B_i), E_i)$ with $1 \leq i \leq K$. For each component $\mathcal{G}_i$, $M_i = |A_i|$ processors send a data volume $M_i.N$ to $N_i = |B_i|$ processors that receive a data volume $N_i.M$. Therefore, $\mathcal{G}_i$ exchange a data volume $V_i = M_i.N = N_i.M$, with M_i and N_i non null. As V_i is multiple of both M and N, one can say $V_i \geq LCM(M, N)$. Consequently, the total volume of communications $M.N = \sum_{i \in [1,K]} V_i$ is superior or equal to $K.LCM(M, N)$. As $GCD(M, N).LCM(M, N) = M.N$, one can deduce $K \leq GCD(M, N)$. As $\mathcal{G}_i$ is a connected graph, its number of edges $|E_i|$ is superior or equal to $M_i + N_i - 1$. And the total number of edges $|E| = \sum_{i \in [1,k]} |E_i|$ is superior or equal to $\sum_{i \in [1,K]} M_i + \sum_{i \in [1,K]} N_i - K = M + N - K$. As a consequence, the total number of communications $|E|$ is superior or equal to $M + N - GCD(M, N)$, for $K \leq GCD(M, N)$. ■

Let us consider the case $M < N$, where the number of processors increases. We can decompose the communication matrix C in two blocks (A, B): a left square block A of dimension $M \times M$ and a right block B of dimension $M \times N - M$.

Theorem 2: The communication matrix $C = (A, B)$ is optimal if the submatrix A minimizes the migration volume and if the submatrix B minimizes the number of communication.

Proof: To minimize the migration volume for C, one must take care to maximize the amount of data remaining in place, i.e., the sum of the terms on the diagonal of A. As a consequence, C optimizes the migration volume if A is diagonal, such as $A = M.I_M$ with I_M the identity matrix of order M. Thus, the minimal migration volume is $M.(N - M)$. In this case, the number of communications of C is $Z(C) = Z(A) + Z(B)$ with $Z(A) = M$. As B is assumed to be optimal, $Z(B) = M + (N - M) - GCD(M, N - M)$ according to theorem 1. As $GCD(M, N - M) = GCD(M, N)$, then $Z(C) = M + N - GCD(M, N)$ and C is optimal. ■

In the case where the number of processors decreases ($M > N$), we obtain a similar result by transposing the previous matrix. These two proofs remain correct for any perfect communication matrix, i.e., when W is not simply equal to $M.N$, but is multiple of M and N.

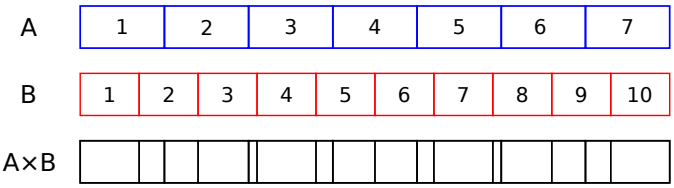


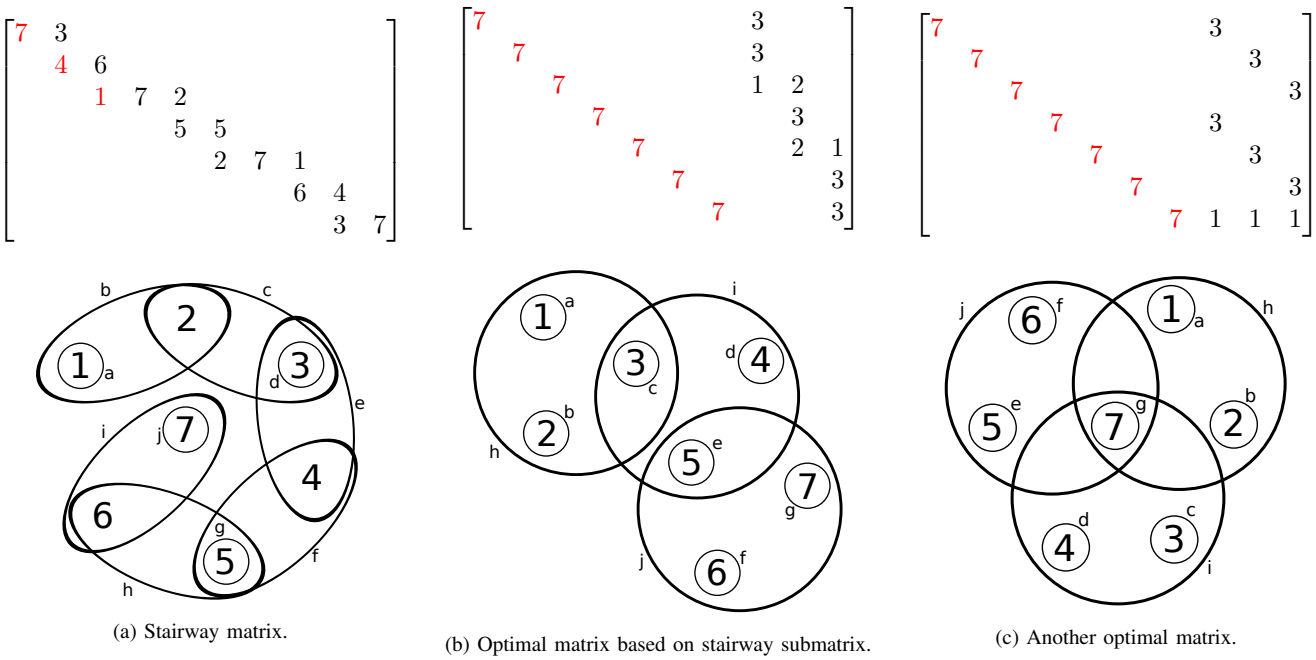
Fig. 1: Partitioning of a "chain graph" (represented as a one-dimensional array of length 70) in 7 and 10 and the resulting intersection pattern $A \times B$ used to construct the stairway communication matrix.

Let us now consider the examples given on figure 2 in the case 7×10. The *stairway matrix* (Fig. 2a) illustrates how to construct a perfect communication matrix with a minimum number of communications. This communication scheme is the one obtained by contiguously partitioning a "chain graph" (i.e., a simple one-dimensional array) in M parts and then in N parts. It is easy to demonstrate that the intersection pattern of these two partitions gives $M + N - GCD(M, N)$ communications, which is the optimal (Fig. 1). For the stairway matrix of dimension 7×10, we find $Z(C) = 16$ that is optimal, but the migration volume is not minimal at all ($Mig(C) = 58$). The figure 2b gives an example of an optimal matrix, based on a stairway submatrix according to theorem 2. In this case, both the number of communication and migration volume are minimal ($Z(C) = 16$ and $Mig(C) = 21$). The figure 2c gives another example of optimal communication matrix, but not based on the stairway matrix.

In the general case where W is any integer, it is no longer possible to maintain a perfect balance because W may not be multiple of M and N. Moreover, actual partitioning tools produce slightly unbalanced partitions, that prevents anyway building perfect communication matrices. Nevertheless in this case, we can obtain similar results, defined up to an unbalance factor, that still maintain an "optimal communication scheme" (represented by the repartitioning hypergraph).

C. Correspondence Problem between the Repartitioning Hypergraph and Quotient Graph

In order to achieve a *good* repartitioning, we have to choose where to place the new parts relatively to the former ones. As the "optimal" communication scheme we want to perform during the migration phase is modeled by a repartitioning hypergraph (Sec. III-A), we have to find a correspondence between vertices of the repartitioning hypergraph with those of the *quotient graph* associated with the initial partition (Def. 2).



(a) Stairway matrix.

(b) Optimal matrix based on stairway submatrix.

(c) Another optimal matrix.

Fig. 2: Three communication matrices in the case 7×10 and their representation as repartitioning hypergraph. Zero elements in matrices are not shown. Rows numbered from 1 to 7 correspond to vertices and columns numbered from a to j correspond to hyperedges. The elements in red are those who remain in place during communications, others will migrate.

Indeed, vertices belonging to the same hyperedge should be matched with *close* vertices in the quotient graph as these parts will send data to the same new part.

Definition 2: Let $P = (V_1, V_2, \ldots, V_M)$ be the initial partition of a graph G into M parts. We note $Q = G/P$ the quotient graph with respect to the partition P. A vertex i of Q represents the part V_i (with weight $w(V_i)$) and there is an edge (i, j) in Q if the parts V_i and V_j are connected. The weight of edge (i, j) in Q is the edge-cut between parts V_i and V_j.

The *closeness* of former parts is modeled by a score. This score is computed from the edges of the quotient graph. To express this score, the repartitioning hypergraph and the quotient graph are represented by matrices. The hypergraph matrix H is a $M \times N$ matrix and its element $H_{v,e}$ is non-zero if the hyperedge e contains the vertex v. The quotient graph is represented by its adjacency matrix Q whose element $Q_{i,j}$ is the weight of the edge (i, j). A matching is represented by a $M \times M$ permutation matrix X whose element $X_{i,j}$ is 1 if the vertex i of H is matched with the vertex j of Q, and 0 otherwise.

In the equation 1, $X_{i,i'}$, $X_{j,j'}$, $H_{i,k}$ and $H_{j,k}$ are binary values, their product is not zero when the vertices i' and j' of Q are respectively matched with the vertices i and j of H which are in the same hyperedge k. The score is the sum of the edge weights $Q_{i',j'}$ whose endpoints are matched with vertices belonging to the same hyperedge. Consequently, matching hyperedges of H with strongly connected subgraph of Q will give higher scores.

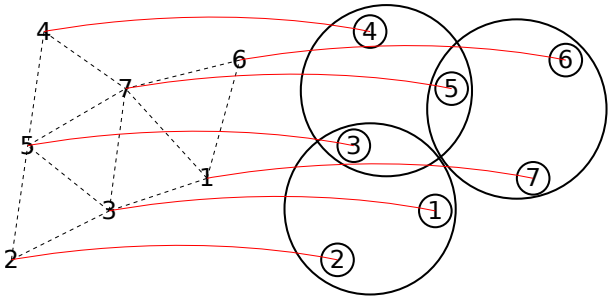


Fig. 3: Sample of a "good" matching in the case 7×10 between a quotient graph (on left) and an repartitioning hypergraph (on right).

$$score(X) = \sum_{i,j,i',j',k} X_{i,i'} X_{j,j'} H_{i,k} H_{j,k} Q_{i',j'}. \quad (1)$$

The score equation can be rewritten as follows:

$$score(X) = \sum_{i,i'} X_{i,i'} \sum_{j,j'} X_{j,j'} (\sum_{k} H_{i,k} H_{j,k}) Q_{i',j'}. \quad (2)$$

Let x be the column vector of size M^2 such that $x_k = X_{i,i'}$ with $k = iM + i'$ and $\otimes$ be the Kronecker product[1], the score can be rewritten as:

[1]Let A be a matrix of size $P \times Q$ and B be a matrix of size $R \times S$. The Kronecker product $A \otimes B$ is a block matrix with $P \times Q$ blocks of size $R \times S$ whose block (i, j) is $A_{i,j} \cdot B$.

$$score(x) = x^T A x \quad \text{with } A = HH^T \otimes Q \text{ of size } M^2 \times M^2. \tag{3}$$

According to the previous formulation, it appears that our problem is a binary quadratic optimization problem, with linear constraints:

$$\begin{cases} \forall i, \sum_{i'} x_{iM+i'} = 1 & \text{(row constraint for } X\text{)}, \\ \forall i', \sum_{i} x_{iM+i'} = 1 & \text{(column constraint for } X\text{)}. \end{cases} \tag{4}$$

The figure 3 gives an example of a good matching in the case 7×10, with the following permutation matrix:

$$X = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The first line of X means the hypergraph vertex 1 is matched with the quotient graph vertex 3. Assuming the weights of the quotient graph edges are all 1, the score is 9, that is the sum of the weights of the following edges: 2-3, 2-5, 3-5, 4-5, 4-7, 5-7, 1-6, 1-7, 6-7.

This optimisation problem is NP-hard [14]. This is a well-studied problem especially in the context of computer vision with a wide variety of applications: segmentation, clustering, graph matching. We can find in literature many heuristics to locate a good approximation to the global optimum: probabilistic metaheuristic like simulated annealing, spectral relaxation methods [15], [16], combinatorial methods like *branch & bound*, *etc.* In this paper, we use a basic simulated annealing with good results.

D. MxN Repartitioning Algorithm based on Fixed Vertices

Our algorithm uses a partitioning technique based on fixed vertices in a similar way to Zoltan [2] or RM-Metis [9]. As already explained in section II, the partitioning of the enriched graph with additional fixed vertices and "migration" edges enables to model the repartitioning problem with a trade-off between edge-cut and migration. Our algorithm extends this model when the number of processors changes, while respecting the chosen communication scheme. It is composed of the following steps:

1) Given an initial partition $P = (V_i)_{1 \leq i \leq M}$ of the graph G in M parts (Fig. 4a), the quotient graph Q is built (Fig. 4b).
2) An optimal communication matrices is chosen, giving us an optimal repartitioning hypergraph H (Fig. 4c). There are several possible choices as discussed in section III-B.
3) The repartitioning hypergraph H is matched to the quotient graph Q associated with the initial partition P, using a simulated annealing algorithm to optimize the score function described in section III-C (Fig. 4d). It give us a permutation matrix X.
4) Fixed vertices are added to graph G. There is one fixed vertex for each new part (or hyperedge in H). They have no weight since they represent processors, not tasks.
5) Then, we add migration edges, connected to these fixed vertices. Let K_j be the set of former processor ranks that will communicate with new processor of rank j, i.e., $K_j = \{i \,|\, \exists k,\, X_{k,i} = 1 \wedge H_{k,j} = 1\}$. Each fixed vertex j is connected with all the vertices of G belonging to former parts V_i with $i \in K_j$ (Fig. 4e). These new edges are weighted with a given migration cost.
6) This enriched graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ is finally partitioned in N parts, giving us the final partition P' of G (Fig. 4f).

While minimizing the edge-cut, the partitioner will try to cut as few migration edges as possible, if the migration cost is high enough. Indeed, each regular vertex v of G is connected to one or more fixed vertices, modeling different new processors where v may be assigned. As exactly one of these migration edges should not be cut, the communication scheme imposed by the repartitioning hypergraph should be respected.

The time complexity of this algorithm is mainly dominated by the partitioning (step 6). Indeed, the matching of the repartitioning hypergraph with the quotient graph (step 3) is a much smaller problem. The building of the enriched graph takes linear time in $|\widetilde{E}|+|\widetilde{V}|$ (step 4-5) with $|\widetilde{V}| = |V|+N$ and $|\widetilde{E}| = |E| + \frac{|V| \times Z_{opt}}{M} = |E| + \mathcal{O}(|V|)$ assuming $N = \mathcal{O}(M)$. Thus, the partitioning of the enriched graph is a little more complex than the one of the original graph.

IV. Experimental Results

Our $M \times N$ graph repartitioning method is compared with a Scratch-Remap method, ParMetis 4.0.2, Scotch 6 beta and Zoltan 3.6[2]. We give in section II further details about the repartitioning algorithms used by these software. These software are designed for repartitioning with a constant number of processors, but can still be used with a different new number of parts[3]. Both the Scratch-Remap method and the $M \times N$ method are achieved with Scotch in our experiments.

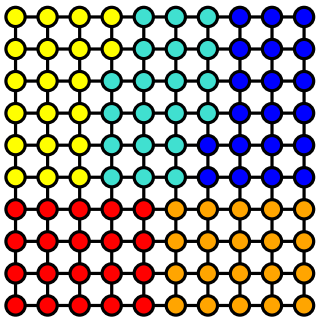
For all experiments, we have set the unbalance factor to 1%, the migration cost to 1 and have used default values for all other options. This latter parameter means we look for a trade-off between cut and migration.

A. Simple case

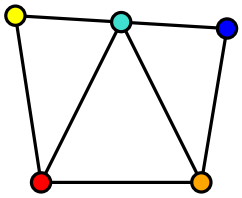
The graph used is based on a regular 3D grid (of dimensions $32 \times 32 \times 32$) with 32768 vertices and 95232 edges. It is initially partitioned in $M = 8$ parts and will be repartitioned in different numbers of new parts N from 2 to 24. All experiments are repeated 10 times and the charts in figure 5 show the average results for edge-cut, migration volume and number of communications.

[2]In order to compare graph partitioners with Zoltan hypergraph partitioner, one simply converts input graph into hypergraph by considering each graph edge as an hyperedge of size 2. In this way, it is correct to compare the classic graph edge-cut with hyperedge-cut (using $\lambda - 1$ cut metric).

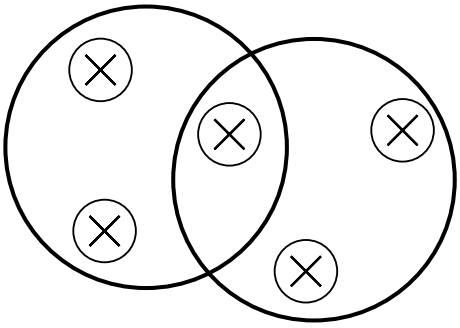
[3]In practice, we just say that the graph was initially partitioned in N parts instead of M. In case $M < N$, it implies that $N - M$ former parts were consider as empty.



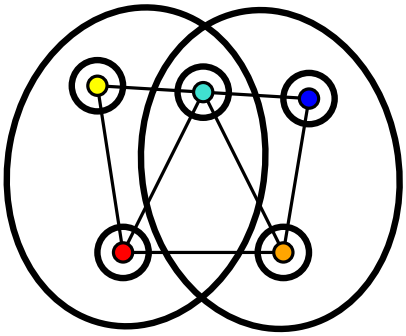
(a) Initial partition in 5 parts.



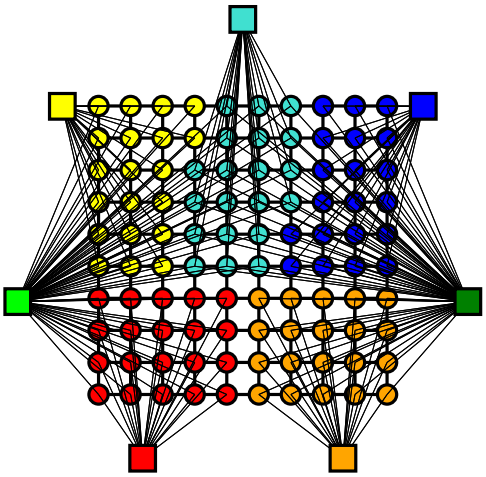
(b) Quotient graph of the initial partition.



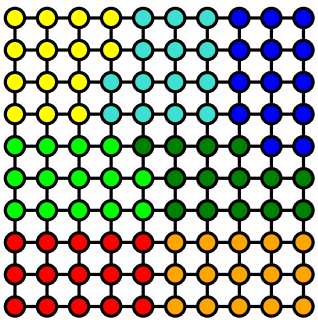
(c) An optimal repartitioning hypergraph for the case 5×7.



(d) Matching between the quotient graph and the repartitioning hypergraph.

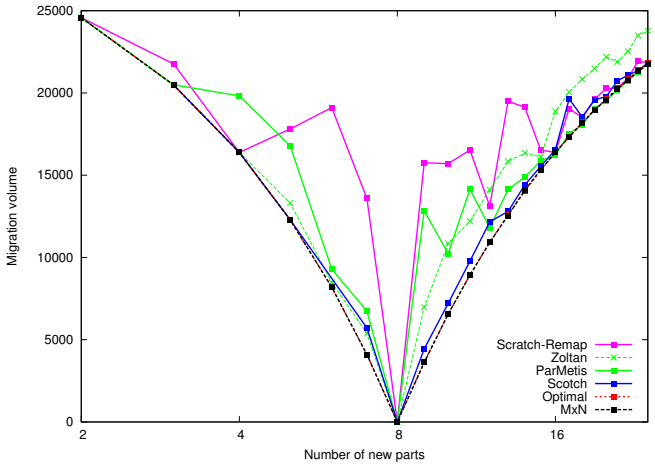


(e) Graph with fixed vertices added according to the matching.

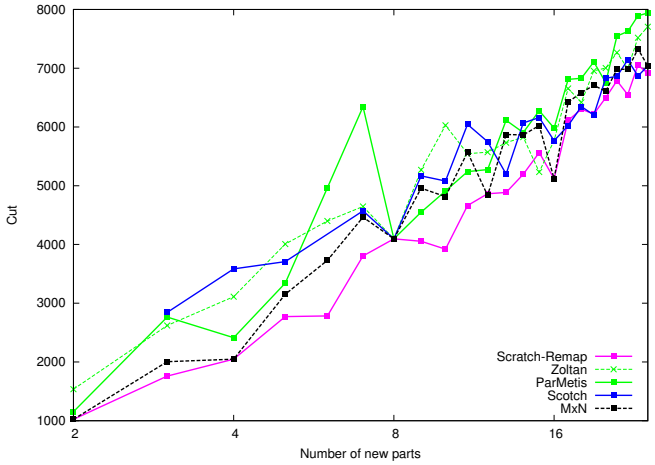


(f) Final partition in 7 parts.

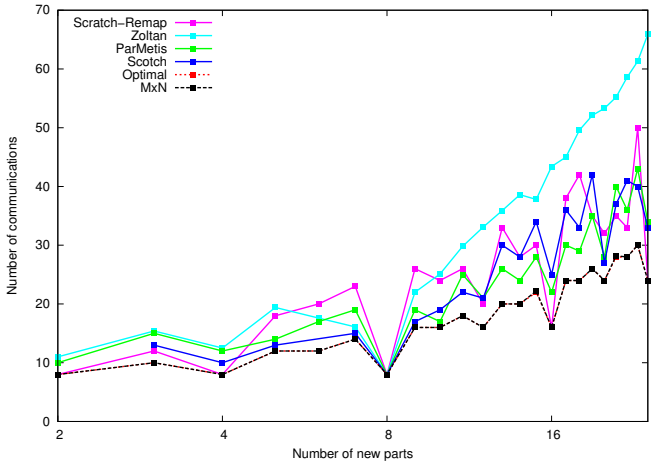
Fig. 4: Repartitioning from 5 to 7 parts.



(a) Migration volume.



(b) edge-cut.



(c) Number of communications.

Fig. 5: Experimental results for the graph repartitioning of a $32 \times 32 \times 32$ grid from $M = 8$ processors to $N \in [2, 24]$.

We can see on figure 5a that the migration for the $M \times N$ approach is optimal, as expected from the chosen communication matrix. For $N >> M$, the use of complex repartitioning methods becomes less relevant and the Scratch-Remap method should be preferred for its simplicity. The figure 5b shows that the low migration obtained by $M \times N$ comes at the expense of higher cut, but not higher than other repartitioning software. The cut for $M \times N$ method is not much higher than the Scratch-Remap method which gives the best cut that the partitioner can provide with no other constraints. The number of communications (including "in-place" communications) needed for the migration is given in the figure 5c. This number is optimal with the $M \times N$ method, while he can reach very high values for other tools, that indicates that the communication pattern for migration is just more complicated.

The communication time of the migration step has been experimentally measured with OpenMPI over an InfiniBand network on INRIA PlaFRIM platform[4]. The migration is up to 10% faster compared with other approaches. This confirms that our theoretical optimal communication matrices improve the migration time.

B. More complex cases

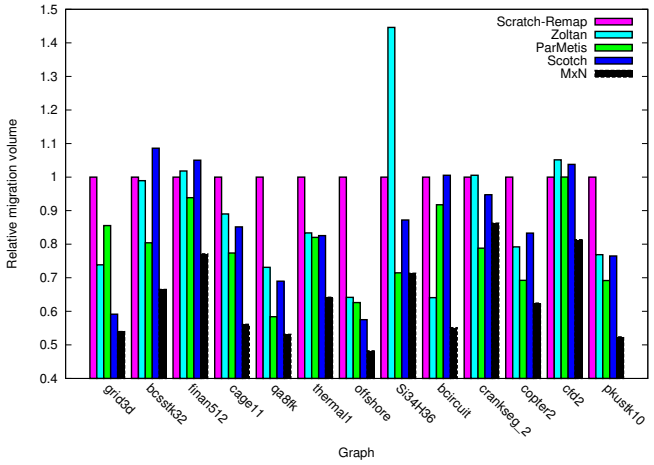
In order to evaluate our method in more complex cases, the same experiment is repeated on real-life graphs from different domains with different topologies. Those graphs are presented in the figure 6. We have seen in the previous experiment that our approach is more relevant when the former and the newer number of parts are close. So, we study two cases: the case 8×11 (Fig. 7) and the case 8×12 (Fig. 8). Remark that the case 8×11 is more irregular than the case 8×12 in terms of communication scheme.

graph	description	$\|V\|$	$\|E\|$	d
grid3d	regular 3D grid	32,768	95,232	5.81
bcircuit	circuit simulation	68,902	153,328	4.45
bcsstck32	structural problem	44,609	985,046	44.16
cage11	DNA electrophoresis	39,082	260,320	13.32
cfd2	computational fluid dynamics	123,440	1,482,229	24.02
copter2	computational fluid dynamics	55,476	352,238	12.70
crankseg_2	structural problem	63,838	7,042,510	220.64
finan512	economic problem	74,752	261,120	6.99
offshore	electromagnetics	259,789	1,991,442	15.33
pkustk10	structural problem	80,676	2,114,154	52.41
qa8fk	accoustic problem	66,127	797,226	24.11
Si34H36	quantum chemistry	97,569	2,529,405	51.85
thermal1	thermal problem	82,654	245,902	5.95

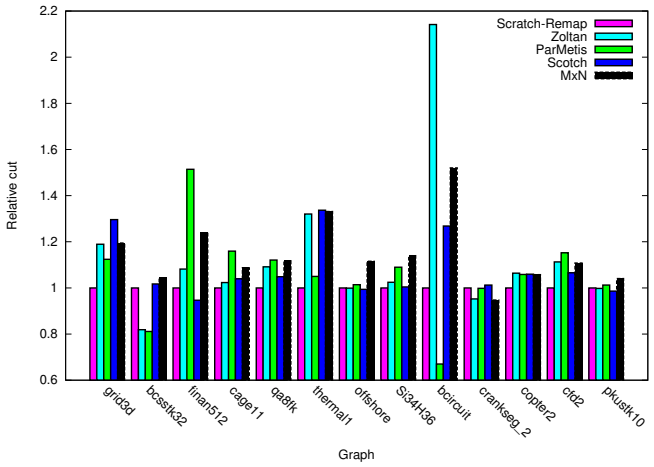
Fig. 6: Description of the test graphs, publicly available from *the university of Florida sparse matrix collection* [17] except for *grid3d*, that is the graph used in section IV-A. The value d represents the average degree of the graph, it is computed from $\frac{2 \times |E|}{|V|}$.

The figures 7a and 8a show the migration volume relatively to the Scratch-Remap method. For almost all graphs, the $M \times N$ method greatly improves the migration volume compared to others. As concerns the edge-cut (relatively to the Scratch-Remap method), we see on figures 7b and 8b that the performance of the different partitioning tools strongly varies depending on the graph. The $M \times N$ method gives an edge-cut

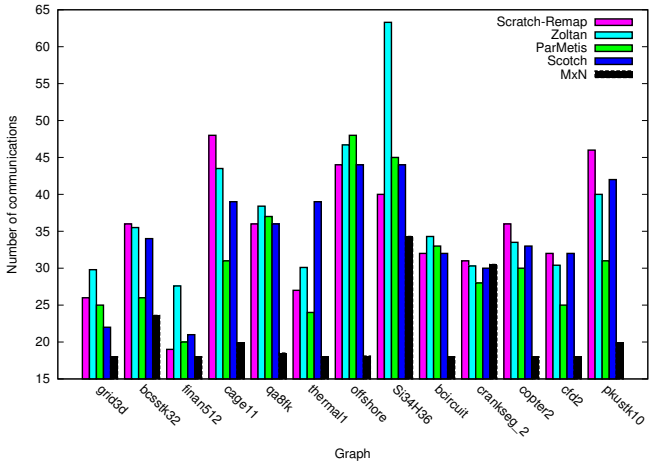
[4]http://plafrim.bordeaux.inria.fr



(a) Relative migration volume.

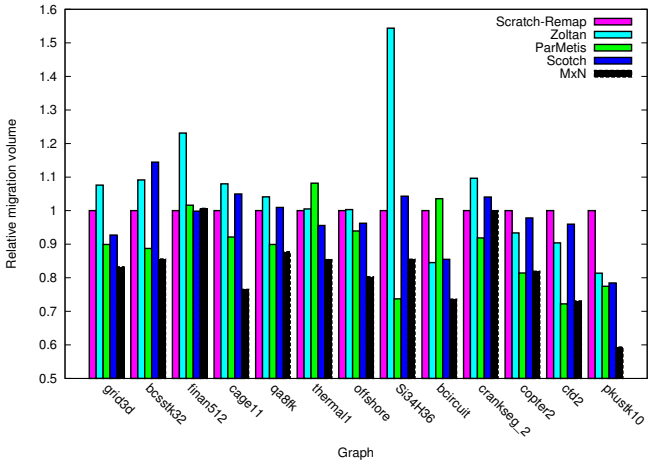


(b) Relative edge-cut.

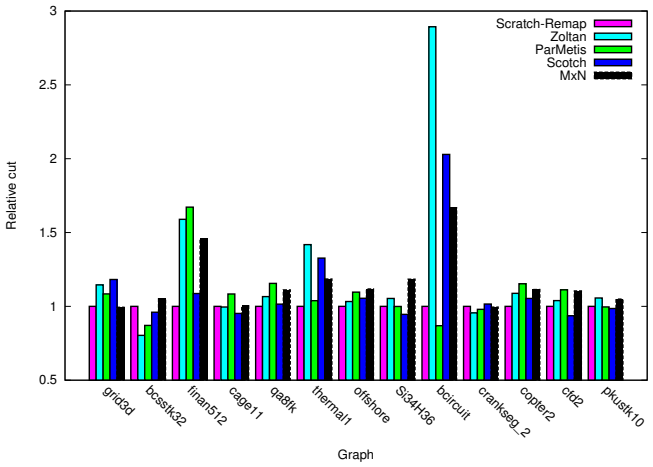


(c) Number of communications.

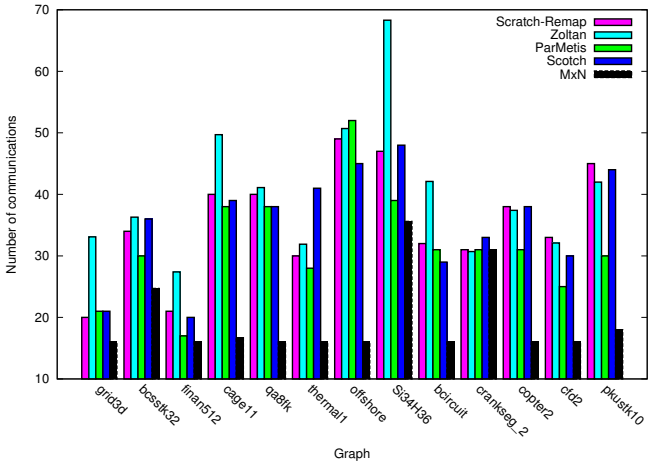
Fig. 7: Experimental results for several graphs described in figure 6 when repartitioning from 8 to 11 parts. Both the edge-cut and the migration volume are presented relatively to the Scratch-Remap method.



(a) Relative migration volume.



(b) Relative edge-cut.



(c) Number of communications.

Fig. 8: Experimental results for several graphs described in figure 6 when repartitioning from 8 to 12 parts. Both the edge-cut and the migration volume are presented relatively to the Scratch-Remap method.

quite comparable to most of the other tools, but slightly worse in several cases. The figures 7c and 8c show the number of communications needed for the migration. We obtain a low number of communication for the $M \times N$ method, that is almost optimal (the optimal numbers are respectively 18 and 16 for the cases 8×11 and 8×12). This confirms the good results obtained in the previous experiment. We see that the communication scheme imposed by the $M \times N$ method is generally well respected, except for the graphs *Si34H36* and *crankseg_2*. As a consequence, the migration volume for these two graphs is not as low as it could be. It is certainly due to some topological issues, that stresses partitioners (e.g., high average degrees of 221 for *crankseg_2*).

V. Conclusion and Future Work

We have presented in this paper a graph repartitioning algorithm, which accepts a variable number of processors, assuming the computational load is already balanced. Our algorithm minimizes both data communication and data migration overheads, while maintaining the load balance in parallel.

The experiments we have presented validate our approach for a large variety of real-life graphs, comparing it against state-of-the-art partitioners. Our $M \times N$ repartitioning provides both a minimal migration volume and a minimal number of communications, while keeping the edge-cut quite low.

We are considering several perspectives to our work. First, we focus on graph repartitioning in the more general case where both the load and the number of processors vary. We expect this work to be really suitable for next generation of adaptive codes. Finally, to be useful in real-life applications, our algorithm needs to work in parallel, that mainly requires to use a direct *k-way* parallel partitioning software that handle fixed vertices, like *Scotch*. This should allow us to partition much larger graph in larger part number.

References

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

[2] 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.

[3] S. Iqbal, G. Carey, M. Padron, J. Suarez, and A. Plaza, "Load balancing with variable number of processors on commodity clusters," in *High Performance Computing Symposium, San Diego*, 2002, pp. 135–140.

[4] S. Iqbal and G. F. Carey, "Performance analysis of dynamic load balancing algorithms with variable number of processors," *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, pp. 934 – 948, 2005.

[5] B. Hendrickson and K. Devine, "Dynamic load balancing in computational mechanics," in *Computer Methods in Applied Mechanics and Engineering*, vol. 184, 2000, pp. 485–500.

[6] 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.

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

[8] 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.

[9] 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.

[10] S. Fourestier and F. Pellegrini, "Adaptation au repartitionnement de graphes d'une méthode d'optimisation globale par diffusion," in *Proc. RenPar'20, Saint-Malo, France*, May 2011.

[11] J. Pilkington and S. Baden, "Dynamic partitioning of non-uniform structured workloads with spacefilling curves," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 3, pp. 288 –300, Mar. 1996.

[12] 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.

[13] B. Hendrickson, R. W. Leland, and R. V. Driessche, "Skewed graph partitioning." in *Eighth SIAM Conf. Parallel Processing for Scientific Computing*, 1997.

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

[15] M. Leordeanu and M. Hebert, "A spectral technique for correspondence problems using pairwise constraints," in *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2*, ser. ICCV '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1482–1489.

[16] O. Duchenne, F. R. Bach, I.-S. Kweon, and J. Ponce, "A tensor-based algorithm for high-order graph matching." in *CVPR*. IEEE, 2009, pp. 1980–1987.

[17] T. A. Davis and Y. Hu, "The university of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices