# Efficient and scalable parallel graph partitioning

Jun-Ho Her[a,1], François Pellegrini[b,2]

[a] *Project Bacchus of INRIA Bordeaux – Sud-Ouest, 351, cours de la Libération, 33405 Talence, France*

[b] *ENSEIRB-MATMECA, LaBRI & Project Bacchus of INRIA Bordeaux – Sud-Ouest, 351, cours de la Libération, 33405 Talence, France*

**Abstract**

The realization of efficient parallel graph partitioners requires the parallelization of the multi-level framework which is commonly used to improve the quality and speed of sequential partitioners. The two most critical issues in this framework are the coarsening phase, and the local refinement step performed in the uncoarsening phase. These two phases are difficult to parallelize, because the direct transposition in parallel of the matching algorithms used for coarsening, and of the inherently sequential Fiduccia-Mattheyses type algorithms traditionally used for local optimization, require much communication and synchronization, which hinder scalability.

This paper describes new parallel algorithms which tackle these two issues: a simplified probabilistic matching algorithm, and a parallel banded diffusive algorithm, both of which are implemented in the PT-Scotch parallel graph partitioning software. Experimental results illustrate the efficiency and the scalability of these methods.

*Key words:* combinatorial optimization, graph partitioning, heuristics, multi-level, parallel algorithms, scientific computing

*Email addresses:* jhher@labri.fr (Jun-Ho Her), francois.pellegrini@labri.fr (François Pellegrini)

[1] This author's post-doctoral position is funded by ANR through the SOLSTICE project (ANR-06-CIS6-010-01).

[2] This author's work is partially supported by ANR through the SOLSTICE project (ANR-06-CIS6-010-01).

## 1. Introduction

Graph partitioning is a combinatorial problem which aims at finding, in a given (possibly weighted) graph, a small vertex set or edge set whose removal splits the graph into a prescribed number of parts of roughly equivalent sizes or weights. This problem is strongly related to many practical problems in the fields of scientific computing and engineering, such as domain decomposition for parallel iterative linear system solvers, the layout of VLSI circuits, or image segmentation [1].

Since the graph partitioning problem is NP-complete in the general case [2], many heuristics have been proposed to yield acceptable partitions in reasonable time. Among them, the *multi-level* approach, which was proposed in the beginning of the 1990s [3, 4], has been proved to be a very successful paradigm, which became the core of many sequential graph partitioning tools [5, 6, 7]. The multi-level method consists of three phases, which are *recursive coarsening*, *initial partitioning*, and *recursive uncoarsening*. In this last phase is most often included a partition *refinement* step, aimed at optimizing the projected partitions at every level, so that the granularity of the solution is the one of the original graph and not the one of the coarsest graph. Most often, this partition refinement is performed by means of local optimization algorithms which derive from the Kernighan-Lin [8] or Fiduccia-Mattheyses (FM) [9] algorithms.

However, because of their ever increasing sizes, the graphs to process can no longer fit in the memory of sequential computers. Several parallel (hyper-)graph partitioning tools have been developed to tackle this issue [10, 11, 12, 13], which base on the parallelization of the multi-level framework.

Because of the inherently sequential nature of the FM refinement algorithm, existing parallel tools which base their refinement algorithms on relaxed versions of the FM heuristic are likely to encounter quality or scalability problems for graphs of billions of vertices processed on thousands of processes. This is why a new refinement algorithm, called *banded diffusive* refinement, has been devised, which simulates the behavior of liquid in a system consisting of barrels and pipes [14]. This nature-inspired algorithm has two advantageous features against the classical FM-like heuristics: its high scalability, due to its intrinsically parallel nature, and its ability to yield small and smooth partition boundaries. The latter feature is an additional quality metric of great interest for iterative linear system solvers [15, 16, 17, 18]. The application considered in this paper is domain decomposition for paral-

lel iterative and hybrid [19] linear system solvers, such as the ones developed within the Bacchus project of INRIA Bordeaux – Sud-Ouest.

This paper presents the algorithms which have been implemented in the PT-Scotch parallel graph partitioning library, including a parallel version of the banded diffusive refinement method. Many building blocks of our implementation are based on the ones which had been formerly developed for the efficient parallel ordering, by nested dissection, of very large symmetric sparse matrices [20]. However, significant improvements have been brought to some algorithms, such as the parallel coarsening phase, which will be discussed in Section 3.2.

The present work adopts a parallel recursive bipartitioning approach for graph partitioning with the following parallel strategy. After some bipartition is computed, the processes which were used for that purpose are also split into two subsets, such that each subset deals recursively with one of the two separated subgraphs. This allows each of the process subsets to work independently at the same time, and increases communication locality. We describe in detail the parallel recursive bipartitioning stratedgy in Section 3.1.

The rest of the paper is organized as follows. Section 2 discusses related work on parallel $k$-way graph partitioning. Parallelization issues regarding recursive bipartitioning and the banded diffusive refinement algorithm are described in Section 3. Section 4 presents experimental results and a comparison with those of ParMeTiS. The final section draws some conclusions and poses future work.

## 2. Related work

### *2.1. Parallel refinement algorithms*

There have already been several significant research efforts on parallel (hyper-)graph partitioning based on the multi-level framework, which can be found in [21, 22, 23, 24]. Since the most related issue to the work presented in this paper is (parallel) refinement algorithms for the uncoarsening phase, we limit ourselves to that issue in the following survey.

The main problem which arises when trying to parallelize iterative local optimization algorithms such as FM is the loss of global synchronization in the computation and update of gains attached to vertex movements. Indeed, two processes can locally decide to move one of the vertices they own to another part, because it will locally improve the partition with respect to the current configuration, while the aggregation of all of these individually

beneficial moves may result in a globally worse partition. Since this problem happens only when neighbor vertices which belong to different processes are swapped, several techniques have been proposed to avoid potentially harmful interactions of distinct processes on neighbor vertices.

For instance, Karypis and Kumar have proposed a parallel *greedy refinement* algorithm which consists of $c$ passes, where $c$ is the number of independent colors of the finer graph [21]. During the $i^{th}$ pass, only vertices of color $i$ are considered, so as to guarantee that no concurrent moves can involve neighboring vertices. After all vertices belonging to the same color have been moved (or not), vertex movement gains are globally recomputed. Their solution is based on synchronization steps which depend only on graph topology, that is, the number of independent colors which can be found in the graph, in this case by means of a Luby coloring [25], irrespective of the distribution of graph vertices onto processes. The problem of the above approach is that it looses some of the local optimization features of the sequential FM algorithm. In this latter, an initial displacement of a frontier vertex often resulted in cascading moves of its neighbors, to level them with the new position of the frontier and reduce the number of cut edges. By allowing independent moves to happen anywhere on the frontier, the probability of having long segments of the frontier leveled in the same direction decreases, thus increasing the number of cut edges.

To address this problem and handle synchronously large portions of the frontier, Walshaw and Cross have proposed another greedy refinement method based on isolated boundary regions, called *interface* regions [22]. Unlike the previous approach, this latter heavily depends on the current state of the partition to isolate portions of the frontier suitable for parallel processing. Devine *et al.* have proposed a parallel two-way refinement method based on the FM heuristic for hypergraph partitioning [23]. It consists of sequences of two consecutive passes, called a *pass-pair*, such that in a given pass only vertices belonging to one of the partitions are moved to the other, therefore avoiding any swapping effect. Trifunovic and Knottenbelt have proposed a $k$-way version of the pass-pair refinement to parallelize the greedy refinement algorithm [24].

Despite these research efforts, the above solutions are still trapped by the intrinsic sequentiality of FM-like heuristics. Relaxing in some way this sequentiality constraint always leads to poorer partition quality, all the more that the proposed methods, which perform multiple moves at the same time, can no longer benefit from the hill-climbing features of the initial FM algo-

rithm, in which sequences of inefficient moves could be undone, and instead behave as gradient-like methods, more likely to be trapped in local optima of the partition cost function. Therefore, the stability of the quality of these algorithms is not guaranteed when the number of processes increase.

### *2.2. Partition quailty metrics*

Since the main application related to this paper is domain decomposition for iterative linear system solvers, we also need to address the specific concerns of this community. Farhat *et al.* [16] and Vanderstraeten *et al.* [18] independently posed the problem that a significant number of iterations are required to converge when the subdomains have longer boundaries or irregular boundary shapes. To quantify these parameters, several authors defined metrics for the boundary shape, referred to as *aspect ratio*, using several geometric invariants of each subdomain. For instance, for meshes related to two-dimensional problems, an acceptable candidate is the ratio between the circumfence of a subdomain (Euclidian length of path consisting of edges linking frontier vertices) and the square root of the enclosed area.

Because it is very hard to incorporate aspect ratio criteria in graph-based local optimization algorithms, several authors proposed global heuristics inspired from natural phenomena. Diekmann *et al.* devised a method simulating the behavior of growing soap bubbles in some finite space through breadth-first search (BFS) traversals of the graph [15]. This method starts from randomly picking as many seed vertices (that is, roots of the BFS-trees) as the desired number of parts, and then performs concurrent BFS traversals from each seed vertex, flagging each vertex with the color of the first BFS tree which hits it, until all vertices have been reached. Then, it tries to seek new positions for the seed vertices, by selecting in each part the vertex having shortest paths to all of the other vertices of the part, as an analogy to finding the mass center of the Euclidian domain. This process is iterated until convergence, that is, seed vertices no longer move and shapes of parts no longer evolve.

Meyerhenke and Schamberger have reinforced this *bubble-growing* model through concurrently growing the bubbles to improve execution time with almost the same quality. However, no significant improvement in execution time was achieved [17, 26]. In [27], Wan *et al.* proposed a distributed network partitioning algorithm making each vertex of a graph diffuse probabilistically its current state information through its neighbors based on a diffusion model, called *influence model*, which had been first introduced in [28]. Since the

target application of this work is sensor networks, leader (source) vertices corresponding to the parts are naturally fixed as the predefined base stations.

However, these meta-heuristics have two drawbacks. Firstly, they lack of proper load balancing mechanisms. For example, in the bubble-growing scheme, a seed vertex unfortunately picked close to the periphery of the graph will have less space to extend its domain, resulting in unbalanced domain sizes. Secondly, they are computationally expensive because they operate on all of the graph vertices, while local optimization methods handle a number of vertices which is smaller by several orders of magnitude, in accordance to separator theorems [29].

Recently, the *banded diffusive* method has been proposed by the second author [14] to address concurrently these two issues. Moreover, this method is highly parallelizable, so that its parallel version is likely to replace advantageously state-of-the-art parallel refinement algorithms. A brief review of the sequential banded diffusive method and its parallelization will be presented in Section 3.3.

## 3. Parallel graph partitioning

### *3.1. Parallel multi-level recursive bipartitioning*

The parallelization of the multi-level framework for graph partitioning raises several algorithmic concerns. Firstly, the matching algorithm used in the parallel coarsening phase must not be biased, for instance by favoring mating with neighbor vertices located on the same process over ones located on other processes. In the sequential algorithm, synchronization of mating decisions was implicitly resolved by the order in which the vertices were considered, and biases could be avoided by considering vertices to be mated in pseudo-random order. In parallel, ties on mating decisions related to non-local edges have to be broken as evenly as possible, without creating dependency chains across processes that could loop into deadlocks or lead to synchronizing all decisions in a sequential way. Secondly, state-of-the-art sequential local optimization algorithms, such as the FM algorithm, are hard to parallelize, while global optimization algorithms are too expensive, as exposed in the previous section.

In a previous work on the parallelization of nested dissection ordering [20], our team devised a parallel multi-level framework which provides workable solutions to these two problems. Our framework comprised three different

levels of concurrency, corresponding to three key steps of the nested dissection process: the nested dissection algorithm itself, the multi-level coarsening algorithm used to compute separators at each step of the nested dissection process, and the refinement of the obtained separators. This parallel framework has been adapted to the parallel recursive graph bipartitioning case, for which edge separators are sought rather than vertex separators.

The first level of concurrency derives from the intrinsically recursive and concurrent nature of the nested dissection and recursive graph bisection methods themselves. Once a separator has been computed in parallel on $p$ processes, all of these processes build the two distributed subgraphs corresponding to the two separated parts, and fold each of the parts to two distinct sets of $\lceil \frac{p}{2} \rceil$ and $\lfloor \frac{p}{2} \rfloor$ processes. Every process can then recursively proceed independently on each subgroup of $\frac{p}{2}$ (then $\frac{p}{4}$, $\frac{p}{8}$, etc.) processes, until each subgroup is reduced to a single process, after which recursive bipartitioning will go on sequentially by means of the sequential routines of the Scotch library.

The second level of concurrency concerns the computation of separators, which is performed by using a multi-level framework. The mating of fine vertices to be coarsened is performed in parallel by means of a synchronous probabilistic algorithm which reduces communication while enforcing unbiased matings, irrespective of the initial distribution of graph vertices; this algorithm will be described in the next section. After this mating phase ends, either all coarsened vertices are kept on their local processes (that is, the processes owning the fine vertices from which mating requests originated), which decreases the number of vertices owned by every process and speeds-up future computations, or else coarsened graphs are folded and duplicated, which reduces inter-process communication in the further stages of the algorithm and increases the number of working copies of the graph, which is likely to improve the final quality of the separators. These two behaviors are illustrated in Figure 1.

The third level of concurrency concerns the refinement heuristics which are used to improve the projected separators. At the coarsest levels of the multi-level algorithm, when computations are restricted to individual processes, the sequential FM algorithm of Scotch is used, while in parallel, for distributed graphs, we have devised a parallel banded diffusion scheme, which will be described in Section 3.3.

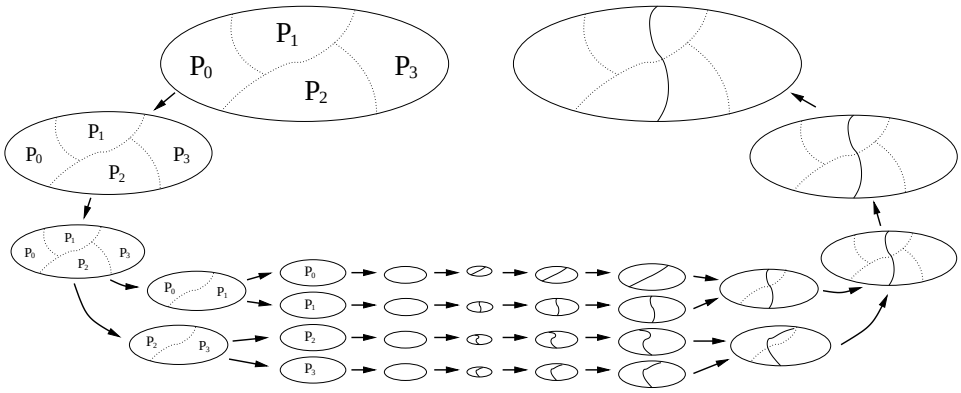


Figure 1: Diagram of the parallel computation of the separator of a graph distributed across four processes, by parallel coarsening with folding-with-duplication, multi-sequential computation of initial partitions that are locally projected back and refined on every process, and then parallel uncoarsening of the best partition encountered.

### *3.2. Parallel matching*

Parallel matching algorithms are a very active research topic, because maximal matchings are an essential tool for the solving of many combinatorial scientific computing problems [30, 31, and included references].

In the context of graph coarsening, this problem is relaxed, because matchings do not need to be maximal. Instead, they should exhibit randomness properties suitable to the preservation, by the coarsened graphs, of the topological properties of the finer graphs to which they are applied. Else, the projection to the finer graphs of partitions computed on the coarser graphs would not be likely to represent good solutions of the original problem, because of the biases introduced by coarsening artifacts.

The critical issue for the computation of matchings on distributed graphs is the breaking of mating decision ties for edges spanning across processes. Any process willing to mate one of its local vertices to a remote adjacent vertex has to perform some form of query-reply communication with the process holding the remote vertex. Because of communication latency and overhead, mating requests are usually aggregated per neighbor process, resulting in a two-phase algorithm. In the first phase, processes send mating requests to their neighbors. In the second phase, neighbors answer positively or negatively depending whether the requested vertices are free, have already been matched in a previous phase, or are temporarily unavailable because they have themselves requested another remote vertex for mating. The asyn-

chronicity between requests and replies creates ties and dependency chains which hinder the convergence of the algorithm. For instance, when several vertices request the same vertex, only one of the requests can be satisfied at most. Moreover, when the requested vertex has itself asked for another vertex, none of the requests will be satisfied, because the vertex cannot know whether its own request will be satisfied or not before replying to its applicants. This phenomenon increases along with the probability of neighbor vertices to be remote: when graphs are arbitrarily distributed (which is usually the case before they are partitioned for the first time), when they have high degrees, and/or when the number of processes is high.

The most popular method to avoid dependency chains is to prevent potential sought-after vertices from being applicants themselves, basing on an independent set coloring of the graph [21]. Each of the colors is considered in turn, and only vertices of the current color can perform mating requests. Since, by definition, independent set coloring guarantees that no two neighbor vertices can have the same color, potential mates cannot be applicants, and thus will always be able to answer positively to one of their own applicants. Colors are processed in a round-robin way until all vertices are matched or have all of their neighbors matched, or until some minimum coarsening threshold is reached.

In all parallel coarsening algorithms we are aware of, independent sets are computed in parallel by means of Luby's algorithm [25]. First, every graph vertex is assigned a random number. Then, vertices having the highest number among all of their neighbors are painted with the first color, after which these vertices are removed from the graph. Remaining vertices having the highest number among all of their remaining neighbors are painted with the second color and removed, and so on until all of the vertices have been considered. This algorithm is simple and elegant, but it has some drawbacks. The sizes of the color sets are usually unbalanced, the last ones being much smaller than the first ones; it is often the case that the last set comprises only one single vertex, as illustrated in Table 1. This imbalanced distribution adversely affects convergence time: larger sets increase the probability that multiple requests are directed to the same vertices, therefore reducing their probability of success, while smaller sets result in less work being carried out and communication rounds being dominated by network latency. A solution to this problem could be to rework Luby's algorithm so as to improve the balance of color sets. However, this is not likely to reduce the number of color sets, that is, communication rounds.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 998699 | 908307 | 848574 | 804471 | 777182 | 757239 | 743009 | 733290 |
| 720049 | 700112 | 661773 | 590857 | 478982 | 338154 | 200174 | 99186 |
| 41975 | 15071 | 4750 | 1377 | 372 | 108 | 20 | 6 |

Table 1: Sizes of Luby's color sets for graph 10MILLIONS. This graph has $10,423,737$ vertices, $78,649,134$ edges and an average degree of 15.09. Our test implementation of Luby's algorithm, applied to the randomly permuted vertex set, yields 24 color sets with the above distribution.

The approach we have chosen bases on probabilities. This idea, first exposed by Chevalier in [32], consists in allowing a vertex to send a request only when the random number it draws is above some threshold. In [32] and [20], this threshold depended on the numbers of local and remote neighbors of the vertex. We opted for a more naive algorithm, less susceptible to graph distribution biases: every unmatched vertex can send a request with probability 0.5. Our algorithm, which consists of successive matching rounds, works as follows. Initially, all local vertices are put in a wait queue, in random order. During a matching round, all queued vertices are processed one by one. For each vertex, a random bit value is considered. If it is zero, the vertex is put back into the queue, else one of its yet unmatched neighbors of highest edge weight is randomly selected for mating, according to the heavy edge matching method [33]. If the selected vertex is local, the matching is immediately accepted; else, a mating request is enqueued in the message destined to its owner process. After all vertices have been considered, mating request messages are exchanged between processes, and are processed in random order by their recipients. Each of the requests is considered in message order. If the sought-after vertex is itself a sender, no reply will be returned. Else, if the vertex is not already matched, the matching is accepted; else, a negative answer is crafted, so that the vertex will no longer be considered as unmatched by the sending process (however, other processes may not yet be aware of this information until one of their local vertices sends a request directed towards this vertex). Then, reply messages are send back, and mating data for local and ghost vertices are updated accordingly. The above communication round is repeated several times, after which a final local matching sweep is performed to match locally, either with a local unmatched neighbor or with itself, every vertex remaining in queue.

In order to evaluate this approach, we have instrumented our matching algorithm to output two values after each round. The first value is the ratio

| Pass | Matching | | Coarsening | |
|---|---|---|---|---|
| | Avg. | M.a.d. | Avg. | M.a.d. |
| C1 | 53.3 | 12.3 | 50.4 | 0.7 |
| C2 | 68.7 | 13.6 | 51.6 | 2.2 |
| C3 | 76.2 | 12.2 | 52.5 | 3.3 |
| C4 | 81.0 | 10.6 | 53.2 | 4.0 |
| C5 | 84.5 | 9.1 | 53.7 | 4.5 |
| LF | 100.0 | 0.0 | 59.4 | 6.8 |

Table 2: Average and mean absolute deviation of the percentage of the vertices processed, and of the coarsening ratio of the processed vertices, after each of five collective matching rounds (C) and after the local final (LF) round. These data were collected by recursively coarsening our test graphs on numbers of processes ranging from 2 to 512.

of matched vertices, expressed as a percentage of the total number of fine vertices. This value is equal to 100 % after the final pass. The second value is the coarsening ratio among matched vertices, that is, the number of coarse vertices computed to date, divided by the number of processed fine vertices. By nature, this value ranges between 50 %, in the ideal case where all fine vertices have been paired into coarse vertices, and 100 %, in the case where no neighbors could be found and all coarse vertices are each made of one single fine vertex.

Table 2 presents the data collected when recursively coarsening our test graphs down to one thousand of vertices per process, for a number of processes ranging from 2 to 512. It shows the mean value and the mean absolute deviation of each of the two aforementioned values, for each of the collective rounds and for the final local round. The number of collective rounds has been set to 5, to keep it small comparatively to the size of the color sets computed by Luby's algorithm. Assuming, as a rule of thumb, a matching probability of a bit lower than 0.5 (depending on the probability of the requested vertex to be inactive and of the probability of collision between matching requests, based on graph topology), 5 collective passes were supposed to be enough to match more than 80 % of the vertices.

Experimental figures corroborate our assumptions and validate our approach. Five collective rounds are enough to match more than 80 % of the graph vertices, with a low resulting coarsening ratio of 53.7 %. This ratio indicates that remote mating is efficient, and that no topological biases, due to initial graph data distribution across processes, are likely to occur. Conse-

quently, the final, local round is not likely to induce an important topological bias, since it only involves 15 % of the vertices on average, after many remote matings have been performed. A sixth collective round could has been tried, but as it did not significantly changed partition results, it was not used for the rest of our tests.

### *3.3. Parallel refinement by banded diffusion*

In a previous contribution [14], the second author proposed a global, diffusion-based, optimization algorithm, called *jug of the Danaides*, as a replacement for parallelized FM-like local optimization algorithms. This method simulates the propagation of two antagonistic liquids (in the bipartitioning case, though the method can be extended to any number of them) flowing from two opposite source vertices as if the graph edges were pipes, until they meet and annihilate, therefore defining a new frontier. Both source vertices flow in $\frac{W}{2}$ units of liquid per iteration, where $W$ is the sum of the graph vertex weights, and each vertex loses in the same time a number of units of liquid equivalent to its own weight, so that the system is bound to converge. In practice, it is not necessary to wait until full convergence, because what matters is to know which liquid dominates in each vertex.

In order to find two relevant source vertices, as well as to reduce problem complexity, our diffusion algorithm does not operate on the full uncoarsened graph but on a band graph which is extracted from it by keeping only vertices that are at small distance from the projected separator (typically less than 3). Vertices which do not belong to the band graph are merged into two *anchor vertices* of weights equivalent to the ones of the merged vertices, as illustrated in Figure 2, which serve as source vertices for the diffusion algorithm.

Band graphs have been successfully implemented in the sequential case to compute vertex separators by means of genetic or constrained FM algorithms [34], and also in parallel to compute distributed band graphs prior to their centralization on each of the processes so as to be able to run sequential FM algorithms on these smaller graphs [20]. Now, by providing a parallel version of the *jug of the Danaides* diffusion algorithm in addition to the parallel extraction of distributed band graphs, all sequential bottlenecks are removed from the parallel multi-level frameworks.

While sequential band graphs have only one anchor vertex per part, we did not transpose this to the parallel case, because anchor vertices would have had very high degrees, possibly creating communication bottlenecks and interfering with the internals of some optimization algorithms. In the

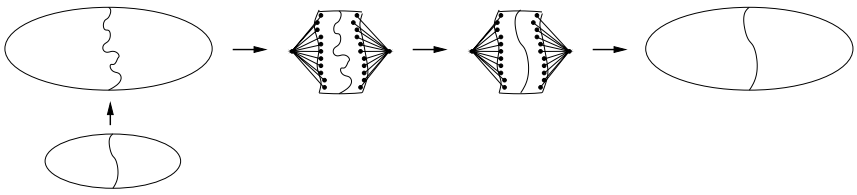

Figure 2: Multi-level banded refinement scheme. A band graph of small width is created around the projected finer separator, with anchor vertices representing all of the removed vertices in each part. After some optimization algorithm (whether local or global) is applied, the refined band separator is projected back to the full graph, and the uncoarsening process goes on.

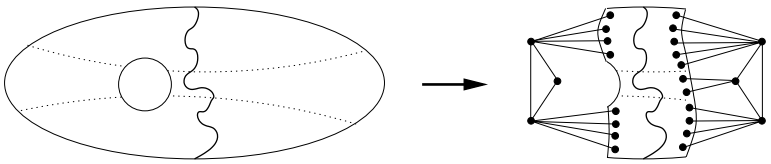

Figure 3: Computation of a distributed band graph from a partitioned graph distributed on three processes. The solid line is the current partition frontier, while dotted lines represent the separation between process domains. Merged vertices in each part are now represented by a clique of local anchor vertices, one per process and per part.

parallel bipartitioning case, there are two anchor vertices per process, so that each of the anchor vertices is connected to all of the local vertices of the last layer belonging to the same part, as well as to all of the remote anchor vertices of the same part, forming two distributed cliques, as illustrated in Figure 3. Of course, for numbers of processes above the ten of thousands, another system should be implemented, for instance some flavor of tree.

The parallel version of our diffusive algorithm does not substantially differ from its sequential counterpart [14]. It relies on the low-level halo exchange routines of the PT-Scotch library to spread on local ghost nodes the amount of liquids borne by each non-local neighbor vertex as the result of the previous iteration, and uses these values to compute the new amounts for all of its local vertices.

## 4. Experimental results

Being part of the PT-Scotch software, the code related to this work has been designed for architectures with distrbuted memory. It is written in

| Graph | Size ($\times 10^3$) | | $\overline{\delta}$ | Description |
|---|---|---|---|---|
| | $\|V\|$ | $\|E\|$ | | |
| 10MILLIONS | 10424 | 78649 | 15.09 | 3D electromagnetics, CEA |
| 23MILLIONS | 23114 | 175686 | 15.20 | 3D electromagnetics, CEA |
| 45MILLIONS | 45241 | 335749 | 14.84 | 3D electromagnetics, CEA |
| 82MILLIONS | 82294 | 609508 | 14.81 | 3D electromagnetics, CEA |
| AUDIKW1 | 944 | 38354 | 81.28 | 3D mechanics mesh, Parasol |
| BRGM | 3699 | 151940 | 82.14 | 3D geophysics mesh, BRGM |
| CAGE15 | 5154 | 47022 | 18.24 | DNA electrophoresis, UF |
| COUPOLE8000 | 1768 | 41657 | 47.12 | 3D structural mechanics, CEA |
| THREAD | 30 | 2220 | 149.32 | Connector problem, Parasol |

Table 3: Characteristics of the graphs that we use in the experiment. $|V|$ and $|E|$ are the vertex and edge cardinalities, in thousands. $\overline{\delta}$ is the average degree. CEA is the French atomic energy agency, BRGM is the leading French public institution involved in the Earth Science field for the sustainable management of natural resources and surface, UF stands for the University of Florida sparse matrix collection [35], and Parasol is a former European project [36].

ANSI C and uses the MPI communication API for message passing.

All experiments were performed on the Platine supercomputer at CCRT. This machine is a Bull Novascale cluster of 932 compute nodes interconnected via an Infiniband network. Each node has four dual-core Intel Itanium II processors.

The main characteristics of the graphs that we have used in our experiments are shown in Table 3.

The metric that we will consider for evaluating the quality of partitions is the cut size, that is, the number of edges the ends of which are assigned to two different processes.

| Test case | Number of processors:Number of parts | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 32:2 | 32:32 | 32:1024 | 128:2 | 128:128 | 128:1024 | 384:2 | 384:256 | 384:1024 | $P_{peak}$:2 |
| 10MILLIONS | | | | | | | | | | |
| $C_{PTS}$ | *4.75E+04* | *6.78E+05* | *3.89E+06* | *4.71E+04* | *1.38E+06* | *3.88E+06* | *4.70E+04* | *1.96E+06* | *3.90E+06* | *4.89E+04* |
| $C_{PM}$ | 5.12E+04 | 7.14E+05 | 3.97E+06 | 5.16E+04 | 1.46E+06 | 3.94E+06 | 5.28E+04 | 2.02E+06 | 3.97E+06 | 5.16E+04 |
| $t_{PTS}$ | *5.10* | 19.11 | 32.53 | *4.05* | 10.27 | 12.27 | *7.40* | *16.15* | *15.81* | *3.51(64)* |
| $t_{PM}$ | 18.43 | 7.89 | 6.81 | 7.65 | 5.27 | 4.16 | 29.60 | 29.76 | 24.65 | 7.65(128) |
| 23MILLIONS | | | | | | | | | | |
| $C_{PTS}$ | *9.10E+04* | *9.45E+05* | *5.19E+06* | *9.26E+04* | *1.88E+06* | *5.18E+06* | *9.07E+04* | *2.65E+06* | *5.20E+06* | *9.26E04* |
| $C_{PM}$ | 9.93E+04 | 9.80E+05 | 5.36E+06 | 9.43E+04 | 1.98E+06 | 5.33E+06 | 1.06E+05 | 2.79E+06 | 5.37E+06 | 9.79E+04 |
| $t_{PTS}$ | *12.49* | 46.89 | 74.29 | *6.15* | 21.50 | 26.64 | *10.25* | 20.21 | 20.67 | *6.15(128)* |
| $t_{PM}$ | 43.07 | 19.67 | 18.00 | 12.99 | 7.36 | 6.25 | 17.53 | 14.00 | 16.09 | 10.35(192) |
| 45MILLIONS | | | | | | | | | | |
| $C_{PTS}$ | *1.15E+05* | *1.13E+06* | *7.24E+06* | *1.11E+05* | *2.52E+06* | *7.26E+06* | *1.06E+05* | *3.65E+06* | *7.29E+06* | *1.05E+05* |
| $C_{PM}$ | 1.26E+05 | 1.38E+06 | 7.57E+06 | 1.33E+05 | 2.72E+06 | 7.58E+06 | 1.39E+05 | 3.81E+06 | 7.62E+06 | 1.26E+05 |
| $t_{PTS}$ | *24.24* | 102.29 | 150.56 | *10.69* | 39.91 | 49.51 | *13.85* | 28.08 | 30.04 | *10.26(192)* |
| $t_{PM}$ | 84.55 | 48.24 | 36.21 | 26.28 | 17.22 | 12.55 | 28.72 | 25.65 | 23.15 | 21.51(256) |
| 82MILLIONS | | | | | | | | | | |
| $C_{PTS}$ | *1.46E+05* | *1.90E+06* | *1.08E+07* | *1.44E+05* | *3.95E+06* | *1.09E+07* | *1.40E+05* | *5.57E+06* | *1.09E+07* | *1.45E+05* |
| $C_{PM}$ | 1.78E+05 | 2.12E+06 | 1.13E+07 | 1.69E+05 | 4.18E+06 | 1.14E+07 | 1.73E+05 | 5.95E+06 | 1.14E+07 | 1.61E+05 |
| $t_{PTS}$ | *46.48* | 189.42 | 297.76 | *17.98* | 75.86 | 91.52 | *23.26* | 46.91 | 61.54 | *16.93(192)* |
| $t_{PM}$ | 176.40 | 85.87 | 76.42 | 48.38 | 24.43 | 21.63 | 32.83 | 30.22 | 26.90 | 30.00(256) |

Table 4: Comparison between PT-Scotch (PTS) and ParMeTiS (PM) for representative numbers of processes and parts. $C_{PTS}$ and $C_{PM}$ ($t_{PTS}$ and $t_{PM}$) are the size of the edge cut (the execution time in seconds) for PTS and PM, respectively. $P_{peak}$ is the number of processors on which PTS and PM recorded peak performance for each graph. The number in parentheses right after the execution time indicates actual $P_{peak}$. Daggers indicate abortion due to an invalid MPI operation.

| Test | Number of processors:Number of parts | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| case | 32:2 | 32:32 | 32:1024 | 128:2 | 128:128 | 128:1024 | 384:2 | 384:256 | 384:1024 | $P_{peak}$:2 |
| AUDIKW1 | | | | | | | | | | |
| $C_{PTS}$ | *1.08E+05* | 2.08E+06 | 1.00E+07 | *1.06E+05* | 4.22E+06 | 9.99E+06 | *1.05E+05* | 5.81E+06 | 9.96E+06 | *1.11E+05* |
| $C_{PM}$ | 1.14E+05 | 2.04E+06 | 9.76E+06 | 1.12E+05 | 4.15E+06 | 9.75E+06 | 1.15E+05 | 5.76E+06 | 9.76E+06 | 1.12E+05 |
| $t_{PTS}$ | *3.51* | 11.84 | 17.35 | 2.90 | 8.72 | 9.29 | 5.87 | 10.72 | 10.06 | 3.01(128) |
| $t_{PM}$ | 3.90 | 3.59 | 5.27 | 2.42 | 2.01 | 2.97 | 4.45 | 4.62 | 4.51 | 2.37(192) |
| BRGM | | | | | | | | | | |
| $C_{PTS}$ | 3.46E+05 | 3.50E+06 | 2.17E+07 | 3.49E+05 | 7.60E+06 | 2.17E+07 | 3.30E+05 | 1.09E+07 | 2.15E+07 | 3.49E+05 |
| $C_{PM}$ | † | † | † | † | † | † | † | † | † | † |
| $t_{PTS}$ | 7.86 | 22.15 | 58.17 | 5.34 | 18.93 | 24.28 | 8.39 | 19.33 | 19.40 | 5.34(128) |
| $t_{PM}$ | † | † | † | † | † | † | † | † | † | † |
| CAGE15 | | | | | | | | | | |
| $C_{PTS}$ | *7.66E+05* | *3.39E+06* | *9.26E+06* | *7.53E+05* | *5.16E+06* | *8.93E+06* | *7.80E+05* | *6.22E+06* | *8.72E+06* | *7.53E+05* |
| $C_{PM}$ | 8.39E+05 | 3.98E+06 | 1.04E+07 | 8.44E+05 | 6.05E+06 | 1.06E+07 | 7.85E+05 | 7.32E+06 | 1.09E+07 | 8.23E+05 |
| $t_{PTS}$ | 31.07 | 82.96 | 100.97 | 29.70 | 62.80 | 64.90 | 41.86 | 85.30 | 79.14 | 29.70(128) |
| $t_{PM}$ | 11.24 | 9.67 | 13.13 | 6.81 | 5.69 | 8.90 | 26.51 | 25.67 | 21.42 | 6.45(64) |
| COUPOLE8000 | | | | | | | | | | |
| $C_{PTS}$ | *3.08E+03* | *9.56E+04* | *3.17E+06* | *3.08E+03* | *3.92E+05* | *3.17E+06* | *3.08E+03* | *7.88E+05* | *3.17E+06* | *3.08E+03* |
| $C_{PM}$ | 3.13E+03 | 9.91E+04 | 3.28E+06 | 3.20E+03 | 4.19E+05 | 3.28E+06 | 3.14E+03 | 8.39E+05 | 3.28E+06 | 3.14E+03 |
| $t_{PTS}$ | *1.68* | 6.76 | 10.65 | *0.83* | 2.96 | 3.73 | 2.05 | 4.80 | 4.88 | *0.83(128)* |
| $t_{PM}$ | 3.46 | 2.84 | 2.51 | 1.47 | 1.62 | 1.31 | 0.87 | 0.89 | 0.94 | 0.87(384) |
| THREAD | | | | | | | | | | |
| $C_{PTS}$ | *5.60E+04* | 6.15E+05 | *1.82E+06* | *5.60E+04* | 1.03E+06 | *1.82E+06* | *5.60E+04* | *1.29E+06* | *1.82E+06* | *5.62E+04* |
| $C_{PM}$ | 5.62E+04 | 6.03E+05 | 1.84E+06 | 5.67E+04 | 1.02E+06 | 1.85E+06 | 5.73E+04 | 1.29E+06 | 1.84E+06 | 5.63E+04 |
| $t_{PTS}$ | *0.53* | 0.97 | *1.07* | *0.60* | 1.08 | *1.05* | *0.85* | 1.27 | *1.28* | *0.47(16)* |
| $t_{PM}$ | 0.77 | 0.75 | 1.99 | 0.70 | 0.67 | 1.98 | 2.00 | 0.89 | 2.07 | 0.52(8) |

Table 5: Continuation of Table 4.

Table 4 presents the execution times and the cut sizes yielded by PT-Scotch and ParMeTiS for our test graphs, for representative numbers of processes and parts.

On most of these test cases, the partitions computed by PT-Scotch compare favorably to the ones produced by ParMeTiS. This gain can be as high as 20 % when bipartitioning graph 82millions, irrespective of the number of processes. PT-Scotch always computes better results for small numbers of parts, while ParMeTiS produces marginally better cuts for three graphs, namely audikw1, thread and brgm, when the number of parts increases. This phenomenon is due to the fact that, to date, PT-Scotch performs $k$-way partitioning by means of recursive bipartitioning, while ParMeTiS uses a direct $k$-way algorithm. Consequently, the quality of the partitions produced by PT-Scotch is likely to degrade for large numbers of parts, because of the greedy nature of the recursive bipartitioning scheme which prevents reconsidering earlier choices. It is therefore not surprising that the three graphs for which ParMeTiS gains over PT-Scotch are the ones of higher degree, for which bad decisions in the earlier bipartitioning stages produce higher penalties in terms of cut.

However, this phenomenon is most often compensated by the improvement in quality yielded by folding-with-duplication and multi-sequential phases. Indeed, for most of the test graphs, both PT-Scotch and ParMeTiS exhibit stable partition quality for all numbers of parts and up to 384 processes (which is the largest number of processes in our experiment), as can be seen for instance for graph 10millions in Figure 4.

However, in the case of graph cage15, partition quality for ParMeTiS decreases as the number of processes and parts increases, while it improves for PT-Scotch. This graph, which is not a mesh, has many topological irregularities (in terms of degrees and connectivity) which are likely to create coarsening artefacts, and therefore require efficient local optimization during the uncoarsening phase of the multi-level framework. As we have exposed in Section 2.1, and as had been already evidenced in the context of parallel ordering [20], the relaxation of the sequentiality constraint in the FM implementation of ParMeTiS hinders its efficiency when the number of processes increases.

To compare the relative efficiency of partition quality between PT-Scotch and ParMeTiS based on the data collected, we have plotted the ratio between the cut sizes yielded by these two tools, in Figures 5 and 6. On mesh graphs, the relative efficiency becomes close to 1 as the number of pars in-

creases. Our assumption is that the decrease in partition quality due to the greedy nature of our recursive bipartitioning algorithm starts to overwhelm the gain of our refinement methods for thousands of partitions.

The negative impact of recursive bipartitioning is of course even more perceptible for run time. While PT-Scotch can be more than three times faster than ParMeTiS in the bipartitioning case, such as for graph 82millions, when the number of parts increases, its execution time suffers a penalty factor which tends to a constant proportional to the inverse of the coarsening ratio, as evidenced in Figures 7 and 8. These figures represent the running times of PT-Scotch for our test graphs, with respect to the numbers of processes and parts. As the number of parts increase, the height of the plots increases and tends to a limit value.

Figure 7 plots the execution times of PT-Scotch for graphs 10millions, 23millions, 45millions and 82millions, which have similar topological characteristics. We could not collect data on 2 processors for 45millions, and from 2 to 16 processors for 82millions, as the pieces of the distributed graphs counld not fit in the memory of the nodes. The same also happened to ParMeTiS.

In order to analyze the time scalability of PT-Scotch, we focus on the bipartitioning case, for which the penalty factor has no impact. A quick look at the plots shows that PT-Scotch is scalable up to 64 processors for graph 10millions, and up to 128 processors for 23millions. However, the peak speed is still reached at 128 processors for 45millions, without significant speed-up for 192 processors. For 82millions, we can see a slight speed-up for 192 processors, while the rise in run time which appeared on 256 processors for graph 45millions is reduced. As argued in [21], significant increase in the graph size was required, for ParMeTiS to achieve constant parallel efficiency. This does not seem to be different with PT-Scotch from the asymptotic point of view. Consequently, to evidence some speed-up on 256 processes, a graph at least four times larger than 23millions is required, as the plots suggest.

Figure 8 plots the execution times of PT-Scotch for graphs audikw1, coupole8000, cage15, and brgm. Even though these graphs may have different topological properties than the ones of Figure 7, the same scalability properties and limits can be evidenced: the small sizes of the graphs limit the time scalability to 128 processes.

As a last comparison between PT-Scotch and ParMeTiS, the last column of Table 4 presents the best execution time, and the associated cut size, obtained by PT-Scotch and ParMeTiS when bipartitioning each of the graphs. We can see that PT-Scotch mostly produces partitions of better quality in smaller time. Moreover, PT-Scotch shows peak performance using less numbers of processors than ParMeTiS for mesh graphs. This lack of scalability is, in our opinion, caused by our recursive bipartitioning scheme, which requires many data movements between processes, all the more when graph pieces are spread across many of them.

## 5. Conclusions

In this paper, we have described the design of a parallel graph partitioning software, based on a parallel multi-level framework taking advantage of novel parallel algorithms for matching and refinement. We compared the results of our implementation with another parallel graph partitioner, ParMeTiS and showed that PT-Scotch most often produces partitions of higher quality. In particular, when considering the bipartitioning case, for which we do not take into account the penalty induced by our recursive bipartitioning method, PT-Scotch can be up to three times faster than ParMeTiS. We evidenced reasonable scalability with respect to the number of processes, for graphs of sufficient sizes. We assume that scalability is hindered by the recursive bipartitioning scheme, which requires many data movements.

Even though we did not present any empirical result on the smoothness of partition boundaries, we believe PT-Scotch produces smoother partition boundaries than the ones yielded by any other existent parallel graph partitioner. We base this assumption on our direct transposition in parallel of the sequential diffusive algorithm which already showed promising results [14].

It has been previously noted that recursive bipartitioning not only results in longer run time but also brings poorer partition quality, due to the greedy nature of the algorithm, compared to direct $k$-way partitioning [37]. As described in the previous section, we evidenced that, for the test graphs we considered, the overhead of recursive bipartitioning starts to overwhelm the gain of our refinement algorithm when considering more than a thousand of parts.

Consequently, we are currently developing a parallel direct $k$-way graph partitioning method, basing on our parallel multi-level framework, by ex-

tending our diffusion method to the $k$-way case. This extension has already been done in the sequential case by [38], basing on our findings, and its parallelization should be as straightforward as it has been for the 2-way case. This should give us the best of both worlds, in term of speed as well as in term of quality.

Yet, several limitations of the graph partitioning model have been reported [39]. A first shortcoming is the intrinsic inaccuracy of the edge cut metric to model actual communication costs. In this respect, hypergraph partitioning has been advocated as a more accurate model [40]. However, hypergraph partitioners are to date much more expensive to run than graph partitioners, which still leaves some room for the latter in practice.

Another limitation is the fact that recent parallel computing architectures are characterized by ever increasing numbers of processors and heavily heterogeneous communication subsystems. Taking into account the underlying topology of the target machine is therefore essential to the effective minimization of running time. Static mapping is the corresponding combinatorial problem, which aims at assigning statically parallel processes onto physical processors, so as to reduce a more realistic message congestion cost.

To date, the sequential Scotch tool allows one to compute static mappings rather than plain graph partitions, basing on recursive bipartitioning of both the source graph and the target architecture graph [41, 42]. This communication model is being extended to the parallel case, so that the upcoming version of our PT-Scotch tool will also be able to compute static mappings in parallel.

## References


[1] I. S. Dhilon, Y. Guan, B. Kulis, Weighted graph cuts without eigenvectors: A multilevel approach, IEEE Trans. Pattern Analysis and Machine Intelligence 29 (11) (2007) 1944–1957.

[2] M. Garey, D. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science 1 (1976) 237–267.

[3] T. Bui, C. Jones, A heuristic for reducing fill in sparse matrix factorization, in: Proc. $6^{th}$ SIAM Conf. on Parallel Processing for Scientific Computing, 1993, pp. 445–452.

[4] J. Cong, M. Smith, A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design, in: Proc. $30^{th}$ Annual ACM/IEEE International Design Automation Conference, 1993, pp. 755–760.

[5] S. T. Barnard, H. D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, Concurrency: Practice and Experience 6 (2) (1994) 101–117.

[6] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, in: Proc. ACM/IEEE conference on Supercomputing (CDROM), 1995, pp. 28–es.

[7] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.

[8] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitionning graphs, BELL System Technical Journal (1970) 291–307.

[9] C. M. Fiduccia, R. M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proc. 19th Design Automation Conference, IEEE, 1982, pp. 175–181.

[10] MeTiS: Family of multilevel partitioning algorithms, `http://glaros.dtc.umn.edu/gkhome/views/metis`.

[11] Jostle: Graph partitioning software, `http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/`.

[12] Zoltan: Parallel partitioning, load balancing and data-management services, `http://www.cs.sandia.gov/Zoltan/`.

[13] Par*k*way: Parallel hypergraph partitioning software, `http://www.doc.ac.uk/~at701/parkway/`.

[14] F. Pellegrini, A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries, in: Proc. Euro-Par'07, Rennes, Vol. 4641 of LNCS, Springer, 2007, pp. 191–200.

[15] R. Diekmann, R. Preis, F. Schlimbach, C. Walshaw, Aspect ratio for mesh partitioning, in: Proc. Euro-Par'98, Vol. 1470 of LNCS, 1998, pp. 347–351.

[16] C. Farhat, N. Maman, G. Brown, Mesh partitioning for implicit computation via domain decomposition: impact and optimization of the subdomain aspect ratio, Int. J. Numer. Math. Eng. 38 (1995) 989–1000.

[17] H. Meyerhenke, S. Schamberger, Balancing parallel adaptive FEM computations by solving systems of linear equations, in: Proc. Europar'05, Vol. 3648 of LNCS, 2005, pp. 209–219.

[18] R. Vanderstraeten, R. Keunings, C. Farhat, Beyond conventional mesh partitioning algorithms, in: SIAM Conf. on Par. Proc., 1995, pp. 611–614.

[19] J. Gaidamour, P. Hénon, A parallel direct/iterative solver based on a Schur complement approach, in: Proc. 11th Int. Conf. on Comp. Sci. and Eng., 2008, pp. 98–105.

[20] C. Chevalier, F. Pellegrini, PT-Scotch: A tool for efficient parallel graph ordering, Parallel Computing 34 (2008) 318–331.

[21] G. Karypis, V. Kumar, Parallel multilevel $k$-way partitioning scheme for irregular graphs, SIAM Review 41 (2) (1999) 278–300.

[22] C. Walshaw, M. Cross, Parallel optimisation algorithms for multilevel mesh partitioning, Parallel Computing 26 (2000) 1635–1660.

[23] K. Devine, E. Boman, R. Heaphy, R. Bisseling, U. Catalyurek, Parallel hypergraph partitioning for scientific computing, in: Proc. $20^{th}$ IEEE International Parallel and Distributed Processing Symposium, 2006.

[24] A. Trifunovic, W. Knottenbelt, Parallel multilevel algorithms for hypergraph partitioning, Journal of Parallel and Distributed Computing 68 (5) (2008) 563–581.

[25] M. Luby, A simple parallel algorithm for the maximal independent set problem, SIAM Journal on Computing 15 (4) (1986) 1036–1055.

[26] H. Meyerhenke, S. Schamberger, A parallel shape optimizing load balancer, in: Proc. Euro-Par'06, Vol. 4128 of LNCS, 2006, pp. 232–242.

[27] Y. Wan, S. Roy, A. Saberi, B. Lesieutre, A stochastic automaton-based algorithm for flexible and distributed network partitioning, in: Proc. Swarm Intelligence Symposium, IEEE, 2005, pp. 273–280.

[28] C. Asavathiratham, S. Roy, B. C. Lesieutre, G. C. Verghese, The influence model, IEEE Control Systems Magazine 21 (6) (2001) 52–64.

[29] R. J. Lipton, R. E. Tarjan, A separator theorem for planar graphs, SIAM J. on Appl. Math. 36 (1979) 177–189.

[30] A. Chan, P. Dehne, F. Bose, M. Latzel, Coarse grained parallel algorithms for graph matching, Parallel Computing 34 (1) (2008) 47–62.

[31] B. Hendrickson, A. Pothen, Combinatorial scientific computing: The enabling power of discrete algorithms in computational science, in: Proc. VECPAR conf., 2006, pp. 260–280.

[32] C. Chevalier, Conception et mise en œuvre d'outils efficaces pour le partitionnement et la distribution parallèles de problèmes numériques de très grande taille, Thèse de Doctorat, LaBRI, Université Bordeaux I (Sep. 2007).

[33] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, Tech. Rep. 95-035, University of Minnesota (Jun. 1995).

[34] C. Chevalier, F. Pellegrini, Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework, in: Proc. Euro-Par'06, Dresden, Vol. 4128 of LNCS, 2006, pp. 243–252.

[35] T. Davis, University of Florida Sparse Matrix Collection, `http://www.cise.ufl.edu/research/sparse/matrices/`.

[36] PARASOL project (EU ESPRIT IV LTR Project No. 20160) (1996–1999).

[37] H. D. Simon, S.-H. Teng, How good is recursive bipartition, SIAM J. Scientific Computing 18 (5) (1997) 1436–1445.

[38] H. Meyerhenke, B. Monien, T. Sauerwald, A new diffusion-based multilevel algorithm for computing graph partitions of very high quality, in: Proc. 22nd IPDPS, 2008.

[39] B. Hendrickson, T. G. Kolda, Graph partitioning models for parallel computing, Parallel Computing 26 (2000) 1519–1534.

[40] U. Çatalyurek, C. Aykanat, A hypergraph-partitioning approach for coarse-grain decomposition, in: Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM), 2001, pp. 28–es.

[41] F. Pellegrini, Static mapping by dual recursive bipartitioning of process and architecture graphs, in: Proc. SHPCC'94, Knoxville, IEEE, 1994, pp. 486–493.

[42] F. Pellegrini, J. Roman, Experimental analysis of the dual recursive bipartitioning algorithm for static mapping, Research Report, LaBRI, Université Bordeaux I, available from `http://www.labri.fr/~pelegrin/papers/scotch_expanalysis.ps.gz` (Aug. 1996).

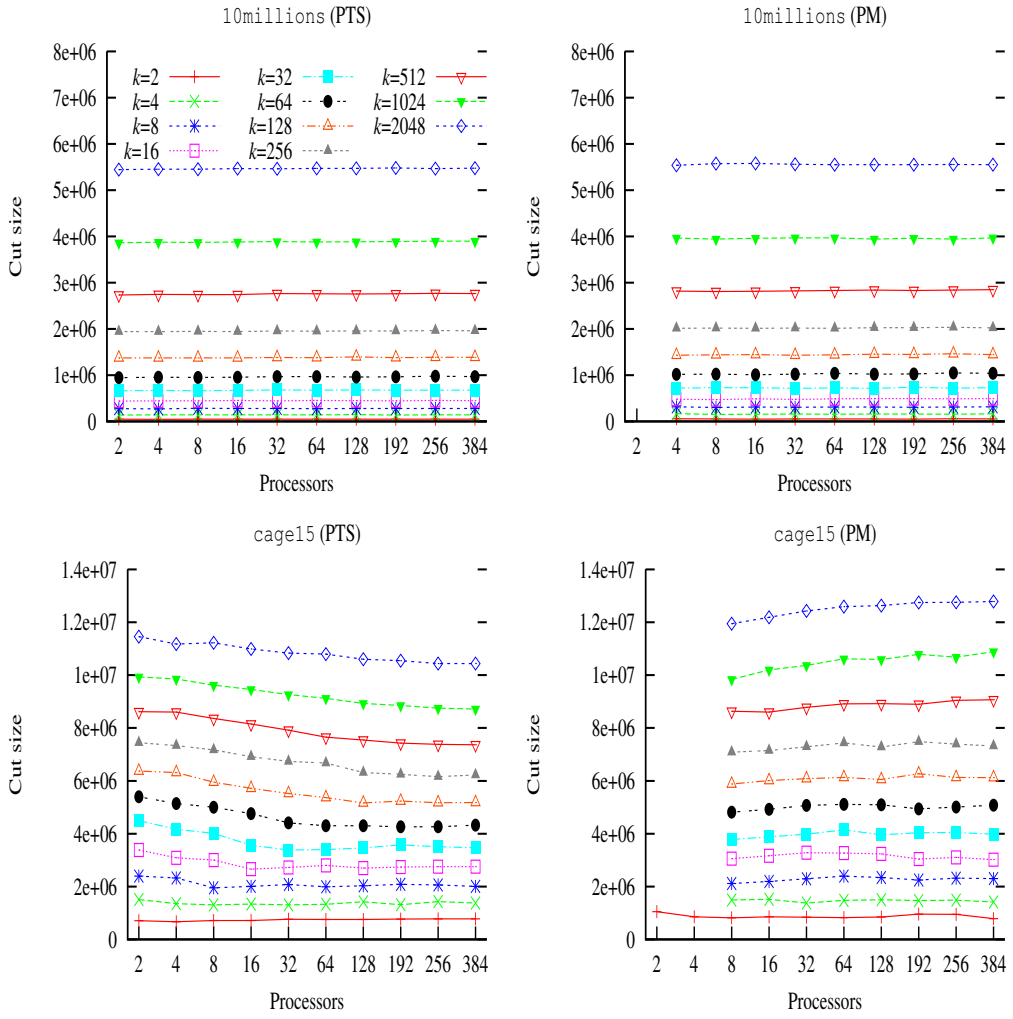


Figure 4: Cut sizes of PT-Scotch (PTS, left) and ParMeTiS (PM, right) for graphs 10millions and cage15.

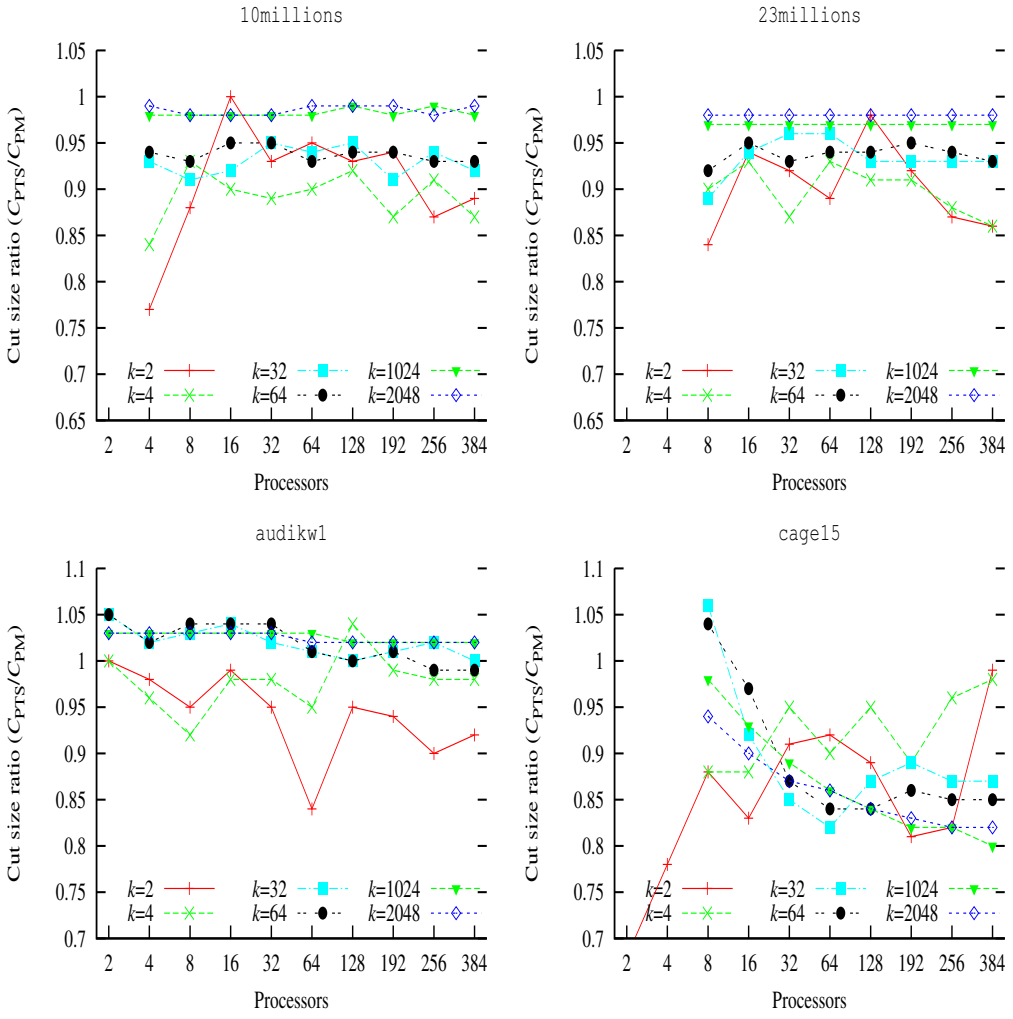


Figure 5: Cut size ratio between PT-Scotch (PTS) and ParMeTiS (PM) for graphs 10millions, 23millions, audikw1, and cage15. $C_{PTS}$ and $C_{PM}$ are the size of the edge cut for PTS and PM, respectively.

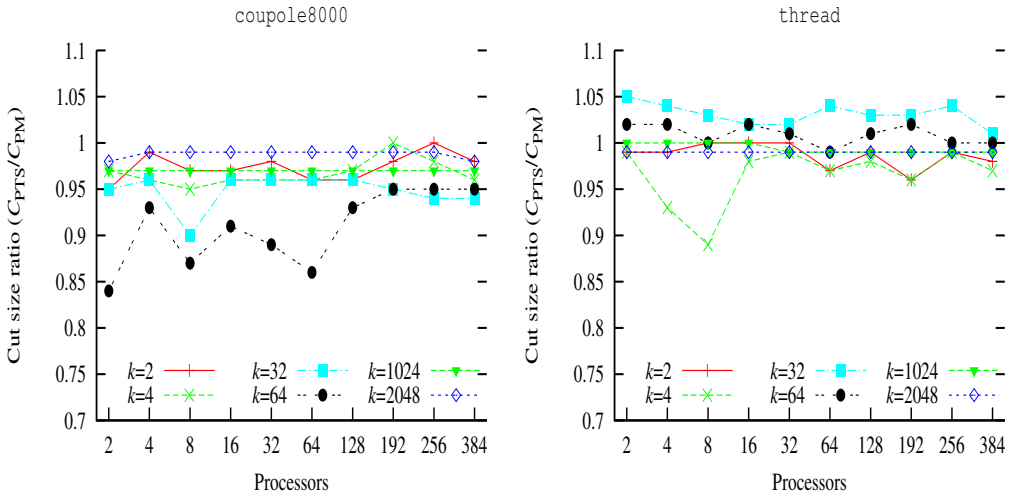


Figure 6: Cut size ratio between PT-Scotch (PTS) and ParMeTiS (PM) for graphs coupole8000 and thread. $C_{PTS}$ and $C_{PM}$ are the size of the edge cut for PTS and PM, respectively.

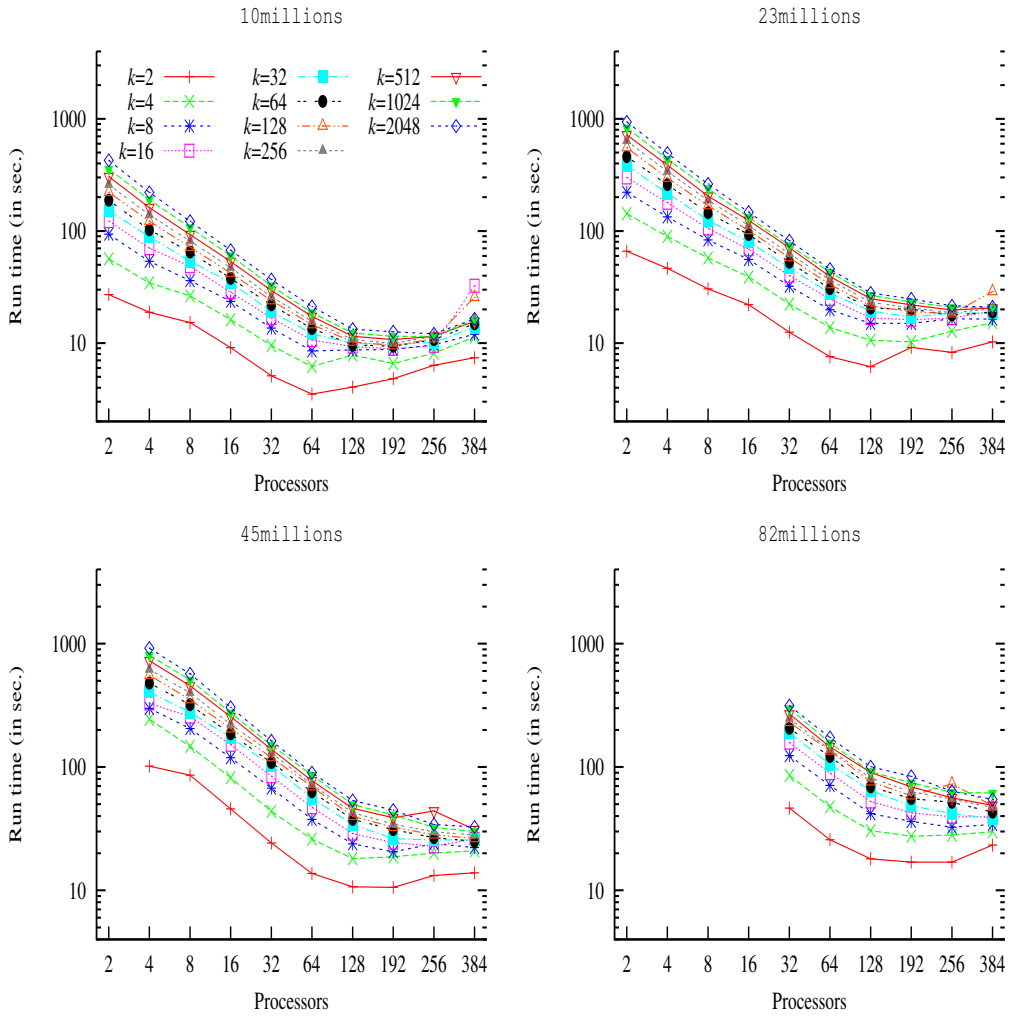


Figure 7: Execution times of PT-Scotch for graphs 10millions, 23millions, 45millions and 82millions.

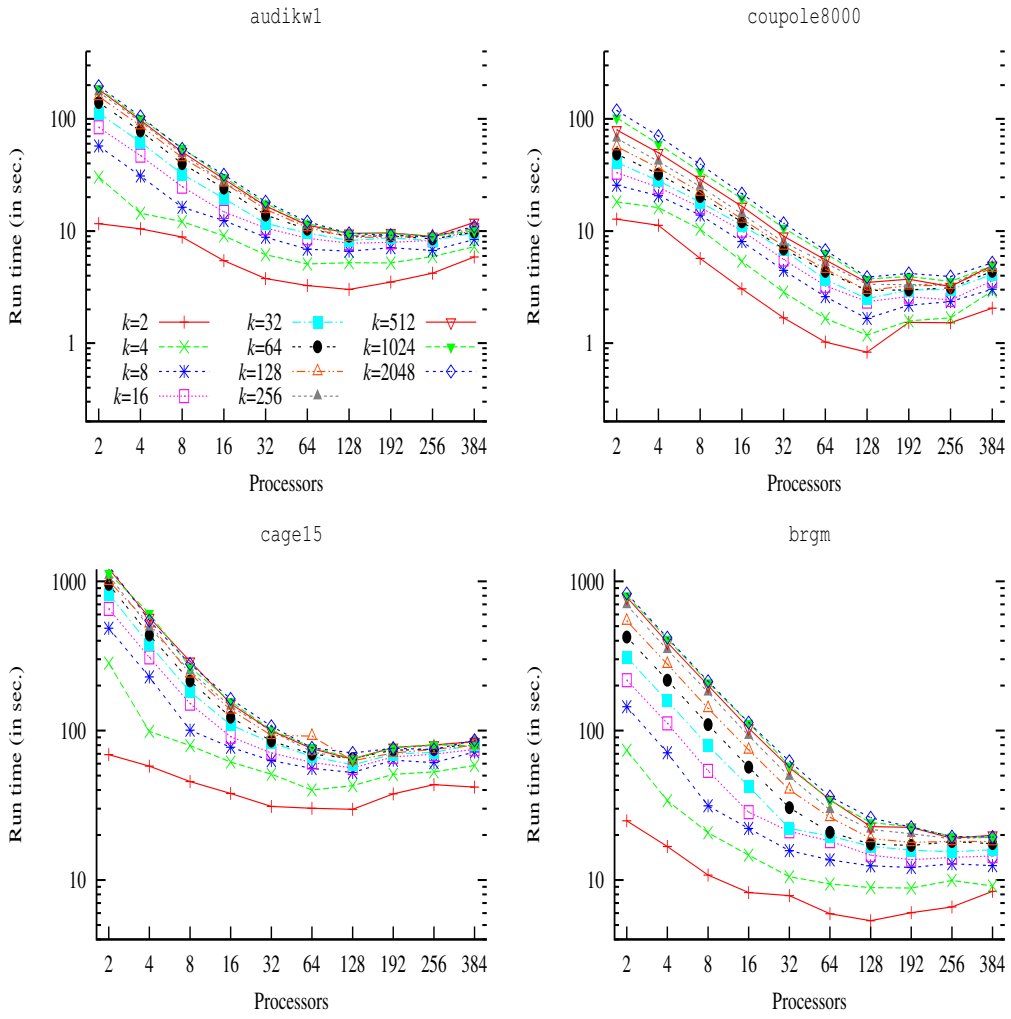


Figure 8: Execution times of PT-Scotch for graphs audikw1, coupole8000, cage15 and brgm.