McGill University
School of Computer Science
ACAPS Laboratory

Advanced Compilers, Architectures
and Parallel Systems

Automatic Partitioning Techniques for Solving Partial Differential Equations on Irregular Adaptive Meshes

Jérôme Galtier

ACAPS Technical Memo 97
September 4, 1995

ACAPS • School of Computer Science • 3480 University St. • Montréal • Canada • H3A 2A7
FTP: ftp-acaps.cs.mcgill.ca • WWW: http://www-acaps.cs.mcgill.ca/

Abstract

We present some original automatic partitioning techniques for irregular sparse matrices arising from Finite-Element discretizations of PDE. We discuss their efficiency in terms of parallel computation, especially from the point of view of adaptive applications, that need rebalancing after small changes on the grid. Some parallel simulations are presented, along with practical experiments on a KSR and a SGI-Challenger.

Contents

List of Figures

1 Introduction

Our work concerns the distribution of a task graph among several processors. This problem is crucial for the parallelization of iterative solvers of partial differential equations, but is also used in direct methods, and has a large number of other applications in computer science, such as VLSI for instance.

More precisely, let $G = (V, E)$ be an undirected graph. We try to find a set $(G_i = (V_i, E_i))_{1 \leq i \leq p}$ of subgraphs of G verifying the following properties:

C	coverage	$V = \bigcup\{V_i/1 \leq i \leq p\}$ and $E = \bigcup\{E_i/1 \leq i \leq p\}$
B	balancing	For all i, for all j, $\mathbf{Work}(G_i) \simeq \mathbf{Work}(G_j)$
C	low communication	For all i, $V_i \cap \bigcup\{V_j/1 \leq j \leq p$ and $j \neq i\}$ is as small as possible
A	adaptivity	Given significant but small changes on G, recompute cheaply a new partitioning $(G_1, \ldots, G_p)$ fitting the previous criterias

The notion of work has to be defined properly. As a first approximation, the reader can assume that $\mathbf{Work}(G = (V, E)) = |V|$, but more sophisticated work functions will be proposed.

Ideally, the three first criterias aim at spreading the graph onto p processors, each of them being assigned a $G_i, 1 \leq i \leq p$. Coverage is a consistency property that insures that each node is owned by one and only one processor. Then, balancing aims at insuring that each processor is assigned an almost equivalent amount of work. The low communication criteria tends to reduce the volume of the communications between the processors. Finaly, adaptivity introduces the idea that the same graph may be slightly modified and reused several times during the computation, and the algorithm has to provide quickly an adapted partitioning (i.e. that fulfill the three first properties). Since this criteria will be satisfied by updating and reusing some kind of information on the graph, we also call it fullfilment reusability.

In this paper, various algorithms will be presented and studied that verify this CBCA criteria. Some people are also concerned with other criterias, such as numerical iteration speedup. We mention this approach later and justify our framework.

This work was more particularly motivated by the last criteria, adaptivity. This problem arises from the resolution of partial differential equations by the Finite-Element method. Three different graphs have to be distinguished in that case:

- G_Ω is the graph that emphases the discretization of the physical space. Nodes of G_Ω are points in the 2D or 3D space, and edges are sides of triangles (or any other element). An example is shown in figure 1. G_Ω can be well characterized because of the underlying physics. For instance, G_Ω is typically a triangulation for 2D problems (i.e. it is planar as a graph), or some more "volumic" structure (3D-triangulation) for 3D problems.

- G_A is the adjacency graph of the matrix A with which we solve the problem

$$A \cdot X = B$$

  Nodes of G_A are entries of the matrix A, and edges are representing non-zero coefficients between entries of A. The structure of G_A is very simply derived from that of G_Ω. For simplicity, it is assumed in the following that $G_A = G_\Omega$.

- G_C is the task-graph of the computation made on A. G_C is derived from G_A, in a way that may be either simple or complicated.

As a consequence, while G_C represents exactly the amount of computation, only G_Ω can be well characterized by its properties derived from physics. Hence, the **Work** function, applied on G_Ω, emphasizes the difference that exists between these two graphs, and allows us to use the interesting physical properties. Depending on the nature of the relationship between the two graphs, this function more or less easy to evaluate.

To improve the numerical solution of a PDE problem, one may rebuild completely the grid and increase *uniformly* the new elements (for instance, by dividing every triangle into four in 2D and into eight in 3D). Another approach, called adaptation, consists in modifying the grid *locally*, concentrating the elements only where the solution needs to be improved, as shown in figure 2. The drawback of such a solution is that the initial partitioning is no longer valid; in particular the load-balancing criteria needs to be revisited.

Partitioning problems have already been investigated along many various directions, so in section 2 we present the position of our work with respect to previous work, and the major assumptions of our framework.

In section 3, three algorithms are described and analyzed with respect to the four CBCA conditions above (Coverage/Balancing/Low Communication/Adaptivity). problem, and study how they fulfill the criterias. The first one is a well-known partitioning scheme that has already been investigated in detail (see [17]), while the second one is, as far as we know, a completely new algorithm, and the third one is a modification of a recursive technique which allows to reduce substantially the amount of operations.

In section 4, those algorithms are tested on a simulator, on a KSR1, and a SGI-Challenge. A distributed-memory programming model was used, and a special numerical treatment of subdomains boundaries was introduced. Finally, the numerical impact of this strategy on the final solution is investigated.

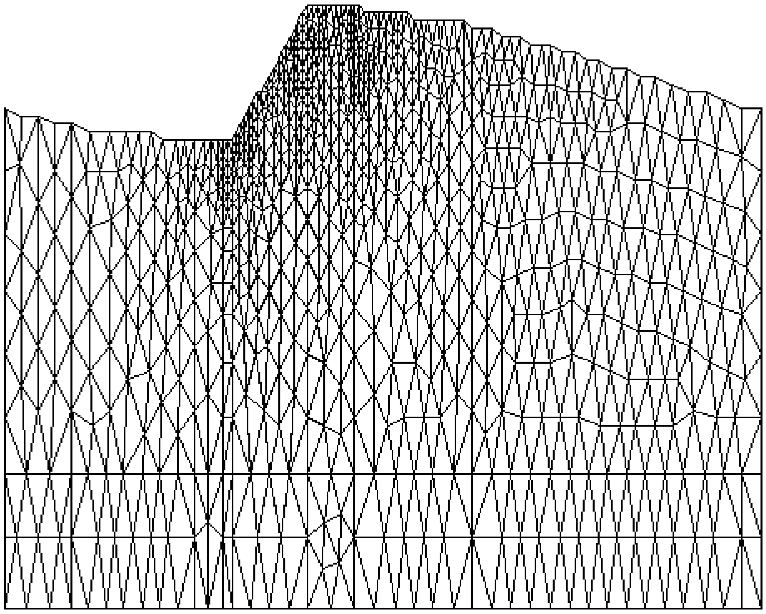


Figure 1: A planar triangulation

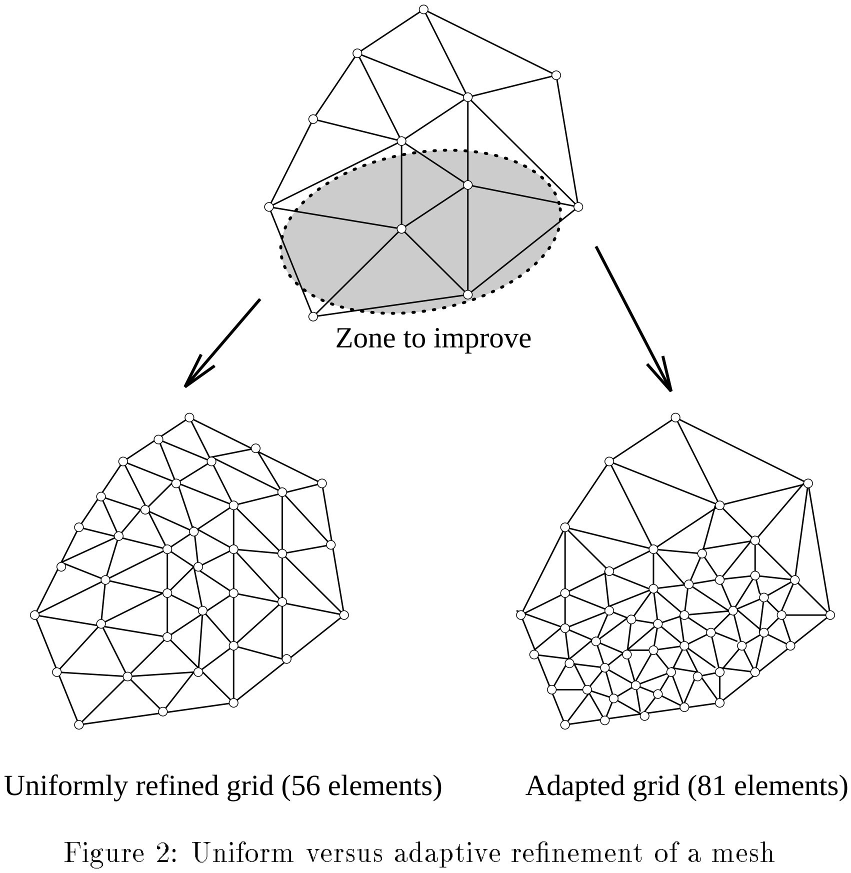


Figure 2: Uniform versus adaptive refinement of a mesh

2 State of the art

2.1 Overview of the field

The problem of partitioning finite-element grids has been studied in many ways and a large number of algorithms and methods have been proposed and investigated. Besides the four criteria listed in the introduction, an additional criteria is the impact of a partitioning on the numerical quality of an iterative solver.

The problem is that the partitioning affects the order in which operations will be performed on the matrix, and thereby has a consequence on the quality of the numerical result obtained. Modelizing this phenomenon leads to take into consideration not only the *iteration speedup* but the *global speedup* of a scheme - see [19, 10] (where the timings measure the delay to reduce the approximate solution to the real one within some fixed error tolerance). However this last criteria is strongly dependent upon the nature of the equations solved, making the problem very difficult to solve in a general setting. Fortunately, in practice, it appears that the impact of a partitioning on the numerical properties is relatively small. Therefore, we will take this criteria into account a posteriori and not a priori. Moreover, some modifications of the numerical method at the subdomain level may be used to decrease the impact of the numerical criteria.

2.2 Related work & adaptivity

A very interesting challenge is to find an adaptive version of each efficient partitioning algorithm. We mention here some popular methods for which adaptivity raises a particularly hard question in our opinion.

Historically, theoretical results that guarantee some minimization of the separator size (cf criteria on low communication) have appeared quite early. For instance, planar graphs with n vertices may be separated in two domains A and B, each of them having less than $\frac{2}{3}n$ vertices, with a separator C having at most $\frac{3}{2}\sqrt{2n}$ vertices [3]. (Some similar work has been recently explored in 3D : [20, 16, 7]). The original proof of Lipton and Tarjan [14] gave also an $O(n)$ algorithm to compute such a separator. But this work, when applied to adaptive grids, suffers from three main drawbacks:

- it lacks precision: we would like to partition the original graph into two subgraphs A and B such that $|A|$ and $|B|$ are as close as possible to $\frac{n}{2}$ (cf criteria on balancing). In fact, the author propose a method to achieve such a property, and the resultant algorithm has still a complexity of $C.\sqrt{(n)}$, but C is a large constant. This last characteristic makes the algorithm unattractive for practical problems.

- Although is complexity is still $O(n)$, the constant is also quite large, so its computation is costful nevertheless.
- It is unable to take into account modifications of the original grid: if the grid is slightly changed, it is necessary to recompute entirely the separator.

Some partitioning methods have been designed for handling more general kinds of graphs than planar. But in general the computational complexity is prohibitive. For instance, spectral methods are based on the analysis of a particular matrix, called Laplacian, built from the graph to be partitioned [15, 18, 4]. The idea is that the second eigenvector of the laplacian matrix contains some information on the connectivity of the graph. Although significant improvements have been realized in this field [6], such an information is still hard to compute. According to the tests in [8], it turns out that the spectral bisection algorithm tends to provide high quality separators at expensive CPU price. This is exactly what we would like to avoid. Nevertheless, integrating adaptivity capabilities in this class of algorithms is an important problem to investigate.

2.3 Criterias of quality

- **Balancing.** Since our work aims to parallelize some numerical computation on a grid, it is expectable to have some kind of work-efficient final algorithm. Each processor will receive a workload proportional to its own speed, therefore we need almost perfectly balanced workloads for the processors on the different subdomains. Of course, this **Work** function depends on the particular numerical application being performed, but some basic functions (that is, amounts of vertices, elements or edges per subdomain) provide a sufficient approximation in many cases. Indeed, these amounts are proportional in practice, and any of them gives a well-fitted work function for the examples we will consider.

  However, some methods require a more sophisticated measure of a domain's work. For instance, hybrid techniques use a partial LU factorization to precondition the iterations. In that case, the work depends on the particular ordering of the vertices the LU factorization has used. Then, it would be interesting to take into consideration some parameters like the domain's diameter, the minimum fill-in number (A NP-hard parameter ! [21]) and the domain's depth. We have no knowledge of an application that makes such a precise work's measure, and it is certainly an interesting field to investigate.

- **Separator size / communication** In general, the amount of communication between two processors is proportional to the size of the separator that they share. It is clear that a perfectly balanced partition having an enormous separator will lead to some unbearable parallel speedup because of the communication bottleneck. It would be possible to include this cost into our subdomain's work function. But this criteria is hard to take into consideration directly into a non-iterative partitioner, since the partition itself is an output. However, an algorithm that manage to guarantee some bounds on the partition's size will be introduced later.

At this point we have to mention that the architecture pattern of the communications between the processors may influence the performance of a partitioning on the numerical computation. For instance, real shared-memory machines are supposed to be independant of data location in memory, and therefore would not need good separators and "low communication". But in practice, nearly all parallel machines make use of cache memories, so that a bad data locality affects dramatically the final performance.

- **Reusability and computational time.** As we mentioned earlier, our goal is to find methods especially fitted to adaptive grids. In particular, the partition should be easily changed if some part of the grid is suddenly refined. Therefore, we expect either the total computational time to be short, or the reusability of the initial partitioning calculation to be great. To explore this criteria, we introduce the "shock-wave" concentration model: a local zone is filled with much more refined elements, while the remaining parts of the grid are unchanged. We imagined two means to characterize the phenomenon:

  - *Weight changes on the elements*: each of the elements of the domain is assigned a *weight* w, such that $w \geq 1$ and the total sum of the weights is less than $K.n$, where K is a threshold. It is clear that the incremental algorithm introduced in that case performs at least a little bit worse than an algorithm that works with the entire information on the grid.

    The signification of weights is as follows: an element of weight w is likely to be replaced by w small elements, in such a way that we are able to rebuild locally the separator, provided that it is balanced according to the weight.

    Since only linear, or almost linear algorithms are considered, an important modification of the grid (for instance, adding more that n elements) justifies the complete rebuilding of the partitioning. Hence the threshold K, that is fixed according to the particular performance of the application.

    We modelize this phenomenon because it is very common nowadays to use this model to realize adaptive grids. We explain in the following lines the basics of the construction. The reader does not have to understand completely the model, and may admit that changing weights represent properly a step of a major class of adaptive numerical computations. An original grid is fixed, and each element receives the level 0. The idea is that one "element" is something fixed that will be represented by a variable number of "triangles". When an element has to be refined, the element's level is increased. If an element has the level L, it will be represented by 4^L triangles.

    Now an interface problem occurs when two elements of a different level share an edge. When the consistency problem of nodes existing on one element and not the other is forgotten, the triangulation is said to be *non-conform* (See figure **??**).

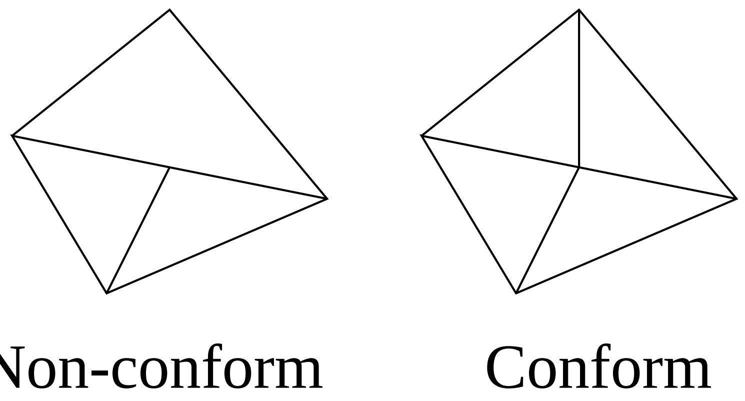


Figure 3: The conformity problem

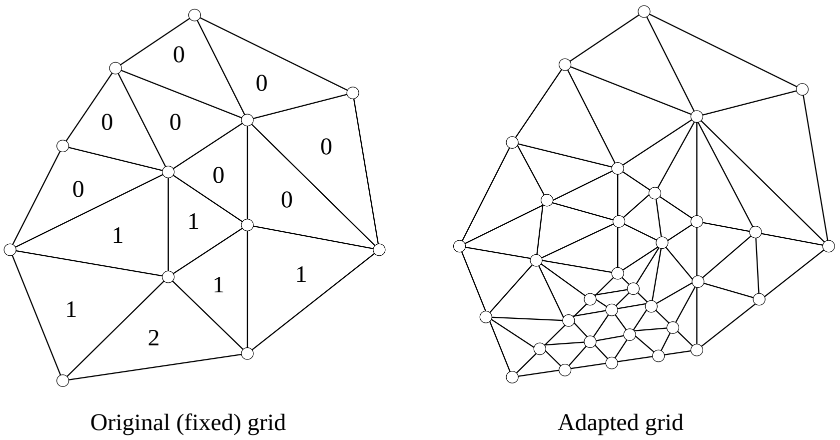


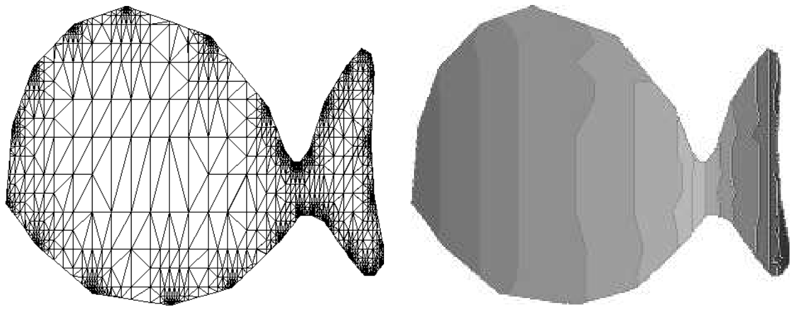
Figure 4: How level (and weights) are used to do adaption

Many finite-element users do not want to handle this case because it implies many other difficulties at various levels, including in the mathematical underlying model. To avoid this, the conformity of the triangulation has to be insured by rules, for instance

* An element of level L does not share an edge with an element of level greater than $L + 1$.
* An element of level L shares an edge with at most one element of level $L + 1$.

Then an element of level L is represented by 4^L triangles if it does not share an edge with an element of level $L + 1$, and $4^L + 2^L$ otherwise. Each edge of a $L + 1/L + 1$ or $L/L + 1$ interface will have two endpoints plus $2^{L+1} - 1$ internal vertices. Figure 4 shows how the new triangulation is built from the old one.

– *Addition of some more elements into the grid*: A more challenging goal would consist in being able to integrate directly some more elements - say $\delta(n)$ - into the data structure produced by the partitioner. This can be achieved by deleting a local zone of the mesh and rebuilding it locally (Such an approach is taken in [**?**]). Handling correctly this case implies that the data structure associated



Original triangulation Geometical sort along x

Figure 5: Examples of partitioning : part 1

to the partitioner is precisely updated according to the new element, whereas in the previous case, only an approximation was necessary. For the same reasons as before, a K threshold is fixed such that $\delta(n) \leq K.n$.

During the calculation, tasks are localized as far as possible inside the subdomains. Then we can expect that requests for grid modifications will frequently concern only the *internal* part of a subdomain. We will wonder how the described algorithm will take advantage of this property.

3 Adaptive partitioning algorithms

The algorithms described here are illustrated on a dummy "fish" example in figures 5 and 6. This shape was chosen for its simplicity and the clarity of the partitions on it.

3.1 Geometrical sort

* Initialization. *cost* $= O(n \log(n))$. For this algorithm, each element is assigned a *coordinate*, generally the x- or the y-coordinate of its center of gravity. It can be also a linear combination of them, or even a completely different function. Then, a sort is applied to it in order to build the partitions.

  In this paper, we studied especially the sort along one coordinate. We could also try to partition along several coordinates, which would certainly reduce the separator's size. But it would also damage the flexibility of the algorithm, and then would not be adequate to our adaptive goals. To maintain the reader aware of that restriction (and its cost), we introduced in the experiments a heuristic that takes advantage from the 2D-shape of the domain.

* Weight modification / Balancing. *cost* $= min(O(p\delta(n)), O(\delta(n) + n))$. In case of a weight modification, we either scan linearly the list or move progressively the partition's bounds (depending on $p\delta(n) \leq \delta(n) + n$ or not).

* ADDITION OF ELEMENTS. *cost* $= O(\delta(n)\log(n+\delta(n)))$*; on a local subdomain of size* s*, cost* $= O(\delta(n)\log(s+\delta(n))$. Adding an element to a sorted list of order n may be realized in $O(\log(n))$ steps if the correct data structured is used (The idea consists in performing a dichotomic choice on a balanced binary tree; see [1, 2] and the leftist binary tree in [13]). Once the sorted list of elements is updated, one needs oftenly to rescale the partitioning (This is exactly equal to the balancing step and costs $min(O(p\delta(n)), O(\delta(n)+n))$ operations; we distinguish nevertheless the two following actions: (1) modify the graph structure while preserving the data associated to it in order to balance the load, and (2) compute effectively the balanced partitioning).

* EXTENSION TO OTHER STRUCTURES OF GRAPH. The method is extensible as far as some *coordinates* may be associated to one node or element. In particular, 3D problems may be handled by this method.

* WARRANTIES. The method guarantees the amount of elements per subdomain, not the size of the separator.

3.2 Deepness analysis

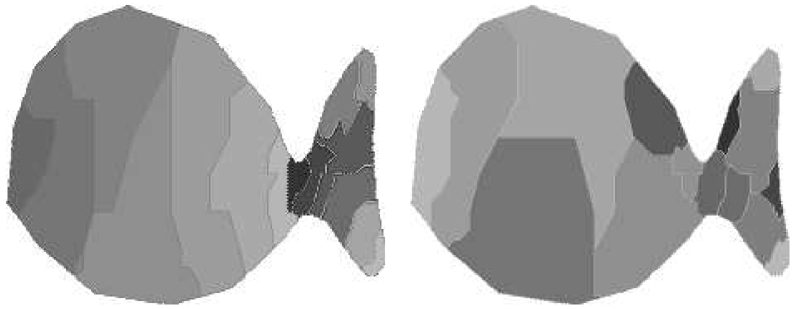
* INITIALIZATION. *cost* $= O(n)$. The method consists in computing a skeleton (called *spine*) of a triangulation. The reader will get more information on the way to build this structure in [11]. The spine length is less than n and in practice around $\sqrt{n}$.

  Similarly to the geometrical sort, it is possible to implement deepness analysis so that it benefits from the two-dimensional structure of the grid to reduce the separator's size. But this would also influence badly the flexibility of the algorithm towards adaptive grids.

* WEIGHT MODIFICATION / BALANCING. *cost* $= min(O(p\delta(n)), O(\delta(n)+n))$. The spine defines a kind of multilinear description of the domain, in the sense that, instead of being described along a list, the elements are ordered along a tree. Thereby, once a separator is chosen along the tree, a separator on the domain is implicitly defined. As a consequence, taking new weights into consideration can be performed in a similar way to the geometrical sort.

* ADDITION OF ELEMENTS. *cost* $= O(n+\delta(n))$*; on a local subdomain of work* s*, cost* $= O(s+\delta(n))$. Since the algorithm gather local and global information, it is necessary to recompute the associated data structure if some elements are randomly introduced. However, since the separators are "depth-optimal" fronts, it is possible to localize this recalculation on only one subdomain.

* EXTENSION TO OTHER STRUCTURES OF GRAPH. The methods works specifically with triangulated planar graphs, and does not need the geometrical coordinates of the nodes. Its extension to 3D is studied, but not established until now.



Deepness analysis Discretized K-means

Figure 6: Examples of partitioning : part 2

* WARRANTIES. The balancing performed in practice is oftenly perfect, but is only guaranteed to be no more than $\frac{3}{2}$ of the minimum possible. It is also warrantied that the total size of the straightforward separator into p subdomains will be at most $O(p\sqrt{n_T})$ for a uniformly regular triangulation, where n_T is the amount of triangles. A proof is given in the appendix.

3.3 Heuristic techniques

There are various iterative techniques that provide good separators in practice. On the one hand, their reusability is great, since they simply start from the previous partition to construct the new one. On the other hand, their granularity is small in general, because they oftenly need at least a linear-time computation on the complete grid. However, we believe that this type of methods will provide interesting adaptive partitioners in the future if sufficient care is taken on granularity aspects. We also find it useful to introduce some high quality separators to measure the cost of adaptivity.

We note that for the same iteration scheme a wide variety of algorithms may appear depending on the strategy of optimization being adopted (Taboo, Depth-First). Among the good iterative schemes, we have to mention the greedy algorithm (introduced by Charbel Fahrat [9]) and the K-means technique, successfully developed by Eric Saltel at the INRIA-Rocquencourt. We will use here a modification of this last method. It can be intuitively described as follows (for p subdomains:).

1 Choose arbitrarily p nodes called "centers" on the domain, numbered from 1 to p.

2 Assign to the subdomain i the nodes for which the center i is the closest center

3 Replace the center i by the center of gravity of all the elements in the subdomain i.

4 Goto 2 until convergence

Step 1	Chose arbitrarily p nodes $v_1, \ldots, v_p$ for i:=1 to p do **Center**[i] := v_i; **Label**[**Center**[i]] := i;
Step 2	for i:=1 to p do **Mark**(**Center**[i]); **Push**(**Center**[i]);
Step 3	While pile not empty do **Pop**(v); for each neighbor w of v if not **TestMark**(w) **Label**[w] := **Label**[v]; **Mark**(w); **Push**(w);
Step 4	**Center**[i] :=Closest node to the center of gravity of all nodes labeled with i
Step 5	Unmark all nodes; goto 2 until convergence

Figure 7: Discretized K-means technique using a FIFO pile

This method is based on notions of distance ("is closer to") and of center of gravity.

We present the details of our modified algorithm in figure 7. The original method is based on a weighted physical distance that tends to make any triangle equilateral, whereas we introduce an algebraic distance, using the graph topology. Since the grid generator is supposed to build elements as close as possible to the equilateral model, the global strategy remains unchanged. But this method has the advantage of (1) producing connected subdomains and (2) having an iteration step of $O(n)$ operations (while the previous one needs $O(np)$ operations, where p is the amount of subdomains).

* Initialization. *cost* $= O(n)$ *per iteration* The convergence is guaranteed statistically. We encountered some (rare) non-stable cases.

* Weight modification/Balancing. *cost* $= O(n)$ *per iteration.* It was possible in the original algorithm to influence the work of a domain - say i - by using a distance $f_i(d(w, v_i))$ from a node w to the center v_i instead of $d(w, v_i)$. In our method, we managed to handle balancing by introducing penalty steps for large domains, without modifying the iteration complexity (we penalized too large domains by introducing later their center into the FIFO pile). The improvement strategy was a depth-first one; we kept the same centers of domains until we managed, by manipulations on the "distance" to have a better balancing. Then we "fixed" the solution by computing new centers as centers of gravity of each subdomain.

  For adaptive applications, this strategy is rather crude. It would probably be interesting to propagate some weight modifications on a local subdomain to the neighboring sub-

Cost	Geometrical sort	Deepness analysis	Discretized K-means
Initialization	$O(n\log(n))$	$O(n)$	$O(n)$ per iteration
Weight modification / Balancing	$min(O(p\delta(n)), O(\delta(n)+n))$	$min(O(p\delta(n)), O(\delta(n)+n))$	$O(n)$ per iteration
Addition of elements	$O(\delta(n)\log(n+\delta(n)))$	$O(n+\delta(n))$	$O(n)$ per iteration

Table 1: Summary of the comparative study

domains only (or a little bit further). But then, the real complexity of the computation becomes hard to evaluate...

* **Addition of elements** *cost* $= O(n)$ *per iteration.* One may also consider that the cost is $O(\delta(n))$, since the algorithm doesn't need to update any structure associated to the graph apart from the partition itself. Then the operation would consists in assigning a subdomain to each element, a thing that may be performed in constant time by some heuristic.

* **Extension to other structures of graph.** The method is easily adaptable to 3D graphs and any other graph that admits a notion of distance and center of gravity.

* **Warranties.** No warranty on the result is known.

All these results are summarized in table 1.

4 Experimental results

4.1 A simulation based on maximum cost

We imagined a simulation program that took a partition and gave the parallel time of an iteration that used it. This simulated time was the maximum local work among all the subdomains, and the local work of a subdomain was given by

$$\text{local work} = \#(\text{internal triangles}) + c.\#(\text{shared nodes})$$

where c was some constant, usually greater than 1, introducing the communication latency. Then the *speedup* of one partition was defined by

$$\text{speedup} = \frac{\#(\text{triangles})}{\text{largest local work}}$$

and the goal was to reach as linear as possible speedups.

In figure 8, we present the speedups obtained for the various algorithms described before. The studied grid (which will be reused on the following experiments) has 5399 nodes (or vertices) and 10219 elements (or triangles), and the constant c equals 3. For the x- and y-geometrical sort and for the discretized K-means technique, we used a triangle-numbering cost function. We tried

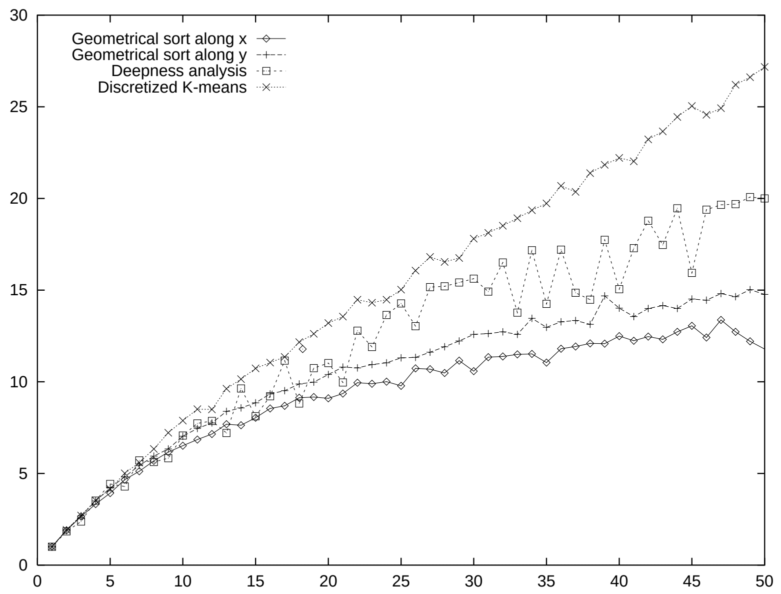


Figure 8: Simulated speedups with communication constant c=3

the "deepness analysis" algorithm with a slightly modified cost, where the spine intersections were taken into consideration, but this hardly improved the results. In order to get results as perfect as possible, K-means had 500 iterations to improve its solution, but a smaller number of iterations gave also good results. The computational time of this method was, however, much longer.

The results show clearly that the more "adaptive" an algorithm is, the worse it performs. We note that deepness can give good results, but is quite irregular. Therefore, we plan to add some regularizing features in the code to avoid deceiving speedups. It also appears that the choice of one direction has an important impact on the geometrical sort. Finally, the contrasted performances between the two extreme cases justify an additional effort to improve the granularity of heuristic methods.

4.2 Practical implementation and related problems

Since the operation matrix-vector product is too simple to be realistic in terms of communication, we tried to experience our techniques with some more sophisticated computation. The physical problem was a heat-conduction equation, assembled using the control-volume finite-element method [5], to obtain a global equation $A.X = B$. Then we chose a Gauss-Seidel iteration process, not for its own performance, but because of the inherent sequentiality of the model, the convergence rate would be affected by the parallelization (Recall that non overlapping domains are used). We discuss later the results we obtained likewise.

The Gauss-Seidel iteration step for dense matrices is presented in figure 9. Normally, our practical matrices for A are sparse, so that, for each vertex i, a small number of entries of both

$$\text{for } i{:=}1 \text{ to vertices do } X[i] := \frac{B[i] - \sum_{1 \le j \le vertices, j \neq i} A[i][j] \times X[j]}{A[i][j]}$$

Figure 9: A Gauss-Seidel iteration step

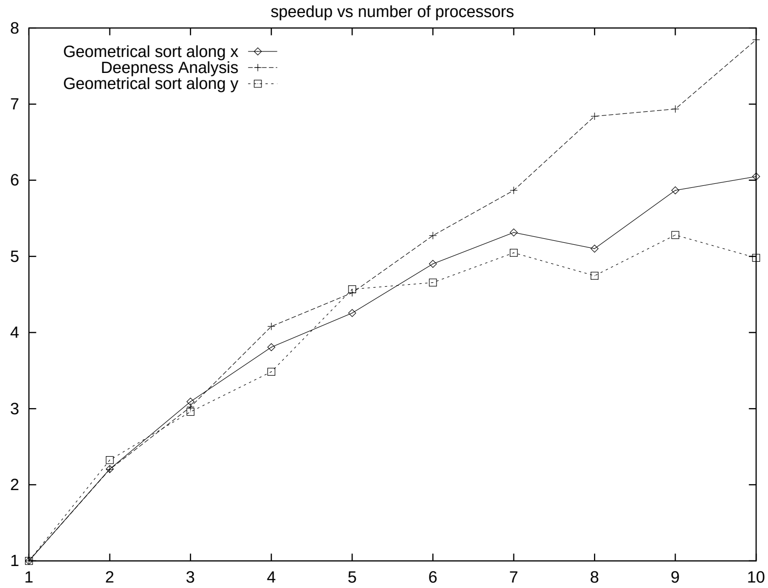


Figure 10: Speedups on the SGI-Challenge (150MHz, R4400)

A and X are accessed. While doing the parallel computation, each vertex is assigned to one or more subdomains (or, equivalently, processors). A vertex is called *internal* if it is affected to a unique domain and *shared* otherwise.

It is clear that a problem occurs when two different processors aim to update the same node at the same time. For instance, by updating randomly the value affected to a shared node, a processor may remove the result of a previous computation, hence the lost of a (local) iteration step. We could also observe on the SGI-Challenge that if no semaphore (i.e. a mutex lock for example) was used to insure consistency, some completely meaningless values may be generated by conflicting writes on the same memory location by two processors. We distinguish two means to insure consistency:

[local] A mutex lock is used whenever a processor updates a shared variable

[global] At each iteration step, some piece of information is spread, so that each processor will take into account others' computations.

On the Challenge, a "local" strategy lead to dramatical results because of the high latency introduced by the lock routine. In figure 12, we show the details of the particular global strategy that was used. This computation presents many similarities with the block-Jacobi method, but should not be identified to it: in fact it allows many more "crossed" references, and we believe that it improved significantly the quality of each parallel iteration step.

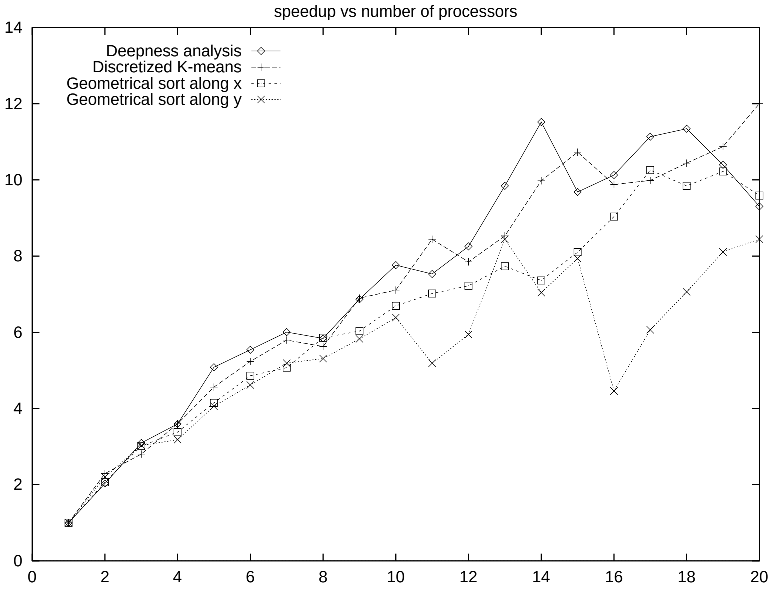


Figure 11: Speedups on the KSR1

Assembly

$Diag = 0_{[n]}$

for each p do
- $A^p = 0_{[n \times n]}$
- for each element e in subdomain p do $A^p := A^p + A^e$

for each p do
- for each node i in subdomain p do $Diag[i] := Diag[i] + A^p[i][i]$
- $PreviousSum^p = O_{[n]}$
- $X^p = O_{[n]}$ (or eventually equals some fixed values on the boundary)

Iterate

for each p do for each node i in subdomain p do

$$Sum^p[i] := \sum_{j \text{node in } p, j \neq i} A^p[i][j] \times X^p[j]$$
$$X^p[i] := \frac{B[i] - Sum^p[i] - \sum_{k \text{subdomain}, k \neq p} PreviousSum^k[i]}{Diag[i]}$$

for each p do for each node i in subdomain p do $PreviousSum^p[i] = Sum^p[i]$

Figure 12: Sequential description of the parallel algorithm

For some reasons, we tried our algorithms on a KSR1, and a SGI-Challenge, that are shared memory machines. But, as a result, we systematically avoid concurrent writing on the same memory locations, so that we used a completely distributed-memory programming model. We also stress the KSR1 is physically distributed.

Finally, the practical speedups obtained by this method are shown in figures 11 and 10

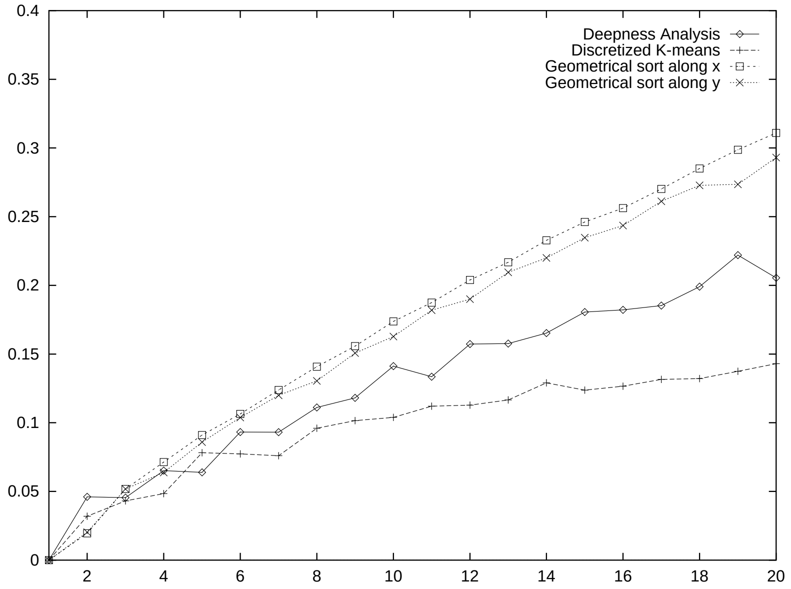


Figure 13: Logarithmic error vs number of processors

respectively for the KSR1 and the Challenge. These results were obtained by compiling our C code with the -O2 flag, and used averages on 10 tests using 1000 iterations each with double floating-point precision on non dedicated machines. A sequential iteration cost respectively 143 ms and 9 ms on the KSR1 and the Challenge (150 MHz, 16 proc., R4400). The measures of the error generated after 1000 iteration compared to the sequential case are given in figure 13, representing logarithmic (log10) errors versus number of processors, for each partitioning method.

We note that, apart from many unpredictable results on the KSR1, the curves correspond to the simulation tests. Another fact is that, in our particular example, the smaller a separator is, the less numerical error we get.

5 Conclusion and further work

Adaptive grids present a new, challenging area in computer science, especially while partitioning them. We presented some answers to this problem, and also showed their defaults. It would be also interesting to parallelize the partitioner itself. At this point, we precise that some algorithms previously mentioned and avoided have parallel implementations [12]. Of course, we are also interested in continuing this work from the point of view of adaptivity in 3D, and testing how these algorithms perform when the grid really adapts.

Another question concerns the grid generator. Is it possible to associate an automatic domain partitioner to it? Some more subtle work may be done in recognizing a smaller (than planar) class of graphs to build fast partitioners still adequate to PDE's domains.

Acknowledgements

I am greatly indebted to Prakash Panangaden, who motivated this work and invited me in McGill to do it, and Clark Verbrugge, who wrote the “cost” simulating function, and helped me for many details.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Ma., USA, 1976.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms.* Addison-Wesley, Reading, Ma., USA, 1983.

[3] Noga Alon, Paul Seymour, and Robin Thomas. Planar separators. *SIAM J. Discrete Math.*, 7(2):184–193, 1994.

[4] K. S. Arun and V. B. Bao. Constructive heuristics and lower bounds for graph partitioning based on a principal-components approximation. *SIAM J. Matrix Anal. Appl.*, 14(4):991–1015, 1993.

[5] B.R. Baliga and S.V. Patankar. Elliptic systems: Finite-element method ii. In W.J. Minkowycz, E.M. Sparrow, G.E. Schneider, and R.H. Fletcher, editors, *Handbook of Numerical Heat Transfer*, pages 421–461. John Wiley & Sons Inc., 1988.

[6] S. T. Barnard and H. D. Simon. A fast implementation of recursive spectral bisection for partitioning unstructured problems. In R. S. Sincovec et al., editor, *Parallel Processing for Scientific Computing*, pages 711–718. SIAM, 1993.

[7] David Eppstein, Gary L. Miller, and Shang-Hua Teng. A deterministic linear time algorithm for geometric separators and its applications. *Fundamenta Infomaticae*, 22:309–329, 1995.

[8] C. Fahrat and H. D. Simon. TOP/DOMDEC: A software tool for mesh partitioning and parallel processing. CU-CSSC-93-11, Center for Space Structures and Control, College of engineering, University of Colorado, 1993.

[9] C. Farhat. A simple and efficient automatic fem decomposer. *Comp. Struct.*, 28:579–602, 1988.

[10] Charbel Farhat and Michel Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Int. J. Numer. Engn*, 36:745–764, 1993.

[11] Jérôme Galtier. Deepness analysis: Bringing optimal fronts to triangular finite element method. ACAPS Technical Memo 73, School of Computer Science, McGill University, Montréal, Québec, September 1994. In ftp://ftp-acaps.cs.mcgill.ca/pub/doc/memos.

[12] Hillel Gazit and Gary L. Miller. A parallel algorithm for finding a separator in planar graphs. pages 238–248, Los Angeles, 1987. IEEE.

[13] Donald E. Knuth. *The Art of Computer Programming*, volume 1 / Fundamental Algorithms. Addison-Wesley Publishing Company, Reading, Massachusetts, 2nd edition, 1973.

[14] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[15] Joseph W. H. Liu. A graph partitioning algorithm by node separators. *ACM Transactions on Mathematical Software*, 15(3):198–219, 1989.

[16] Gary L. Miller, Shang-Hua Teng, and Stephen A. Vavasis. A unified geometric approach to graph separators. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 538–547, 1993.

[17] Claude Pommerell, Marco Annaratone, and Wolfgang Fichtner. A set of new mapping and coloring heuristics for distributed-memory parallel processors. *SIAM Journal on Scientific and Statistical Computing.*, 13(1):194–226, January 1992.

[18] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 1(3):430–452, 1990.

[19] P. Le Tallec. Domain decomposition method in computational mechanics. *Computational Mechanics Advances*, 1(2):121–220, 1993.

[20] Stephen A. Vavasis. Automatic domain partitioning in three dimensions. *SIAM J. Sci. Stat. Comput.*, 12(4):950–970, 1991.

[21] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Method.*, 2(1):77–79, 1981.

A Appendix

Definition 1 *A planar graph $G = (V, E)$ is a simply connected triangulation if it admits a connected subgraph $\mathcal{B} =< B >$ such that the graph*

$$G' = (V \bigcup \{w\}, E \bigcup \{xy\} \bigcup \{uw, u \in B\})$$

, where w is an additional vertex, is planar and any addition of an edge in G' makes G' non-planar (we ignore double edges).

Definition 2 *Given a simply connected triangulation $\mathcal{T} = (V, E)$, a bijection $\gamma : V \mapsto \mathbb{R}^2$ is a geometrical mapping if it embeds a straight-line planar representation of G.*

This last notion allows us to talk about "triangles", "areas" , "distance" $d(x, y)$ between two vertices x and y, and so on. We recall that deepness analysis does not need such a representation of G.

Definition 3 *A simply connected triangulation $\mathcal{T}$ associated to a geometrical mapping γ is said to be (σ, τ)-uniformly regular if for any triangle $T = xyz$ of $\mathcal{T}$, we have*

$$\tau h \leq h_T \leq h \tag{1}$$

$$\frac{h_T}{\rho_T} \leq \sigma \tag{2}$$

where

$$\left| \begin{array}{l} h = max\{d(u, v), v \in E\} \\ h_T = max\{|d(x, y), d(x, z), d(y, z)|\} \\ \rho_T = maximum \ radius \ of \ a \ circle \ included \ in \ T \end{array} \right.$$

Definition 4 *The depth of a vertex v in a simply connected triangulation $\mathcal{T} = (V, E)$ having a boundary ¡B¿, is the minimum length p of a series $\{v_0, \ldots, v_p\}$ such that, for any i in $\{1, \ldots, p\}$, $v_{i-1}v_i \in E$ and $v_o = v$ and $v_p \in B$. Then the depth of $\mathcal{T}$, noted* $\mathbf{depth}(\mathcal{T})$ *is the maximum depth of all vertices in V.*

Theorem 1 *[11] Given a triangulation $\mathcal{T}$, with weights associated to $\mathcal{T}$'s triangles, it is possible to find a $1/3 - 2/3$ separator in linear time having less than $2.\mathbf{depth}(\mathcal{T}) + 2$ vertices*

Theorem 2 *If $\mathcal{T}$ is a $(\sigma - \tau)$-uniformly regular triangulation, having n triangles drawn on a domain Ω (of area $|\Omega|$), then*

$$\mathbf{depth}(T) \leq \frac{(R(\sigma) + 1)^2 \sigma^2}{\tau^2} + \frac{diam(\Omega)}{\varepsilon(\sigma, \tau)} \sqrt{\frac{\pi}{|\Omega|}} \sqrt{n}$$

where R, ε and $diam$ are functions that may be expressed explicitly, respectively on (σ), (σ, τ) and (Ω).

Proof

- Link between h and n. We bound the area A_T of a triangle T in $\mathcal{T}$ by 1 and 2:

$$\frac{\pi}{h^2} \leq A_T \leq \pi \left(\frac{\tau h}{\sigma}\right)^2 \tag{3}$$

Then, since $\mathcal{T}$ has n triangles and

$$\sum_{T \in \mathcal{T}} = |\Omega|$$

we have :

$$\frac{|\Omega|}{\pi h^2} \leq n \leq \frac{|\Omega|\sigma^2}{\pi \tau^2 h^2} \tag{4}$$

- Link between h and the triangulation's depth.

Let S be a node in B.

Property 1 *There exists $R(\sigma)$ and $\varepsilon(\sigma,\tau)$ ¿0, such that, for any vertex M in $V - B$ verifying $d(M,S) > R(\sigma)h$, one can find N adjacent to M (i.e. $MN \in E$) such that*

$$d(M,S) - D(N,S) > \varepsilon(\sigma,\tau)h$$

Proof Since any triangle T verifies $\frac{h_T}{\rho_T} \leq \sigma$, there exists $\theta(\sigma)$ chosen in $(\frac{\pi}{3};\pi)$ that bounds the maximum angle of all the triangles. Now let us draw the line D from M to S. Clearly, since $M \notin B$, there exists a vertex N such that

$$|(\vec{MN}, \vec{MS})| \leq \frac{\theta(\sigma)}{2}$$

And we have, by 1 and 2,

$$\frac{\tau h}{\sigma} \leq d(M,N) \leq h$$

Then, we take $\beta(\sigma)$ in $(0;\frac{\pi}{2})$ such that $\beta(\sigma) + \theta(\sigma) < \pi$ and $R(\sigma)$ such that $R(\sigma) > \frac{1}{tan(\beta(\sigma))} + 1$. Draw M' and N' perpendicular projections of M and N on (SM), (SN) respectively. We have obviously $d(S,M') \geq d(S,M) - d(M,N') \geq (R(\sigma)-1).h$ and $d(N,M') \leq h$. Then $|(\vec{SN}, \vec{SM'})| \leq arctan\frac{1}{R(\sigma)-1} \leq \beta(\sigma)$

So we have

$$\begin{aligned} d(S,M) - d(S,N) &\geq d(S,M') - d(S,M) = d(N.M') \\ &\geq d(M,N)\cos(\tfrac{\theta(\sigma)+\beta(\sigma)}{2}) \\ &\geq \tfrac{\tau}{\sigma}\cos(\tfrac{\theta(\sigma)+\beta(\sigma)}{2})h = \varepsilon(\sigma,\tau)h \end{aligned}$$

□

Now, take M_0 such that $\mathbf{depth}(M_0)$ is the depth of $\mathcal{T}$. We build recursively $M_{k=1}$ from M_k by the previous proposition until $d(M_{k+1}, S) \geq R(\sigma)h$ or $M_{k+1} \in B$.

Then the series $(M_i)_{0 \leq i \leq p}$ is bounded by

$$p \leq \frac{d(M_0, S)}{\varepsilon(\sigma, \tau)h} \leq \frac{diam\Omega}{\varepsilon(\sigma, \tau)h}$$

If M_p is on the boundary, we already have a path from M_0 to the boundary. Otherwise $d(M_p, S) < R(\sigma)h$. But since the area of one triangle is bounded by $\pi \left(\frac{\tau h}{\sigma}\right)^2$, there is at most K triangles in the disk of center S and radius $(R(\sigma)+1)h$, with K verifying :

$$K \leq \pi \frac{(R(\sigma)+1)^2 h^2}{\pi \left(\frac{\tau h}{\sigma}\right)^2} = \frac{(R(\sigma)+1)^2}{\tau^2}\sigma^2$$

And then one may reach the boundary from M_p in at most K steps. So we have

$$\mathbf{depth}(T) \leq \frac{(R(\sigma)+1)^2\sigma^2}{\tau^2} + \frac{diam(\Omega)}{\varepsilon(\sigma, \tau)}\frac{1}{h}$$

and

$$\mathbf{depth}(T) \leq \frac{(R(\sigma)+1)^2\sigma^2}{\tau^2} + \frac{diam(\Omega)}{\varepsilon(\sigma, \tau)}\sqrt{\frac{\pi}{|\Omega|}}\sqrt{n}$$

□