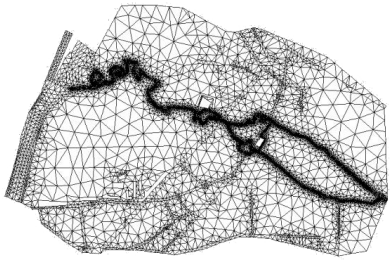
Université d'Évry
Département Informatique
91 000 Évry

CEA/DIF
Campus Teratec
91 680 Bruyères-Le-Chatel

Mémoire de stage de master

Heuristique parallèle et distribuée pour la génération automatique de maillages triangulaires de Delaunay



Rivière « l'Automne », Mairie de Verberie

Résumé

Notre étude porte sur la construction d'une heuristique parallèle pour la génération automatique de maillages triangulaires de Delaunay en contexte HPC. L'objectif est double : explorer les approches parallèles suffisamment génériques pour la construction automatique de ces maillages, et fournir une solution expérimentale pouvant tirer parti de centaines de processus légers (de 100 à 200 threads environ).

Après une étude préalable de l'état de l'art, une implémentation séquentielle incrémentale est proposée. Ensuite, une heuristique parallèle est construite, basée sur un partage de tâches entre les n PE en charge de générer le maillage. Enfin, une heuristique distribuée basée sur un partitionnement géométrique du domaine est établie. Les résultats obtenus sont ensuite analysés, en terme d'accélération, d'efficacité et de répartition de charges entre PE.

Étudiant

Hoby RAKOTOARIVELO
M2 Modèles, Optimisation, Programmation et Services
2013 – 2014

Encadrants

M. Franck LEDOUX
M. Nicolas LEGOFF

Table des matières

Remerciements

Je remercie mon tuteur, M. Franck Ledoux, de m'avoir encadré durant toute la durée du stage dans une ambiance de travail agréable. Je le remercie également pour ses conseils instructifs tant sur le plan technique que professionnel.

Je remercie également mon co-tuteur M. Nicolas Le-Goff pour m'avoir épaulé et guidé tout au long du stage. Même occupé, il a toujours été disponible pour m'aider (debugging, déploiement), ou pour me débloquer quand j'étais dans une impasse sur des points précis.

Je tiens également à remercier M. Jean-Christophe Weill pour m'avoir accueilli avec bienveillance au sein de l'équipe, M. Cédric Chevalier pour son aide, M. Denis Lubin et M. Thao Le-Chi pour leur assistance et les divers visites auxquelles on a eu droit durant le stage.

Mes remerciements vont également à mes collègues stagiaires pour la bonne ambiance générale au sein de l'open-space. Je remercie en particulier Alexis, Corentin, Rémi, Jonathan, Gautier, Julien, Fabien et Guillaume d'avoir contribué à une ambiance conviviale et chaleureuse (blagues, foot-bouchon, grand huit etc.). Je remercie Agnès et Alexis pour les discussions/blagues dans le car, rendant les trajets quotidiens plus agréables.

Introduction

La simulation numérique de phénomènes physiques nécessite la résolution approchée des équations aux dérivées partielles qui modélisent ces phénomènes. Pour ce faire, les méthodes numériques (différences finies, éléments finis, et volumes finis) utilisent une discrétisation du domaine : le maillage. En effet, les équations initiales sont approchées par un système d'équations s'appuyant sur les éléments du maillage. Ce maillage doit approcher au maximum la géométrie du domaine, et sa qualité influe directement sur la convergence des schémas et la précision des solutions obtenues.

Maillages de Delaunay

La construction séquentielle de maillages a été largement étudiée, et il existe une multitude d'algorithmes, qui peuvent se différencier selon la topologie du domaine géré (2D/3D, géométrie régulière/-quelconque), selon la méthode de génération (automatique, semi-automatique ou manuelle), ou selon le type d'éléments gérés (triangles, quadrangles, tétraèdres, hexaèdres etc.) .
Dans le cas des maillages triangulaires, il existe actuellement 3 approches de construction : à base de décomposition spatiale [1, 2], par avancée de front [3] et la méthode dite « de Delaunay » [4, 5, 6, 7, 8, 9, 10]. Cette dernière est particulièrement intéressante car elle garantit intrinsèquement d'obtenir des mailles « de qualité », en maximisant l'angle minimal de chaque triangle, ce qui tend à préserver leur équilatéralité [11].

Génération parallèle

La génération de maillages peut induire des temps de calcul très longs, en fonction de la taille du domaine initial, et le degré de raffinement cible des éléments. A ce titre, le recours au calcul parallèle s'avère être une solution, car il permet de traiter des maillages de plus grande taille, et de répartir la charge de calcul et/ou la consommation mémoire sur plusieurs machines ou unités de calcul.
Il y a plusieurs approches pour la génération parallèle de maillages [12, 13, 14, 15, 16], néanmoins celles-ci sont très spécialisées pour la plupart, et il en existe peu qui soient suffisamment génériques pour garantir des performances constantes d'une configuration (de maillage et/ou de machine) à une autre.

Objectifs

Dans ce contexte, notre étude porte sur la construction d'une heuristique parallèle pour la génération automatique de maillages triangulaires de Delaunay en contexte HPC [1].
L'objectif est double : explorer les approches parallèles suffisamment génériques pour la construction automatique de ces maillages, et fournir une solution expérimentale pouvant tirer parti de centaines de processus légers (de 100 à 200 threads environ).

1. High Performance Computing – Calcul Haute Performance

Solution mise en œuvre

Après avoir introduit les notions nécessaires à la lecture de ce document au chapitre 1, nous effectuons une étude préalable de l'état de l'art au chapitre 2.

Une première solution séquentielle incrémentale est ensuite proposée au chapitre 3. Celle-ci comporte 3 phases : la triangulation, la génération du maillage et l'optimisation.
Partant d'une liste de points $\mathcal{S}$ issue de la discrétisation du domaine Ω, la triangulation $\mathcal{T}$ est obtenue par insertion successive de chaque point $p \in \mathcal{S}$ dans $\mathcal{T}$, sur la base de l'algorithme de Lawson [5], et en utilisant une structure arborescente pour la localisation des faces dans $\mathcal{T}$ [17]. Ensuite, le maillage $\mathcal{M}$ est obtenu en raffinant chaque face $\Delta \in \mathcal{T}$ selon une métrique définie : une taille cible d'éléments $\bar{\rho}$ et un aplatissement maximal $\bar{\theta}$. Enfin, on optimise la qualité du maillage $\mathcal{M}$ obtenu grâce à un lissage laplacien. Cela consiste à repositionner chaque nœud de $\mathcal{M}$, de manière à ce qu'il soit le barycentre de ses voisins.

Au chapitre 4, deux solutions parallèles sont proposées : La première s'appuie sur la compétition et la coopération des PE pour la génération du maillage ; la seconde est basée sur un partitionnement géométrique du domaine. De manière plus détaillée, la première solution, ou heuristique, H_1 est basée sur une parallélisation des 3 phases de l'algorithme précédent. Ici, on cherche à décomposer l'algorithme en un ensemble de taches élémentaires, qui vont être réparties entre n PE en charge de générer le maillage. Pour la phase de triangulation, on s'inspire de [13] et l'idée sera de partitionner et de répartir les points du domaine Ω entre les n PE. Pour la phase de génération du maillage, l'idée est de constituer une file de taches $\mathcal{Q}$ à effectuer (raffinements de faces) et les PE devront récupérer et traiter chaque tache, et ce tant que cette file n'est pas vide. Pour la phase de lissage, l'idée est de répartir la liste de points du maillage courant entre les PE.

La seconde heuristique H_2 est basée sur une décomposition géométrique du domaine Ω, comme dans [18]. A partir d'une triangulation $\mathcal{T}$ ou d'un maillage initial $\mathcal{M}$, l'idée sera de créer n_w partitions de tailles géométriques sensiblement égales, de manière à ce que chaque partition $\mathcal{P}_i$ soit traitée individuellement par un PE (n_w en tout).

Enfin, nous présentons au chapitre 5 des résultats obtenus avec les heuristiques H_1 et H_2. Ceux-ci seront analysés en terme d'accélération, d'efficacité et de répartition de charges entre PE.

Chapitre 1

Notions préliminaires

Ce chapitre présente l'ensemble des notions préliminaires requises pour la lecture de ce document.
Dans un premier temps, on abordera les maillages dans la section 2.1, en y présentant les classes de maillages, leurs représentations, les structures de données sous-jacentes, ainsi que les familles d'algorithmes de maillages triangulaires.
Dans un second temps, on abordera le parallélisme dans la section 2.2, en y présentant les architectures parallèles, les modèles de programmation, les stratégies de parallélisation et les méthodes de décomposition de calcul.

1.1 Maillages

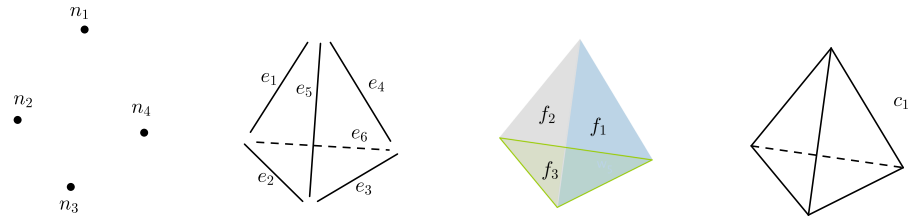
En simulation numérique, un maillage fournit une représentation discrète d'un domaine géométrique. Ce caractère discret permet de visualiser ou manipuler informatiquement des phénomènes ou systèmes physiques continus, modélisés par des équations aux dérivées partielles.

Définition 1 (MAILLAGE). Un maillage $\mathcal{M}$ est une discrétisation d'un domaine Ω continu et borné de $\mathbb{R}^n, 0 \leq n \leq 3$ par des éléments géométriques finis $K_i \in \mathbb{R}^k, k \leq n$, tels que :

$$\Omega = \bigcup_{K \in \mathcal{M}} K \tag{1.1}$$

$$\forall K_i, K_j \in \mathcal{M}, \text{ on a } K_i \cap K_j = \begin{cases} \emptyset \\ \text{un point } K_0 \in \mathcal{M} \\ \text{une arête } K_1 : (u, v) \in \mathcal{M} \\ \text{une face } K_2 : (i, j, k) \in \mathcal{M} \end{cases} \tag{1.2}$$

D'un point de vue topologique, un maillage $\mathcal{M}$ est une collection d'éléments K_i de dimensions différentes et interconnectés (cf. figure 1.1). Une cellule de dimension 0 (ou 0-cellule) est appelée noeud ou sommet.
Une 1-cellule est une arête.
Une 2-cellule est une face.
Une 3-cellule est usuellement appelée maille, ou simplement cellule [1].



(a) Noeuds (b) Segments (c) Faces (d) Cellule

FIGURE 1.1 – Elements constitutifs d'un maillage

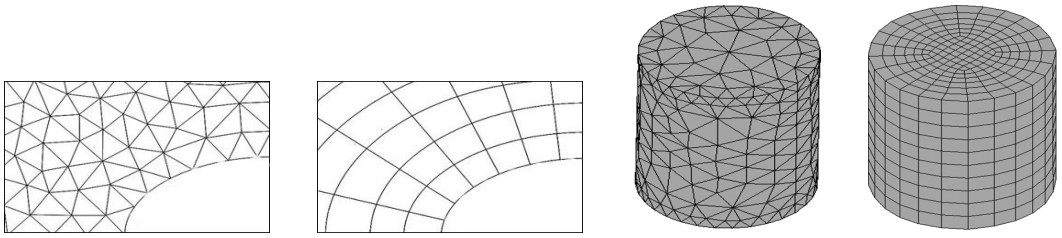
1. Pour un maillage en dimension 2, une face est également appelée cellule

1.1.1 Classification

Selon la topologie et la connectivité de ses éléments, un maillage peut être structuré ou non-structuré.

Définition 2 (MAILLAGE STRUCTURÉ). Un maillage $\mathcal{M}$ est dit structuré si sa connectivité est régulière. Les éléments d'un tel maillage sont des quadrangles en 2D, et des hexaèdres en 3D (voir figure 1.2).
Il permet de retrouver implicitement le voisinage de chaque cellule, et ne nécessite aucun stockage de données topologiques. Toutefois, il est limité par la régularité géométrique du domaine, et ne permet de mailler tout type de surface/volume.

Définition 3 (MAILLAGE NON-STRUCTURÉ). Un maillage $\mathcal{M}$ est dit non-structuré si sa connectivité est quelconque. Il est généralement constitué de simplexes (triangles/tétraèdres), mais peut contenir des éléments géométriques de natures différentes (voir figure 1.2).
Il permet de discrétiser des domaines quelconques. Néanmoins, il est plus difficile de contrôler localement la densité des mailles, et nécessite un stockage local des données topologiques.



(a) triangulaire (b) quadrangulaire (c) tétraédrique (d) hexaédrique

FIGURE 1.2 – Classes usuelles de maillages surfaciques et volumiques

1.1.2 Modèles de représentation

D'un point de vue informatique, il y a plusieurs manières de représenter un maillage et la connectivité de ses éléments. Le choix des structures de données découle de la représentation choisie, et impacte directement sur la complexité de l'algorithme. En effet, il faudra choisir entre une structure plus riche permettant un accès direct aux données topologiques (localisation, voisinage) mais impliquant un stockage important et une mise à jour couteuse, et une structure plus simple nécessitant peu d'espace mémoire, mais fournissant peu d'informations également.

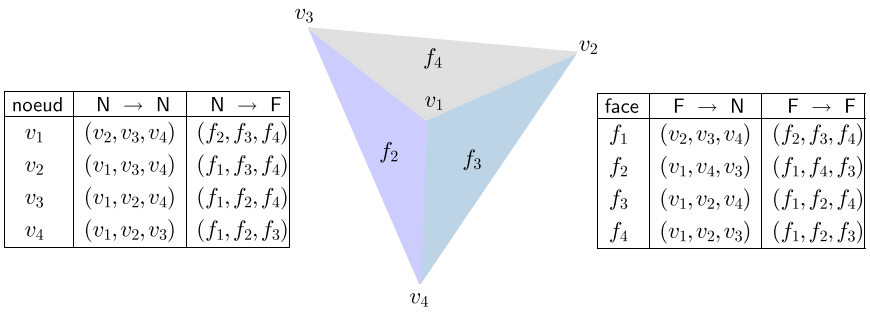
On distingue usuellement 2 familles de représentations : à base de cellules, et à base d'arêtes.

1.1.2.1 REPRÉSENTATION À BASE DE CELLULES

Dans cette représentation, un maillage de dimension n, avec $n = 2$ ou 3 est vu comme une collection de cellules de dimension $k \leq n$ connectées. Ici, les cellules sont représentées explicitement et directement stockées en mémoire. Selon la connectivité de ses éléments, on distingue 3 modèles de représentation :

- ✏ Nœud $\rightarrow$ Nœud : ici, chaque sommet connait l'ensemble des sommets voisins $\mathcal{N}(s)$. Centrée sur les nœuds, cette représentation permet également de retrouver le voisinage d'une face Δ, en identifiant l'arête $(i, j) \in \Delta$ opposée pour chaque $k \in \mathcal{N}(s)$ et en reconstituant la face $\Delta' : (i, j, k)$.
- ✏ Nœud $\rightarrow$ Face : ici, chaque nœud connait les faces qui l'utilisent.
  Le voisinage d'une face Δ est obtenu en récupérant les faces $\mathcal{N}_F(i)$ incidentes à chaque nœud $i \in \Delta$ et en les intersectant.
- ✏ Face $\rightarrow$ Face : ici, chaque face connait directement les faces qui lui sont voisines. Il s'agit de la représentation qu'on utilisera au sein de notre heuristique.

Toutefois, ces 3 modèles nécessitent une mise à jour constante des données locales, lors d'une modification d'une cellule ou de ses voisins – voir figure 1.3.



noeud	N → N	N → F
v_1	(v_2, v_3, v_4)	(f_2, f_3, f_4)
v_2	(v_1, v_3, v_4)	(f_1, f_3, f_4)
v_3	(v_1, v_2, v_4)	(f_1, f_2, f_4)
v_4	(v_1, v_2, v_3)	(f_1, f_2, f_3)

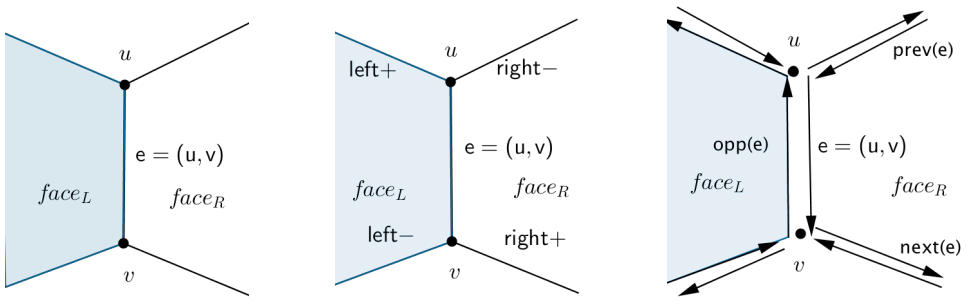
face	F → N	F → F
f_1	(v_2, v_3, v_4)	(f_2, f_3, f_4)
f_2	(v_1, v_4, v_3)	(f_1, f_4, f_3)
f_3	(v_1, v_2, v_4)	(f_1, f_2, f_4)
f_4	(v_1, v_2, v_3)	(f_1, f_2, f_3)

FIGURE 1.3 – Modèles de représentation à base de cellules

1.1.2.2 REPRÉSENTATION À BASE D'ARETES

Dans cette représentation, le maillage est vu comme une collection d'arêtes (ou de « morceaux » d'arêtes) connectées. Ici, les cellules sont représentées implicitement Selon les références stockées au sein des arêtes, on peut distinguer 3 modèles de représentation :

- ✏ Quad-edge : chaque arête contient une référence vers les noeuds source/destination, ainsi que les faces voisines – voir figure 1.4(a).
- ✏ Winged-edge : chaque arête contient une référence vers les noeuds sources/destination, les faces voisines, ainsi que les winged-edge voisins – voir figure 1.4(b).
- ✏ Half-edge : similaire aux winged-edges, mais ici une arête est décomposée en demi-arête (une par face incidente). Ainsi, seule la moitié des informations est stockée, l'autre étant déduite de son complémentaire – voir figure 1.4(c).



(a) Quad-edge (b) Winged-edge (c) Half-edge

FIGURE 1.4 – Modèles de représentation à base d'arêtes

1.1.3 Maillages triangulaires

Dans le cadre de notre étude, nous allons nous restreindre aux maillages triangulaires.
S'agissant du cas de base de maillage simplicial, il permet de définir/déduire les outils et propriétés utiles à l'extension en 3D (tétraèdres).

Définition 4 (TRIANGULATION 2D). Soit $\mathcal{S}$ un ensemble de points de $\mathbb{R}^2$, et $\Omega \subset \mathbb{R}^2$ le domaine défini par l'enveloppe convexe de $\mathcal{S}$. Une triangulation $\mathcal{T}_S$ est un ensemble de simplexes [2] de dimension 2, qui

2. triangles

forme un recouvrement de Ω vérifiant :

$$\mathcal{S} = \bigcup_{p \in \Delta, \Delta \in \mathcal{T}_S} p, \text{ et } \Omega = \bigcup_{\Delta_i \in \mathcal{T}_S} \Delta_i$$

$$\forall \Delta_i, \Delta_j \in \mathcal{T}_S \text{ on a } \Delta_i \cap \Delta_j = \begin{cases} \emptyset \\ \text{un point } p \in \mathcal{T}_S, \text{ ou une arête } (u, v) \in \mathcal{T}_S \end{cases}$$

Différence entre maillage et triangulation

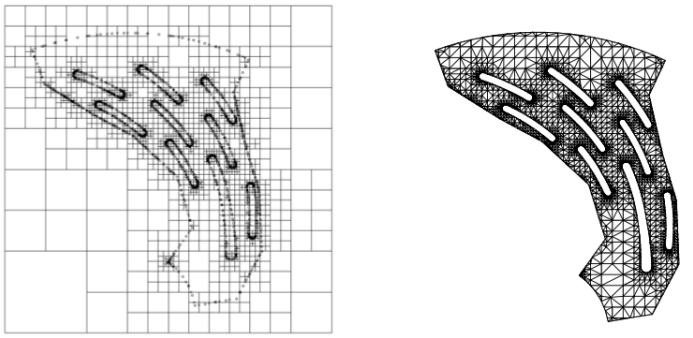
Un maillage $\mathcal{M}$ diffère d'une triangulation $\mathcal{T}_S$ en 3 points :

- ✏ initialisation : tous les points sont connus d'avance dans $\mathcal{T}$, alors que seuls les points issus de la discrétisation de la frontière de Ω sont connus dans $\mathcal{M}$.
- ✏ type d'élément : $\mathcal{T}$ est constitué uniquement de simplexes (triangle, tétraèdre), alors que $\mathcal{M}$ peut être constitué de n'importe quel élément géométrique.
- ✏ respect de la frontière : $\mathcal{M}$ doit être conforme à la frontière du domaine, ce qui n'est pas le cas de $\mathcal{T}$.

Il existe de nombreux algorithmes de maillages triangulaires. Selon l'approche adoptée et les structures de données sous-jacentes, ils peuvent être regroupés en 3 catégories : les méthodes de décomposition spatiale [1, 2], les méthodes frontales [3], et celles de Delaunay [4, 5, 6, 7, 8, 9, 10].

1.1.3.1 MÉTHODE DE DÉCOMPOSITION SPATIALE

Elle se base sur une décomposition du domaine en utilisant une structure arborescente pour ranger les cellules du maillage et y accéder rapidement. Elle stocke aussi les données topologiques (voisinage) et métriques (distribution de tailles, densité etc.). Cette structure permet de créer les cellules de manière à ce que leurs tailles soient liées à celles des cases qui les contiennent [1, 2]. Sur la figure 1.5, une décomposition par quad-tree est proposée en (a). Les points internes seront utilisés pour le maillage proposé en (b).



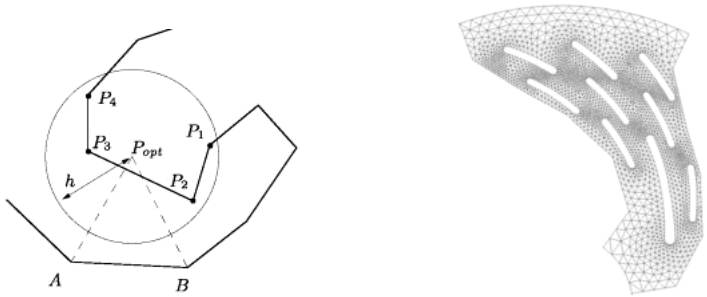
(a) Décomposition (b) Maillage obtenu

FIGURE 1.5 – Maillage par décomposition spatiale

1.1.3.2 MÉTHODE FRONTALE

Elle se base sur une construction itérative du maillage en partant de la frontière du domaine. Cette méthode consiste à créer, insérer et connecter les points à ceux du front pour créer les nouveaux éléments [3].
Le front courant est défini par la frontière séparant la partie déjà maillée de celle qui reste à traiter. Ce front est continuellement mis à jour à chaque maille créée, et la procédure se termine dès lors qu'il

devient vide. A chaque itération, une cellule est choisie, et un nouveau sommet est calculé. Si ce dernier permet de créer une maille satisfaisant le critère de qualité, alors il est inséré – voir figure 1.6(a). Cette méthode garantit d'obtenir des mailles de bonne qualité (cf. figure 1.6). Toutefois, les procédures de sélection des éléments du front, celle des points optimaux, et la validation de ces éléments à partir des points doivent être robustes pour assurer une bonne convergence de l'algorithme [11].



(a) Choix du prochain point (b) Maillage

FIGURE 1.6 – Maillage par avancée de front

1.1.3.3 MÉTHODE DE DELAUNAY

Définition 5 (CRITÈRE DE DELAUNAY & TRIANGULATION DE DELAUNAY). .
Une triangulation $\mathcal{T}_S$ d'un ensemble de points $\mathcal{S}$ vérifie le critère de Delaunay si le cercle circonscrit de n'importe quel $\Delta \in \mathcal{T}_S$ ne contient aucun autre sommet de $\mathcal{S}$.
Une telle triangulation est dite de Delaunay.

Propriétés

Une telle triangulation confère 2 propriétés importantes [11] :

1. elle maximise le plus petit angle de chaque face de $\mathcal{T}_S$, évitant les faces trop applaties (inexploitables par les méthodes numériques).
2. elle peut être calculée en temps polynomial $\mathcal{O}(n \log n)$, n étant le nombre de points de $\mathcal{S}$.

Définition 6 (DIAGRAMME DE VORONOI). Le diagramme de Voronoï d'un ensemble fini de points $S = (p_i)_{i=1,\dots,n} \in \mathbb{R}^d$ est l'ensemble des cellules V_i définies par :

$$V = \bigcup_{p_i \in S} V_i, \text{ avec } V_i = \{p \mid d(p,p_i) \leq d(p,p_j), \forall j \neq i \wedge p_i, p_j \in \mathcal{S}\} \text{ où } d \text{ est la distance euclidienne}$$

Une cellule V_i contient donc l'ensemble des points géométriquement plus proches de p_i que de n'importe quel autre point de S. Il s'agit du problème dual de la triangulation de Delaunay [11].

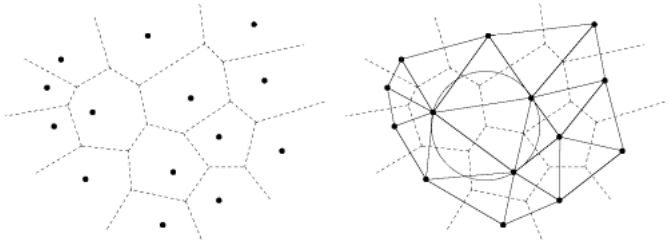
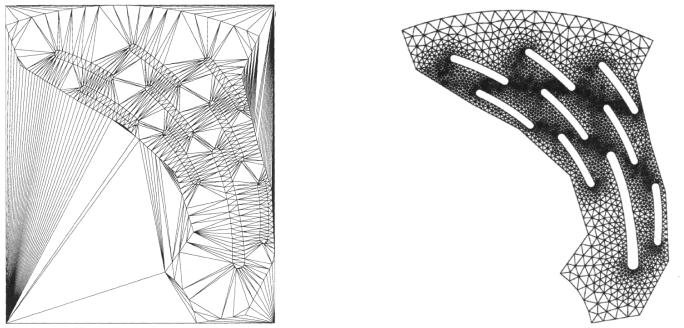


FIGURE 1.7 – Diagramme de Voronoi et triangulation de Delaunay correspondante

CONSTRUCTION DU MAILLAGE La méthode directe de Delaunay consiste à construire le maillage à partir d'une triangulation de Delaunay de la frontière du domaine, et en la raffinant par insertion de points internes, tout en préservant le critère du cercle circonscrit vide pour chaque face [10, 5, 8, 9].
Un exemple de maillage de Delaunay est donnée sur la figure 1.8.



(a) Triangulation (b) Maillage obtenu

FIGURE 1.8 – Maillage par la méthode de Delaunay

Tirant profit des propriétés induites par ce critère, elle permet d'obtenir des mailles de bonne qualité en préservant l'équilatéralité lors du raffinement, avec une implémentation moins complexe comparée aux 2 précédentes méthodes.

1.2 Parallélisme

Le recours au calcul parallèle vise à accélérer le temps de traitement en permettant de tourner des algorithmes plus couteux, à obtenir de meilleurs résultats sur une même durée, et à résoudre des instances de très grande taille.
Cette section présente les notions minimales requises pour la mise en oeuvre d'un algorithme parallèle.

1.2.1 Architectures parallèles

Afin de comprendre les modèles de programmation parallèle, on fait un bref rappel des architectures existantes. En effet, ces premiers exploitent généralement les spécificités de l'architecture, et le couplage hardware/software dépendra du niveau de granularité choisi.
La classification de Flynn permet de caractériser les architectures en fonction des flots de données et de contrôle [19] – cf. table 1.1.

- ✏ SISD correspond aux machines séquentiels : une seule instruction est exécutée sur une seule donnée, à tout instant.
- ✏ MISD (très peu courant) permet l'exécution de plusieurs instructions sur une même donnée, et est utilisé dans certains équipements critiques, ou pour la tolérance aux pannes.
- ✏ SIMD permet l'exécution d'une même instruction sur plusieurs données simultanément, et est implémenté dans la plupart des processeurs actuels (VMX, SSE etc.)
- ✏ MIMD correspond aux machines multiprocesseurs/clusters. Chaque cœur/processeur exécute des instructions (potentiellement différentes) sur des données différentes.

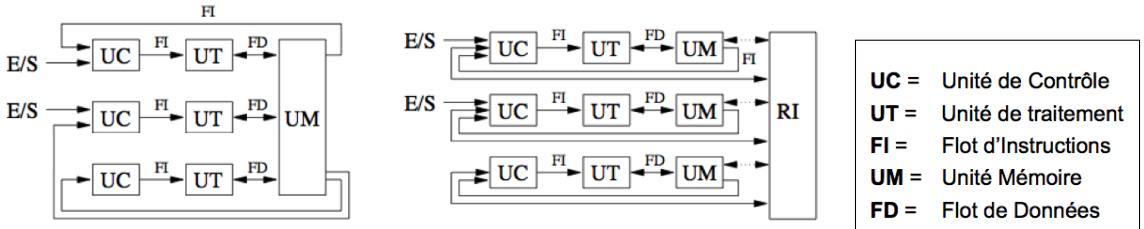
	Single instr.	Multi. instr.
Single data	SISD	MISD
Multi. data	SIMD	MIMD

Table 1.1 – Classification de Flynn des architectures parallèles

On peut classer les MIMD selon :

- ✏ le type de mémoire et l'espace d'adressage qui peuvent être partagé ou distribué.
- ✏ le mode d'accès mémoire : NORMA (pas d'accès données distantes), UMA (uniforme), NUMA (non uniforme), CC-NUMA (non uniforme avec caches), OSMA (accès données distantes gérées par l'OS), COMA (mémoire locale se comportant comme un cache).

En mémoire partagée, l'espace mémoire est commun à tous les processeurs/cœurs. Il permet d'avoir un cout d'accès réduit aux données, mais demande parfois d'avoir recours à des section critiques car les processus (ou PE[3]) ne doivent pas accéder à une même zone mémoire pendant que l'un d'entre eux la modifie. En distribué, chaque PE dispose de ses propres données et instructions mais nécessite un mécanisme explicite de communication pour le partage de données entre PE - cf. figure 1.9(b).



(a) MIMD mémoire partagée (b) MIMD mémoire distribuée

Figure 1.9 – Architecture d'un MIMD classique [20]

1.2.2 Modèles de programmation

Un modèle de programmation parallèle est une abstraction permettant de décrire les mécanismes de contrôle, de gestion des données et de synchronisation entre processus d'un programme parallèle, en fonction de l'architecture de déploiement.
En d'autres termes, il désigne la manière de concevoir le code parallèle en adaptant le programme séquentiel, en vue de l'exploiter sur un type de machine parallèle.

1.2.2.1 concepts de base

Il y a plusieurs manières d'extraire le parallélisme au sein d'un programme séquentiel, en fonction des instructions et des données d'entrées. On distingue usuellement 3 approches : le parallélisme de données, le parallélisme de tâches et le parallélisme de flux.

parallélisme de données Il consiste à distribuer les données au sein de k unités de calcul, et ensuite d'y appliquer un même traitement.

parallélisme de taches Il consiste à décomposer le programme en taches, et à les exécuter[4] au sein de k unités de calcul. Il permet d'appliquer des traitements différents sur les mêmes données, et nécessite des échanges de données et des directives de contrôle pour la synchronisation des tâches. Le

3. Processing Element : il peut s'agir d'un processus lourd ou léger (threads)
4. En pratique, les taches sont implémentées par des threads, ou des processus selon le type de mémoire de la machine

découpage en tâches s'effectue en fonction de 2 critères : la granularité (finesse de découpage), et les dépendances entre les tâches.

PARALLÉLISME DE FLUX Il s'agit d'un parallélisme de taches avec une dépendance particulière entre les taches, de telle sorte que la sortie de l'une soit l'entrée de l'autre. Lié à l'enchainement des tâches, il permet de traiter efficacement les flots de données, par la construction d'un pipeline.

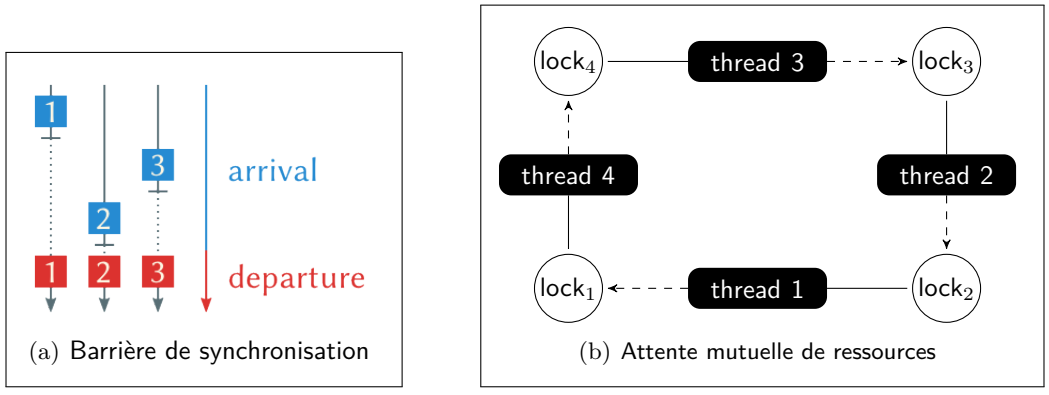
Bien que les concepts soient les mêmes, les modèles de programmation se différencient toutefois en fonction de l'architecture utilisée. A cet effet, on distingue usuellement 3 modèles : la vectorisation, le multithreading, le modèle à passage de messages.

1.2.2.2 MULTITHREADING

Il s'agit du modèle adapté aux machines à mémoire partagée. Ici, le programme consiste en un ensemble de processus légers (ou threads), qui communiquent via la lecture et l'écriture de variables partagées. Chaque thread dispose de données privées (pile locale), et peut accéder aux variables partagées (pile globale). Leur gestion peut être explicite (instanciation par l'utilisateur, et affectation explicite des instructions à exécuter [5]) ou implicite (directives de compilation et création implicite à l'exécution [6]).

EXCLUSION MUTUELLE Du fait de l'asynchronisme d'exécution (géré par le système d'exploitation), ce mode de communication induit une situation de compétition (race condition) lorsque plusieurs threads tentent d'accéder et/ou modifier simultanément une variable partagée. Cela conduit à un indéterminisme du résultat, qui dépendra de l'ordre d'exécution des threads.
Ce problème peut être géré par un contrôle d'accès par exclusion mutuelle : un seul thread à la fois peut exécuter la section critique, et ainsi accéder à la variable partagée, comme illustré dans la table 1.2.

BARRIÈRE Il s'agit d'un contrôle de séquence qui consiste à resynchroniser l'exécution des threads. Les threads seront en attente tant que le dernier n'aura atteint la barrière, ensuite l'exécution continue comme décrit dans la figure 1.10(a).



(a) Barrière de synchronisation (b) Attente mutuelle de ressources

FIGURE 1.10 – Barrière de synchronisation et interblocage

INTERBLOCAGE Le controle d'accès aux variables partagées peut induire une situation d'interblocage : 2 threads ayant acquis chacun un verrou attendent mutuellement que l'autre relache son verrou pour pouvoir continuer leur exécution – voir figure 1.10(b). Lorsque cela se produit, les threads concernés sont définitivement bloqués.

5. comme les Posix Threads par exemple – https://computing.llnl.gov/tutorials/pthreads/
6. comme OpenMP par exemple – https://computing.llnl.gov/tutorials/openMP/

thread 1

```
s1 = 0;
for(i=0, i < n/2, i++)
  s1 = s1 + f(A[i]);
lock(m);
s = s + s1;
unlock(m);
```

thread 2

```
s2 = 0;
for(i=n/2, i < n, i++)
  s2 = s2 + f(A[i]);
lock(m);
s = s + s2;
unlock(m);
```

TABLE 1.2 – Contrôle d'accès d'une variable globale par exclusion mutuelle

3 stratégies contre l'interblocage

1. la PRÉVENTION : elle se base sur des contraintes sur la manière de demander l'accès aux ressources critiques : soit en réservant toutes les ressources avant de commencer, soit en relâchant toutes les ressources avant d'accéder à de nouvelles, soit en ordonnançant l'accès aux ressources.
2. l'ÉVITEMENT : consiste à contrôler pas à pas la progression des processus, de telle sorte à toujours prévoir la possiblité d'échapper à un interblocage. Pour cela, il faudrait stocker localement les données sur les ressources et les threads dans un graphe, qui détermine la séquence d'actions permettant une terminaison correcte des threads. Néanmoins cette méthode dépend fortement de l'algorithme.
3. la DÉTECTION : si un état s du système est bloqué, alors tous les états atteignables à partir de s le sont également. Il est donc possible de détecter un interblocage grâce à un algorithme de détection qui ne doit pas influencer l'exécution de l'algorihtme (non-interference), mais devra détecter l'interblocage en temps fini s'il y en a.

1.2.3 Indicateurs de performances

Les performances d'un algorithme parallèle sont mesurées à l'aide de 2 indicateurs : l'accélération et l'efficacité.

Définition 7 (ACCÉLÉRATION ou SPEEDUP). Elle mesure le gain de temps d'exécution d'un programme parallèle par rapport à sa version séquentielle. Si T_u est le temps d'exécution sur 1 processeur, et $T(n)$ le temps d'exécution sur n processeurs, alors le speedup est donnée par :

$$S(n) = \frac{T_u}{T(n)}$$

Définition 8 (EFFICACITÉ). Elle mesure l'accélération par unité de calcul, et est donné par :

$$E(n) = \frac{S(n)}{n}$$

Définition 9 (SCALABILITÉ). Elle mesure la capacité de l'algorithme à maintenir l'accélération et l'efficacité en cas de montée de charge, soit en terme de données, soit en unités de calcul

Définition 10 (SCALABILITÉ FORTE). Elle mesure l'évolution de $S(n)$ et $E(n)$ en fonction du nombre d'unités de calcul, pour une taille des données fixée.

Définition 11 (SCALABILITÉ FAIBLE). Elle mesure l'évolution de $S(n)$ et $E(n)$ en fonction du nombre d'unités de calcul et de la taille des données.

La loi d'Amdhal [21] est utilisée pour calculer le gain maximal théorique que l'on peut attendre de la modification d'un système. Elle rappelle qu'en particulier dans un programme parallèle, l'augmentation du nombre d'unités de calcul n'aura un impact que sur la partie parallèle de ce programme, la partie séquentielle n'étant pas affectée.
Si T_s désigne la durée de la partie séquentielle, T_p celle de la partie parallèle, alors on a :

- ✏ durée totale $T = T_s + T_p$
- ✏ temps d'exécution sur n unités de calcul : $T(n) = T_s + \frac{T_p}{n}$.

Loi d'Amdhal

$$S(n) = \frac{T}{T(n)} = \frac{T_s + T_p}{T(n)} = \frac{T_s + T_p}{T_s + \frac{T_p}{n}} \leq \frac{T}{T_s}$$

.

La loi de Gustafson [22] permet de calculer le gain maximal théorique que l'on peut attendre d'un programme parallèle, en supposant que le volume de données à traiter soit extensible : elle traduit simplement l'intuition que plus d'unités de calcul permettent de traiter plus de données à temps d'exécution constant (en faisant la supposition importante que la valeur des données n'affectent pas la partie séquentielle du programme).
Si n désigne la taille des données et le nombre d'unités de calcul, alors on a :

- ✏ temps d'exécution total sur n unités de calcul : $T(n) = T_s + n \cdot (\frac{T_p}{n}) = T_s + T_p$
- ✏ durée totale : $T = T_s + n \cdot T_p$

Loi de Gustafson

$$S(n) = \frac{T}{T(n)} = \frac{T_s + T_p \cdot n}{T(n)} = \frac{T_s + T_p \cdot n}{T_s + T_p} \leq n$$

.

Chapitre 2

Etat de l'art

Dans le cadre de notre étude, on se restreindra aux maillages triangulaires de Delaunay.
Afin de mieux cerner les objectifs et l'intérêt de cette étude, on dressera un aperçu succinct des approches existantes pour la génération de maillages de Delaunay, en séquentiel dans la section 2.1, et en parallèle dans la section 2.2. Les approches usuelles de localisation de cellules sont présentées dans la section 2.3.

2.1 Approches séquentielles

La génération séquentielle de maillages de Delaunay a été largement étudiée, et il existe de nombreux algorithmes dans la littérature.De manière synthétique, on distingue 3 approches :

1. les méthodes à base de diagramme de voronoï centroïdal [6].
2. les méthodes type « divide-and-conquer » [10].
3. les méthodes par construction incrémentale [5, 8, 9].

2.1.1 Méthodes à base de diagramme de Voronoï centroïdal

Il s'agit d'une méthode indirecte consistant à approximer le diagramme de Voronoï, puis à en extraire la triangulation de Delaunay (cf. définition 6, page 8).

À partir d'un ensemble $\mathcal{S}$ de points générés aléatoirement, le partitionnement consiste à regrouper chaque $x \in \mathcal{S}$ de manière itérative, par un algorithme proche des K-means, en tenant compte de la densité cible des triangles $\rho(x)$ [16]. A chaque itération, le centre de gravité (ou centroïde) z_i de chaque cellule c_i s'obtient par :

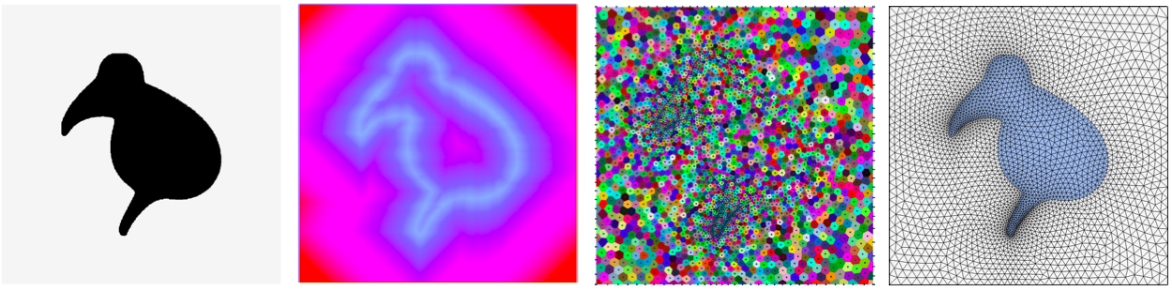
$$z_i = \frac{\int_{c_i} x \cdot \rho(x) dx}{\int_{c_i} \rho(x) dx}$$

L'algorithme vise à minimiser le critère d'énergie suivant :

$$E = \sum_{i=1}^{n} \left(\sum_{p_j \in c_i} \int_{p_j} \rho(x) ||x - z_i||^2 dx \right) \text{ avec } \left[\begin{array}{l} x \text{ point de } \mathcal{S} \\ \rho(x) \text{ la fonction de densité} \\ z_i \text{ le centroïde de la cellule } c_i \end{array} \right.$$

La triangulation est ensuite obtenue en reliant ensuite les centroïdes z_i par insertion d'arêtes comme sur la figure 2.1.

COMPLEXITÉ Elle est de $O(n^2)$ dans sa version incrémentale [6], et $O(n \log n)$ dans sa version « diviser pour régner » [4].
Cette approche permet d'obtenir des maillages adaptatifs de bonne qualité avec une densité non uniforme des triangles [16], mais son implémentation est plus compliquée comparée aux méthodes directes décrites dans les sous-sections 2.1.2 et 2.1.3.



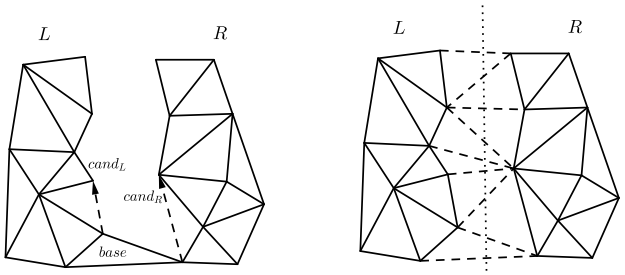
(a) Domaine (b) Carte de densité $\rho(x)$ (c) Cellules de Voronoi c_i (d) Maillage de Delaunay

FIGURE 2.1 – Génération d'un maillage de Delaunay à base de D. de Voronoi [16]

2.1.2 Méthodes D&C

Il s'agit de subdiviser récursivement l'ensemble de points S jusqu'à obtenir des partitions de 2 à 3 sommets. Chaque partition est fusionnée avec celle qui a été formée en même temps qu'elle.

PRINCIPE on commence par trier chaque point de S par ordre croissant de leurs abscisses, et on divise S en k partitions $\pi_i, 2 \leq i \leq k$ contenant chacun 2 ou 3 sommets.
On procède ensuite à leur triangulation, en rajoutant 1 segment si la partition contient 2 points, et 1 triangle si elle en contient 3. On fusionne ensuite chaque paire de partitions $(\mathcal{P}_1, \mathcal{P}_2)$, en créant les arêtes transversales comme sur la figure 2.2(b).



(a) Vérification et sélection de la prochaine arête (b) Rajout de toutes les arêtes transversales

FIGURE 2.2 – Fusion de 2 partitions dans le paradigme « Divide and conquer »

FUSION DE PARTITIONS En premier lieu, il est nécessaire de distinguer 3 types d'arêtes :

$$\begin{array}{lll} \text{gauche :} & \mathsf{L} & = \{(i,j) \in \mathcal{P} \mid i,j \in \mathcal{P}_1 \wedge i,j \notin \mathcal{P}_2\} \\ \text{transversale :} & \mathsf{LR} & = \{(i,j) \in \mathcal{P} \mid i \in \mathcal{P}_1 \wedge j \in \mathcal{P}_2\} \\ \text{droite :} & \mathsf{R} & = \{(i,j) \in \mathcal{P} \mid i,j \in \mathcal{P}_2 \wedge i,j \notin \mathcal{P}_1\} \end{array}$$

On commence par insérer l'arête transversale de base [AB] (la plus basse et qui n'intersecte aucune autre arête de $\mathcal{P}_1$ ni de $\mathcal{P}_2$). Le point délicat de cette phase est de déterminer la prochaine arête transversale à connecter à la base, comme sur la figure 2.2.

Sélection du sommet candidat pour l'arête transversale

On sélectionne un sommet candidat P de $\mathcal{P}_1$ tel que $\measuredangle(P,B,A) = \min_{P_i \in \mathcal{P}_2} \measuredangle(P_i,B,A)$, et on vérifie :

1. $\measuredangle(P,B,A) \leq 180$.
2. le cercle circonscrit au triangle $\Delta_{(A,B,P)}$ ne contient aucun autre sommet candidat – voir figure 2.3.

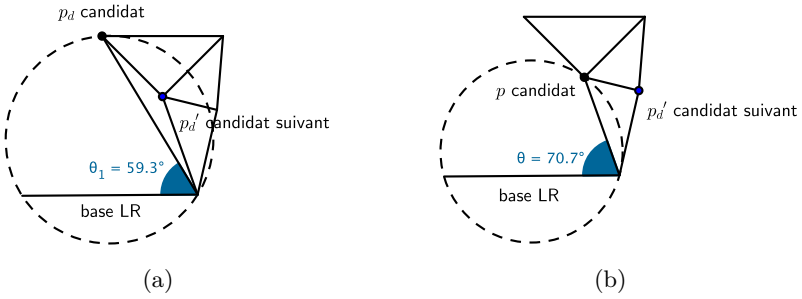


FIGURE 2.3 – Sélection du sommet candidat de la partition de droite

Choix de l'arête transversale

- ✏ si P satisfait ces 2 critères, alors l'arête $[PB]$ sera créée.
- ✏ si P ne satisfait pas (1), alors aucune arête ne sera créée.
- ✏ si P satisfait (1) mais pas (2), alors on supprime l'arête $[PB]$ de $\mathcal{P}_2$ (voir figure 2.3).

On itère le traitement jusqu'à obtenir le candidat final de $\mathcal{P}_2$, ou si aucun sommet candidat n'a été retenu. On fait de même pour $\mathcal{P}_1$, mais de manière symétrique (voir figure 2.4).

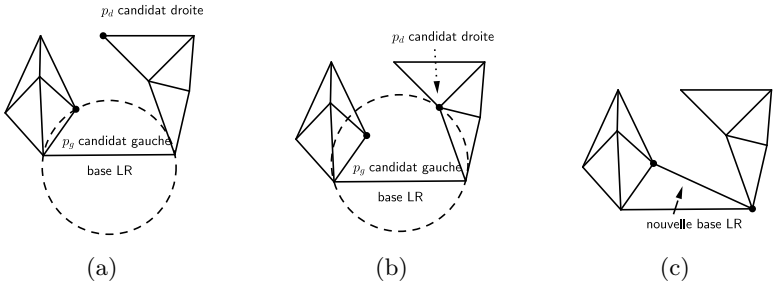


FIGURE 2.4 – Sélection du sommet final et complétion d'arête dans la phase de fusion

La fusion des 2 partitions courantes se termine quand tous les sommets candidats auront été vérifiés, et les arêtes transversales insérées. On réitère le traitement jusqu'à ce que toutes les partitions soient fusionnées.

COMPLEXITÉ Elle est de $\mathcal{O}(n \log n)$. Cet algorithme est théoriquement plus performant que les 2 autres décrites dans les sous-sections 2.1.1 et 2.1.3. Néanmoins, ces performances doivent être nuancées car l'algorithme ne garantit pas d'obtenir des partitions de taille égales à chaque pas de récursion [23], et la fusion des cavités nécessite généralement une phase de correction en pratique.

2.1.3 Méthodes incrémentales

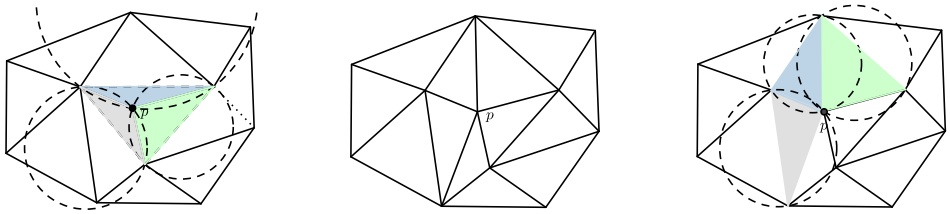
Il s'agira de construire la triangulation par insertion itérative des points du domaine.
On distingue 2 approches qui diffèrent selon la méthode de légalisation [1] des faces impactées lors de cette insertion :

- ✏ celle de Lawson [5].
- ✏ celle de Bowyer-Watson [8, 9].

1. transformation des triangles qui ne respectent pas le critère de cercle circonscrit vide de Delaunay

ALGORITHME DE LAWSON Etant donné un point p à insérer dans la triangulation $\mathcal{T}_S$, il consiste à insérer p dans un triangle $\Delta \in \mathcal{T}_S$, et légaliser les faces impactées par une procédure de bascules récursives d'aretes. De manière succincte, cette procédure consiste à :

1. localiser le triangle Δ qui contient géométriquement le point p à insérer ;
2. subdiviser Δ ;
3. vérifier récursivement si le critère de Delaunay est vérifié par les triangles adjacents à Δ, et basculer l'arête commune le cas contraire – cf. figure 2.5.



(a) Insertion d'un point et vérification

(b) Basculement des arêtes illégales

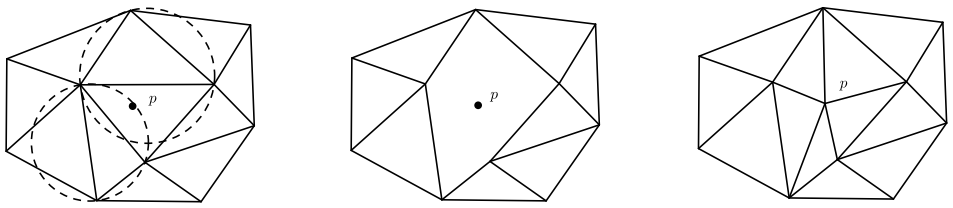
(c) Propagation de la vérification

FIGURE 2.5 – Algorithme de Lawson

COMPLEXITÉ Elle accuse une complexité en $\mathcal{O}(n^2)$.
Néanmoins il est possible d'obtenir une complexité en $\mathcal{O}(n \log n)$ en moyenne, en utilisant une heuristique adaptée pour la localisation des triangles [6].

ALGORITHME DE BOWYER-WATSON Etant donné un point p à insérer dans la triangulation $\mathcal{T}_S$, il consiste à déterminer tous les triangles dont le cercle circonscrit contient p, à supprimer ces triangles et à remailler la cavité sur la base de p. De manière succincte, cette procédure consiste à :

1. localiser le triangle Δ qui contient le point p à insérer ;
2. chercher tous les triangles dont le cercle circonscrit contient p ;
3. supprimer ces triangles (entrainant la formation d'une cavité vide) ;
4. re-trianguler la cavité vide à partir du point p, en créant de nouveaux triangles – cf. figure 2.6.



(a) Insertion d'un point et vérification

(b) Suppression des faces illégales

(c) Retriangulation de la cavité

FIGURE 2.6 – Algorithme de Bowyer-Watson

COMPLEXITÉ Elle a également une complexité en $\mathcal{O}(n^2)$.
De même que pour l'algorithme de Lawson, il est possible d'obtenir une complexité en $\mathcal{O}(n \log n)$, en utilisant une heuristique rapide pour trouver les faces adjacentes à un point p [24].

2.2 Approches parallèles

La génération parallèle de maillages constitue un domaine de recherche actif, et de nombreuses approches existent pour le cas des maillages de Delaunay [12, 13, 14, 15, 16].
Néanmoins, ces algorithmes sont adaptés pour des cas spécifiques la plupart du temps, et les gains de performances (speedup) dépendent fortement :

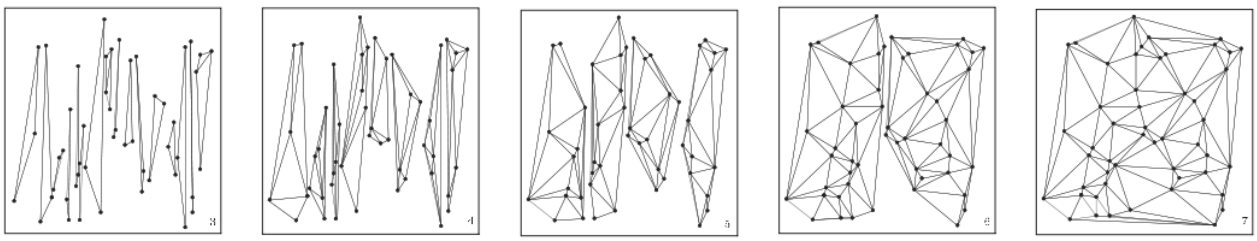
1. des données en entrée (distribution des sommets, topologie du domaine).
2. des optimisations sur l'architecture et sur les structures de données (conteneurs et allocateurs mémoire spécifiques, instructions atomiques dédiés à la place de verrous classiques etc.).

Dans le cadre de notre étude, on se restreindra aux méthodes directes, telles que décrites dans les sous-sections 2.1.2 et 2.1.3. A ce titre, on présente de manière succincte les approches directes existantes pour la génération parallèle de maillages triangulaires de Delaunay.

2.2.1 Méthodes D&C

Il s'agit de l'approche la plus fréquemment parallélisée dans la littérature.
De part la distribution des données, ces algorithmes fournissent une méthode de parallélisation directe : en divisant récursivement les données et en allouant une partition par PE [2].
Néanmoins ces méthodes souffrent de 2 inconvénients non triviaux à résoudre :

- ✏ la phase de fusion reste difficilement parallélisable, et nécessite généralement une phase de correction (voir figure 2.7). [25] présente une approche de parallélisation de cette phase de fusion, au prix d'une implémentation complexe pour des gains de performances limités.
- ✏ un déséquilibre des charges entre chaque PE, principalement due à la taille des taches partielles variant d'un niveau de subdivision à un autre.



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

FIGURE 2.7 – Fusion récursive de partitions dans le paradigme « divide and conquer »

Davy et Dew présentent une approche basée sur une subdivision du domaine en k partitions de taille égales dans [26]. Chaque partition sera allouée à 1 PE, et les triangulations partielles seront ensuite fusionnées. Les tests ont permis d'obtenir un speedup maximal de 5.07 pour 8 PE.

Dans [27], Clematis et Puppo raffinent l'approche présentée dans [26] en alternant des lignes de partitionnement horizontales et verticales, ce qui permet d'éviter des partitions trop étirées qui peuvent etre source de problèmes numériques.

Cignoni et al. présentent 2 approches de triangulation 3D basée sur ce paradigme dans [28].
La première consiste à traiter les éléments de la frontière en premier lieu, et à trianguler de part et d'autre de cette frontière par la suite. Cette méthode ne nécessite pas de phase de fusion, mais induit un déséquilibre important de charges dûe à la taille inégale des tâches entre les PE. Les tests ont été effectués sur 8 000 sommets uniformément distribués, avec un speedup maximal de 3.35 pour 16 PE.

2. Thread ou processus

La seconde approche consiste à diviser le domaine en k partitions réparties sur k PE. Elle vise à faciliter la reconnexion des partitions, en permettant la duplication des faces de la frontière lors du traitement des PE. De ce fait, la phase de fusion consiste juste à éliminer les faces redondantes. Les auteurs ont implémenté leur solution sur une machine nCUBE 2, avec 20 000 points uniformément distribués pour un speedup maximal de 19,01 pour 64 PE.

Hardwick présente une approche basée sur l'extraction d'une enveloppe convexe 3D dans [29]. En premier lieu, les sommets sont projetés sur un paraboloide $\mathcal{P} \in \mathbb{R}^3$, comme sur la figure 2.8. L'enveloppe convexe est ensuite calculée, et la triangulation déduite est ensuite re-projetée dans le plan. En effet, pour chaque point $(x, y) \in \mathbb{R}^2$, sa projection verticale dans le paraboloide correspond à $(x, y, x^2 + y^2)$ [30].

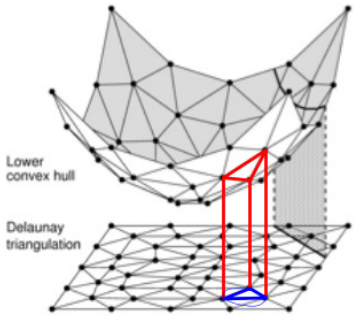


FIGURE 2.8 – Projection des points sur un paraboloide et extraction de la triangulation [31]

Sa parallélisation consiste ensuite à partitionner les points initiaux via une ligne médiane, selon l'abscisse des points. La ligne est ensuite remplacée par les arêtes déduites du calcul de l'enveloppe convexe dans le paraboloide. L'approche ne nécessite aucune phase de fusion, et est moins sensible à la distribution des sommets, néanmoins sa mise en œuvre est nettement plus compliquée comparé à la plupart des méthodes existantes selon [13]. Les auteurs ont implémenté leur solution sur un SGI Power Challenge en mémoire partagée avec 128 000 points initiaux uniformément distribués, pour un speedup maximal de 5.8 pour 8 PE.

Dans [32], Chen et al. présentent une approche qui consiste à créer une interface [3] sur la frontière, de manière incrémentale. On fusionne ensuite les 2 interfaces de la frontière, et on rajoute les triangulations partielles. La procédure de fusion reste séquentielle, néanmoins sa durée reste négligeable comparé à celles des autres procédures. Les auteurs ont implémenté leur solution sur un IBM SP2 avec 96 000 points uniformément distribués pour un speedup maximal de 4.95 pour 8 PE.

Dans [12], Lee et al. présentent une approche basée sur une combinaison de [29] et [28], qui consiste à créer les partitions de sommets en une seule passe, et non plus de manière récursive. Les auteurs ont implémenté leur solution sur un INMOS TRAM pour un speedup maximal de 12.5 pour 32 PE. Ils n'ont toutefois pas précisé le nombre de points utilisés dans leurs cas-tests.

2.2.2 Méthodes incrémentales

Bien que plus rare, des approches de parallélisation d'algorithmes incrémentaux existent également dans la littérature. De part la nature séquentielle de cette approche, il est difficile d'obtenir un speedup substantiel, sans le recours à diverses heuristiques (rééquilibrage de charges, optimisation des communications, etc.).

3. Ensemble de faces longeant une frontière

A proprement parler, il n'existe pas d'algorithme parallèle incrémental optimal. La plupart des approches existantes se basent sur des heuristiques, dont les performances varient sensiblement en fonction de la distribution des données et de l'architecture utilisée.

Puppo et al. présentent une méthode de triangulation basé sur une insertion sélective de points dans [33]. Le critère retenu se base sur un calcul de la distance entre le point et la triangulation, et pour chaque face, on choisit d'insérer le point le plus distant. Les conflits entre PE sont gérés par une règle de priorité en fonction de l'erreur d'approximation de cette distance. Ils ont implémenté l'algorithme sur une machine CM-2, avec 512^2 points initiaux, pour un speedup maximal limité à 80 pour 16 000 PE.

Dans [34], Okusanya et Peraire présentent une parallélisation de l'algorithme décrit dans [35].
La triangulation initiale est répartie sur k processeurs, et si l'insertion d'un nouveau point affecte les faces distantes (sur un autre processeur), alors le PE récupère ces faces par passage de messages (MPI ou PVM). Ce dernier les verrouille et les envoie au PE maitre, qui les déverrouillera une fois l'insertion terminée. Les auteurs ont implémenté leur solution sur un IBM SP2, avec 1 million de points uniformément distribués, pour un speedup maximal de 3 pour 8 PE.

Dans [36], Chrisochoides et Nave présentent une implémentation de l'algorithme de Bowyer-Watson en mémoire distribuée (cf. sous-section 2.1.3), en utilisant un parallélisme à base de tâches. En premier lieu, une triangulation initiale est générée par la version séquentielle, et les faces sont ensuite répartis sur k processeurs. Quand une cavité est partagée par plusieurs PE, elle nécessite une synchronisation par passage de messages. Les auteurs n'ont toutefois pas présenté de résultats concrets et complets, permettant de mesurer les performances réelles de leur méthode.

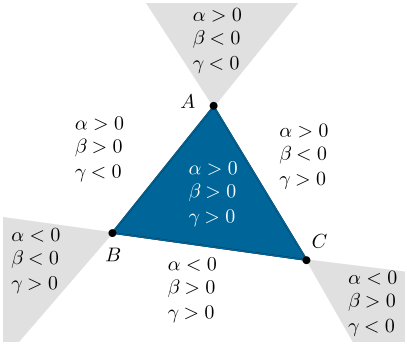
Kohout et al. présentent une approche en mémoire partagée dans [17] et [13]. Elle se base sur l'utilisation d'un graphe orienté acyclique (DAG) pour la localisation des faces. Ce graphe peut être modifié simultanément par plusieurs PE lors de l'insertion des points. Les auteurs ont implémenté leur solution sur un DELL PowerEdge 8450, avec 1 million de points uniformément distribués, pour un speedup maximal de 5.84 pour 8 PE.

2.3 Heuristiques de localisation

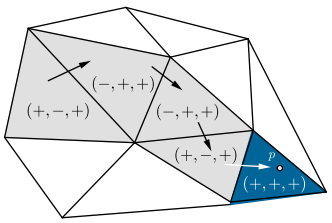
Toutes les approches décrites précédemment nécessitent une procédure efficace pour la localisation des cellules, lors de la phase de triangulation.
On distingue usuellement 3 approches dans la littérature :

1. PARCOURS BARYCENTRIQUE [5] : Partant d'une face Δ : (A,B,C) choisie aléatoirement, on calcule les coordonnées barycentriques (α, β, γ) de p par rapport au repère $(\vec{AB}, \vec{AC})$, et on détermine la position de p en fonction des signes de α, β, et γ. On réapplique ainsi la même procédure jusqu'à tomber sur la face pour lequel $\alpha > 0, \beta > 0$ et $\gamma > 0$. Cette méthode ne nécessite aucune structure de données annexe, mais nécessite de parcourir potentiellement tout le maillage pour chaque insertion, soit en $\mathcal{O}(n)$ – voir figure 2.9.

2. DÉCOMPOSITION SPATIALE [37] : elle se base sur une subdivision du domaine en grille (bucketing), en utilisant une structure arborescente (2-tree, 4-tree) pour mémoriser les partitions et les points qu'il contient. Pour localiser la face qui contient le point p à insérer, on effectue un parcours en profondeur de la structure jusqu'à tomber sur la partition qui contient géométriquement p. Parmi les faces qui intersectent cette cellule, on récupère celle qui contient p. On partitionne ensuite la cellule, et on met à jour la structure. Sa construction s'effectue en $\mathcal{O}(n \log n)$ et son parcours en $\mathcal{O}(\log n)$. – voir figure 2.10



(a) Coordonnées (α, β, γ) de p dans le repère $(\vec{AB}, \vec{AC})$



(b) Parcours en fonction du signe de (α, β, γ)

FIGURE 2.9 – Localisation de face par parcours barycentrique

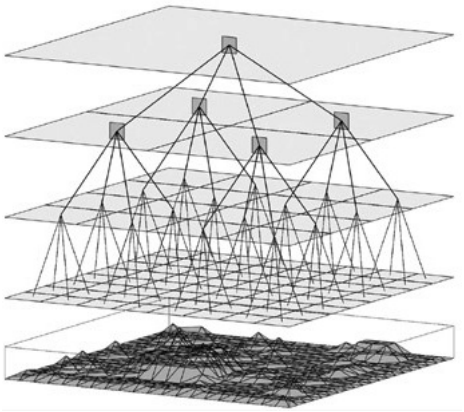


FIGURE 2.10 – Partitionnement en grille du domaine et quadtree correspondant

3. GRAPHE DE LOCALISATION [38] : on stocke l'historique des modifications dans un graphe orienté : chaque face partitionnée garde une référence vers celles qui l'ont remplacée, de même pour le basculement. Il suffit de parcourir la structure chainée, en testant à chaque fois si p est géométriquement contenu dans la face courante Δ. En utilisant une structure de données adéquate, cette méthode permet une localisation de faces en $\mathcal{O}(\log n)$ en moyenne.

Chapitre 3

Algorithme séquentiel

Ce chapitre présente l'algorithme séquentiel qui nous servira de base pour la mise en œuvre de l'heuristique. En premier lieu, on présente l'approche adoptée et la structure de maillage utilisé dans les sections 3.1 et 3.2. Ensuite, on détaillera l'algorithme dans la section 3.3, et sa complexité dans la section 3.4. Enfin, on terminera par une présentation des résultats obtenus dans la section 3.5.

3.1 Modèle de maillage

Pour la génération séquentielle du maillage, notre approche se base sur l'algorithme de Lawson décrit à la page 17.
Malgré une complexité de $\mathcal{O}(n^2)$ en pire cas, cette méthode permet de générer une triangulation de Delaunay en $\mathcal{O}(n \log n)$ en moyenne, à condition d'utiliser une structure de données adaptée pour la localisation des faces, ceci lui vaut d'être aussi performante que les méthodes D&C.

Cette approche confère plusieurs avantages. D'une part, elle maintient intrinsèquement le critère de Delaunay lors d'une insertion ou suppression de points, ce qui évite d'avoir à regénerer le maillage lors d'une modification locale. D'autre part, elle peut également être adaptée pour la prise en compte de contraintes d'arêtes, sans nécessiter de profondes modifications [1]. Enfin, il est possible de la rendre insensible à la distribution des points, en « randomisant » l'ordre d'insertion des sommets.

CONNECTIVITÉ DES ÉLÉMENTS Les procédures de bascules et partitionnement d'arêtes nécessitent la récupération du voisinage topologique de la cellule en cours de modification.
Pour ce faire, nous avons choisi d'implémenter le modèle [face → face], permettant la récupération du voisinage en $\mathcal{O}(1)$. Ainsi, chaque cellule stocke localement une référence de ses 3 faces voisines, qu'il faudra mettre à jour pour toute modification relative à cette cellule ou à une de ses voisines.
A titre informatif, la matrice de connectivité est une matrice carrée $\mathcal{M}_n$ symétrique, décrite par :

$$\mathcal{M}_n = (a_{i,j}), \text{ avec } (a_{i,j}) = \begin{cases} 1 & \text{si } \exists (u,v) \in \mathcal{M} | \Delta_i \cap \Delta_j = (u,v) \\ 0 & \text{sinon} \end{cases}$$

GRAPHE DE LOCALISATION Pour la localisation des faces, on choisit l'approche utilisant une structure annexe, ce qui nous semble un bon compromis entre le cout en temps de calcul et le cout mémoire [2].
Pour cela, on utilise une structure arborescente appelée DAG (Directed Acyclic Graph).
De manière concise, il s'agit d'un graphe orienté $\mathcal{G}(V, A)$ avec :

1. V : ensemble des nœuds définis par :
   - ✏ 1 triangle du maillage en cours de construction [3]
   - ✏ 2 références vers les noeuds parents

1. Il en est de même de son extension en dimension 3 (maillage tétraédrique)
2. En $\mathcal{O}(n)$
3. En pratique, on stockera plutôt l'identifiant et les sommets du triangle

- ✏ 3 références vers les noeuds fils

2. $A = \{(n_{\Delta_1}, n_\Delta) \in V^2 | \Delta$ provient d'une subdivision de Δ_1 ou d'une bascule de $e \in \Delta_1 \cap \Delta_2\}$.
   Il s'agit des connexions orientées représentant le fait qu'un triangle Δ a été obtenu par partitionnement de Δ_1, ou par bascule d'une arête commune entre Δ_1 et Δ_2.

Les faces courantes du maillage sont répertoriées au niveau des feuilles du DAG, et leur localisation s'effectue donc en $\mathcal{O}(\log n)$, n étant le nombre total de faces créées [13].
L'avantage d'une telle structure est qu'elle permet d'effectuer des « rollbacks » [4] en cas de nécessité. Néanmoins, cela implique une mise à jour constante du graphe pour toute modification du maillage pendant la procédure de triangulation.

3.2 Le conteneur GMDS

Le choix des structures de données influe grandement sur la complexité d'un algorithme de maillage. A ce titre, le CEA a développé une plateforme générique GMDS (Generic Mesh Data Structure) pour la représentation de maillage et le référencement de ses éléments [39].
Afin d'exploiter les spécificités et optimisations du conteneur, mais également de comprendre les contraintes inhérentes, il est nécessaire de comprendre la mécanique générale du conteneur.

REPRÉSENTATION DU MAILLAGE Afin de réduire l'empreinte mémoire, GMDS est optimisé de manière à éviter les recopies, tout en garantissant l'extensibilité (gestion de la réallocation mémoire [5], éléments 3D). En effet, il permet d'accéder aux données topologiques en $\mathcal{O}(1)$ sous certaines conditions [6], et est extensible car il permet l'ajout ultérieur de nouvelles connectivités.
Tous les éléments (faces, nœuds et arêtes), et leurs connectivités sont gérés par un système de références [7].
Lors de la récupération d'une face, celle-ci est reconstituée à la volée, et seuls les données relatives aux sommets (position, valeurs nodales etc.) seront stockées telles quelles.

RÉFÉRENCEMENT DES ÉLÉMENTS Comme décrite sur la figure 3.1, le conteneur est composé de 3 structures de données :

1. MAILLAGE : il contient les références des cellules constituant le maillage.
   Il s'agit d'un tableau dynamique (type `std::vector` en `C++`) de booléens $\mathcal{M}[i]$, indiquant si la $i^{ème}$ cellule existe dans le maillage ou non. A chaque création de cellules, il sélectionnera l'indice de la première case vide, afin de ne réserver de nouvelles cases que si le conteneur est plein.

2. GESTIONNAIRE DE CELLULES : pour une cellule i, il fournit son type (triangle, quadrangle etc.), et son ID interne au sein du gestionnaire de connectivité associé à chaque type.

3. GESTIONNAIRE DE CONNECTIVITÉS : maintient 1 tableau dynamique par type d'adjacence.
   Selon le modèle spécifié, il allouera les tableaux correspondants (`T2N` et `T2F` dans notre cas). Chaque case sera référencée par un ID interne, et indiquera :
   - ✏ un vecteur d'indices correspondant aux sommets du triangle k pour `T2N`
   - ✏ un vecteur d'indices correspondant aux 3 voisins du triangle k pour `T2F`.
     Il s'agit du tableau qu'il faudra mettre manuellement à jour à chaque modification (création, suppression, partitionnement, basculements) d'une cellule ou de ses voisins.

4. Annulation des modifications et retour à l'état précédent
5. En cas d'ajout d'un nouvel élément dans un conteneur plein, celui-ci réserve une nouvelle plage mémoire. Cela entraine potentiellement une recopie des données existantes, ce qui invalide les pointeurs
6. Si on stocke les connectivités des faces F → F
7. Correspondance entre identifiants globaux et internes

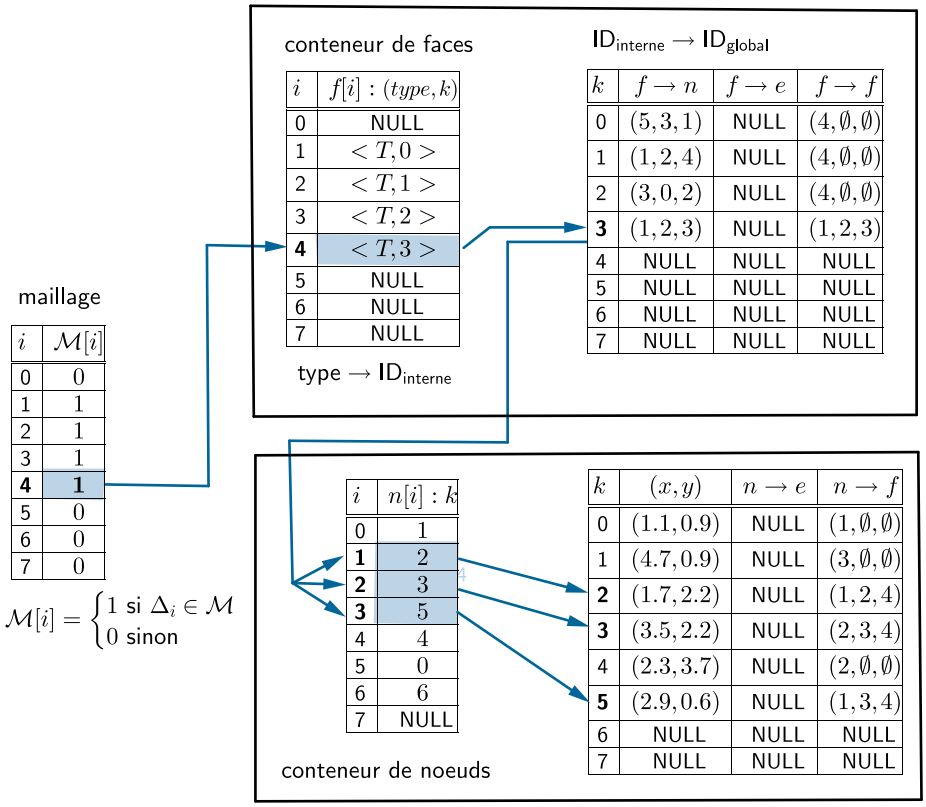


FIGURE 3.1 – Stockage interne et référencement des éléments du maillage dans GMDS

3.3 Détails de l'algorithme

De manière succincte, l'algorithme comporte 3 grandes étapes :

1. une phase de triangulation $\mathcal{T}_S$ à partir d'une discrétisation $\mathcal{S}$ de la frontière du domaine Ω.
2. une phase de génération du maillage $\mathcal{M}$ par raffinement de $\mathcal{T}_S$.
3. une phase d'optimisation de $\mathcal{M}$.

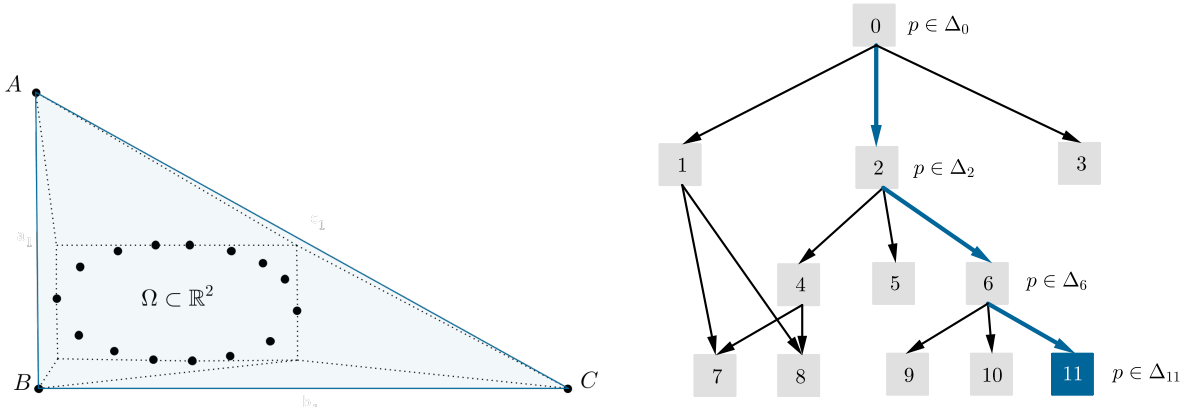
3.3.1 Triangulation

Dans cette partie (uniquement), on s'est intéressé à [13] qui propose une approche de triangulation adaptée pour une parallélisation à base de threads.
En premier lieu, on discrétise la frontière du domaine en une liste de points $\mathcal{S}$, et on crée un triangle englobant Δ_c couvrant chaque $p \in \mathcal{S}$ – voir figure 3.2(a) – et en initialisant la racine du graphe à Δ_c. Une fois la liste de points $\mathcal{S}$ obtenue, on procède à l'insertion successive de chaque $p \in \mathcal{S}$ dans la triangulation courante.

LOCALISATION Pour chaque $p \in \mathcal{S}$, il faut localiser la cellule Δ qui contient p. En partant de la racine du DAG, on effectue un parcours en profondeur, en vérifiant pour chaque nœud si p se trouve à l'intérieur du triangle Δ' référencé. La procédure se termine en retournant l'indice de la feuille dont le triangle référencé contient p – voir figure 3.2(b).

SUBDIVISION On procède ensuite à la subdivision de $\Delta = (I, J, K)$:

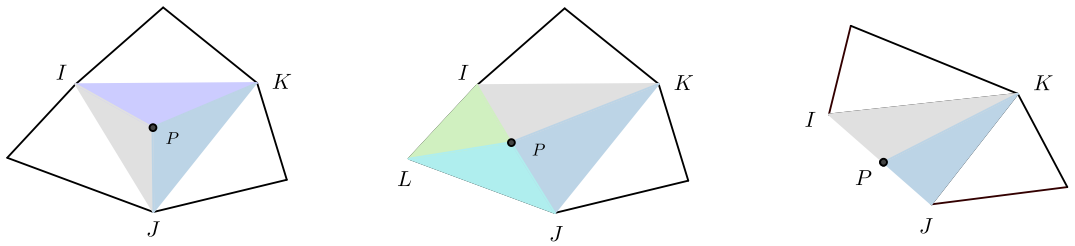
✏ Si p se trouve strictement à l'intérieur de Δ, alors on subdivise Δ en 3 faces, comme sur la figure 3.3(a), tout en veillant à respecter l'orientation anti-horaire des arêtes.



(a) Discrétisation de la frontière et triangle initial Δ_0 (b) Graphe de localisation des faces

FIGURE 3.2 – Initialisation du triangle conteneur et graphe de localisation pour l'étape de triangulation
Les coordonnées de (A, B, C) sont déduites du rectangle englobant les points de Ω.
Pour récupérer la face où insérer p, on parcourt en profondeur et on retourne la feuille qui contient p. Les arcs croisés correspondent à une bascule de l'arête commune à Δ_1 et Δ_4.

- ✏ Si p se trouve sur une arête (I,J) de Δ, alors on récupère la face Δ' adjacente à Δ par cette arête, et on scinde Δ et Δ' en 4 faces, comme sur la figure 3.3(b).
- ✏ Si p est sur une arête de la frontière, alors on a juste à scinder Δ en 2 comme sur la figure 3.3(c).



(a) p à l'intérieur de Δ (b) p sur une arête (c) p sur la frontière

FIGURE 3.3 – Partitionnement des faces en fonction de la position du point p

Pour chaque création de face, il est important de garder la même orientation (anti-horaire) des arêtes. En effet, les prédicats géométriques utilisent des heuristiques basées sur des produits vectoriels, dont les signes dépendront de l'orientation des arêtes de la face concernée (cf. lemme 1, page 60).

LÉGALISATION Ensuite, pour chaque face Δ créée, on procède à la légalisation des arêtes externes. En d'autre termes, on vérifie le critère de Delaunay sur cette face, c'est-à-dire qu'aucun voisin Δ' n'est inclus dans le cercle circonscrit à Δ – voir définition 5, page 8. Si tel est le cas, alors il faut basculer l'arête (i, j) commune aux 2 faces, et ré-appliquer récursivement le critère de Delaunay sur chaque nouvelle paire de faces, jusqu'à ce que l'intégralité de la triangulation soit conforme – voir figure 2.5.
Par ailleurs, on mettra à jour le DAG en initialisant un nouveau nœud $n_{\Delta'}$, en récupérant le nœud parent n_Δ (face subdivisée ayant conduit à Δ'), et en rattachant $n_{\Delta'}$ à n_Δ par une mise à jour des références.

La figure 3.4 donne un aperçu partiel de la procédure de triangulation décrite dans l'algorithme 1.

Algorithm 1 Triangulation du domaine Ω

procedure TRIANGULATE($\mathcal{S}$, λ)
$\mathcal{S}$: l'ensemble de sommets issus de la discrétisation de Ω
λ : distance minimale entre les sommets du triangle initial et ceux de $\mathcal{S}$

$\Delta_0 \leftarrow$ COMPUTE-TRIANGLE($\mathcal{S}$, λ)
DAG_INSERT(Δ_0) ▷ Initialisation de la racine du graphe

while $\mathcal{S} \neq \emptyset$ **do**
$\Gamma \leftarrow \emptyset$ ▷ File qui contiendra les nouvelles faces créées
$p \leftarrow \mathcal{S}[0]$
$\Delta \leftarrow$ DAG_FIND(p) ▷ Localisation de la face qui contient P
$\Gamma \leftarrow$ INSERT(p, Δ, TRUE)
LEGALIZE(Γ) ▷ Vérification du critère de Delaunay sur chaque face créée
$\mathcal{S} \leftarrow \mathcal{S} - \{p\}$
end while
end procedure

---

procedure COMPUTE-TRIANGLE($\mathcal{S}$, λ)
$\mathcal{S}$: l'ensemble de sommets issus de la discrétisation de Ω
λ : distance minimale entre les sommets du triangle initial et ceux de $\mathcal{S}$

$(x_{min}, y_{min}) \leftarrow \min(\mathcal{S})$
$(x_{max}, y_{max}) \leftarrow \max(\mathcal{S})$ ▷ Construction du triangle initial
$p_1 \leftarrow (x_{min} - \lambda, y_{min} - \lambda)$
$p_2 \leftarrow (x_{min} - \lambda, y_{max} + \lambda)$
$p_3 \leftarrow (x_{max} + \lambda, y_{min} - \lambda)$
return $\Delta \leftarrow (p_1, p_2, p_3)$;
end procedure

---

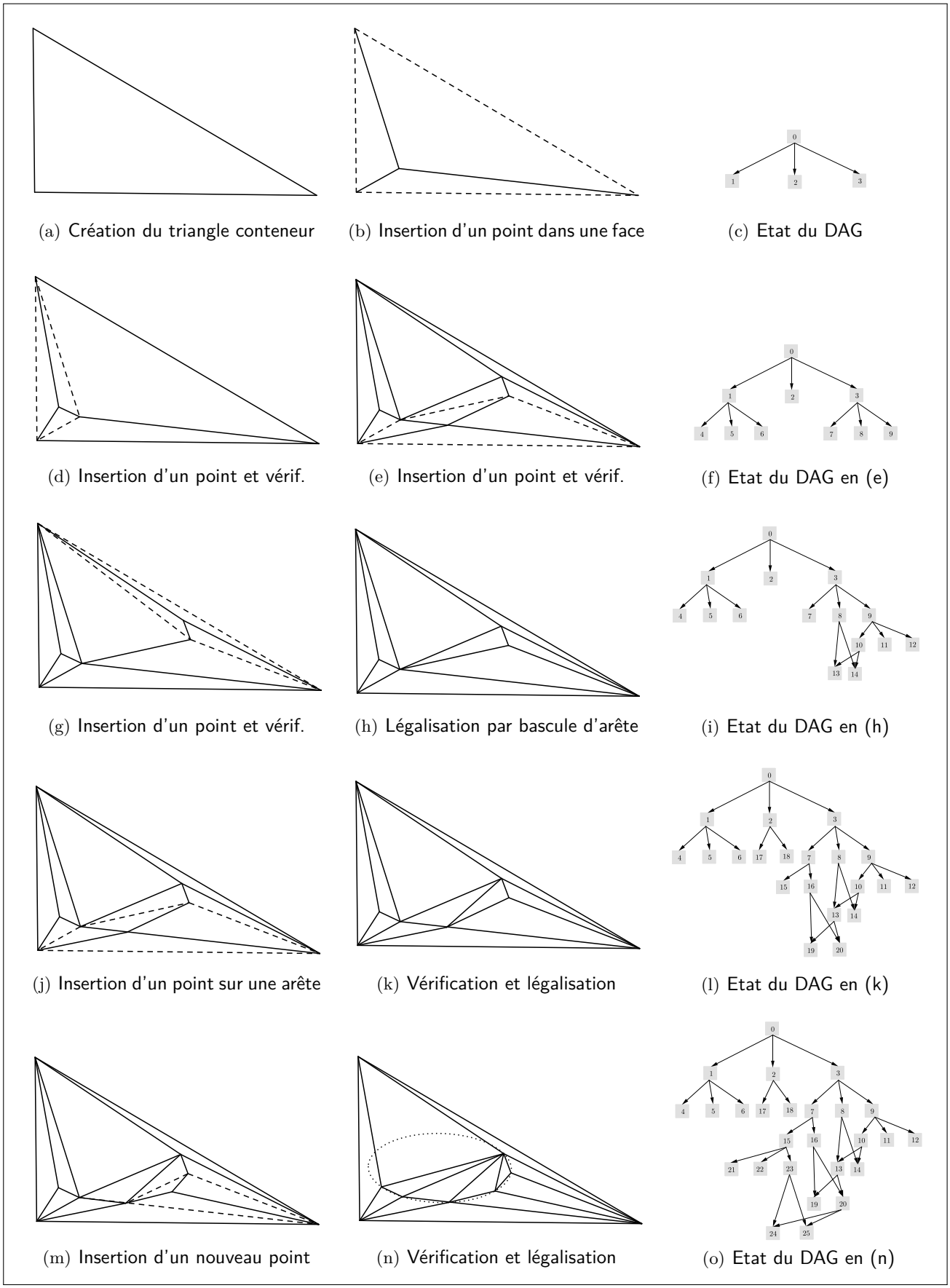
procedure INSERT(P, Δ, DAG)
P : le point à insérer dans la triangulation
Δ : la face où insérer P
DAG : booléen indiquant s'il faut mettre à jour le DAG ou non
$\mathcal{N}_\Delta = \{\Delta' \mid \exists u, v \in \mathcal{S} \text{ tq } \Delta \cap \Delta' = (u, v)\}$ ▷ voisinage topologique de la face Δ

if $\det(\vec{PB}, \vec{AB}) = 0$ **then** ▷ P sur l'arête [AB]
$\Gamma \leftarrow$ SPLIT(Δ, P, $\vec{AB}$)
else if $\det(\vec{PC}, \vec{BC}) = 0$ **then** ▷ P sur l'arête [BC]
$\Gamma \leftarrow$ SPLIT(Δ, P, $\vec{BC}$)
else if $\det(\vec{PA}, \vec{CA}) = 0$ **then** ▷ P sur l'arête [CA]
$\Gamma \leftarrow$ SPLIT(Δ, P, $\vec{CA}$)
else ▷ P à l'intérieur de Δ
$\Gamma \leftarrow$ SUBDIVIDE(Δ, P)
end if

SET_ADJACENCY(Γ, $\mathcal{N}_\Delta$, $\mathcal{N}_{\Delta'}$) ▷ mise à jour du voisinage

---

if DAG **then** ▷ S'il faut mettre à jour le DAG
if $|\Gamma| = 4$ **then** ▷ si P sur une arête interne
DAG_LINK(Δ, Γ[1,2]) ▷ Création des nœuds fils n_{Δ_i} pour chaque $\Delta_i \in \Gamma$ et connexion à n_Δ
DAG_LINK(Δ', Γ[3,4])
else ▷ si P à l'intérieur de Δ ou sur une arete du bord
for $\Delta_i \in \Gamma$ **do**
DAG_LINK(Δ, Δ_i)
end if
end if
end procedure



(a) Création du triangle conteneur (b) Insertion d'un point dans une face (c) Etat du DAG

(d) Insertion d'un point et vérif. (e) Insertion d'un point et vérif. (f) Etat du DAG en (e)

(g) Insertion d'un point et vérif. (h) Légalisation par bascule d'arête (i) Etat du DAG en (h)

(j) Insertion d'un point sur une arête (k) Vérification et légalisation (l) Etat du DAG en (k)

(m) Insertion d'un nouveau point (n) Vérification et légalisation (o) Etat du DAG en (n)

Figure 3.4 – Triangulation d'un domaine elliptique, et mise à jour du graphe de localisation

3.3.2 Génération du maillage

Une fois la triangulation $\mathcal{T}_S$ obtenue, on procède à son raffinement, selon une métrique fournie.
En premier lieu, on identifie les faces qui ne respectent pas les critères définis par cette métrique (longueur maximum d'arêtes $\bar{\rho}$, angle maximal des triangles $\bar{\theta}$). Ces faces sont insérées itérativement dans une file $\mathcal{Q}$. On procède ensuite au raffinement de chaque $\Delta \in \mathcal{Q}$ – voir algorithme 2.
A noter que la métrique et les coordonnées du point de raffinement sont des paramètres qu'il faudra adapter à la nature de la surface à mailler.

3.3.2.1 PRINCIPE

Pour chaque face $\Delta : (A, B, C) \in \mathcal{Q}$, on calcule les coordonnées du point de raffinement (x_P, y_P), en fonction de l'aplatissement $\bar{\theta}$, et de la taille cible des arêtes $\bar{\rho}$ – voir algorithme 3.
Une fois les coordonnées du point P obtenues, on vérifie la position de P par rapport aux 3 arêtes AB, BC et CA de Δ, en calculant respectivement $(\vec{PB} \cdot \vec{AB})$, $(\vec{PC} \cdot \vec{BC})$, et $(\vec{PA} \cdot \vec{CA})$.
Si l'un de ces 3 produits scalaires est nul, alors l'arête correspondante sera scindée, sinon on procède à une subdivision classique de Δ, en vérifiant récursivement le critère de Delaunay dans les 2 cas.
Pour chaque nouvelle face créée, on vérifie si le critère de raffinement est respecté, et on le rajoute à la fin de la file $\mathcal{Q}$ dans le cas contraire.
La procédure complète est décrite dans l'algorithme 2.

Algorithm 2 Génération du maillage

procedure MESH($\mathcal{M}$, $\bar{\rho}$, $\bar{\theta}$)
$\mathcal{M}$: le maillage issu de la triangulation $\mathcal{T}$
$\bar{\rho}$: longueur max. d'arête
$\bar{\theta}$: aplatissement max. d'une face

  $\mathcal{Q} \leftarrow \{\Delta \in \mathcal{M} \mid (\max_{\vec{u} \in \Delta} ||\vec{u}|| \geq \bar{\rho}) \vee (\max_{\theta \in \Delta} \measuredangle(\theta) \geq \bar{\theta})\}$ ▷ Récupération des faces de mauvaise qualité

  while $\mathcal{Q} \neq \emptyset$ **do**
    $\Delta_s \in \mathcal{Q}$;
    if $\Delta_s \in \mathcal{M}$ **then** ▷ Si la face existe toujours (suppression possible après un swap)
      REFINE($\mathcal{M}$, Δ_s, $\mathcal{Q}$, $\bar{\rho}$, $\bar{\theta}$)
    end if
    $\mathcal{Q} \leftarrow \mathcal{Q} - \{\Delta_s\}$;
  end while
end procedure

procedure REFINE($\mathcal{M}$, Δ_s, $\mathcal{Q}$, $\bar{\rho}$, $\bar{\theta}$)
$\mathcal{M}$: le maillage courant
Δ_s : la face courante
$\mathcal{Q}$: file des cellules de mauvaise qualité (selon la métrique définie)
$\bar{\rho}$: longueur max. d'arête
$\bar{\theta}$: angle max. d'une face

  $\Gamma \leftarrow \emptyset$ ▷ File qui contiendra les nouvelles faces créées
  $p \leftarrow$ MIDPOINT(Δ_s) ▷ On récupère le point de raffinement p
  $\Gamma \leftarrow$ INSERT(p, Δ_s,FALSE) ▷ On insère ensuite p dans Δ (on ne met pas à jour le DAG)
  for $\Delta_c \in \Gamma$ **do** ▷ On vérifie la taille des nouvelles faces selon la métrique
    if $(\max_{\vec{u} \in \Delta_c} ||\vec{u}|| \geq \bar{\rho}) \vee (\max_{\theta \in \Delta_c} \measuredangle(\theta) \geq \bar{\theta})$ **then**
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\Delta_c\}$
    end if
  end for
  LEGALIZE(Γ) ▷ On légalise les arêtes de chaque face créée
end procedure

3.3.2.2 POINT DE RAFFINEMENT

Il existe plusieurs approches pour le choix du point de raffinement dans la littérature.
Dans [40] et [35], les auteurs utilisent les centres des cercles circonscrits aux faces, en définissant une borne supérieure des angles internes, [7] utilise le barycentre de la face à raffiner,et [41] utilise la notion de « variogramme » (géostatistique).
Dans [42, 43, 44], les auteurs préconisent une approche basée sur la prédétermination des points à insérer, tandis que [11] se base sur une saturation d'arêtes, qui consiste à insérer autant de points sur chaque arête du maillage jusqu'à ce que $d(u,v) = \bar{\rho}$ pour toute arête $(u,v) \in \mathcal{M}$.

Notre approche se base sur celle de [7], avec en plus la prise en compte de l'aplatissement (angle max θ_Δ de la face Δ), auquel cas on choisit d'insérer sur l'arête opposée à θ_Δ.

Principe

Pour une face Δ, on vérifie si toutes les arêtes de Δ ont une longueur inférieure à la taille cible $\bar{\rho}$. Si ce n'est pas le cas, alors on examine ses angles $\alpha = \angle(\vec{AB}, \vec{AC})$, $\beta = \angle(\vec{BA}, \vec{BC})$, et $\gamma = \angle(\vec{CB}, \vec{CA})$. Si l'un de ces angles dépasse l'angle max. spécifié $\bar{\theta}$, alors on retourne le milieu de l'arête opposée (i,j), ainsi l'arête (i,j) elle-même.
La procédure complète est décrite dans l'algorithme 3.

Cette méthode permet de réduire le facteur d'aplatissement des faces, particulièrement ceux du bord du domaine, et on évite ainsi les erreurs numériques liées à la précision.

Algorithm 3 Récupération du point de partitionnement

procedure MIDPOINT(Δ, $\bar{\rho}$, $\bar{\theta}$)
$\bar{\rho}$: longueur max d'arête
$\bar{\theta}$: angle max autorisé
$P = (x_P, y_P)$: le point de partitionnement, et (i,j) : l'arête à segmenter (si nécessaire)

$(A,B,C) \leftarrow \Delta_c$
if $(||\vec{AB}|| > \bar{\rho}) \vee (||\vec{BC}|| > \bar{\rho}) \vee (||\vec{CA}|| > \bar{\rho})$ **then** ▷ Vérification de la longueur des arêtes de Δ

$$\alpha = \arccos\left(\frac{\vec{AB}\cdot\vec{AC}}{||\vec{AB}|| \times ||\vec{CA}||}\right) \quad \beta = \arccos\left(\frac{\vec{BA}\cdot\vec{BC}}{||\vec{BA}|| \times ||\vec{BC}||}\right) \quad \gamma = \arccos\left(\frac{\vec{CB}\cdot\vec{CA}}{||\vec{CB}|| \times ||\vec{CA}||}\right)$$

if $\measuredangle(\alpha) > \bar{\theta}$ **then**
$(x_P, y_P) \leftarrow \frac{1}{2}(x_B + x_C, y_B + y_C)$ $(i,j) \leftarrow (B,C)$ ▷ P : milieu de [BC]
else if $\measuredangle(\beta) > \bar{\theta}$ **then**
$(x_P, y_P) \leftarrow \frac{1}{2}(x_A + x_C, y_A + y_C)$ $(i,j) \leftarrow (C,A)$ ▷ P : milieu de [CA]
else if $\measuredangle(\gamma) > \bar{\theta}$ **then**
$(x_P, y_P) \leftarrow \frac{1}{2}(x_A + x_B, y_A + y_B)$ $(i,j) \leftarrow (A,B)$ ▷ P : milieu de [AB]
else
$(x_P, y_P) \leftarrow \frac{1}{3}(x_A + x_B + x_C, y_A + y_B + y_C)$ ▷ P : barycentre de Δ
end if
end if
return $[P, (i,j)]$
end procedure

3.3.3 Optimisation

Une fois le maillage brut obtenu, on cherche à améliorer la qualité géométrique des faces, de manière à les rendre les plus régulières possibles (des faces équilatérales idéalement).

MESURE DE QUALITÉ La qualité d'un simplexe est déterminée par [11] :

$$Q_\Delta = \left(\alpha \cdot \frac{\rho_\Delta}{r_\Delta}\right)^{-1}, \text{ avec } \left[\begin{array}{l} \rho_\Delta \text{ longueur du plus grand coté de } \Delta \\ r_\Delta \text{ rayon du cercle inscrit à } \Delta \\ \alpha \text{ coefficient de normalisation tel que } \mathcal{Q}_\Delta = 1 \Leftrightarrow \Delta \text{ équilatéral.} \end{array}\right.$$

Le terme $\frac{\rho_\Delta}{r_\Delta}$ mesure la distorsion de la face Δ par rapport à une face de référence Δ^* (équilatérale). La qualité du maillage $\mathcal{M}$ équivaut à celle de la face la plus distordue, et est définie par $\mathcal{Q}_\mathcal{M} = \min_{\Delta \in \mathcal{M}} \mathcal{Q}_\Delta$

Principe

Optimiser la qualité du maillage revient à maximiser $\mathcal{Q}_\mathcal{M}$.
Pour ce faire, on distingue usuellement 2 approches :

- ✏ celle qui consiste à modifier la connectivité des points, en conservant leur position
- ✏ celle qui consiste à bouger les points, en gardant leur connectivité (lissage).
  Il s'agit de la méthode que nous adoptons pour notre algorithme.

LISSAGE LAPLACIEN Il s'agit de l'opérateur d'optimisation locale que nous allons appliquer au maillage. Il consiste à déplacer le point de manière à ce qu'il soit le barycentre $\bar{p}$ de tout ses voisins p_j, avec :

$$\bar{p} = \frac{1}{n} \sum_{j=1}^{n} p_j$$

Néanmoins déplacer un point vers sa position « optimale » peut causer des chevauchements des faces de $\mathcal{M}$. Pour y remédier, on utilise plutôt une version relaxée qui consiste à approcher ce point optimal.

Lissage laplacien relaxé de p

$$p = (1 - \omega)p + \omega\bar{p}, \text{ avec } \omega \leq 1$$

Le paramètre de relaxation ω est relatif à la qualité des faces incidentes à p ($\mathcal{Q}_{\Delta_i}$).

On itère ensuite l'opération k fois, comme décrit sur l'algorithme 4.

Algorithm 4 Lissage du maillage par un opérateur laplacien

procedure LAPLACIAN($\mathcal{M}$, ω, k)
$\mathcal{M}$: le maillage à optimiser
ω : le paramètre de relaxation
k : nombre d'itérations

```
while i < k do
    for p ∈ M do
        for p_j ∈ N_p : voisins de p do
            p̄ ← p̄ + p_j
        end for
        p̄ ← 1/|N_p| · p̄
        p ← (1 − ω)p + ωp̄
    end for
    i ← i + 1
end while
end procedure
```

3.4 Complexité théorique

Mesurer la complexité d'un algorithme consiste à évaluer le nombre d'opérations en fonction de la taille des données d'entrée. Ici, on s'intéresse à la complexité moyenne car elle est plus représentative des performances réelles de l'algorithme.

Soit n le nombre initial de points, et m le nombre total de faces créées.
La création/suppression d'une face et la mise à jour des connectivités ne dépendent pas de n, et s'effectuent en $\mathcal{O}(1)$.

OPÉRATIONS ÉLÉMENTAIRES Il s'agit des procédures de localisation, de subdivision (faces/arêtes), de légalisation et de mise à jour du DAG (pour la phase de triangulation) :

1. Localiser la face Δ_p qui contient p revient à effectuer un parcours en profondeur du DAG. Son coût dépend donc de la hauteur h du DAG, soit en $\mathcal{O}(\log m)$ en moyenne.[8]. Comme il y a n points, la localisation s'effectue donc en $\mathcal{O}(n \log m)$ en moyenne.
2. Subdiviser une face Δ revient à créer 3 faces, à supprimer Δ et à mettre à jour l'adjacence de ces 3 faces. Elle s'effectue donc en $\mathcal{O}(1)$. Scinder une arête revient à créer 2 ou 4 faces, et à supprimer 1 ou 2 faces. Elle s'effectue aussi en $\mathcal{O}(1)$.
3. Durant la légalisation, on crée récursivement 2 faces, soit $\mathcal{O}(m^2)$ au cas où celle-ci se propagerait sur tout le maillage. Néanmoins elle est de $\mathcal{O}(m \log m)$ dans la plupart des configurations (sauf cas dégénérés).
4. La mise à jour du DAG implique la recherche du nœud parent pour chaque création de face, soit un cout en $\mathcal{O}(n \log m)$.

TRIANGULATION Les coûts de chaque opération élémentaire étant fixés, la complexité de la phase de triangulation se déduit du nombre de faces créées au sein de chaque étape de l'algorithme.
Pour trouver le nombre exact de faces créées (m), il faut calculer le degré maximal d'un point p (nombre d'arêtes adjacentes à p), et exprimer m en fonction de n.
Le calcul suivant se base sur le dénombrement décrit dans [45] :

- ☞ Si p a un degré k à la fin de la triangulation, alors on aura $2k-3$ triangles.
- ☞ Selon la relation d'Euler, le nombre d'arêtes de la triangulation vaut $3(i+3)-6 = 3i+3$, dont 3 appartiennent au triangle conteneur. Donc la somme des degrés est inférieure à $2(3i+3-3) = 6i$.
- ☞ Chaque sommet a donc en moyenne un degré de $k = 6$ au cours de la triangulation.
  A l'étape i, on crée donc : $2k - 3 = 2 \cdot 6 - 3 = 9$ triangles.

Au cours de la triangulation, on crée donc en tout $m = 9n + 1$ triangles (faces), soit une complexité de $\mathcal{O}(n \log(9n+1))$.
Or $n \log(9n+1) < n \log(n^9) = 9n \log(n)$.

Complexité de l'étape de triangulation

La complexité de la triangulation est de $\mathcal{O}(n \log(9n+1)) = \mathcal{O}(9n \log n) = \mathcal{O}(n \log n)$.

GÉNÉRATION DU MAILLAGE Cette procédure prend en entrée la file $\mathcal{Q}$ des « mauvais » triangles, issue de la triangulation. Ici, on cherche donc à évaluer le nombre de raffinements en fonction de la taille initiale de $\mathcal{Q}$. Dans ce cas, on cherche à borner la taille maximale de $\mathcal{Q}$, qui détermine le nombre d'itérations. Dans le pire cas, toutes les faces de la triangulation sont « mauvaises » (m en tout), donc

8. en $\mathcal{O}(m)$ dans le cas où l'ajout d'un nouveau point modifie la totalité de la triangulation, ce qui est très rare en pratique

$|\mathcal{Q}_0| = m$.

Sans perte de généralité, on suppose que l'on dépile entièrement $\mathcal{Q}$, avant de rajouter les nouvelles faces à chaque itération.
A l'étape i, on supprime 1 face et on ajoute au plus 3 faces dans $\mathcal{Q}$ lors de la subdivision, et on rajoute au plus 2 faces à chaque pas de légalisation (dans le cas où tous les triangles qu'on aura créés sont de mauvaise qualité).
Or, en pire cas, toutes les faces initiales ont été rajoutées dans $\mathcal{Q}$.
Dans ce cas, les faces qu'on aura supprimées au cours de la légalisation y sont également, et ne doivent plus être traitées à la prochaine itération.
A chaque pas de la légalisation (m au maximum), on aura donc ajouté 2 faces et supprimé 2 autres.
Donc à l'étape i, on rajoute au plus 3 faces à $\mathcal{Q}$, et lorsque $\mathcal{Q} = \emptyset$, on aura rajouté en tout $3m$ faces dans $\mathcal{Q}$, en pire cas. On réitère ensuite le traitement (au plus $3m$ étapes).

Complexité de l'étape de génération du maillage

Donc en tout, on effectue moins de $\mathcal{O}(9m^2) = \mathcal{O}(m^2)$ raffinements.

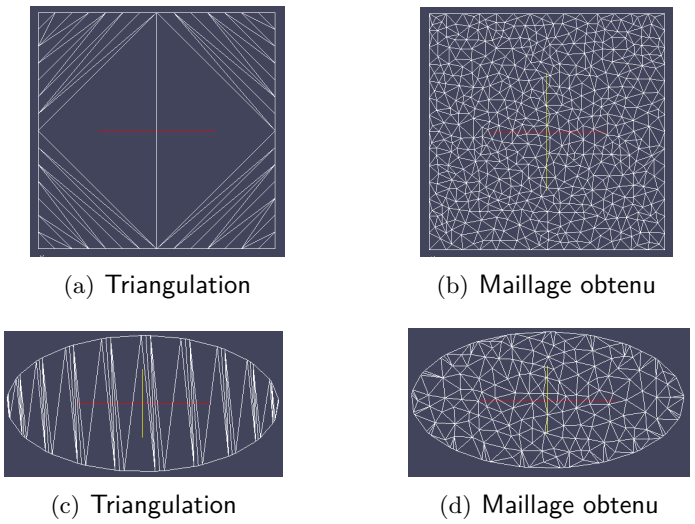
LISSAGE Cette procédure prend en entrée les points du maillage $\mathcal{M}$, et consiste à repositionner chaque $p \in \mathcal{M}$ en fonction de son voisinage $\mathcal{N}_p$.
Soit n le nombre de sommets de $\mathcal{M}$.
En pire cas, le nombre de voisins de chaque point $p \in \mathcal{M}$ est borné par n pour tout maillage $\mathcal{M}$, donc $|\mathcal{N}_p| \leq n$ pour tout p. La modification des coordonnées d'un point s'effectue en $\mathcal{O}(1)$, et le traitement du voisinage de chaque point se fait en tout $\mathcal{O}(n^2)$ en pire cas.
Par ailleurs on itère la procédure k fois, k étant une constante fixée.

Complexité de l'étape de lissage

Le lissage s'effectue donc en $\mathcal{O}(k \cdot n^2) = \mathcal{O}(n^2)$ en pire cas

3.5 Résultats

L'algorithme a été implémenté en `C++`, et on utilise GMDS pour la représentation du maillage, ainsi que la bibliothèque standard (STL) pour les structures de données de base (listes, files etc.).
A l'heure où ces lignes sont écrites, la phase de lissage n'a pas encore pu être implémentée.
La figure 3.5 donne un aperçu de maillages obtenus à partir d'une discrétisation régulière de la frontière de 2 domaines géométriques.



(a) Triangulation (b) Maillage obtenu

(c) Triangulation (d) Maillage obtenu

FIGURE 3.5 – Exemples de maillages générés par l'algorithme.

La figure 3.6 indique le temps d'exécution (en sec.) de l'algorithme sur un domaine Ω carré – cf. figure 3.5(a) – pour les phases de triangulation et de génération du maillage.

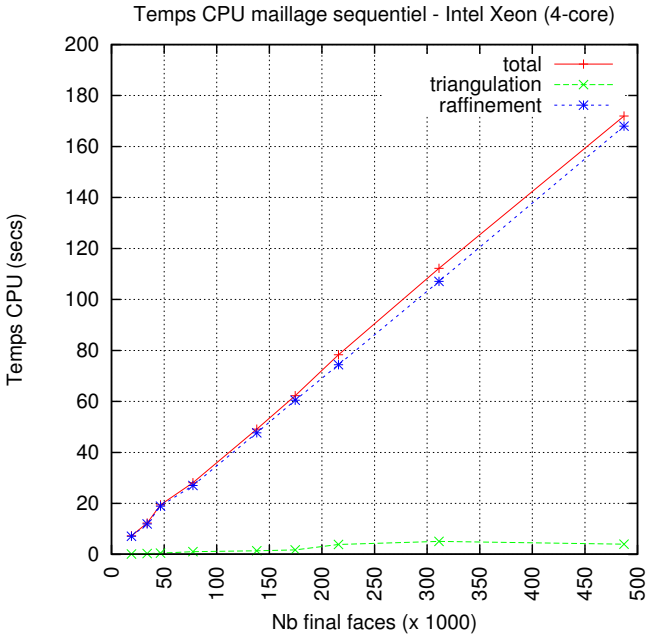


FIGURE 3.6 – Durées phases de l'algorithme en fonction de la taille du maillage

Sur la figure 3.6, on voit que la durée de la phase de génération du maillage évolue de manière linéaire en fonction du nombre de faces du maillage. Ce résultat concorde avec la complexité théorique évaluée

(en $\mathcal{O}(n^2)$ en pire cas) – cf. section 3.4.
Par ailleurs, on remarque que cette durée représente la quasi-totalité du temps CPU global.
Ceci peut s'expliquer par :

- une taille cible $\bar{\rho}$ fixé à 10% de la longueur d'un côté, ce qui induit un nombre important de faces à raffiner.
- l'absence de points internes au moment de la triangulation, ce qui induit la génération de faces très étirées, avec une longueur max d'arêtes ρ beaucoup plus importante par rapport à la taille cible $\bar{\rho}$ – voir figure 3.5(a).

La figure 3.7 décrit la répartition du temps par opération (localisation, subdivision, légalisation).

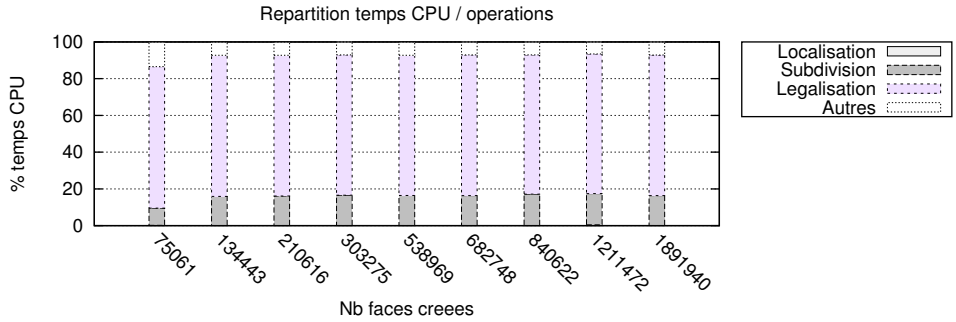


Figure 3.7 – Répartition temps CPU par opérations

Le temps dédié à la légalisation est plus important que pour les autres étapes, ce qui est normal compte-tenu :

- du nombre de faces à raffiner (faces de $\mathcal{T}_S$ très étirées, et degré de raffinement $\bar{\rho} \ll \frac{1}{n}\sum_{i=1}^{n}\rho_{\Delta_i}$)
- de la propagation de la légalisation, qui s'effectue en $\mathcal{O}(n \log n)$ en moyenne, n étant le nombre de faces créées.

La table 3.1 contient les différentes mesures qui ont été utilisées pour générer les courbes.

$\lvert\mathcal{S}_0\rvert$	$\lvert\mathcal{S}_f\rvert$	nb total f.	nb final f.	temps total	durée TR	durée RF	local.	subd.	leg.
236	9 580	75 061	18 904	7 318 888	184 611	7 130 930	9 186	1 156 643	5 639 789
316	17 147	134 443	33 958	12 299 893	312 604	11 983 642	12 403	1 942 758	9 449 707
396	26 974	210 616	46 569	19 415 144	492 735	18 918 671	17 895	3 112 421	14 875 258
476	38 968	303 275	77 440	2 811 815	1 100 885	27 013 913	23 049	4 632 086	21 494 208
636	69 331	538 969	138 006	4 915 114	1 437 323	47 708 806	30 475	8 044 190	37 510 230
716	87 691	682 748	174 646	62 237 462	1 782 844	60 450 789	34 873	10 152 160	47 690 675
796	108 298	840 622	215 780	78 298 266	3 871 776	74 422 618	42 698	13 383 701	59 389 648
956	156 121	1 211 472	311 266	112 217 684	5 113 395	107 099 846	52 238	19 077 499	85 284 537
1 196	244 193	1 891 940	487 170	171 974 611	3 996 063	167 974 124	67 179	28 052 356	131 618 297

Table 3.1 – Mesures temps d'exécution (10^{-6} secondes) et nombre d'éléments du maillage

Chapitre 4

Mise en œuvre de la solution parallèle

Ce chapitre présente les 2 heuristiques parallèles, construites sur la base de l'algorithme précédent.
Dans la section 4.1, on aborde une approche concurrente, dans laquelle toutes les structures utilisées seront partagées entre les PE qui devront coopérer et se synchroniser pour la construction du maillage. En particulier, on présente la stratégie mis en œuvre pour la gestion de la concurrence entre PE, ce qui constitue un point délicat de cette approche.
Dans la section 4.2, on aborde une approche distribuée, dans laquelle le maillage sera partitionné et réparti entre les PE qui traiteront localement chaque étape (triangulation, génération et lissage). On présente en particulier la gestion des interfaces entre les partitions, ce qui constitue un point sensible de cette seconde approche.

4.1 Heuristique concurrente

L'idée de cette approche est de décomposer l'algorithme séquentiel précédent en un ensemble taches élémentaires répartis entre n threads [1] en charge de générer le maillage.
A ce titre, ces n PE vont partager le conteneur de maillage, ainsi que toutes les structures de données nécessaires à chaque phase de l'algorithme (DAG, conteneur de faces de mauvaise qualité).
Pour la phase de triangulation, on s'inspire de [13] et l'idée sera de partitionner et de répartir les points du domaine Ω entre les n PE. Pour la phase de génération du maillage, l'idée est de constituer une file de taches $\mathcal{Q}$ à effectuer (raffinements de faces) et les PE devront récupérer et traiter chaque tache, et ce tant que cette file n'est pas vide. Pour la phase de lissage, l'idée est de répartir la liste de points du maillage courant entre les PE.

Cette approche soulève une problématique : celle de la cohérence de la topologie du maillage. En effet, la compétition des PE peut induire des situations incohérentes telles que :

- ✏ localisation incohérente dû à l'obsolescence des informations du DAG. Cela se produit quand un PE récupère un face Δ qui contient un point p, pendant qu'un autre PE était en train de modifier Δ au cours d'une subdivision (ou légalisation). Cela entraine donc un état incohérent du maillage, à cause des chevauchements de faces.
- ✏ incohérence des données topologiques dûe à une mauvaise mise à jour du voisinage d'une face. En effet, un PE pourrait être en train de mettre à jour les relations d'adjacences des voisins $(\Delta_1, \Delta_2, \Delta_3)$ d'une face Δ, tandis qu'un autre était en train de modifier l'un d'entre eux.
- ✏ raffinement simultané d'une même face par 2 PE, dû à une consultation concurrente de la file $\mathcal{Q}$ des faces de mauvaise qualité ; ou encore le raffinement d'une face inexistante dû à une une mise à jour incorrecte de la file. En effet, la légalisation d'une face Δ peut potentiellement impacter une face Δ' déjà incluse dans $\mathcal{Q}$ (en supprimant Δ et Δ', et en créant Δ_1 et Δ_2).

1. Ou PE

4.1.1 Granularité et décomposition fonctionnelle

On se base sur un parallélisme de taches à granularité moyenne. Ce niveau de granularité permet une gestion explicite des taches et des instructions de contrôle nécessaires à la synchronisation des PE (verrous, barrières, notifications). En effet, la dépendance intrinsèque de données (connectivité des faces et nœuds), et l'entrelacement des opérations (localisation, subdivision, légalisation) au sein de chaque étape, impliquent une gestion explicite de la concurrence entre PE.

Pour la répartition des taches, on se base sur le principe de « task farming » [46]. De manière simple, chaque étape est découpée en tâches individuelles, et les résultats partiels sont ensuite regroupés, évalués et redistribués entre les PE pour l'étape suivante. Un PE particulier (master) se charge de la génération des autres PE (workers), ainsi que de l'ordonnancement des phases, et la synchronisation entre les workers.

1. Dans la phase de triangulation $\mathcal{T}$, le PE master partitionne la liste de points $\mathcal{S}_0$ en n_w partitions égales, n_w étant le nombre de PE workers.
2. Dans la phase de génération du maillage, il constitue la file $\mathcal{Q}$ des faces de mauvaise qualité, sur la base d'une métrique spécifiée. Cette file est ensuite partagée par les workers.
3. Dans la phase de lissage, le master récupère la liste $\mathcal{S}_1$ des points de $\mathcal{M}$, et crée de nouveau n_w partitions, qu'il affectera aux n_w workers, qui vont itérer leur traitement x fois.

A chaque fin de phase correspond une barrière de synchronisation (cf. page 11), afin que le master puisse initialiser la prochaine étape. La figure 4.1 donne une vue globale de cette décomposition à l'aide d'un réseau de Pétri.

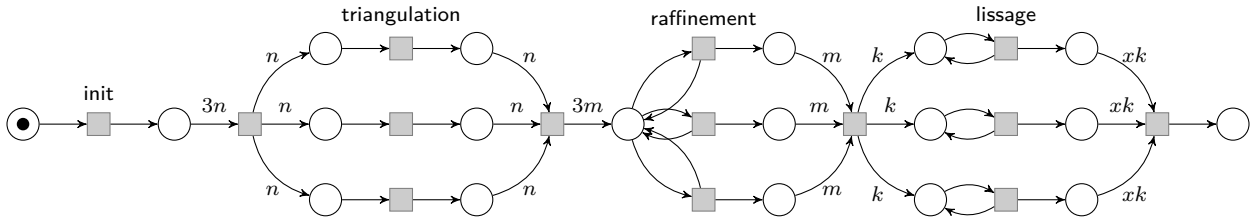


Figure 4.1 – Parallélisation des phases de l'heuristique, avec 1 master et $n_w = 3$ workers
$|\mathcal{S}| = 3n$: nombre de points initiaux issus de la discrétisation de la frontière de Ω
$|\mathcal{Q}| = 3m$: nombre de faces issues de la triangulation
$|\mathcal{S}| = 3k$: nombre de points après la phase de raffinement
x : nombre d'itérations pour la phase de lissage

Cette décomposition garantit l'équilibre de charges entre les PE :

1. Dans la phase de triangulation, le nombre de taches de chaque PE équivaut au nombre de points à insérer.
2. Dans la phase de génération du maillage, le fait de partitionner le conteneur $\mathcal{Q}$ en n_w conteneurs ne garantit pas que les n_w auront raffiné le même nombre de faces. Comme illustré sur la figure 4.1, le fait de récupérer et traiter unitairement les éléments de $\mathcal{Q}$ garantit que les PE auront effectué le même nombre de raffinements, à condition que l'accès à $\mathcal{Q}$ soit équitable.
3. Dans la phase de lissage, le nombre de taches de chaque PE est relatif au nombre de points qu'il aura à repositionner. Ainsi partitionner et répartir la liste de points $\mathcal{S}$ garantit l'équilibre de charges de cette phase.

4.1.2 Gestion de la concurrence entre PE

Pour chaque phase, la principale difficulté sera donc de trouver une stratégie de synchronisation qui garantit une cohérence de l'état du maillage à tout moment (gestion des accès en lecture/écriture au conteneur de maillage, au DAG, et à la file des faces de mauvaises qualité) d'une part, et qui soit moins

lourde (minimiser les sections critiques de manière à ce que l'overhead lié à la synchronisation soit négligeable par rapport à la durée totale) d'autre part.

4.1.2.1 SYNCHRONISATION DES PHASES

TRIANGULATION Pour cette phase (uniquement), on s'est intéressé à l'approche décrite dans [13] et [17] pour la triangulation parallèle en mémoire partagée, basée sur l'utilisation d'un DAG.
Le point essentiel de cette approche consiste à gérer l'accès à chaque élément du DAG, plutôt que de le verrouiller dans son intégralité. Pour ce faire, les auteurs distinguent 3 stratégies de synchronisation :

- ✏ Approche « batch » : ici plusieurs PE s'occupent de la phase de localisation, et un seul thread se charge de la phase de subdivision et de légalisation
- ✏ Approche pessimiste : ici, tous les PE effectuent les mêmes traitements, néanmoins les phases de subdivision et de légalisation ne doivent s'effectuer qu'au sein d'une section critique, afin d'éviter les accès concurrents au DAG
- ✏ Approche optimiste : ici, tous les PE effectuent simultanément les 3 phases de la triangulation, néanmoins chacun doit verrouiller la face qu'il souhaite modifier à un instant t.

Ici, nous adoptons l'approche dite « optimiste » pour notre heuristique, en y apportant quelques modifications toutefois – voir 4.1.3.

GÉNÉRATION DU MAILLAGE Partant de la même logique consistant à minimiser les régions critiques au sein de l'heuristique, l'idée sera de permettre aux PE d'accéder et de modifier simultanément le conteneur $\mathcal{Q}$ de faces de mauvaise qualité, tout en veillant à ce qu'il reste cohérent à tout moment.

LISSAGE Cette phase ne nécessite pas de stratégie de synchronisation particulière, et est hautement parallélisable. A l'itération k, le PE courant calcule la nouvelle position d'un point p sur la base de celles de son voisinage $\mathcal{N}_p$, qui peuvent être repositionnés par d'autres PE entre-temps. Néanmoins, à l'itération $k+1$, le même PE aura recalculé la nouvelle position de p, en tenant compte des nouvelles positions de $\mathcal{N}_p$. L'algorithme converge car les variations de position de chaque point tendent à diminuer au cours des itérations, et la position d'un point p finit par se stabiliser à une itération k_F.

4.1.2.2 INTERBLOCAGES

Pour chaque face Δ, on lui affecte un verrou v_Δ. En outre, on utilise 2 verrous globaux : v_C pour le conteneur de maillage, et v_{DAG} pour la récupération de l'instance du DAG.

Pour éviter les interblocages, on adopte une approche basée sur la prévention et l'évitement (cf. page 12). Pour cela, on utilise des verrous non-bloquants au niveau des faces, et on contraint chaque PE à relâcher ses verrous le plus tôt possible. Ainsi, un PE qui tente de verrouiller une face Δ ne se bloque pas si v_Δ n'a pas encore été relâché : il abandonne la tâche courante (en la remettant dans la file [2]), et prend une autre. Par ailleurs, si le PE doit verrouiller une liste de faces $\mathcal{F} = (\Delta_i)_{i=1\ldots n}$, mais qu'il n'a pas acquis tous les verrous v_{Δ_i}, il devra relâcher ceux qu'il a acquis entre-temps.

4.1.2.3 COHÉRENCE DE LA TOPOLOGIE

Durant la triangulation, à chaque fois qu'on accède à une feuille n_Δ du DAG, il faut acquérir v_Δ. Cela permet d'éviter toute modification intempestive de Δ (rajout de nœuds fils à n_Δ, suppression de Δ) – voir figure 4.2.

De même, avant chaque subdivision/légalisation d'une face Δ, il faut verrouiller Δ et son voisinage direct $\mathcal{N}_\Delta$, afin d'éviter les modifications multiples d'une part (chevauchement) et maintenir à tout moment la cohérence des connectivités d'autre part – voir figure 4.3.

2. liste des sommets à insérer, file des faces à raffiner, ou liste des points à repositionner

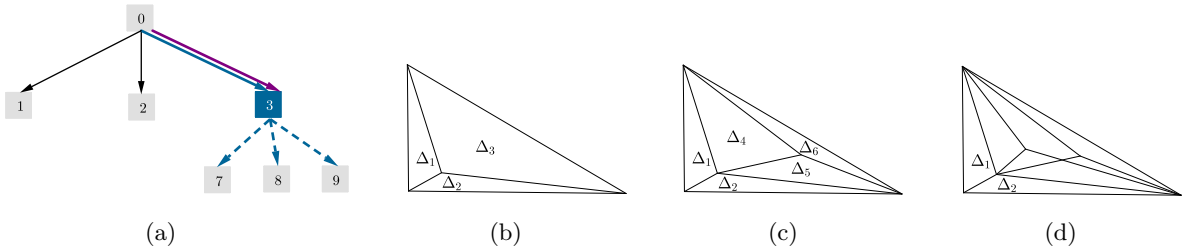


FIGURE 4.2 – Inconsistance de la triangulation due à l'incohérence des informations recueillies du DAG. Ici, w_1 (à l'étape de localisation) retourne la feuille n_8 car $p_1 \in \Delta_8$. Entre-temps, w_2 (à l'étape de subdivision) finit de créer les 3 faces, et rajoute 3 nœuds fils à n_8. Enfin w_1 (à l'étape de subdivision) partitionne de nouveau Δ_8

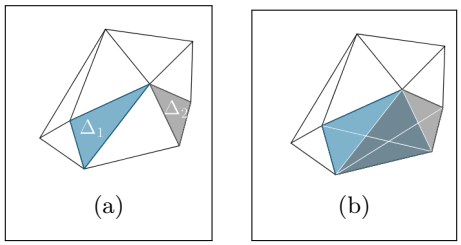


FIGURE 4.3 – Inconsistance du maillage du à la modification multiple d'une même face par 2 PE.

La figure 4.4 décrit le problème lié à une mauvaise mise à jour du voisinage d'une face Δ_3.

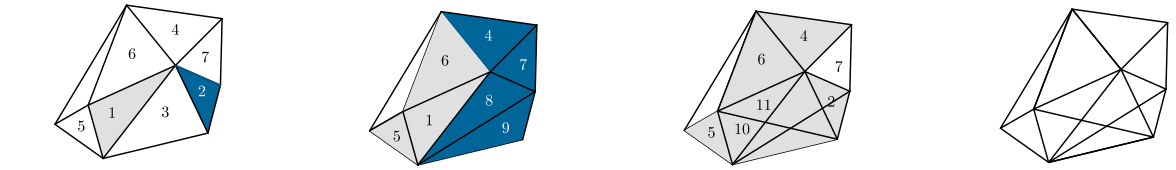


FIGURE 4.4 – Conséquence d'une mauvaise mise à jour du voisinage d'une face Δ_3 Ici, w_1 et w_2 tentent de légaliser 2 faces Δ_1 et Δ_2 disjointes, et verrouillent leurs voisinages respectifs $\mathcal{N}_1$ et $\mathcal{N}_2$. w_2 réussit à verrouiller Δ_3 en premier et bascule l'arete $(u, v) : \Delta_2 \cap \Delta_3$, et renseigne les connectivités des nouvelles faces Δ_8 et Δ_9. Ensuite, il relâche Δ_3 et Δ_2, qui sont immédiatement verrouillées par w_1. Ayant terminé de verrouillé sa liste de faces, w_1 bascule l'arête $(i, j) : \Delta_1 \cap \Delta_3$ et crée un chevauchement.

Ce problème est lié au fait qu'on sauvegarde localement les connectivités ($\mathsf{F} \rightarrow \mathsf{F}$) d'une part, et le fait que la procédure d'acquisition des verrous de faces n'est pas indivisible d'autre part.
Une solution naïve serait de verrouiller l'acquisition des verrous et de demander au conteneur si la face Δ_i existe encore avant sa modification. Toutefois cela implique de verrouiller le conteneur (afin qu'il ne soit modifié entre-temps), qui devient un goulot d'étranglement.
Pour ce faire, on préfère re-tester l'adjacence effective de chaque $\Delta_i \in \mathcal{N}_\Delta$ une fois qu'on aura verrouillé $\mathcal{N}_\Delta$, avant de poursuivre le traitement. Si tel n'est pas le cas, on abandonne la tache courante.

4.1.3 Triangulation

Ici, on se base sur la méthode dite « optimiste » décrite dans [17].
Néanmoins, notre approche se distingue sur 2 points :

- ✏ le mode de verrouillage : uniquement les feuilles du DAG (et non les noeuds parents des feuilles).
- ✏ la gestion des deadlocks : les auteurs s'appuient sur une heuristique de détection d'interblocage, alors que notre approche est basée sur la prévention (ordonnancement des accès, verrous non-bloquants sur les faces) et l'évitement (relâchement des verrous précédemment acquis par le PE s'il n'a pas réussi à verrouiller sa liste des faces à traiter).

En effet, la mise en place d'une procédure de détection n'est pas triviale, et l'overhead lié au déblocage des PE limite la scalabilité d'une telle approche.

4.1.3.1 PROCÉDURE GLOBALE

Dans un premier temps, le PE master initialise le triangle conteneur Δ_0, ainsi que la racine n_R du DAG. Ensuite, il divise la liste de points $\mathcal{S}$, en k partitions égales – voir algorithme 6. Après il initialise k threads correspondant aux k workers, en leur fournissant chacun une partition de points.
Chaque worker w_i traite ensuite sa liste locale Γ_i de points. A chaque $p \in \Gamma_i$, il localise la face Δ qui contient p (cf. algorithme 7), et essaie de verrouiller Δ ainsi que son voisinage $\mathcal{N}_\Delta$. S'il y parvient, alors il subdivise Δ en mettant à jour $\mathcal{N}$, sinon il abandonne le traitement et remet p à la fin de Γ. Dans tous les cas, il déverrouille Δ, afin de permettre aux autres workers de la verrouiller si besoin.

Algorithm 6 Partitionnement de la liste initiale de points, et triangulation par les PE workers

procedure WORKER_TRIANGULATE(n_R, Γ, $\mathcal{T}_S$, n)
n_R : racine du DAG
Γ : partition de points à insérer dans $\mathcal{T}_S$
$\mathcal{T}_S$: triangulation en cours de construction
n : compteur global de workers ayant terminé cette phase

```
Σ ← ∅                                              ▷ liste des voisins verrouillés
while Γ ≠ ∅ do
    p ← Γ[0]
    Γ ← Γ − {p}
    Δ ← WORKER_FIND(p, n_R)                        ▷ Récupération de la face Δ qui contient p via le DAG
    ---
    if P(v_Δ) then                                 ▷ Si on a réussi à verrouiller Δ
        for Δ_i ∈ N_Δ do
            if P(v_Δi) then                        ▷ On essaie de verrouiller chaque voisin de Δ
                Σ ← Σ ∪ v_Δi
        end for
        if |Σ| = 3 then                            ▷ Si on a réussi à verrouiller les 3 voisins
            INSERT(p, Δ, n_R, T_S, TRUE)           ▷ on insère p dans Δ
        else
            Γ ← Γ ∪ {p}                            ▷ Sinon on remet p à la fin de la file
        end if
        V(Σ)                                       ▷ Relâcher les verrous des voisins acquis dans tous les cas
        Σ ← ∅
    end if
    ---
end while
n ← n + 1                                          ▷ On incrémente le compteur statique pour signaler qu'on a terminé
end procedure
```

Algorithm 5 Partitionnement de la liste initiale de points et phase de triangulation par le PE master

procedure MASTER_PARTITION($\mathcal{S}$, k)
$\mathcal{S}$: liste des points à trianguler, k : nombre de workers

```
Γ ← ∅                                              ▷ Liste de partitions à retourner
j ← 0, P_0 ← ∅
for i de 1 à |S| do
    P_j ← P_j ∪ {S[i]}
    if i mod |S|/k = 0 then
        Γ ← Γ ∪ {P_j}
        j ← j + 1
        P_j ← ∅
    end if
end for
return Γ
end procedure
```

procedure MASTER_TRIANGULATE($\mathcal{S}$, k)
$\mathcal{S}$: liste des points à trianguler
k : nombre de PE workers

```
n_R ← ∅                                            ▷ racine du DAG
Γ[k] ← ∅                                           ▷ tableau de k partitions de points
n ← 0                                              ▷ compteur global de workers ayant terminé
---
Δ_0 ← COMPUTE-TRIANGLE(S)                          ▷ Initialisation du triangle conteneur
T_S ← {n_R}
n_R ← (Δ_0, ∅)                                     ▷ Initialisation du DAG
Γ ← MASTER_PARTITION(S, k)                         ▷ Partitionnement de S en k listes de tailles égales
for i de 1 à k do
    WORKER_TRIANGULATE(n_R, Γ[i], T_S, n)          ▷ Création et exécution des k workers
end for
---
while n ≤ k do WAIT                                ▷ Attente de terminaison des workers
---
for Δ ∈ T_S do                                     ▷ Suppression des faces externes à la frontière
    if Δ ∩ Δ_0 ≠ ∅ then
        T_S ← T_S − {Δ}
end for
end procedure
```

4.1.3.2 GESTION DU DAG

Ici, l'idée est de permettre un accès concurrent aux noeuds, tant en lecture qu'en écriture.

Durant la localisation, le PE parcourt le DAG, tout en vérifiant que le noeud courant n_Δ ne s'agit pas d'une feuille. En effet, les feuilles représentent les faces réellement présentes dans la triangulation en cours de construction. S'il s'agit d'une feuille, il essaie d'acquérir v_Δ, avant de poursuivre. En effet, si Δ est verrouillée, cela signifie qu'elle est soit en cours de consultation, ou en cours de modification (auquel cas elle ne sera plus une feuille). Ainsi, une fois qu'il aura acquis v_Δ, le PE devra re-tester si n_Δ est toujours une feuille, auquel cas il relâche v_Δ et renvoie Δ, sinon il poursuit sa recherche. Durant la subdivision ou la légalisation d'une face Δ, le PE localise et verrouille le nœud n_Δ en suivant la même procédure, crée et rattache les nouveaux nœuds n_{Δ_i}, puis relâche v_Δ – voir figure 4.5.

Cette méthode permet à 2 PE travaillant sur 2 faces non adjacentes (Δ_i, Δ_j) d'accéder et de modifier le DAG simultanément. En effet, les contentions n'interviennent que si Δ_i et Δ_j sont voisines, ou s'il y a un lien de parenté entre leurs nœuds n_{Δ_i} et n_{Δ_j} – voir figure 4.5.

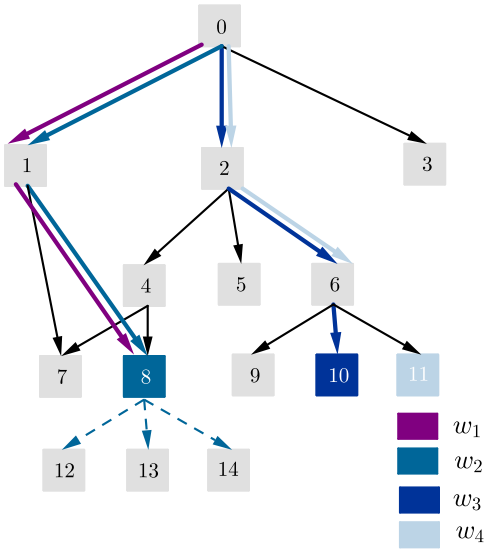


FIGURE 4.5 – Accès concurrent aux éléments du DAG par 4 workers durant la phase de triangulation. Ici, w_2 a verrouillé le noeud n_8, et crée 3 noeuds $[n_{12}, n_{13}, n_{14}]$, suite à une subdivision de Δ_8. En même temps, w_1 attend que w_2 déverrouille n_8 avant de poursuivre son parcours.

L'algorithme 7 décrit la procédure de localisation concurrente d'une face via le DAG.

Algorithm 7 Localisation d'une face par un PE worker

procedure WORKER_FIND(p, n_R)
p : le point à insérer dans la triangulation, n_R : la racine du DAG

$\Gamma \leftarrow \{n_R\}$ ▷ File des nœuds à parcourir
while $\Gamma \neq \emptyset$ **do**
$n_i : (\Delta, c[\,]) \leftarrow \Gamma[0]$ ▷ On récupère le nœud courant n_i

if $c[1] = \emptyset \wedge c[2] = \emptyset \wedge c[3] = \emptyset$ **then** ▷ Si n_i est une feuille (et n'a donc pas de nœud fils)
while $\neg P(v_\Delta)$ **do** WAIT ▷ Attente jusqu'à acquisition de v_Δ
if $c[1] = \emptyset \wedge c[2] = \emptyset \wedge c[3] = \emptyset$ **then** ▷ Si n_i est toujours une feuille
if $p \in \Delta$ **then**
$V(v_\Delta)$ ▷ On relâche v_Δ et on retourne la face Δ
return Δ
end if
$V(v_\Delta)$ ▷ On relâche v_Δ dans tous les cas
end if

for $n_k : (\Delta_k, k[\,]) \in c[\,]$ **do** ▷ Rajout des nœuds fils dans la file Γ des nœuds à parcourir
if $n_k \neq \emptyset \wedge p \in \Delta_k$ **then**
$\Gamma \leftarrow \Gamma \cup \{n_k\}$ ▷ On ne rajoute que si la face Δ_k contient p
end for
$\Gamma \leftarrow \Gamma - \{n_i\}$
end while
end procedure

4.1.4 Génération du maillage

Au départ, le PE master constitue la file $\mathcal{Q}$ en vérfiant pour chaque face $\Delta \in \mathcal{T}_S$ sa longueur maximum d'arêtes ρ_Δ, et son aplatissement θ_Δ. Ensuite, il notifie les k workers, afin que ces derniers puissent commencer la phase de génération.

Une fois notifié, chaque worker w_i essaie de récupérer (et retirer) une face Δ de $\mathcal{Q}$. S'il y parvient, alors w_i essaie de verrouiller Δ, récupère ses voisins $\mathcal{N}_\Delta$ puis essaie de les verrouiller un à un. S'il réussit à verrouiller intégralement $[\Delta, \mathcal{N}_\Delta]$, alors il procède au raffinement de Δ en mettant à jour $\mathcal{N}_\Delta$ (cf. sous-section 3.3.2.1), sinon il remet Δ dans la file et essaie de prendre une autre face.

Algorithm 8 Phase de génération du maillage

procedure MASTER_MESH($\mathcal{M}$, k, $\bar{\rho}$, $\bar{\theta}$)
$\mathcal{M}$: le maillage issu de la triangulation $\mathcal{T}$
k : nombre de PE workers
$\bar{\rho}$: longueur max d'arête
$\bar{\theta}$: aplatissement max d'une face

```
n ← 0                                          ▷ compteur global de workers ayant terminé cette phase
Q ← ∅
for Δ ∈ M do                                   ▷ Constitution de la file Q des faces de mauvaise qualité
    if (max_{u⃗∈Δ} ||u⃗|| ≥ ρ̄) ∨ (max_{θ∈Δ_c} ∠(θ) ≥ θ̄) then
        Q ← Q ∪ {Δ}
    end if
end for

for i de 1 à k do                              ▷ Relance des k workers
    WORKER_MESH(M, Q, ρ̄, θ̄, n)

while n ≤ k do WAIT                            ▷ Attente de terminaison des workers
```

end procedure

procedure WORKER_MESH($\mathcal{M}$, $\mathcal{Q}$, $\bar{\rho}$, $\bar{\theta}$, , n)
$\mathcal{Q}$: la file des faces de mauvaise qualité
n : compteur global de workers ayant terminé cette phase

```
while Q ≠ ∅ do                                 ▷ Parcours de la file (partagée entre les PE)
    if Δ ← Q[0] ∧ Δ ∈ M then                   ▷ Si on a pu récupérer une face, et qu'elle existe encore
        Q ← Q − {Δ}

        Σ ← ∅                                  ▷ Liste des voisins verrouillés
        if P(v_Δ) then                         ▷ Si on a réussi à verrouiller Δ
            for Δ_i ∈ N_Δ do
                if P(v_{Δ_i}) then             ▷ On essaie de verrouiller chaque voisin de Δ
                    Σ ← Σ ∪ v_{Δ_i}
            if |Σ| = 3 then                    ▷ si on a réussi à verrouiller les 3 voisins
                REFINE(M, Δ, Q, ρ̄, θ̄)          ▷ on raffine Δ dans ce cas
            else
                Q ← Q ∪ {Δ}                    ▷ sinon on remet Δ à la fin de Q
            end if
            V(Σ)                               ▷ On relâche les verrous des voisins acquis dans tous les cas
        end if

    end if
end while
n ← n + 1                                      ▷ On incrémente le compteur statique pour signaler qu'on a terminé
```

end procedure

Pour gérer les accès concurrents, les requêtes de récupération ou d'insertion d'éléments peuvent être sérialisées en utilisant 2 buffers : un en début de file (pour la récupération), et un autre en fin de file (pour l'insertion). Ainsi les requêtes seront ordonnées et traitées par ordre d'arrivée.

4.2 Heuristique distribuée

Pour cette heuristique, on s'inspire de [18] qui fournit une approche distribuée pour la génération de maillages tétraédriques, avec un traitement en amont des éléments frontaliers qui minimise les synchronisations entre les PE.
Ici, on se base sur un partitionnement géométrique du domaine Ω. A partir d'une triangulation $\mathcal{T}$ (ou d'un maillage initial $\mathcal{M}$), l'idée sera de décomposer $\mathcal{T}$ (ou $\mathcal{M}$) en k partitions de tailles géométriques sensiblement égales, de manière à ce que chaque partition $\mathcal{P}_i$ soit traitée individuellement par un PE.

Cette approche soulève 2 problématiques : la cohérence des éléments de la frontière entre 2 partitions (interface) d'une part, et la répartition de charges entre les PE d'autre part. En effet :

1. Toute opération locale à une partition ne doit altérer la cohérence topologique du maillage (connectivités, non-chevauchement de faces). Ainsi, toute modification (subdivision, légalisation, mise à jour des connectivités) d'un élément d'une interface $\mathcal{I}$ commune à 2 partitions $\mathcal{P}_i$ et $\mathcal{P}_j$ doit être répercutée sur ces 2 partitions, ce qui nécessite une synchronisation explicite entre les PE en charge de $\mathcal{P}_i$ et $\mathcal{P}_j$.
2. Son efficacité dépend de l'équilibre de charges entre les PE, qui dépend du partitionnement initial. Ainsi, cette décomposition doit tenir compte de la géométrie du domaine Ω d'une part (forme, répartition des points de Ω), et du nombre d'éléments dans chaque partition d'autre part.

4.2.1 Granularité et décomposition fonctionnelle

Ici, on se base sur un parallélisme de données à granularité forte. Ce niveau de granularité minimise la synchronisation nécessaire entre les PE, ce qui facilite la montée en charge en nombre de PE. En effet, le but recherché sera de décomposer le traitement de manière à ce que les taches des PE soient mutuellement indépendantes, ce qui minimise les communications. Cela permet également d'adapter l'heuristique afin qu'elle puisse être implémentée en mémoire partagée (pthread), ou en mémoire distribuée (MPI).

Tout comme dans la première approche, on s'appuie également sur le principe du task farming (1 PE master et k PE workers). Le PE master se charge de la génération des PE workers, qui devront le signaler à chaque fin de phase. Pour la décomposition des taches, l'heuristique s'inspire du paradigme D&C. La figure 4.6 donne une vue globale de cette décomposition à l'aide d'un réseau de Petri.

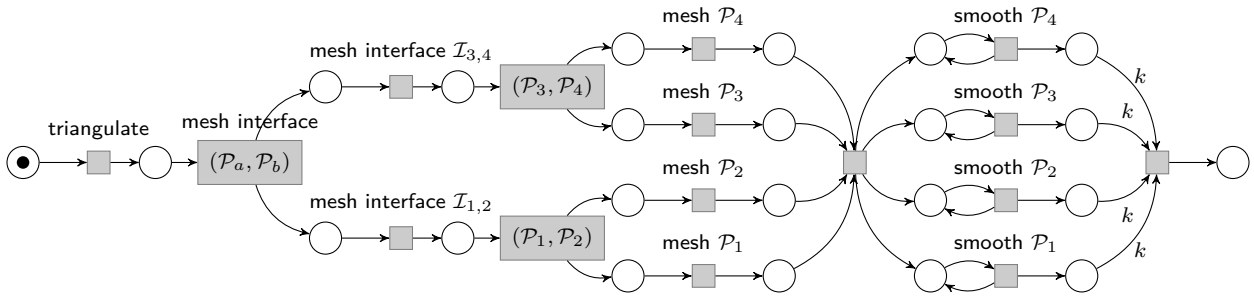


FIGURE 4.6 – Vue globale des phases de l'heuristique, avec 1 master et $n_w = 4$ workers

La phase de triangulation est effectuée en séquentiel par le PE master. Le temps dédié à cette phase étant négligeable par rapport à celui de la génération du maillage, son exécution séquentielle dégrade peu les performances de l'heuristique. Par ailleurs, cela permet également de s'affranchir de l'utilisation des verrous, qui dans notre cas, n'est requise que pour cette phase.

La phase de génération du maillage se décompose en 2 étapes : une étape de partitionnement du domaine, et une étape de raffinement. Initié par le PE master, ce partitionnement s'effectue de manière

récursive en fonction du nombre de PE workers cible. Ainsi pour n_w workers, on partitionne le domaine initial $(\log_2 n_w)$ fois. Une fois les partitions finales $(\mathcal{P}_i)_{i=1\cdots n_w}$ obtenues, chaque worker w_i procède au raffinement des faces de leurs partitions respectives $\mathcal{P}_i$.

Comme on est en mémoire partagée, la phase de lissage est identique à celle de la première heuristique : le master récupère la liste $\mathcal{S}$ des points de $\mathcal{M}$, et crée de nouveau n_w listes de points, qu'il affectera aux n_w workers, qui vont itérer leur traitement k fois.

4.2.2 Partitionnement du domaine

Il existe plusieurs approches pour le partitionnement du domaine, basées sur une décomposition d'un maillage initial $\mathcal{M}$. Le partitionnement optimal d'un maillage, ou plus exactement le partitionnement d'un graphe, n'est pas un problème trivial, et constitue un domaine de recherche à part entière [47, 48].

Ici, l'idée est juste de fournir une méthode de décomposition simple et géométrique, mais on peut la substituer par une autre heuristique de partitionnement telle que la méthode gloutonne [49], la méthode globale de Lin-Kerninghan [50], celle de Cuthill-McKee [51], ou encore la méthode spectrale [52, 53, 54].

Dans notre cas, nous nous limiterons à un partitionnement en grille ($p-$partition récursive) basé sur la géométrie du domaine à mailler. Comme énoncé précédemment, il s'effectue de manière récursive en fonction du nombre cible de partitions, qui équivaut au nombre de PE workers. Toutefois, il impose que ce nombre de workers n_w soit une puissance de 2.

PRINCIPE Dans un premier temps, le PE master calcule la boite englobante des points du domaine $\Omega : [A, B, C, D] = [(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max})]$.
A partir des coordonnées de $[A, B, C, D]$, il détermine ensuite la verticalité du rectangle englobant Ω en comparant les valeurs de $\delta_X = x_{max} - x_{min}$ et $\delta_Y = y_{max} - y_{min}$. En effet, si $\delta_X > \delta_Y$ alors on effectue un partitionnement vertical, sinon il s'agit d'un partitionnement horizontal – voir algorithme 9.

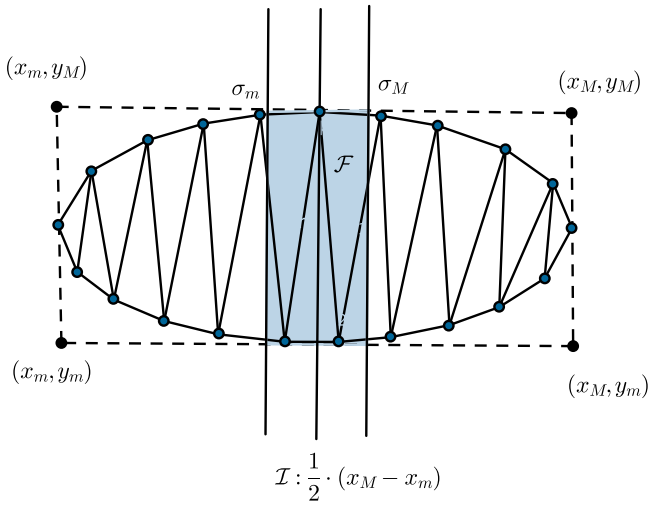


FIGURE 4.7 – Bissection du domaine Ω et création de la fenêtre $\mathcal{F}$

Une fois la direction de subdivision (l'interface $\mathcal{I}$ entre les 2 sous-maillages) déterminée, on calcule ensuite une fenêtre $\mathcal{F} : (\sigma_{min}, \sigma_{max})$ de part et d'autre de cette interface. $\mathcal{F}$ correspond à une zone identifiant les faces proches de $\mathcal{I}$ selon une métrique fixée – voir figure 4.7.

Algorithm 9 Création des partitions

procedure GET_INTERFACE($\mathcal{S}$, λ)
$\mathcal{S}$: ensemble de points issus de la discrétisation Ω
λ : largeur de l'interface

$\mathcal{I} \leftarrow 0$ ▷ Coordonnées de l'interface
$\mathcal{F} : (\sigma_m, \sigma_M)$ ▷ Coordonnées de la fenêtre
$(x_m, y_m) \leftarrow \min(\mathcal{S})$
$(x_M, y_M) \leftarrow \max(\mathcal{S})$
$(\delta_X, \delta_Y) \leftarrow (x_M - x_m, y_M - y_m)$

if $\delta_X > \delta_Y$ **then** ▷ On détermine le sens de partitionnement (horizontal/vertical)
  $\mathcal{I} \leftarrow \frac{1}{2}\delta_X + x_m$
else
  $\mathcal{I} \leftarrow \frac{1}{2}\delta_Y + y_m$
end if
$\sigma_m \leftarrow \mathcal{I} - \lambda$, $\sigma_M \leftarrow \mathcal{I} + \lambda$
return $[\mathcal{I}, \mathcal{F}]$ ▷ On retourne l'interface et la fenêtre associée
end procedure

On crée ensuite les 2 partitions courantes $(\mathcal{P}_1, \mathcal{P}_2)$ en parcourant chaque face Δ_i du maillage, et en comparant les coordonnées de leurs sommets par rapport à celles de la fenêtre $\mathcal{F} : (\sigma_{min}, \sigma_{max})$, avec $\sigma = x$ s'il s'agit d'une interface verticale et $\sigma = y$ sinon – voir figure 4.8.

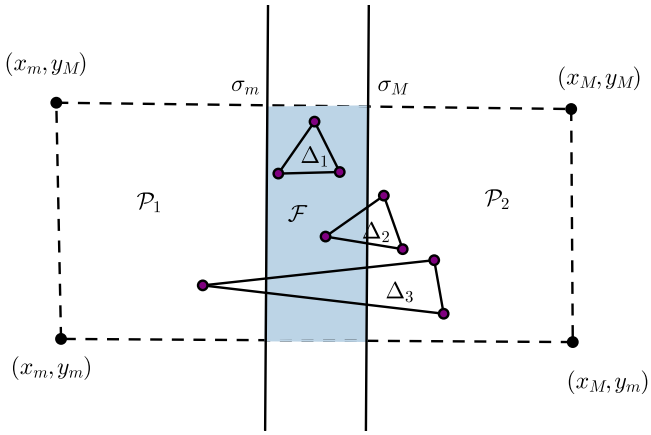


FIGURE 4.8 – Elements intersectant la fenêtre $\mathcal{F} : [\sigma_m, \sigma_M]$ de l'interface.
Ici, les 3 faces Δ_1, Δ_2 et Δ_3 appartiennent à la zone à raffiner avant la création des 2 partitions $\mathcal{P}_1$ et $\mathcal{P}_2$. Elles ne seront donc rajoutées ni à $\mathcal{P}_1$, ni à $\mathcal{P}_2$.

4.2.3 Gestion des éléments aux interfaces

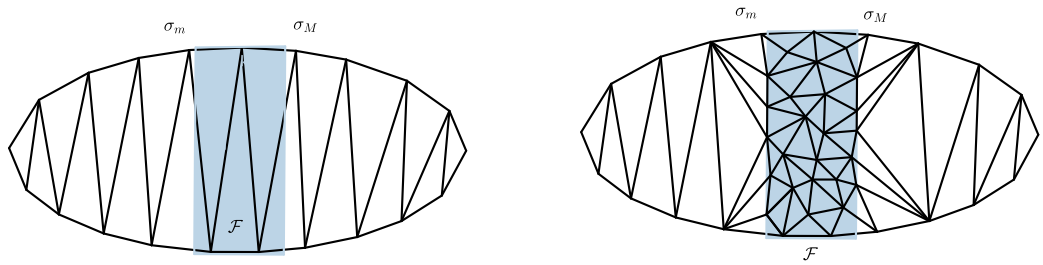
La gestion des éléments des interfaces constitue un point sensible de cette approche, indépendamment de la méthode de décomposition implémentée. En effet, la modification (subdivision, suppression, légalisation) d'une face $\Delta_i \in \mathcal{P}_1$ touchant l'interface $\mathcal{I} : \mathcal{P}_1 \cap \mathcal{P}_2$ impacte nécessairement son voisin $\Delta_j \in \mathcal{P}_2$, qui doit être mis à jour également (relation d'adjacence, suppression de Δ_2 dans le cas d'une bascule d'arêtes).

Pour gérer cela, on distingue usuellement 3 méthodes de synchronisation :

- ✏ méthode directe : à chaque fois qu'un PE w_i modifie une maille Δ de l'interface $\mathcal{I} : \mathcal{P}_i \cap \mathcal{P}_j$, celui-ci doit prévenir le PE en charge de $\mathcal{P}_j$, soit par la modification d'une variable partagée associée à Δ, soit par passage de messages. L'inconvénient majeur de cette approche provient du nombre important de synchronisations, qui dépend de la taille de l'interface $\mathcal{I}$.

- ✏ méthode de mailles fantômes : pour chaque interface $\mathcal{I} : \mathcal{P}_i \cap \mathcal{P}_j$, on recopie chaque $\Delta_i \in \mathcal{I} \cap \mathcal{P}_i$ en $\mathcal{P}_j$, et vice-versa. Lors d'une modification d'une maille fantôme $\Delta'_i \in \mathcal{P}_i$ (qui n'appartient donc pas réellement à $\mathcal{P}_i$), le PE courant w_i notifie celui en charge de $\mathcal{P}_j$. Ce dernier (w_j) met ensuite à jour la maille originale Δ_i et notifie w_i. Cette méthode permet d'agréger les communications entre PE, car ces dernières peuvent être effectuées après que tous les éléments de l'interface $\mathcal{I}$ soient traités [55].
- ✏ méthode a priori : ici, les éléments de chaque interface $\mathcal{I} : \mathcal{P}_i \cap \mathcal{P}_j$ sont traités en amont (avant le raffinement de $\mathcal{P}_i$ et $\mathcal{P}_j$), ce qui permet de s'affranchir des synchronisations puisque justement ils ne sont plus modifiés [18, 11]. Il s'agit de l'approche que nous adopterons pour notre heuristique.

Pour chaque interface $\mathcal{I}$, le but sera de raffiner directement les éléments proches de $\mathcal{I}$, avant la soumission des partitions aux PE workers.



(a) Identification des éléments intersectés par $\mathcal{F} : [\sigma_m, \sigma_M]$

(b) Raffinement de ces éléments

FIGURE 4.9 – Raffinement des éléments de l'interface

A partir de l'interface $\mathcal{I}$, on calcule la fenêtre $\mathcal{F}_I : (\sigma_{min}, \sigma_{max})$, et on identifie ensuite les faces qui intersectent cette fenêtre $\mathcal{F}_I$ – voir figure 4.8 et 4.9.
Afin de ne raffiner que les éléments de $\mathcal{F}$, il nous faut modifier la métrique, au moment de la constitution du conteneur $\mathcal{Q}$ de faces de mauvaise qualité. Ainsi, si la face Δ intersecte $\mathcal{F}$, et que la longueur maximale des arêtes $\rho_\Delta \geq \bar{\rho}$, alors on rajoute Δ dans $\mathcal{Q}$, sinon on passe à la face suivante – voir algorithme 10.

Algorithm 10 Mise à jour de la métrique

procedure METRIC_VALID(Δ, $\mathcal{F}$, VERTICAL)
$\Delta : (A, B, C)$: la face courante à vérifier
$\mathcal{F} : (\sigma_m, \sigma_M)$: la fenêtre correspondant à l'interface de subdivision
VERTICAL : indique le sens de subdivision

```
for (x_i, y_i) ∈ Δ do
    if VERTICAL then
        if (σ_m ≤ x_i ≤ σ_M) ∨ (x_i ≤ σ_m ∧ x_j ≥ σ_M, i ≠ j) then
            return (max_{u⃗∈Δ} ||u⃗|| ≥ ρ̄) ∨ (max_{θ∈Δ} ∠(θ) ≥ θ̄)
        else
            return TRUE
    else
        if (σ_m ≤ y_i ≤ σ_M) ∨ (y_i ≤ σ_m ∧ y_j ≥ σ_M, i ≠ j) then
            return (max_{u⃗∈Δ} ||u⃗|| ≥ ρ̄) ∨ (max_{θ∈Δ} ∠(θ) ≥ θ̄)
        else
            return TRUE
    end if
end for
end procedure
```

4.2.4 Procédure globale

Dans un premier temps, le PE master commence par calculer la boite englobante du domaine Ω, ainsi que l'interface de subdivision $\mathcal{I}$ et la fenetre correspondante $\mathcal{F}_I$.
Après, il procède à la triangulation des points issus de la frontière de Ω. Il construit ensuite le conteneur $\mathcal{Q}$ de faces de mauvaise qualité (cf. sous-section 4.2.3), et procède au raffinement de chaque face de $\mathcal{Q}$. A partir de la fenêtre $\mathcal{F}$, il initialise les partitions $(\mathcal{P}_1, \mathcal{P}_2)$ (cf. sous-section 4.2.2), crée les n_w PE workers, en leur fournissant une partition chacune.

Une fois qu'il a reçu sa partition $\mathcal{P}_i$, chaque PE worker w_i calcule l'interface $\mathcal{I}_i$ et la fenêtre associée $\mathcal{F}_I$. Ensuite, w_i procède au raffinement des éléments intersectant avec $\mathcal{F}$, et initialise la nouvelle paire de partition $(\mathcal{P}_{i,1}, \mathcal{P}_{i,2})$. Enfin, ce dernier activera les 2 prochains workers en leur fournissant $(\mathcal{P}_{i,1}, \mathcal{P}_{i,2})$.

Algorithm 11 Procédure globale

procedure MASTER_PROCESS($\mathcal{S}$, λ, n_w, k)
$\mathcal{S}$: ensemble de points issus à trianguler
λ : largeur de l'interface, n_w : nombre de workers, k : nombre d'itérations pour la phase de lissage

$n_{sub} \leftarrow \log n_w$ ▷ Nombre de subdivisions
$\mathcal{Q} \leftarrow \emptyset$ ▷ File des faces à raffiner
$\mathcal{F} : (\sigma_m, \sigma_M) \leftarrow (0, 0)$ ▷ Fenêtre correspondant à la zone à raffiner

---

$\mathcal{M} \leftarrow$ TRIANGULATE($\mathcal{S}$, λ) ▷ Triangulation initiale de $\mathcal{S}$
$[\mathcal{I}, \mathcal{F}] \leftarrow$ GET_INTERFACE($\mathcal{S}$, λ) ▷ Récupération de l'interface de subdivision
PROCESS_PARTITION($\mathcal{M}$, $\bar{\rho}$, $\bar{\theta}$, $\mathcal{F}$, $\mathcal{I}$, n_{sub})
while $n_{sub} > 0$ **do** WAIT ▷ Attente de terminaison des workers
for i de 1 à n_w **do**
WORKER_LAPLACIAN($\mathcal{M}$, $\mathcal{P}_i$, k)
end procedure

---

procedure WORKER_PROCESS($\mathcal{M}$, $\mathcal{P}_i$, n_{sub})
$\mathcal{P}_i$: partition courante, et n_{sub} : nombre de subdivisions restantes

if $n_{sub} > 0$ **then**
PROCESS_PARTITION($\mathcal{P}_i$, $\bar{\rho}$, $\bar{\theta}$, $\mathcal{F}$, $\mathcal{I}$, n_{sub})
else
$\mathcal{M} \leftarrow$ MESH($\mathcal{M}$, $\bar{\rho}$, $\bar{\theta}$, $\mathcal{Q}$)
end procedure

---

procedure PROCESS_PARTITION($\mathcal{P}$, $\bar{\rho}$, $\bar{\theta}$, $\mathcal{F}$, $\mathcal{I}$, n_{sub})
$\mathcal{P}$: partition courante, n_{sub} : nombre de subdivisions restantes
$\mathcal{F}$: zone de raffinement de l'interface $\mathcal{I}$, $\bar{\rho}$: longueur max d'arête, $\bar{\theta}$: aplatissement max d'une face

for $\Delta \in \mathcal{P}$ **do**
if $\neg$ METRIC_VALID(Δ, $\mathcal{F}$, $\mathcal{I}$) **then**
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\Delta\}$;
end for

$\mathcal{M} \leftarrow$ MESH($\mathcal{M}$, $\bar{\rho}$, $\bar{\theta}$, $\mathcal{Q}$) ▷ Raffinement des éléments intersectés par $\mathcal{F}$

---

if $n_{sub} > 0$ **then** ▷ Si le pas de subdivision n'est pas encore atteint
$(\mathcal{P}_1, \mathcal{P}_2) \leftarrow$ PARTITION($\mathcal{M}$, $\mathcal{F}$)
WORKER_PROCESS($\mathcal{M}$, $\mathcal{P}_1$, n_{sub})
WORKER_PROCESS($\mathcal{M}$, $\mathcal{P}_2$, n_{sub})
$n_{sub} \leftarrow n_{sub} - 1$
end if
end procedure

Chapitre 5

Résultats

L'algorithme a été implémenté en C++, et représente 4 754 lignes de code en tout.
Celui-ci utilise le conteneur GMDS, et la bibliothèque standard C++ (STL) pour les structures de données de base.

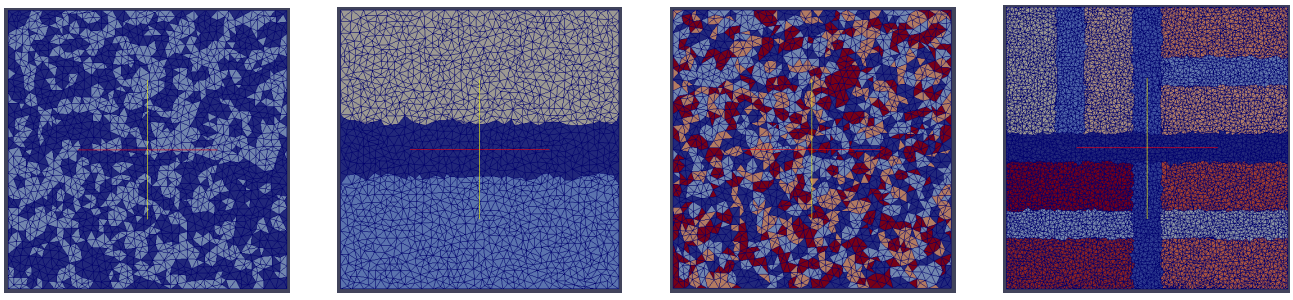
ENVIRONNEMENT DE TEST Les PE sont implémentés par des threads Unix natifs (via STL), et les tests sont exécutés sur 2 nœuds de calcul d'un cluster, constitués de :

- ✏ coprocesseur Intel KNC/Xeon-Phi, cadencé à 1Ghz avec 1 processeur/nœud. Il est doté de 60 cœurs avec 4 threads/cœur (240 threads max), avec un cache total de 20Mo et une mémoire interne totale de 8Go. Il s'agit d'une nouvelle génération de cartes accélératrices permettant l'exécution de programmes parallèles en mémoire partagée sur un nombre très élevé de PE, et nécessitant peu de données.
- ✏ processeur Intel Xeon, cadencé à 2.70Ghz avec 2 processeurs/nœud, avec chacun 12 coeurs (24 threads max), un cache total de 30Mo et une mémoire de 2,5Go/PE. Il est plus performant (par cœur) que le Xeon-Phi et moins limité en terme de mémoire, mais est limité en nombre de PE natifs pouvant être exécutés en même temps.

5.1 Génération du maillage

5.1.1 Cas-test

Pour mesurer les performances des 2 heuristiques (notées H_1 et H_2), on s'est basé sur le maillage d'un domaine carré vide Ω, avec 25 points sur chaque coté, et espacés chacun de 500 unités, comme sur la figure 5.1.



(a) H_1 avec $n_w = 2$ (b) H_2 avec $n_w = 2$ (c) H_1 avec $n_w = 4$ (d) H_2 avec $n_w = 8$

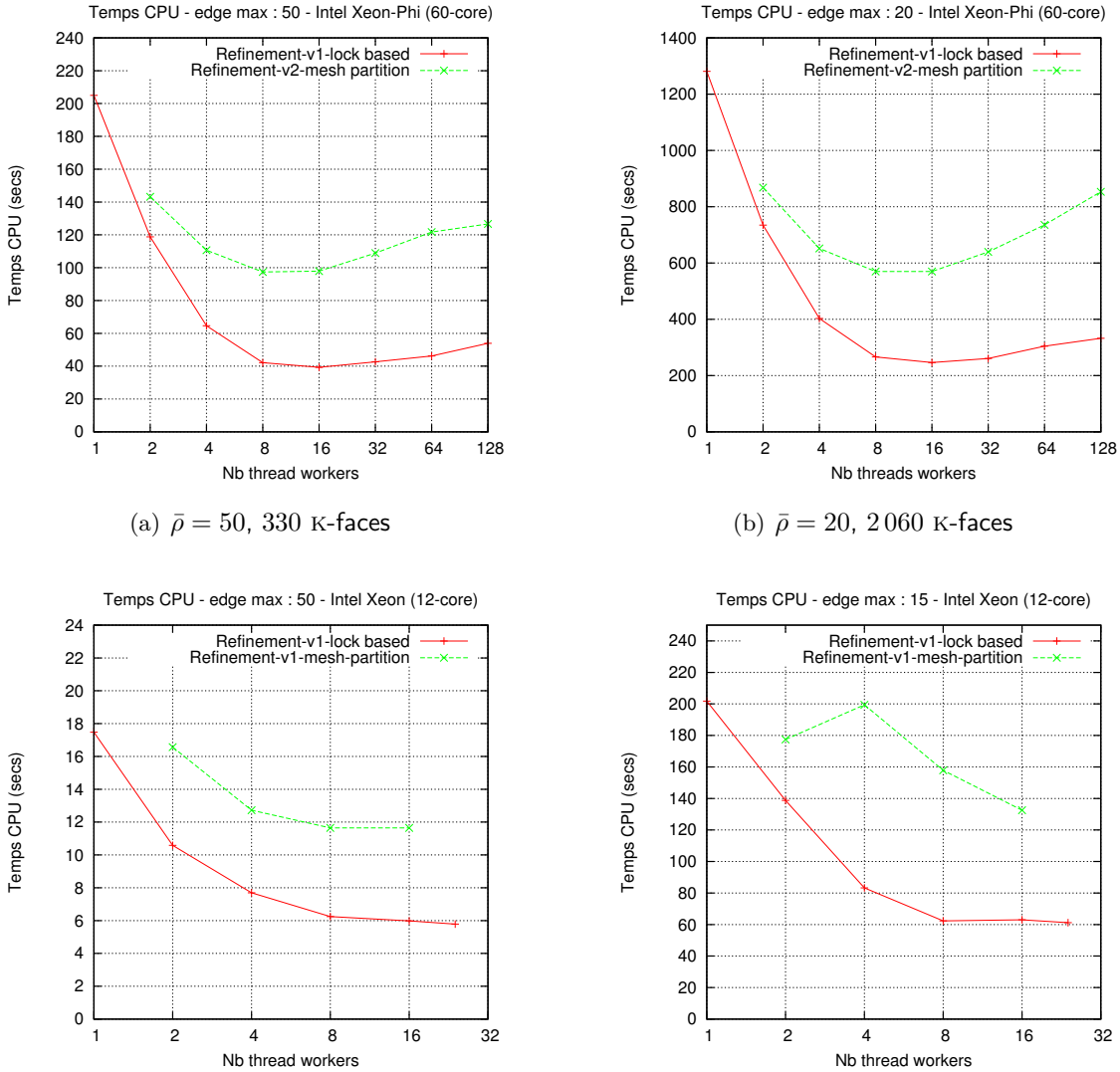
FIGURE 5.1 – Exemple de maillage généré par H_1 et H_2. La couleur d'une face Δ_j correspond à l'ID du PE worker w_i qui l'a créée.

Dans un premier temps, on compare le temps d'exécution des 2 heuristiques en fonction du nombre de PE workers, et ce sur les 2 plateformes – voir sous-section 5.1.2.

Ensuite, on calcule l'accélération $S(n_w)$ et l'efficacité $E(n_w)$ de H_1 et H_2 pour une taille de maillage fixe (défini par le degré de raffinement $\bar{\rho}$) – voir sous-section 5.2.1.
Enfin, on calcule $S(n_w)$ et $E(n_w)$ pour une taille de maillage proportionnelle au nombre de PE workers – voir sous-section 5.2.2.

5.1.2 Performances brutes

La figure 5.2 donne un comparatif des performances brutes de H_1 et H_2 pour le maillage de Ω sur les 2 architectures, avec 2 tailles cibles d'arêtes $\bar{\rho}_1$ et $\bar{\rho}_2$ fixées respectivement à 50 et 20.



(a) $\bar{\rho} = 50$, 330 K-faces

(b) $\bar{\rho} = 20$, 2 060 K-faces

(c) $\bar{\rho} = 50$, 330 K-faces

(d) $\bar{\rho} = 15$, 3 660 K-faces

FIGURE 5.2 – Comparaison des temps CPU des 2 heuristiques sur les 2 architectures

Sur la figure 5.2, on voit que H_1 est indiscutablement plus performante que H_2, à taille de maillages $|\mathcal{M}|$ et nombre de PE n_w égales. Cela peut s'expliquer par :

- La finesse de décomposition des taches (granularité) de H_1 qui minimise le temps d'attente des PE lors d'une synchronisation.
- L'étape de partitionnement/raffinement récursif des éléments proches de chaque interface de subdivision $\mathcal{I}_i$ pénalise H_2. A l'étape i, le temps dédié au partitionnement de $\mathcal{P}_i$ est proportionnel à la taille de l'interface $\mathcal{I}_i$ et au nombre de workers n_w, ce qui limite les performances de H_2.

Toutefois, pour un domaine Ω rectangulaire, ce résultat devrait s'améliorer, en raison d'une taille max d'interface ($|\mathcal{I}_i|$) plus petite par rapport au périmètre de Ω.

Notons aussi qu'on peut apercevoir qu'à partir de $n_w = 16$, les performances de H_1 et H_2 commencent à se dégrader (point d'inflexion). Il s'agit du nombre maximal de PE pour lequel l'overhead [1] lié à la synchronisation des PE reste inférieure au gain de temps apporté par l'augmentation du nombre de PE workers.

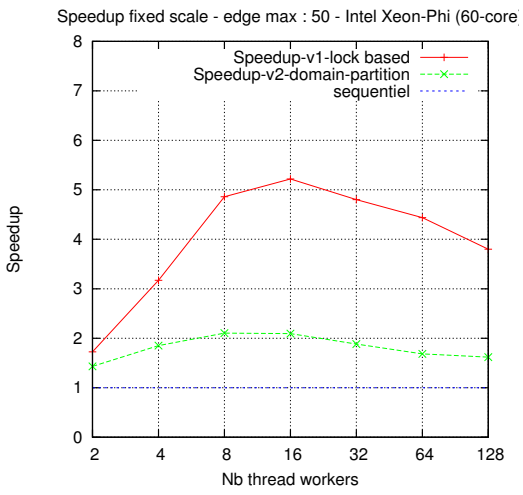
- ✏ pour H_1 cela est probablement lié au fait du nombre croissant de contentions (attente verrou) lors de la mise à jour du conteneur de maillage (création/suppression de faces).
- ✏ pour H_2 cela est probablement lié à la durée de partitionnement de Ω (7 étapes en tout pour $n_w = 128$) qui dépasserait la durée de raffinement final des faces de $\mathcal{M}$.

Par ailleurs, sur la figure 5.2(d), on peut apercevoir un pic pour $n_w = 4$ pour H_2. Ce pic reste pour le moment inexpliqué.

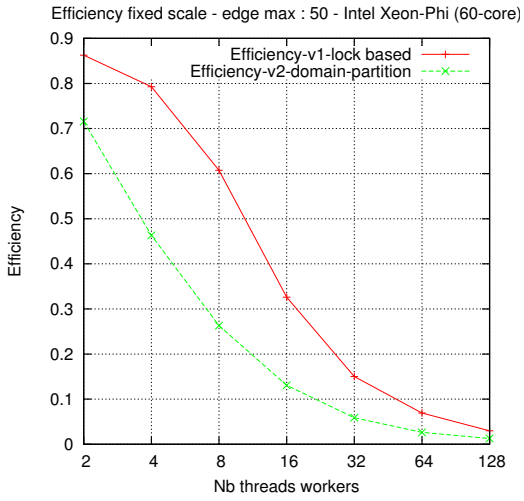
5.2 Scalabilité

5.2.1 Scalabilité forte

Ici, on cherche à évaluer et comparer la scalabilité forte des 2 heuristiques (cf. définition 10, page 12). Les figures 5.3 et 5.4 décrivent l'accélération et l'efficacité de H_1 et H_2, en fonction du nombre de PE workers, pour une taille de maillage fixe - cf. définition 7 et 8.



(a) Accélération pour $\bar{\rho} = 50$, $|\mathcal{M}| \approx 330$ K-faces

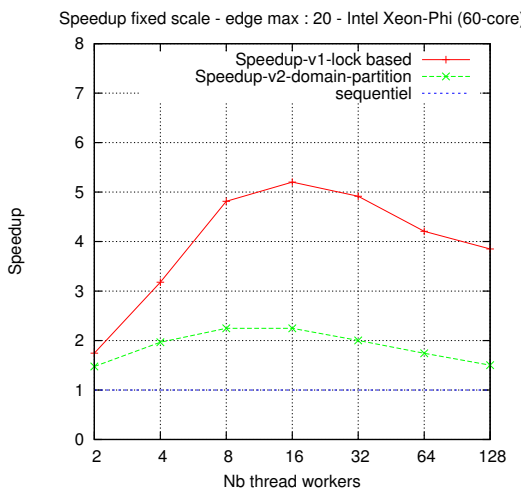


(b) Efficacité pour $\bar{\rho} = 50$, $|\mathcal{M}| \approx 330$ K-faces

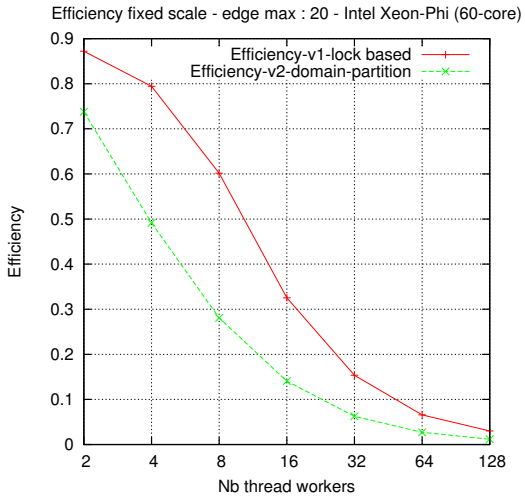
FIGURE 5.3 – Accélération et efficacité de H_1 et H_2 pour $\bar{\rho} = 50$ – Intel Xeon-Phi

Conformément aux résultats discutés dans 5.1.2, H_1 a une meilleure scalabilité que H_2, avec une accélération max $S_1(n_w) = 5.2$ contre $S_2(n_w) = 2.1$ pour H_2, toutes atteintes à $n_w = 16$.
Les résultats obtenus pour H_1 se rapprochent de ceux obtenus dans [14] (coarse + fine grain), avec une accélération crête de 6 (13 PE).
Par ailleurs, H_1 a une meilleure efficacité que H_2, qui peut s'expliquer par une bonne répartition des charges des PE – voir sous-section 4.1.1.
En terme d'efficacité, H_2 est moins performante en raison de la disproportion du nombre d'éléments à raffiner à chaque étape de partitionnement de Ω. En effet, au fur et à mesure que l'on partitionne

1. Surcoût en temps



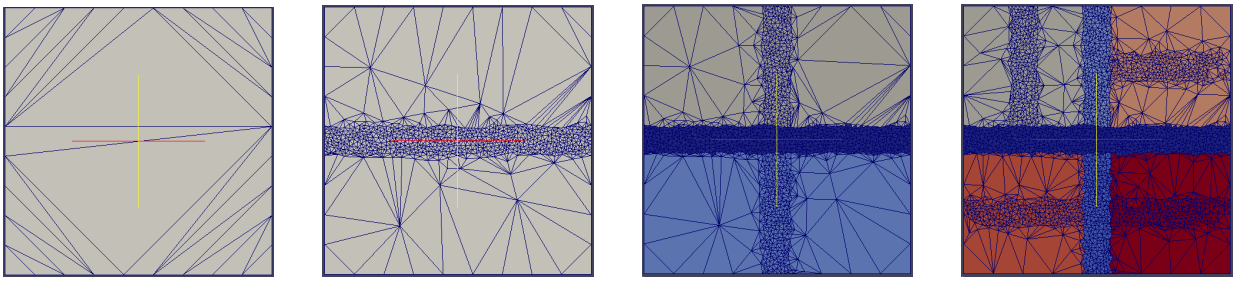
(a) Accélération, $\bar{\rho} = 20$, $|\mathcal{M}| \approx 2\,060$ K-faces



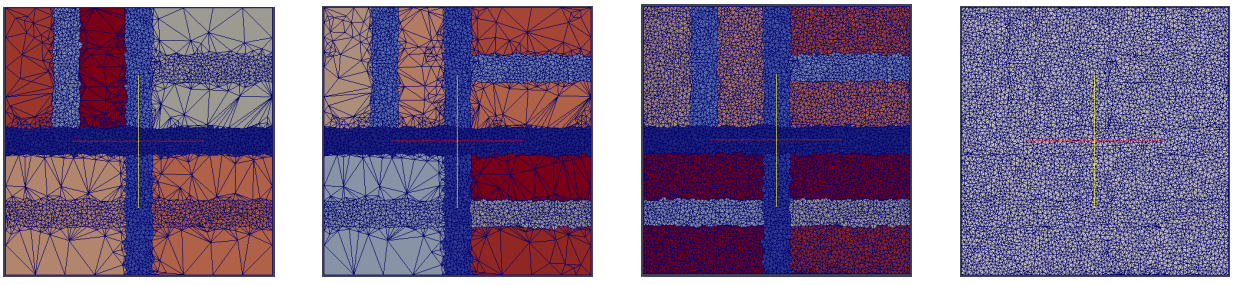
(b) Efficacité pour $\bar{\rho} = 20$, $|\mathcal{M}| \approx 2\,060$ K-faces

FIGURE 5.4 – Accélération et efficacité de H_1 et H_2 pour $\bar{\rho} = 20$ – Intel Xeon-Phi

Ω, la taille de chaque interface $\mathcal{I}_i$ décroit logarithmiquement en fonction de n_w. Ainsi d'un pas de partitionnement à un autre, les PE n'auront pas le même nombre de faces à raffiner – voir figure 5.5.



(a) triangulation (b) raffinement $i = 1$ (c) raffinement $i = 2$ (d) raffinement $i = 3$



(e) raffinement $i = 4$ (f) raffinement $i = 5$ (g) raffinement $i = 6$ (h) Maillage final

FIGURE 5.5 – Partitionnement récursif de Ω par H_2 pour $n_w = 8$

5.2.2 Scalabilité faible

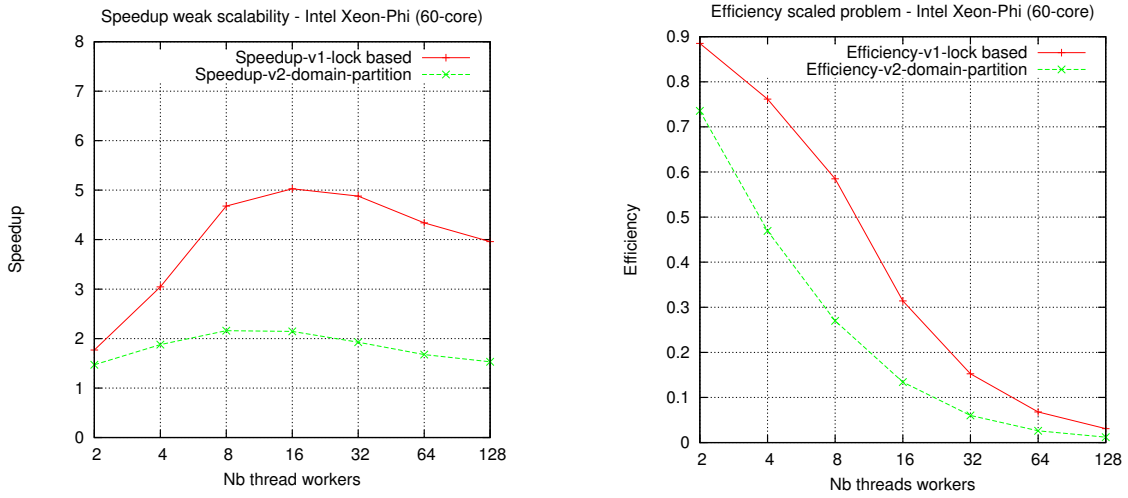
Cette fois, on cherche à évaluer et comparer la scalabilité faible des 2 heuristiques (cf. déf. 11, page 12). En d'autres termes, on cherche à estimer le gain de performances de H_1 et H_2 quand on fait évoluer le nombre de PE workers proportionnellement à la taille du maillage – voir figure 5.6.

Le nombre final de faces étant un résultat et non un paramètre, on l'estime en fonction du degré de raffinement $\bar{\rho}$, grâce à la formule de récurrence : $\rho_i = \rho_0 \cdot \frac{1}{\sqrt{n_w}}$.

La table 5.1 donne le nombre de faces (créées/finales) en fonction de n_w.

n_w	$\bar{\rho}$	nb. créées	nb. final
2	150	140 943	36 548
4	106	282 488	73 660
8	75	565 835	147 288
16	53	1 128 702	293 954
32	37	2 317 184	602 180
64	26	4 696 730	1 221 404
128	18	9 794 925	2 545 704

TABLE 5.1 – Nombre de faces (total/final) en fonction du nombre de PE workers



(a) Accélération en fonction de $\bar{\rho}$ et n_w

(b) Efficacité en fonction de $\bar{\rho}$ et n_w

FIGURE 5.6 – Accélération et efficacité pour $36\,000 \leq |\mathcal{M}| \leq 2\,502\,000$ faces, et $2 \leq n_w \leq 128$

La figure 5.6 montre que H_1 a une meilleure scalabilité que H_2, quand on fait évoluer conjointement $\bar{\rho}$ et n_w. On constate que ces résultats ne diffèrent pas de ceux obtenus pour la scalabilité forte dans 5.2.1.

Avec une accélération crête de 5 pour H_1, et de 2.2 pour H_2, cela confirme que la pénalité liée au partitionnement récursif résorbe l'accélération obtenue lors du raffinement final des faces du maillage pour H_2. En effet, les performances de H_2 dépendent fortement de la géométrie du domaine, et devraient s'avérer meilleures pour une géométrie allongée (auquel cas la taille des interfaces se verraient diminuer) – voir figure 5.5.

De même, avec une efficacité quasi-identique à celle dans 5.2.1, cela confirme la bonne répartition des charges des PE dans H_1, même avec une taille de maillage importante.

Chapitre 6

Conclusion

Dans le cadre de notre étude, on a proposé 2 heuristiques pour la génération parallèle automatique de maillages de Delaunay. La première solution, H_1, est basée sur un partage de tâches entre les n PE en charge de générer le maillage, tandis que la seconde, H_2, est basée sur une décomposition du domaine et la distribution du traitement des parties entre les n PE. Les deux solutions ont été étudiées et validées expérimentalement.

Observations

Les résultats expérimentaux obtenus montrent que notre implémentation de H_1 est plus performante que celle de H_2. Ceci est conforme à deux avantages théoriques de H_1 : une finesse de décomposition qui minimise les synchronisations d'une part, et une répartition équilibrée des charges entre les PE d'autre part. Pour cela, elle requiert une gestion des accès concurrents de toutes les structures utilisées (conteneur de maillage, DAG, conteneur des faces à raffiner), afin d'éviter les incohérences topologiques induites par la compétition des PE.

Bien que moins performante, H_2 bénéficie d'une meilleure extensibilité. En effet, comme la gestion des éléments aux interfaces est effectuée en amont, chaque PE peut traiter sa partie indépendamment de ses pairs, ce qui limite les communications entre PE.

En outre, l'implémentation de ces 2 heuristiques a permis d'identifier des modifications souhaitables à apporter à GMDS pour le support d'algorithmes parallèles de maillages en mémoire partagée. En particulier, pour permettre à plusieurs PE de modifier simultanément un maillage, GMDS pourrait fournir des fonctionnalités d'accès verrouillé aux éléments du maillage, sans que l'utilisateur n'ait à l'implémenter explicitement au sein de son code.

Perspectives

La durée du stage n'a pas permis d'analyser totalement les résultats obtenus, et de les consolider par des cas-tests supplémentaires. Ceci est un travail nécessaire à faire à court terme pour comprendre comment optimiser différents points de contention. Cependant, nous avons déjà pu en identifier dans la solution proposée :

1. Pour la solution H_1, un premier point de contention est lié à notre utilisation du conteneur de faces à raffiner. Pour rappel, le principe de la phase de raffinement est que, à un instant donné, chaque PE va essayer de raffiner un triangle ce qui nécessite de verrouiller ce triangle et une partie de son voisinage. Si un triangle de ce voisinage est déjà protégé par un autre PE, la tâche est abandonnée. La spécification FIFO [1] du conteneur fait que de nombreuses tâches vont justement être abandonnées. En effet, les PE ne vont pas récupérer des triangles géométriquement distants mais des triangles proches car insérés par « blocs » (voir l'opération de subdivision par exemple). Fournir un accès en lecture totalement aléatoire serait une première avancée. Tenir en plus compte de la distance géométrique entre éléments dans ce processus en serait une seconde.

1. FIFO pour *First In, First Out.*

2. Un second point de contention est lié à la scalabilité du conteneur lui-même. Dans notre approche, tous les PE y accèdent de manière concurrente. Il est donc nécessaire que celui-ci garantisse des accès concurrents efficaces à la fois en consultation et en insertion, et ce pour un nombre important de PE (de l'ordre de 100 à 200).
3. La solution H_2 repose sur un partitionnement géométrique relativement simple (découpe axiale de la boîte englobante du domaine). Ce découpage ne tient pas compte du nombre de mailles à insérer ce qui ne garantit pas un bon équilibrage de charge. Ce point mérite d'être étudié à l'avenir.
4. De même, pour H_2, le rapport d'aspect entre les faces d'une partie et de la zone de raffinement génère des configurations géométriques numériquement instables. La prise en compte d'une gradation [2] sur le maillage permettrait d'éviter ce problème.

Enfin,il est important de rappeler que la seconde solution se prête bien à une implémentation en mémoire distribuée (MPI), ce qui permet de s'affranchir des limitations mémoire. A terme, cela pourra permettre de traiter des maillages de plus grande taille d'une part, et de fournir une approche hybride [3] basée sur le couplage de ces 2 heuristiques d'autre part.

2. Assurer un rapport d'aspect de 2 au plus entre mailles voisines par exemple.
3. En mémoire partagée + distribuée

Bibliographie

[1] Mark Yerry and Mark Shephard. « Three-Dimensional Mesh Generation by Modified Octree Technique ». In *International Journal for Numerical Methods in Engineering*, 1984.

[2] Mark Shephard and Marcel Georges. « Three-Dimensional Mesh Generation by Finite Octree Technique ». In *International Journal for Numerical Methods in Engineering*, pages 709–749, 1991.

[3] S.H. Lo. « A New Mesh Generation Scheme for Arbitrary Planar Domains ». In *International Journal for Numerical Methods in Engineering*, volume 21, pages 1403–1426, 1985.

[4] Michael Ian Shamos and Hoey Dan. « Closest-Point Problems ». In *Proceeding of* 16^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[5] Charles Lawson. « Generation of a Triangular Grid with Applications to Contour Plotting ». In *Technical Memo n.299*, 1977.

[6] Peter Green and R. R. Sibson. « Computing Dirichlet Tessellations In The Plane ». In *Computer Journal*, volume 21, pages 168–173, 1978.

[7] François Hermeline. « Une Méthode Automatique de Maillage en Dimension n ». *Thèse de doctorat*, 1980.

[8] Adrian Bowyer. « Computing Dirichlet Tessellations ». In *Computing Journal*, volume 24, pages 162–166, 1981.

[9] David Watson. « Computing The n-dimensional Delaunay Tessellation With Application to Voronoi Polytopes ». In *Computing Journal*, volume 24, pages 167–172, 1981.

[10] Leonidas Guibas and Jorge Stolfi. « Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams ». In *ACM Transactions on Graphics*, volume 4, pages 74–123, 1985.

[11] Pascal Jean Frey and Paul-Louis George. « Maillages, applications aux éléments finis ». *HERMES Science Publications*, 1999.

[12] Sangyoon Lee, Chan-Ik Park, and Chan-Mo Park. « An Improved Parallel Algorithm for Delaunay Triangulation on Distributed Memory Parallel Computers ». In *Parallel Processing Letters*, pages 341–352, 2001.

[13] Josef Kohout, Ivana Kolingerova, and Jiri Zara. « Practically Oriented parallel Delaunay Triangulation in $\mathbb{R}^2$ for Computers in Shared Memory ». In *Computers & Graphics*, pages 703–718, 2004.

[14] Nikos Chrisochoides, Filip Blagojevic, Andrey Chernikov, and Christos Antonopoulos. « A Multigrain Delaunay Mesh Generation Method for Multicore SMT-based Architectures ». In *Journal of Parallel and Distributed Computing*, pages 589–600, 2009.

[15] Nikos Chrisochoides, Christos Antonopoulos, Filip Blagojevic, Andrey Chernikov, and Dimitrios Nikolopoulos. « Algorithm, Software and Hardware Optimizations for Delaunay Mesh Generation on Simultaneous Multithreaded Architectures ». In *Journal of Parallel and Distributed Computing*, pages 601–612, 2009.

[16] Sébastien Valette, Julien Dardenne, Nicolas Siauve, and Rémy Prost. « Génération de Maillages Triangulaires Adaptatifs 2D avec des Diagrammes de Voronoi Centroidaux ». In $22^{ème}$ *Colloque GRETSI – Traitement du Signal et des Images*, 2009.

[17] Josef Kohout and Ivana Kolingerova. « Optimistic Parallel Delaunay Triangulation ». In *The Visual Computer*, volume 18, pages 511–529, 2002.

[18] Aurélien Alleaume, Laurent Francez, Mark Loriot, and Nathan Maman. « Large Out-of-core Tetrahedral Meshing ». In *Proceedings of the* 16^{th} *International Meshing Roundtable*, pages 461–476, 2008.

[19] Michael Flynn. « Some Computer Organizations And Their Effectiveness ». In *IEEE Transactions on Computing*, volume 21, pages 948–960, 1972.

[20] Frédéric Desprez. *« Support de Cours Parallélisme »*. Inria/ENS Lyon, 2012.

[21] Gene Amdahl. « Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities ». In *AFIPS Conference Proceedings*, 1967.

[22] John Gustafson. « Reevaluating Amdahl's Law ». In *Communications of the ACM*, 1988.

[23] Guy Blelloch and Franklin Cho. « Divide-and-Conquer Delaunay Triangulation ». In *Algorithms in the Real World*, 1998.

[24] S. Rebay. « Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer–Watson Algorithm ». In *Journal of Computational Physics*, volume 106, pages 125–138, 1993.

[25] Leonidas Guibas, Alok Aggarwal, Bernard Chazelle, Colm O'Dunlaing, and Chee Yap. « Parallel Computational Geometry ». In *Algorithmica*, volume 3, pages 293–327, 1988.

[26] JR Davy and P.M. Dew. « A Note on Improving the Performance of Delaunay Triangulation ». In *Proceedings of Computer Graphics International*, 1989.

[27] Andrea Clematis and Enrico Puppo. « Effective Parallel Processing of Irregular Geometric Structures ». In *Proceedings AICA, International Section : Parallel and Distributed Architectures and Algorithms*, 1993.

[28] Paolo Cignoni, Claudio Montani, Raffaele Perego, and Roberto Scopigno. « Parallel 3D Delaunay Triangulation ». In *Computer Graphics Forum*, volume 12, 1993.

[29] Jonathan Hardwick. « Implementation and Evaluation of an Efficient Parallel Delaunay Triangulation Algorithm ». In *Proceedings of the* 9^{th} *ACM Symposium on Parallel Algorithms and Architectures*, pages 239–248, 1997.

[30] Subhash Suri and Robert Pless. « Delaunay Triangulations and Convex Hulls ». *Departement of Computer Science, University of California*, 2000.

[31] Jean-Daniel Boissonnat. « Convex Hulls, Voronoi Diagrams and Delaunay Triangulations ». *Winter School on Algorithmic Geometry*, 2010.

[32] Min-Bin Chen, Tyng-Ruey Chuang, and Jan-Jan Wu. « Efficient Parallel Implementation of 2D Delaunay Triangulation with High Performance Fortran ». In *Proceedings of* 10^{th} *SIAM Conference on Parallel Processing for Scientific Computing*, 2001.

[33] Enrico Puppo, Larry Davis, Daniel DeMenthon, and Ansel Teng, Y. « Parallel Terrain Triangulation ». In *Proceeding of the* 5^{th} *International Symposium on Spatial Data Handling*, pages 632–641, 1992.

[34] Tolulope Okusanya and Jaume Peraire. « Parallel Unstructured Mesh Generation ». In *Proceeding of the* 5^{th} *International Conference on Numerical Grid Generation in Computational Fluid Dynamic and Related Fields*, 1996.

[35] Paul Chew. « Guaranteed-Quality Triangular Meshes ». In *Technical Report n.89-983*, 1989.

[36] Nikos Chrisochoides and Démian Nave. « Simultaneous Mesh Generation and Partitioning for Delaunay Meshes ». In *Proceeding of the* 8^{th} *International Meshing Roundtable*, pages 55–66, 1999.

[37] Rex Dwyer. « Higher-Dimensionnal Voronoi Diagrams in Linear Expected Time ». In *Proceedings of* 5^{th} *annual symposium on Computational Geometry*, pages 326–333, 1988.

[38] Ivana Kolingerova. « Modified DAG Location for Delaunay Triangulation ». In *Proceedings of the International Conference on Computational Science*, pages 125–134, 2002.

[39] Franck Ledoux, Jean-Christophe Weill, and Yves Bertrand. « GMDS : A Generic Mesh Data Structure ». In 17^{th} *International Meshing Roundtable*, 2008.

[40] D.G. Holmes and D.D. Snyder. « The Generation of Unstructured Triangular Meshes using Delaunay Triangulation ». In *Numerical grid generation in computational fluid mechanics*, pages 643–652, 1988.

[41] Laurent Tacher and Aurèle Parriaux. « Automatic Nodes Generation in n-Dimensional Space ». In *Communications in Numerical methods in Engineering*, volume 12, pages 243–248, 1996.

[42] Nigel Weatherill. « Generation of Unstructured Grids Using Dirichlet Tesselations ». *Princeton University*, 1985.

[43] Timothy Baker. « Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets ». In *Proceedings of the* 8^{th} *AIAA Computational Fluid Dynamics Conference*, 1987.

[44] Dimitri Mavriplis. « Adaptative Mesh Generation for Viscous Flows using Delaunay Triangulation ». In *Journal of Computational Physics*, volume 90, pages 271–291, 1990.

[45] Éric Béchet. *« Cours de Géométrie Algorithmique »*. Université de Liège, 2013.

[46] Steven McDonagh, Cigdem Beyan, Phoenix Huang, and Robert Fisher. « Applying Semi-synchronised Task Farming to Large-scale Computer Vision Problems ». In *International Journal of High Performance Computing Applications*, 2014.

[47] G Karypis and V. Kumar. « Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs ». In *SIAM Review*, volume 41, pages 278–300, 1999.

[48] Cédric Chevalier and François Pellegrini. « Improvement of the Efficiency of Genetic Algorithms for Scalable Parallel Graph Partitioning in a Multi-Level Framework ». In *Proceedings of Euro-Par*, pages 243–252, 2006.

[49] C. Farhat and M. Lesoinne. « Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics ». In *International Journal for Numerical Methods in Engineering*, volume 36, pages 745–764, 1993.

[50] B.W. Kerninghan and S. Lin. « An Efficient Heuristic Procedure for Partitioning Graphs ». In *Bell System Technical Journal*, volume 49, page 291–307, 1970.

[51] E. Cuthill and J. McKee. « Reducing the Bandwidth of Sparse Symmetric Matrices». In *Proceeding of* 24^{th} *National Conference Association in Computing Mach.*, pages 157–172, 1969.

[52] P. Gervasio, E. Ovtchinnikov, and A. Quarteroni. « The Spectral Projection Decomposition Method for Elliptic Equations in Two Dimensions ». In *SIAM Journal on Numerical Analysis*, volume 34, pages 1616–1639, 1997.

[53] H. Simon. « Partitioning of Unstructured Problems for Parallel Processing ». In *Computing Systems in Engineering.*, volume 2, pages 135–148, 1991.

[54] R. Natarajan. « Domain Decomposition using Spectral Expansion of Steklov-Poincaré Operators II : A Matrix Formulation ». In *SIAM Journal on Scientific Computing*, volume 18, pages 1187–1199, 1997.

[55] Orion Lawlor, Sayantan Chakravorty, Terry Wilmarth, Nilesh Choudhury, Isaac Dooley, Gengbin Zheng, and Laxmikant Kalé. « Parfum : A Parallel Framework for Unstructured meshes for Scalable Dynamic Physics Applications. ». In *Engineering with Computers*, volume 22, pages 215–235, 2006.

Table des figures

Annexe A

Annexes

A.1 Prédicats géométriques

Lors des procédures de localisation et de légalisation, il faudra utiliser une heuristique rapide pour vérifier l'appartenance d'un point à un triangle, ou à un cercle circonscrit d'un triangle.

Lemme 1. Pour savoir si un point p est inclus dans un triangle $\Delta = (\mathsf{A},\mathsf{B},\mathsf{C})$, on calcule les coordonnées barycentriques (α, β, γ) de p dans le repère $(\vec{AB}, \vec{AC})$, et on vérifie son signe :

$$\alpha = \begin{vmatrix} x_B - x_P & x_C - x_P \\ y_B - y_P & y_C - y_P \end{vmatrix} \quad \beta = \begin{vmatrix} x_C - x_P & x_A - x_P \\ y_C - y_P & y_A - y_P \end{vmatrix} \quad \gamma = \begin{vmatrix} x_A - x_P & x_B - x_P \\ y_A - y_P & y_B - y_P \end{vmatrix}$$

$$\begin{array}{rcl} \alpha = 0 & \Leftrightarrow & \det(\vec{PB}, \vec{PC}) = 0 \quad \Leftrightarrow \quad p \text{ est sur l'arête } \mathsf{BC} \\ \beta = 0 & \Leftrightarrow & \det(\vec{PA}, \vec{PC}) = 0 \quad \Leftrightarrow \quad p \text{ est sur l'arête } \mathsf{CA} \\ \gamma = 0 & \Leftrightarrow & \det(\vec{PA}, \vec{PB}) = 0 \quad \Leftrightarrow \quad p \text{ est sur l'arête } \mathsf{AB} \end{array}$$

$$\text{Donc } p \text{ est à l'intérieur de } \Delta \Leftrightarrow \begin{cases} \alpha > 0 \\ \beta > 0 \\ \gamma > 0 \end{cases}$$

Toutefois, les signes de α, β, γ dépendant de l'orientation du repère $(\vec{AB}, \vec{AC})$, il faudra veiller à garder la même orientation (anti-horaire) pour tous les triangles créés.

Lemme 2. Pour vérifier si un point p se trouve à l'intérieur du cercle circonscrit au triangle $\Delta = (\mathsf{A},\mathsf{B},\mathsf{C})$, il faudra vérifier le signe du déterminant suivant :

$$p \in \mathcal{C}_{(\mathsf{A},\mathsf{B},\mathsf{C})} \quad \Leftrightarrow \quad \begin{vmatrix} x_A & y_A & {x_A}^2 + {y_A}^2 & 1 \\ x_B & y_B & {x_B}^2 + {y_B}^2 & 1 \\ x_C & y_C & {x_C}^2 + {y_C}^2 & 1 \\ x_P & y_P & {x_P}^2 + {y_P}^2 & 1 \end{vmatrix} > 0$$

Une preuve complète de ce lemme est fournie dans [10].

A.2 Architecture du code

Le programme a été implémenté en `C++` selon un modèle orienté objet.
Les entités relatifs au maillage (faces, nœuds, conteneur principal) sont implémentées par des classes de GMDS (voir section 3.2, page 23), dont on ne détaillera pas ici.
Le diagramme de classes de l'application est décrit sur la figure A.1.

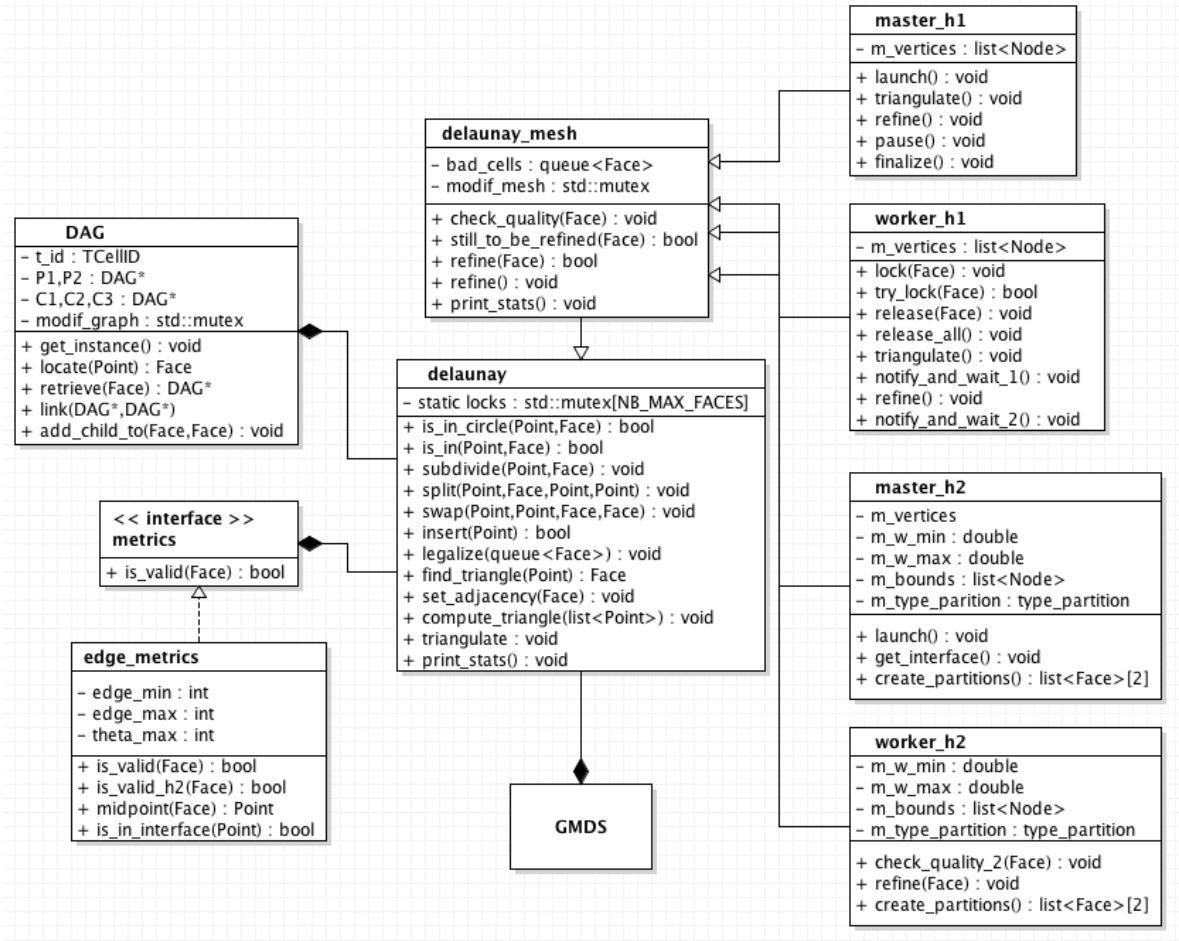


Figure A.1 – Diagramme de classes de l'application

Outre celles de GMDS, l'application est constituée de 8 classes principales :

- `delaunay` : Elle fournit les méthodes nécessaires pour la mise en œuvre de la triangulation. Elle contient un tableau global de verrous des faces, ainsi qu'une instance du DAG (pour la localisation de faces). Elle contient également les méthodes relatives aux prédicats géométriques (appartenance d'un point à une face ou à son cercle circonscrit, adjacence de faces etc.), ainsi que les opérations élémentaires sur les cellules (localisation, subdivision, légalisation de face, insertion d'un point, mise à jour du voisinage).

- `DAG` : Elle fournit une représentation du DAG (graphe chainé), ainsi que les méthodes nécessaires à sa manipulation : récupération d'une instance, localisation de la face contenant un point donné, connexion de 2 nœuds du DAG etc.

- `delaunay_mesh` : Elle contient le conteneur de faces de mauvaise qualité, et fournit les méthodes nécessaires pour la génération du maillage (raffinement d'une face, vérification de la qualité d'une face selon la métrique etc.).

- ✏ `metrics` et `edge_metrics` : Elles implémentent la métrique utilisée dans les phases de triangulation et génération du maillage. Elles contiennent des prédicats vérifiant si une face donnée respectent la métrique implémentée ou non (taille cible d'arête, aplatissement maximale etc.).

- ✏ `master_h1` et `worker_h1` : Elles correspondent à l'implémentation de la première heuristique H_1. La classe `master_h1` contient les méthodes nécessaires à la synchronisation des workers (création des threads, barrières), et à la finalisation des phases (suppression triangle conteneur, statistiques etc.). La classe `worker_h1` contient les méthodes nécessaires au verrouillage des faces, la notification des workers, ainsi que l'implémentation des phases de H_1.

- ✏ `master_h2` et `worker_h2` : Elles correspondent à l'implémentation de la seconde heuristique H_2. La classe `master_h2` contient les méthodes relatives à la triangulation initiale, à l'initialisation des workers, au calcul de l'interface et à la création de la première partition. La classe `worker_h2` contient les méthodes relatives au calcul de l'interface $\mathcal{I}$ correspondant à sa partie, et au raffinement des éléments proches de $\mathcal{I}$, et à la création de la partition suivante.

Annexe B

Fiche de synthèse

B.1 Le CEA

Acteur majeur de la recherche, du développement et de l'innovation, le Commissariat à l'énergie atomique et aux énergies alternatives intervient dans les trois grand domaines : les énergies décarbonées, la Défense et la sécurité globale, les technologies pour l'information et la santé.

Pour être au plus haut niveau de la recherche, le CEA compte plusieurs atouts :

- ✏ une culture croisée entre ingénieurs et chercheurs, propice aux synergies entre recherche fondamentale et innovation technologique ;
- ✏ des installations exceptionnelles (supercalculateurs, réacteurs de recherche, lasers de puissance...) ;
- ✏ une forte implication dans le tissu industriel et économique

Le CEA est structuré autour de cinq pôles opérationnels (Défense, énergie nucléaire, recherche technologique, sciences de la matière et sciences du vivant) et quatre pôles fonctionnels (maitrise des risques, ressources humaines et formation, stratégie et relations extérieures, gestion des systèmes d'information), implantés dans 10 centres répartis dans toute la France.
Il développe de nombreux partenariats avec les autres organisme de recherche, les collectivités locales et les universités.
Reconnu comme un expert dans ses domaines de compétences, le CEA est pleinement inséré dans l'espace européen de la recherche et exerce une présence croissante au niveau international.
Le CEA compte actuellement 15700 personnes pour un budget de 4,3 milliards d'euros.

B.1.1 La Direction des Applications Militaires

La Direction des Applications Militaires (DAM) constitue le pôle Défense du CEA. Ce pôle a pour mission principale de concevoir, fabriquer, maintenir en condition opérationnelle puis démanteler les têtes nucléaires qui équipent les forces océaniques et aéroportées.

Apportant aux pouvoirs publics la garantie que ces têtes sont fiables et sûres, il est l'un des principaux artisans de la crédibilité de la dissuasion nucléaire française.
L'objectif du Pôle Défense est de continuer à assurer sur le long terme cette capacité de dissuasion sans recourir aux essais nucléaires, définitivement arrêtés depuis 1996.

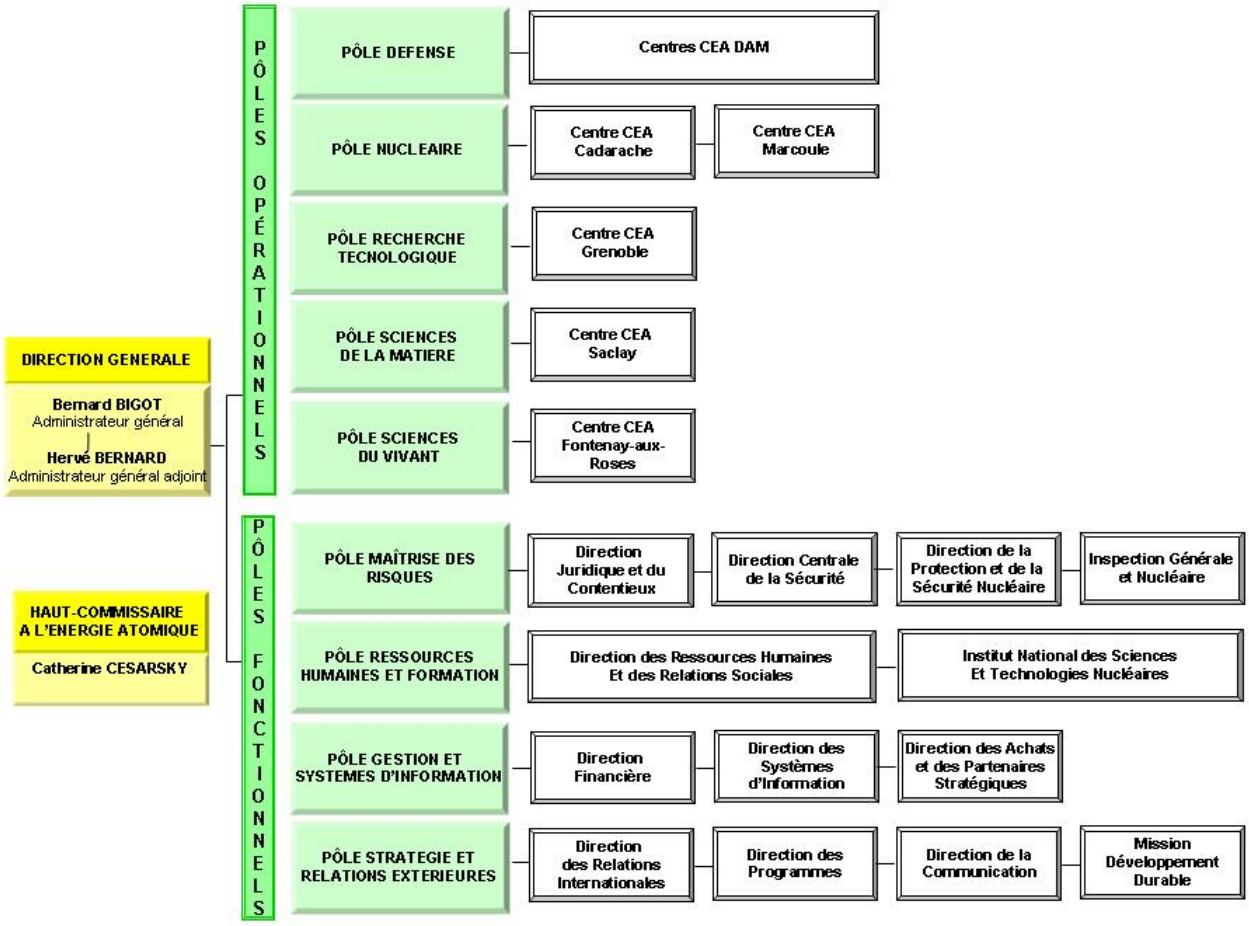


Figure B.1 – Organigramme du CEA

Ses missions portent également sur :

- ✏ l'approvisionnement en matières nucléaires pour les besoins de la Défense ;
- ✏ la propulsion nucléaire navale ;
- ✏ la lutte contre la prolifération et le terrorisme et la sécurité globale.

Le Pôle Défense publie bon nombre de ses résultats scientifiques et développe des collaborations avec d'autres acteurs de la Recherche. Il valorise ses activités auprès des industries par le transfert de technologies et le dépôt de brevets. Il s'est également engagé dans une importante démarche de certification qualité de l'ensemble de ses activités liées aux armes nucléaires.

La DAM compte aujourd'hui 4750 collaborateurs, menant des activités réparties entre la recherche de base, le développement et la fabrication. Son budget est d'environ 1,8 milliards d'euros.

Elle comprend :

- ✏ Un échelon Direction ;
- ✏ Cinq directions d'objectifs (DOB), qui ont pour mission la définition et la gestion des programmes et le pilotage des projets ;
- ✏ Cinq directions opérationnelles (DOP), qui ont pour mission la réalisation des produits conformément aux directeurs d'objectifs. Les directeurs opérationnels de la DAM sont les directeurs de centre ;

✏ Quatre directions fonctionnelles (DF), qui ont des missions de directive, de soutien et de contrôle.

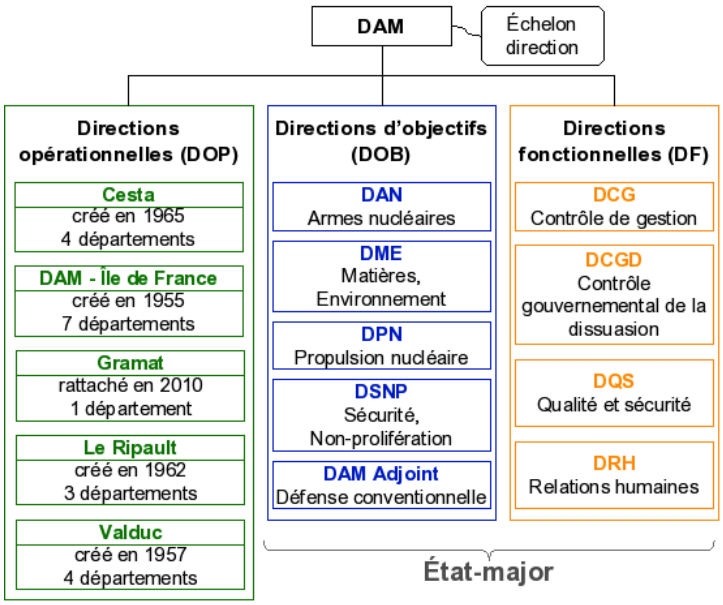


FIGURE B.2 – Organigramme de la DAM

La DAM est implantée sur cinq centres :

✏ Valduc, en Bourgogne ;

✏ Le Ripault, en Touraine ;

✏ Le Cesta, en Aquitaine ;

✏ DAM-Île de France (DIF) ;

✏ Gramat (CEG) en Midi-Pyrénées.

B.1.2 Le centre DAM Ile de France

Le CEA/DAM-Île de France (DIF) est l'une des directions opérationnelles de la DAM. Le site de la DIF compte environ 2000 salariés CEA et accueille quotidiennement 600 salariés d'entreprises extérieures.

Elle comprend deux sites :

✏ Le site de Bruyères-le-Châtel, situé à environ 40km au sud de Paris, dans l'Essonne ;

✏ le site du Polygone d'expérimentation de Moronvilliers (PEM), situé à 20km de Reims, dans la Marne.

Les missions de la DIF comprennent :

✏ La conception et garantie des armes nucléaires, grâce au programme Simulation. L'enjeu consiste à reproduire par le calcul les différentes phases du fonctionnement d'une arme nucléaire et à confronter ces résultats aux mesures des tirs nucléaires passés et aux résultats expérimentaux obtenus sur les installations actuelles (machine radiographique Airix, lasers de puissance, accélérateurs de particules) ;

✏ La lutte contre la prolifération et le terrorisme, en contribuant notamment au programme de garantie du Traité de Non Prolifération et en assurant l'expertise technique française pour la mise en œuvre du Traité d'Interdiction Complète des Essais Nucléaires (TICE) ;

- ✏ L'expertise scientifiques et technique, dans le cadre de la construction et du démantèlement d'ouvrages complexes ainsi que pour la surveillance de l'environnement et les sciences de la terre ;
- ✏ L'alerte des autorités, mission opérationnelle assurée 24h sur 24, 365 jours par an, en cas d'essai nucléaire, de séisme en France ou à l'étranger, et de tsunami dans la zone Euroméditerranéenne. La DIF fournit aux autorités les analyses et synthèses techniques associées.

Depuis 2003, le centre DAM-Île de France héberge le complexe de calcul scientifique du CEA, qui regroupe l'ensemble des supercalculateurs du CEA, et qui comprend :

- ✏ Le supercalculateur TERA-100 pour les besoins Défense, premier calculateur européen à dépasser la barre du petaflops, c'est à dire capable d'effectuer un million de milliards d'opérations à la seconde ;
- ✏ Les ordinateurs du Centre de Calculs pour la Recherche et la Technologie (CCRT), ouverts à la communauté civile de la recherche et de l'industrie, pour une puissance globale de 500 teraflops ;
- ✏ Le supercalculateur Curie, d'une puissance de 2 petaflops, deuxième élément d'un réseau de supercalculateurs de classe pétaflopique destiné au TGCC (Très Grand Centre de Calcul) et exploité par les équipes CEA qui apporte ainsi sa contribution à la participation de la France au projet PRACE (Partnership for Advanced Computing in Europe), dans le cadre de la recherche européenne.

B.1.3 Le campus TERATEC

Le Campus TERATEC est l'une des deux composantes de la Technopole TERATEC, l'autre composante étant le « Très Grand Centre de Calcul du CEA » (TGCC). La Technopole, créée à l'initiative du CEA afin de développer et promouvoir la simulation numérique haute performance, est située dans la zone dite de « Morionville », à proximité du site du CEA, sur la commune de Bruyères-le-Châtel, dans le département de l'Essonne.

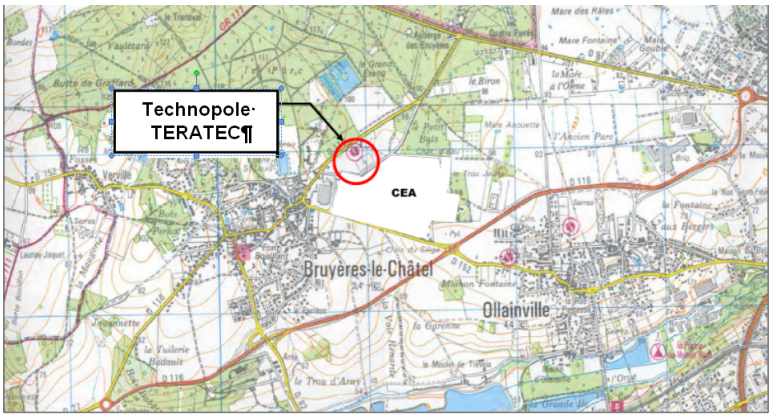


Figure B.3 – Position du campus TERATEC

L'accès principal à la Technopole TERATEC se fait par une entrée commune au TGCC et au Campus, située au nord-ouest, sur le chemin de la Piquetterie.

Le Campus TERATEC est un ensemble immobilier à usage de bureau. L'accès au Campus TERATEC est contrôlé et vidéo-surveillé. Deux hôtesses d'accueil, membres du personnel de la Chambre de Commerce et de l'Industrie (CCI), se relaient à l'entrée principale (de 8h30 à 18h du lundi au jeudi, de 8h30 à 17h le vendredi).

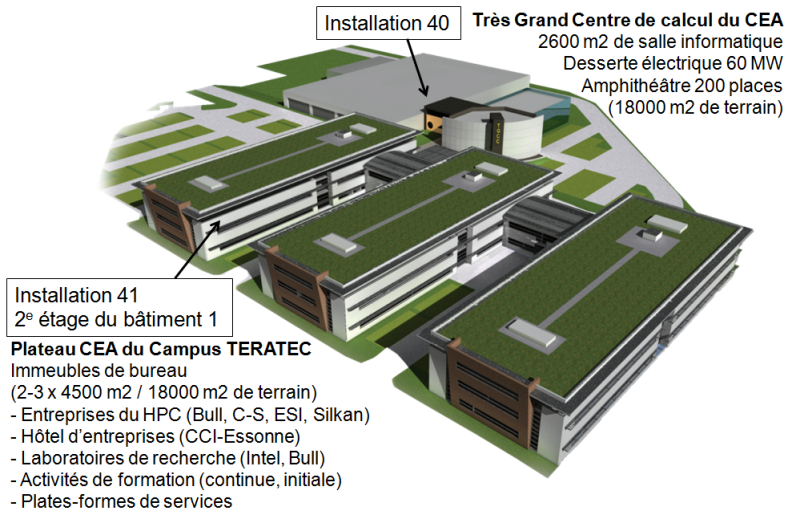


Figure B.4 – Vue d'ensemble des bâtiments du campus Teractec

L'installation Sciences Informatiques

Le centre CEA/DIF est locataire d'un plateau de 1500m^2 situé au 2e étage du bâtiment 1 du Campus.

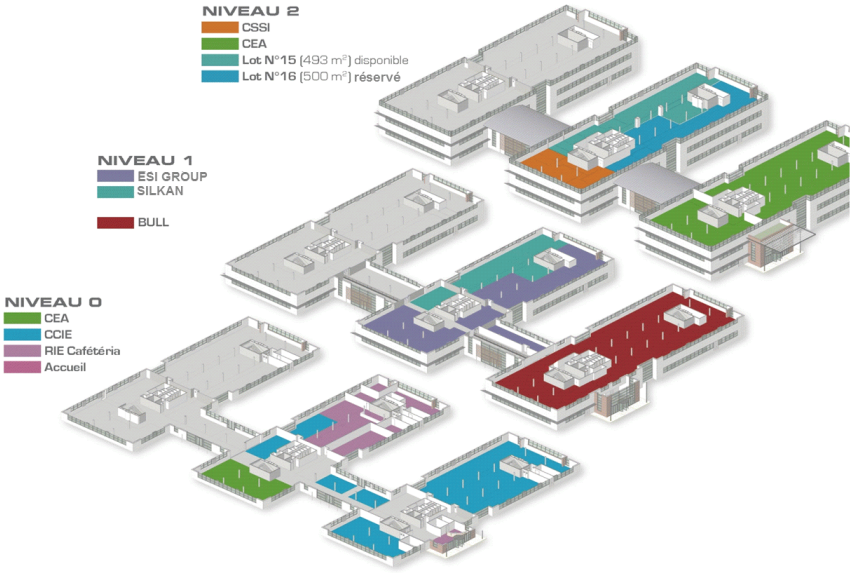


Figure B.5 – Vue en éclaté du premier bâtiment du campus Teractec

Le plateau CEA est organisé en 3 zones :

- La zone CEA pour l'accueil de stagiaires, de thésards et de post-doc,
- La zone ACCUEIL pour les réunions de travail, séminaires, conférences, formations...
- La zone ECR pour l'hébergement du laboratoire « Exascale Computing Research ».

B.2 Résumé des travaux

B.2.1 Encadrants

- ✏ Franck Ledoux – Ingénieur-chercheur
- ✏ Nicolas Le-Goff – Ingénieur-chercheur

B.2.2 Travaux proposés

Le sujet initial s'articulait autour de la génération parallèle de triangulation de Delaunay 2D, en mémoire partagée. L'objectif était de :

- ✏ proposer une heuristique parallèle, qui soit à la fois générique (insensible à la distribution des points), et scalable (en nombre de threads) pour la triangulation parallèle de Delaunay.
- ✏ fournir une implémentation en C++, et d'analyser les résultats obtenus.

Par la suite, il a été décidé d'étendre l'étude aux maillages de Delaunay, incluant les phases de triangulation et de raffinement parallèles.

B.2.3 Travaux effectués

Ils comprennent :

- ✏ État de l'art des techniques de maillages non-structurés 2D, en séquentiel et en parallèle.
- ✏ Études des approches existantes pour la triangulation et le maillage de Delaunay.
- ✏ Apprentissage du C++ et de la bibliothèque standard (STL), mise en place des outils (compilateurs, debugger, gestionnaire de version etc.)
- ✏ Prise en main de la plateforme GMDS, et implémentation de programmes-tests.
- ✏ Conception de l'algorithme séquentiel, codage et premiers tests de performances.
- ✏ Étude d'une première heuristique parallèle H_1, et codage.
- ✏ Débogage et refonte de la solution H_1.
- ✏ Étude d'une seconde heuristique parallèle H_2, codage et débogage.
- ✏ Déploiement sur le cluster de tests, et configuration (scripts, variables d'environnement, modules etc.)
- ✏ Exécution des tests de performances, génération des courbes et analyse des résultats.
- ✏ Rédaction du mémoire, et préparation pour la soutenance interne (slides + oral).

Technologies utilisées

Red Hat Enterprise Linux, C/C++, Git, Intel TBB, GDB, Valgrind, Gnuplot, Vim, Python, C-Make, SLURM, Paraview

B.3 Synthèse du sujet

B.3.1 Présentation du sujet de mémoire

Sujet : Heuristique parallèle pour la génération automatique de maillages de Delaunay en mémoire partagée.
Problématique : Trouver une approche parallèle générique pour la génération automatique de maillages de Delaunay, et fournir une solution expérimentale pouvant tirer parti de centaines de processus légers (100 à 200 threads environ).

B.3.1.1 Objectifs

La génération parallèle de maillages constitue un domaine de recherche actif, et il existe de nombreuses approches pour le cas des maillages de Delaunay [12, 13, 14, 15, 16]. Toutefois, il est difficile de trouver une approche qui concilie généricité (indépendance vis-à-vis de la géométrie du domaine à mailler) et performances (scalabilité). Ici, le but du travail était d'établir et de comparer des solutions expérimentales, visant à répondre à ces 2 objectifs, en s'appuyant sur une combinaison des approches existantes [13, 15, 7, 18]. En parallèle, un objectif secondaire était d'identifier les modifications souhaitables à apporter à GMDS pour le support d'algorithmes parallèles de maillages en mémoire partagée.

B.3.1.2 Motivations

D'une part, je suis intéressé par les problématiques informatiques relatives à la simulation numérique. D'autre part, je suis aussi intéressé par tout ce qui concerne le calcul (massivement) parallèle [1]. A ce titre, ce sujet me permettait de traiter conjointement les 2 problématiques, ainsi que d'acquérir des compétences relatives aux 2 domaines (maillages et HPC).

B.3.2 Sources d'information

Elles proviennent essentiellement d'articles scientifiques listés dans la bibliographie (page 54).

B.3.3 Innovation

A notre connaissance, il s'agit d'une des premières solutions expérimentales (sur les maillages de Delaunay) ayant été menée sur un nombre aussi important de threads (128 à 240 PE). Ceci est en parti rendu possible par les moyens expérimentaux utilisés, à savoir des processeurs Intel Xeon-Phi. A titre de comparaison, la plupart des résultats du même ordre proviennent de solutions à base de tâches MPI (à granularité forte) – voir section 2.2, page 18.
En recherche appliquée, les résultats sont importants car la cohérence et la validité de la solution théorique proposée en dépend d'une part, et ils permettent également d'identifier les pistes d'améliorations et/ou recherche d'autre part.

B.3.4 Utilisation potentielle des travaux

Bien que conçues pour les maillages triangulaires de Delaunay, les solutions parallèles proposées peuvent être utilisées pour le maillage surfacique parallèle, ou être étendues en 3D (maillages tétraédrique de Delaunay).
En effet, la solution H_1 étant basée sur une décomposition de tâches, elle peut être aisément modifiée afin de supporter d'autres opérations/contraintes (anisotropie, carte de métrique pour l'adaptation, raffinement de tétraèdre, bascule de faces en 3D etc.). Par ailleurs,la solution H_2 peut être raffinée en utilisant une autre méthode de partitionnement, et couplée à H_1 pour obtenir une heuristique hybride multi-grain (threads + MPI), permettant d'obtenir une meilleure scalabilité.

B.3.5 Principales perspectives des travaux

Comme énoncé dans la conclusion (page 53), il faudrait analyser plus finement les résultats obtenus dans le chapitre 5 (page 48), et effectuer 2 autres cas-tests afin de consolider les résultats théoriques, et/ou d'identifier d'éventuels points de contention supplémentaires.
Faute de temps, la procédure de lissage n'a pas encore été implémentée et on projette de la coder dans les temps qui restent. En parallèle, on effectuera des tests de performances sur la triangulation parallèle, et on analysera les résultats.

1. Communément appelé HPC : *High Performance Computing*