



**UFC**

**UNIVERSIDADE FEDERAL DO CEARÁ  
DEPARTAMENTO DE COMPUTAÇÃO  
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**DANIEL NASCIMENTO TEIXEIRA**

**GERAÇÃO PARALELA DE MALHAS UTILIZANDO  
DECOMPOSIÇÃO A PRIORI**

**FORTALEZA, CEARÁ**

**2016**

## LISTA DE FIGURAS

Figura 1.1 Arquitetura de um Multiprocessador Simétrico (SMP).	9
Figura 1.2 Arquitetura de um MPP.	9
Figura 1.3 Arquitetura de uma NOW ou aglomerado de computadores.	9
Figura 1.4 Arquitetura de uma grade computacional.	10
Figura 1.5 <i>Cluster</i> Solaris da Sun Microsystems.	10
Figura 1.6 Arquiteturas paralelas: memória compartilhada à esquerda e memória distribuída à direita.	11
Figura 1.7 Arquiteturas paralela mista ou híbrida.	12
Figura 1.8 As 4 etapas do modelo de programação paralela Foster.	13
Figura 1.9 Agrupamento de tarefas para reduzir a comunicação e intensificar a computação.	14
Figura 1.10 Agrupamento de tarefas para concentrar a comunicação em um canal e intensificar a computação.	14
Figura 1.11 Vantagens e desvantagens na escolha da granularidade das tarefas.	15
Figura 1.12 Cenário de exemplo onde somente a <i>thread</i> A tem acesso a pilha objetos.	16

Figura 1.13 Exemplo de um semáforo controlando o acesso de 4 <i>threads</i> a recursos compartilhados.	16
Figura 1.14 Exemplo de um monitor controlando o acesso de 4 <i>threads</i> a determinados métodos.	17
Figura 2.1 Exemplo duas malhas triangulares, uma bidimensional e outra tridimensional.	22
Figura 2.2 Exemplo de malhas uma bidimensional que não é uma triangulação válida.	23
Figura 2.3 Exemplo de malha quadrilateral estruturada e não-estruturada triangular.	23
Figura 2.4 Exemplo de um elemento triangular de boa qualidade (esquerda) e dois de péssima qualidade (direita).	24
Figura 2.5 Avanço de fronteira (FREITAS, 2010).	25
Figura 2.6 a) Critério de Delaunay falhando para os dois triângulos. b) Triangulação válida respeitando o critério de Delaunay.	26
Figura 2.7 Triangulação por inserção de pontos (FREITAS, 2010).	27
Figura 2.8 Exemplo de uma decomposição espacial feita para renderização e teste de colisão. Fonte: <a href="http://togeckov.net/">http://togeckov.net/</a>	28
Figura 2.9 Uma subdivisão feita por <i>quadtree</i> com cinco níveis e sua representação em árvore.	29
Figura 2.10 Uma subdivisão feita por <i>octree</i> com três níveis e sua representação em árvore.	29

Figura 2.11 Uma subdivisão feita com BSP e sua representação em árvore. ....	30
Figura 2.12 Exemplo do método de divisão e conquistar de (VIDWANS A.; KALLINDERIS, 1994) para equilibrar a carga entre quatro processadores. (a) distribuição de carga inicial. (b) Distribuição de carga após o passo 1. (c) Distribuição de carga após o passo 2. ....	33
Figura 2.13 Passo a passo da técnica de (GAITHER, 1996). ....	33
Figura 2.14 O passo a passo da técnica de (WU POTING; HOUSTIS, 1996). ....	34
Figura 2.15 Exemplo da formação dos subdomínios em (SAID, 1999). Fronteira de entrada, grade auxiliar inicial e 6 subdomínios gerados juntamente com as suas discretizações respectivamente. ....	35
Figura 2.16 As três formas de particionar. 1 - planos equidistantes. 2 - volume dos subdomínios iguais. 3 - centro de massa (IVANOV; ANDRÄ; KUDRYAVTSEV, 2006). ....	35
Figura 2.17 Exemplo do processo de particionamento em (JURCZYK TOMASZ; GŁUT, 2007). ....	36
Figura 2.18 Exemplo de um particionamento feito por (ANDRÄ, 2008) para 8 subdomínios. ....	37
Figura 2.19 Os principais passos da técnica de (PIRZADEH; ZAGARIS, 2009) para gerar os segmentos de interface. ....	38
Figura 2.20 Da esquerda para direita: todas as arestas que sofrem interseção, a fronteira inicial, e a fronteira suavizada. Em (CHEN, 2012). ....	39

Figura 2.21 Passos da técnica baseada na malha de superfície. (a) malha de superfície; (b) corte; (c) seção transversal; (d) malha final (GLUT; JURCZYK, 2008). ....	39
Figura 2.22 Passos da técnica baseada na malha volumétrica grosseira. (a) malha volumétrica grosseira; (b) refinamento da seção transversal; (c) seção transversal; (d) malha final (GLUT; JURCZYK, 2008). ....	40
Figura 2.23 Decomposição em 4 subdomínios de uma malha multiconectada na técnica de (FARHAT, 1988). ....	40
Figura 2.24 Fluxograma da triangulação em paralelo (BARNARD STEPHEN T.; SIMON, 1994). ....	41
Figura 2.25 Oito subdomínios criados com quantidades iguais de elementos e do lado direito a otimização das partições em (NIKISHKOV, 1999). ....	41
Figura 2.26 Técnica de (LÖHNER, 2001). ....	42
Figura 2.27 Junção das malhas dos subdomínios em (LOHNER, 2014). ....	43
Figura 2.28 Regiões de corte inválidas em cinza (LARWOOD et al., 2003). ....	43
Figura 2.29 Malha de elementos finitos (esquerda) e a partição da malha com sua representação em grafo (à direita) em (CHARMPIS DIMOS C.; PAPADRAKAKIS, 2005). ....	44
Figura 2.30 Construção do grafo de particionamento de (LINARDAKIS; CHRISOCHOIDES, 2006). ....	45
Figura 2.31 Malha tetraédrica inicial, malha de superfície refinada nos subdomínios e melhorias nas faces de superfície no trabalho de (ITO et al., 2007). ....	45

Figura 2.32 Decomposição feita pela técnica de (PANITANARAK THAP; SHONTZ, 2011). 45

Figura 2.33 Pontos organizados em células. À esquerda por partição regular e à direita por *kd-tree* (LO, 2012). 46

Figura 2.34 Fluxograma da triangulação em paralelo (LO, 2012). 47

Figura 2.35 Malha gerada, espalhada entre os processos escravos, e as células da *quadtree* de decomposição deslocadas para a direção +X (FREITAS et al., 2013). 47

Figura 2.36 Passos da geração da malha no trabalho de (FREITAS; CAVALCANTE-NETO; VIDAL, 2014). Cada cor representa a malha gerada por um processador. 48

## SUMÁRIO

<b>1 Tema Secundário I : Programação Paralela</b>	<b>8</b>
<b>1.1 Computação de Alto Desempenho</b> .....	8
1.1.1 Classes de Arquiteturas Existentes .....	8
1.1.2 Paradigmas de programação paralela - Modelos .....	11
<b>1.2 Metodologia da programação paralela</b> .....	12
1.2.1 Decomposição .....	13
1.2.2 Comunicação .....	13
1.2.3 Aglomeração .....	14
1.2.4 Mapeamento .....	15
<b>1.3 Problemas na programação paralela</b> .....	15
1.3.1 Seção Crítica .....	15
1.3.1.1 Semáforos .....	16
1.3.1.2 Monitores .....	17
<b>1.4 Métricas de Desempenho</b> .....	18
1.4.1 Desempenho e Escalabilidade .....	19
<b>1.5 Ambientes para programação paralela</b> .....	19
<b>2 Tema Secundário II : Particionamento de Modelos para geração Malha em paralelo</b>	<b>21</b>
<b>2.1 Conceitos e Definições</b> .....	21
2.1.1 Malhas Triangulares .....	21
2.1.2 Geração de Malha .....	24
2.1.2.1 Avanço de Fronteira .....	24
2.1.2.2 Delaunay .....	26
2.1.2.3 Arbitrária .....	27

2.1.3	Estrutura de Dados .....	28
2.1.3.1	<i>Quadtree</i> .....	28
2.1.3.2	<i>Octree</i> .....	29
2.1.3.3	<i>Binary Space Partitioning</i> (BSP) .....	29
2.1.4	Estimativa de Carga .....	30
2.1.5	Particionamento de Malhas .....	31
2.1.6	Balanceamento de carga .....	31
<b>2.2</b>	<b>Particionamento Baseado na Geometria</b> .....	32
<b>2.3</b>	<b>Particionamento Baseada em Estruturas de Dados</b> .....	40
<b>2.4</b>	<b>Considerações Finais</b> .....	49
	<b>Referências Bibliográficas</b>	<b>50</b>

# 1 TEMA SECUNDÁRIO I : PROGRAMAÇÃO PARALELA

## 1.1 Computação de Alto Desempenho

O termo Computação de Alto Desempenho ou HPC (do inglês *High-performance computing*) refere-se à prática de agregar o poder computacional de uma forma que proporciona um desempenho muito superior do que se poderia obter de um computador de mesa, a fim de resolver os grandes problemas da ciência, engenharia, ou de negócios. O uso eficiente desses recursos é o principal foco de estudo nessa área.

Em decorrência disso, criou-se a possibilidade de resolver problemas mais complexos como tratamento de conjuntos de imagens, biologia computacional, mineração de dados, simulação de modelos científicos e de engenharia entre outros.

### 1.1.1 Classes de Arquiteturas Existentes

Uma aplicação paralela pode ser composta por uma ou mais tarefas. As tarefas que compõem uma aplicação paralela podem executar em vários processadores, caracterizando desta forma o paralelismo da execução da aplicação. Os tipos de processadores utilizados por uma determinada aplicação e o meio de comunicação entre eles caracterizam a classe de arquitetura de execução da aplicação.

Podemos agrupar as arquiteturas hoje existentes em cinco grandes grupos: SMPs, MPPs, NOWs, Grades computacionais e *Clusters*.

- SMPs (*Symmetric multi-processing* ou multiprocessadores simétricos)

São máquinas em que vários processadores compartilham a mesma memória ((HWANG; XU, 1998)). A Figura 1.1 mostra um exemplo de uma SMPs.

- MPPs (*Massively parallel processing* ou processadores maciçamente paralelos)

São compostos por vários nós (processador e memória) independentes, interconectados por redes dedicadas e de alta velocidade. A Figura 1.2 mostra um exemplo de uma MPPs.

- NOWs (*Network of workstations* ou redes de estações de trabalho) ou aglomerados de computadores

São um conjunto de estações de trabalho ou PCs, ligados por uma rede local. As NOWs são arquiteturalmente semelhantes aos MPPs. A principal diferença entre NOWs e MPPs

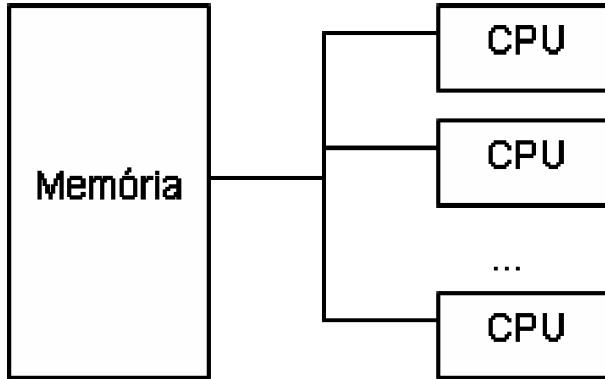


Figura 1.1: Arquitetura de um Multiprocessador Simétrico (SMP).

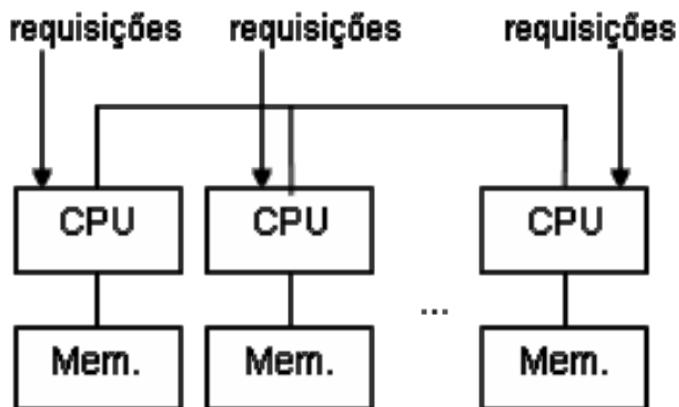


Figura 1.2: Arquitetura de um MPP.

é que os nós que compõem uma MPP tipicamente são conectados por redes desenvolvidas especificamente para o MPP, enquanto uma NOW é composta por equipamentos de rede e processadores tradicionais das lojas. A Figura 1.3 mostra um exemplo de uma NOWs.

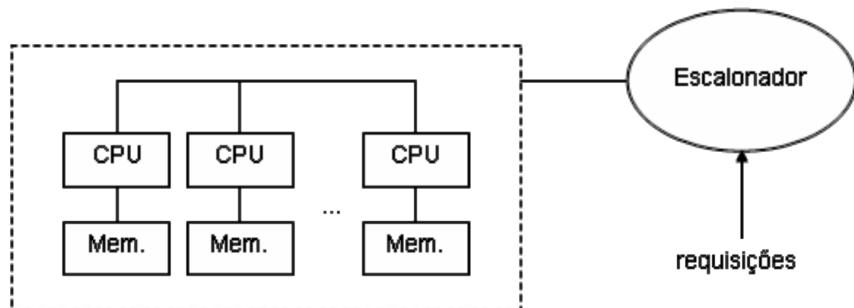


Figura 1.3: Arquitetura de uma NOW ou aglomerado de computadores.

- Grades computacionais

São a expansão das NOWs, onde os componentes de uma grade não se restringem a processadores, podendo ser SMPs, MPPs e PCs, como também outros dispositivos digitais. Todos estes dispositivos estão espalhados geograficamente e estão conectados através da internet. A Figura 1.4 mostra um exemplo de uma grade computacional.

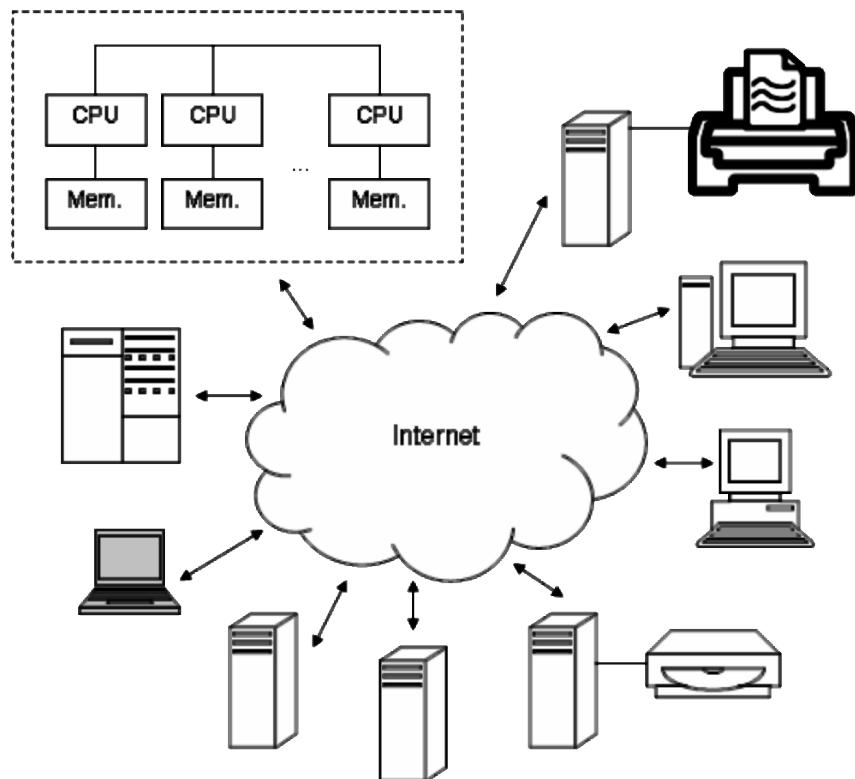


Figura 1.4: Arquitetura de uma grade computacional.

- *Clusters*

Diversos processadores que estão fisicamente organizados em uma mesma máquina (ou, pelo menos, em uma mesma sala) e conectados através de uma rede comum ou de alta velocidade (*ethernet* ou *infiniband* são as mais difundidas atualmente). A Figura 1.5 mostra um exemplo de *cluster*.



Figura 1.5: *Cluster* Solaris da Sun Microsystems.

### 1.1.2 Paradigmas de programação paralela - Modelos

Em programas que executam sequencialmente não existe a preocupação que uma dada posição de memória seja alterada no mesmo tempo que ela esteja sendo lida. Em computação paralela há essa preocupação e existem várias técnicas para manter a ordem de leitura e escrita na memória. Basicamente há três tipos de paradigmas:

- Memória Compartilhada

Engloba basicamente os sistemas UMA (*Uniform Memory Access*), ou seja, o acesso à memória é feito de forma uniforme através de endereçamento direto. Assim, todos os processadores de um computador compartilham um mesmo espaço de memória e para isso deve haver um controle na leitura e escrita na memória (à direita da Figura 1.6).

- Memória Distribuída

Cada um dos processadores têm acesso a um espaço único de endereçamento de memória privada. Cada módulo da memória pode ser acessado diretamente por apenas um dos processadores (à esquerda da Figura 1.6). A comunicação entre os processos ocorre através de troca de mensagens.

- Mista ou híbrida

Ocorre quando um modelo de memória distribuída é criado utilizando processadores multinúcleos, ou seja, temos uma arquitetura de memória distribuída com vários núcleos acessando um mesmo espaço de memória. A Figura 1.7 ilustra esse tipo de paradigma.

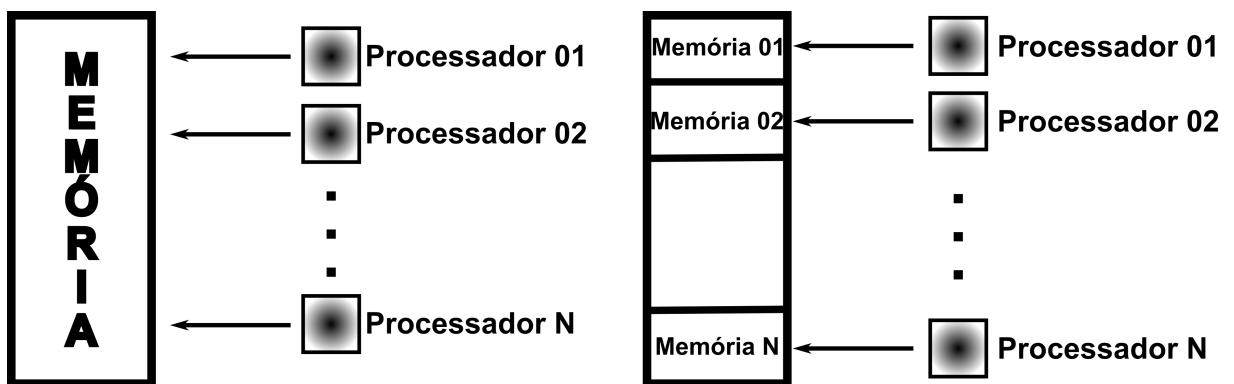


Figura 1.6: Arquiteturas paralelas: memória compartilhada à esquerda e memória distribuída à direita.

Ao se afirmar que um programa roda em um dado modelo teórico de paradigmas não necessariamente corresponde a estrutura física da máquina que está sendo executado o programa.

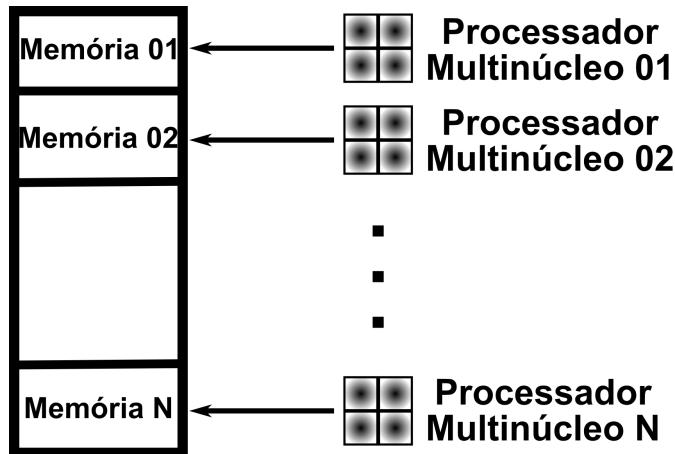


Figura 1.7: Arquiteturas paralela mista ou híbrida.

Por exemplo, uma máquina com arquitetura distribuída pode executar programas com memória compartilhada através de uma emulação via *software*, assim como o contrário possa acontecer, uma máquina com estrutura de memória compartilhada executar um programa com memória distribuído.

## 1.2 Metodologia da programação paralela

Apesar das arquiteturas paralelas serem atualmente uma realidade, a programação paralela continua a ser uma tarefa bastante complexa quando comparada a programação sequencial. Alguns dos problemas que surgiram com a programação paralela foram::

- Concorrência: identificar as partes da computação que podem ser executadas em simultâneo.
- Comunicação e Sincronização: desenhar o fluxo de informação de modo a que a computação possa ser executada em simultâneo pelos diversos processadores evitando situações de *deadlock* e *race conditions*.
- Balanceamento de Carga e Escalonamento: distribuir de forma equilibrada e eficiente as diferentes partes da computação pelos diversos processadores de modo que todos os processadores possam ficar ocupados durante toda a execução, evitando assim processadores ociosos.

Um dos métodos mais conhecidos para estruturar algoritmos paralelos é a metodologia de (FOSTER, 1995). Este modelo permite que o programador se concentre inicialmente nos aspectos não-dependentes da arquitetura, como sejam a concorrência e a escalabilidade, e só

depois considere os aspectos dependentes da arquitetura, como sejam aumentar a localidade e diminuir a comunicação da computação.

O modelo de programação de Foster é composto por 4 etapas: Decomposição; Comunicação; Aglomeração e Mapeamento. A Figura 1.8 ilustra as etapas do modelo Foster.

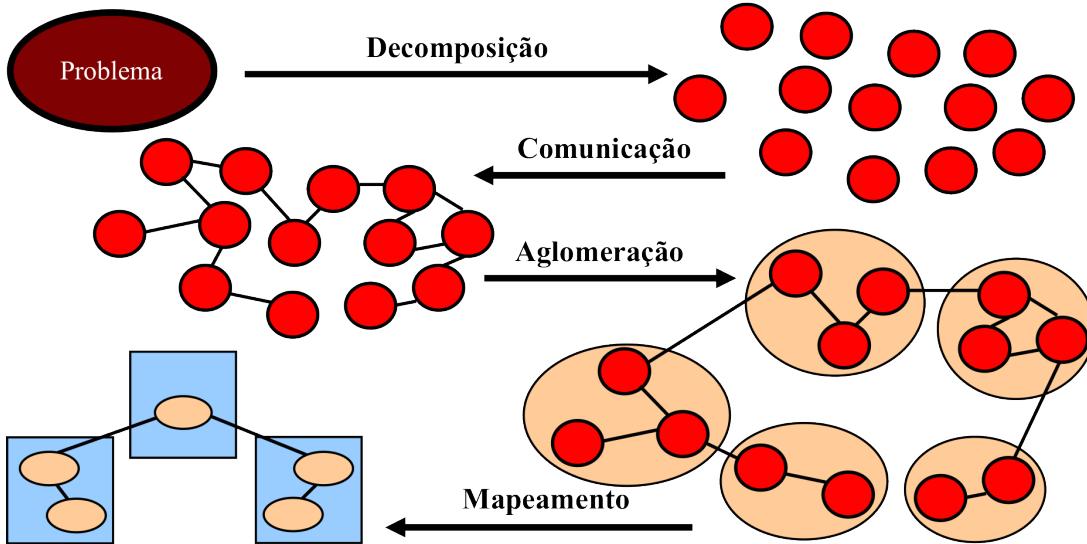


Figura 1.8: As 4 etapas do modelo de programação paralela Foster.

### 1.2.1 Decomposição

Uma forma de diminuir a complexidade de um problema é conseguir dividi-lo em tarefas mais pequenas. Para isso existem duas estratégias principais de decompor um problema:

- **Decomposição do Domínio:** decompor o problema em função dos dados. Ou seja, os dados de entrada serão decompostos em entradas menores.
- **Decomposição Funcional:** decompor o problema em função da computação. Ou seja, são criadas diversas tarefas para serem executadas entre os processadores.

Vale lembrar que uma boa decomposição tanto divide os dados como a computação em múltiplas tarefas mais pequenas, tendo assim maior proveito do paralelismo.

### 1.2.2 Comunicação

A natureza do problema e o tipo de decomposição determinam o padrão de comunicação entre as diferentes tarefas. A execução de uma tarefa pode envolver a sincronização/acesso a dados pertencentes/calculados por outras tarefas.

Para haver cooperação entre as tarefas é necessário definir algoritmos e estruturas de dados que permitam uma eficiente troca de informação assim como a sincronização dos dados. Existem dois padrões de comunicação:

- Comunicação Síncrona: As tarefas executam de forma coordenada e sincronizam na transferência de dados. A comunicação somente é encerrada quando as duas tarefas estão sincronizadas.
- Comunicação Assíncrona: As tarefas executam de forma independente não necessitando de sincronizar para transferir dados. Assim, o envio das mensagens não interfere com a execução do emissor.

### 1.2.3 Aglomeração

Aglomeração é o processo de agrupar um conjunto de tarefas em uma tarefa maior, diminuindo assim os custos de implementação do algoritmo paralelo e os custos de comunicação entre as tarefas.

O agrupamento de tarefas elimina os custos de comunicação entre essas tarefas e aumenta a granularidade da computação (Figura 1.9).

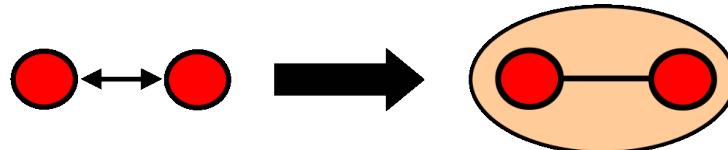


Figura 1.9: Agrupamento de tarefas para reduzir a comunicação e intensificar a computação.

O agrupamento de tarefas com pequenas comunicações individuais em tarefas com comunicações maiores permite aumentar a granularidade das comunicações e reduzir o número total de comunicações (Figura 1.10).

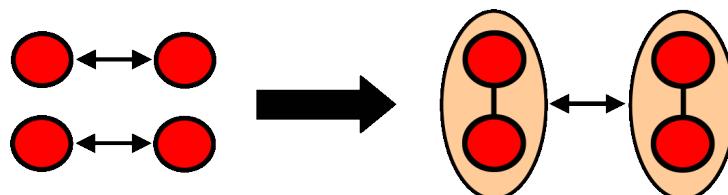


Figura 1.10: Agrupamento de tarefas para concentrar a comunicação em um canal e intensificar a computação.

### 1.2.4 Mapeamento

Mapeamento ou Balanceamento de Carga é a fase de atribuir tarefas a processadores de modo a reduzir o tempo ocioso e minimizar a comunicação entre os processadores. O balanceamento de carga pode ser estático (em tempo de compilação) ou dinâmico (em tempo de execução).

Dependendo da granularidade das tarefas o balanceamento pode ser mais preciso. Com uma granularidade fina, o balanceamento de carga é mais eficiente, já com uma granularidade grossa é mais difícil conseguir uma boa eficiência. É preciso analisar bem o problema antes de escolher uma estratégia (Figura 1.11).

	Balanceamento de carga	Tempo de Computação	Melhora do Desempenho
Granularidade Fina			
Granularidade Grossa			

Figura 1.11: Vantagens e desvantagens na escolha da granularidade das tarefas.

## 1.3 Problemas na programação paralela

### 1.3.1 Seção Crítica

É um trecho de um algoritmo que acessa recursos compartilhados por outros processos, que deve ser executado por somente um processo por vez. O objetivo é tornar as operações desta seção sobre os recursos compartilhados atômica. Isso evita que dois ou mais processos acessem ou alterem valores no mesmo espaço de memória, podendo resultar na invalidez de certos dados e comprometendo a corretude do programa. Este problema é um acontece somente em arquiteturas compartilhadas. A Figura 1.12 ilustra uma seção crítica para três *threads* tentando acesso a uma pilha de objetos.

A corretude de um programa depende das seguintes propriedades da seção crítica:

- Exclusão mútua

Instruções dentro da seção crítica não podem ser executadas por dois ou mais processos ou *threads* ao mesmo tempo, isto é, devem todas ser executadas por somente um processo do início ao fim da seção crítica.

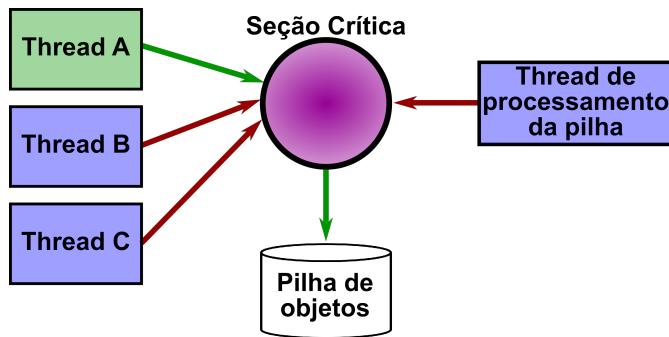


Figura 1.12: Cenário de exemplo onde somente a *thread A* tem acesso a pilha objetos.

- Inexistência de *deadlock*

Dois ou mais processos não podem ficar impedidos de continuar suas execuções na espera de um pelo outro. Deve ser garantido que um deles deve conseguir ser executado.

- Inexistência de inanição

Todos os processos que estão esperando para entrar na seção crítica, devem, em algum momento no futuro, conseguir ser executado.

Existem duas classes de algoritmos de permissão que são utilizados para garantir as três propriedades acima descritas. As duas classes de algoritmos são semáforos e monitores, que resolvem trivialmente o problema da seção crítica, mas podem também ser utilizados para outros propósitos na computação.

### 1.3.1.1 Semáforos

Semáforos são tipos de dados protegidos contendo um inteiro, que determina quantos processos (no máximo) podem executar um certo trecho de código, e uma lista de processos ou *threads* em espera para executar aquele trecho de código. A Figura 1.13 ilustra um semáforo para 4 *threads*.

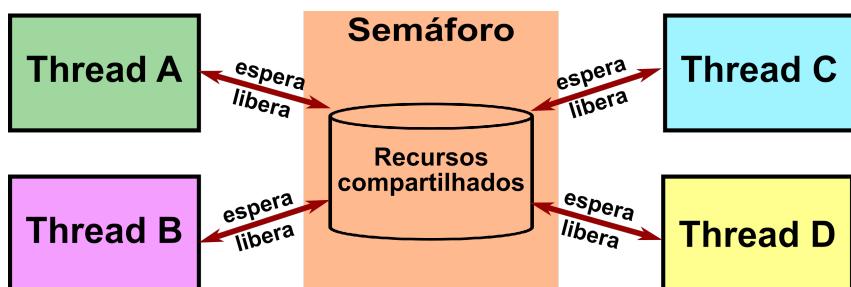


Figura 1.13: Exemplo de um semáforo controlando o acesso de 4 *threads* a recursos compartilhados.

Se for necessário que somente um processo execute em certo trecho de código, deve-se definir o inteiro do semáforo como 1, este semáforo também é chamado de mutex (mutual exclusion).

Além disso, são definidas duas operações atômicas, de **espera** e de **liberação**, que são chamados no início e no final do trecho de código de interesse, respectivamente. Diz-se que uma operação é atômica quando ela é feita de uma vez só, com auxílio de hardware. Assim, os semáforos devem ter suporte do sistema operacional utilizado.

Na operação de espera, é feito um teste de permissão, ou seja, se o número de processos executando o trecho de código de interesse é menor que o número máximo permitido. Se passar no teste, o processo executa o trecho de código de interesse. Caso não passe no teste, o processo entra em espera. Quando um processo termina o trecho de código de interesse, a função libera é chamada, liberando um processo em espera para executar o mesmo trecho.

### 1.3.1.2 Monitores

Monitores são estruturas que sincronizam *threads*, permitindo que duas *threads* possam executar ou esperar para acessar uma determinada região até que uma certa condição seja verdadeira. Este monitor define as operações possíveis de serem feitas nesse objeto, operações estas que são executadas de forma exclusiva. Além disso, operações de monitores diferentes podem ser executadas simultaneamente. A Figura 1.14 ilustra um monitor para 4 *threads*.

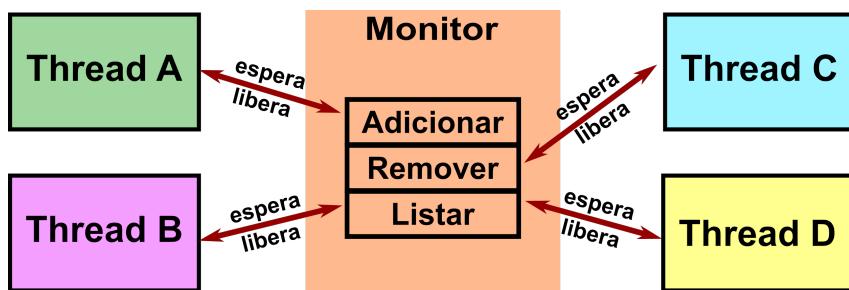


Figura 1.14: Exemplo de um monitor controlando o acesso de 4 *threads* a determinados métodos.

É importante mencionar que é possível implementar monitores utilizando semáforos, e é também possível simular semáforos utilizando monitores. Assim, os dois tipos têm a mesma capacidade de expressão.

## 1.4 Métricas de Desempenho

Ao utilizar uma aplicação paralela, surge o interesse em saber o ganho de velocidade quando comparado a uma aplicação sequencial. Para isso existem algumas métricas utilizadas:

- Escalabilidade

É a propriedade de um sistema que lhe confere a capacidade de aumentar seu desempenho sob uma determinada carga, quando mais recursos (processadores) são acrescentados a esse sistema. Ou seja, pode-se falar que um algoritmo é escalável se ele pode ser utilizado em uma grande quantidade de processadores sem que aconteça uma queda em sua velocidade.

Pode-se dizer que um sistema é escalável quando ele resolve um problema de magnitude  $\gamma$  com um recurso  $R$ , e consegue resolver um problema de magnitude  $n\gamma$  com um recurso  $nR$ . Ou seja, sempre que aumentar os recursos computacionais, aumentará proporcionalmente a capacidade de resolver problemas maiores.

- *Speed-up*

Esta métrica mostra quantas vezes um programa paralelo é mais rápido que um serial. Para obter um *speed-up* linear tem-se que obter um programa com tempo de execução  $x$  vezes mais rápido quando aumentado em  $x$  o número de processadores. Já um *speed-up* super linear seria obter um ganho maior que  $x$  quando aumentado em  $x$  o número de processadores.

O *speed-up*  $S$  para  $p$  processadores é calculado pela seguinte formula:  $S(p) = T_s/T_p$ , onde  $T_s$  é o tempo de execução do programa sequencialmente e  $T_p$  é o tempo do programa executando em paralelo para  $p$  processadores.

Na prática um *speed-up* linear é difícil de se obter. À medida que a quantidade de processadores aumentam, a comunicação entre os processos aumenta e isso faz o tempo de execução cair, derrubando assim o *speed-up*.

- Eficiência

Outra medida importante é a eficiência, que trata da relação entre o *speed-up* e o número de processadores. Ela retorna a porcentagem de tempo para o qual um processador foi empregado de forma útil, ou seja, qual a porcentagem de tempo que foi utilizada com a computação do algoritmo. A eficiência é definida como a razão do *speed-up* pelo número de processadores. Em um sistema ideal, onde o *speed-up* é linear, a eficiência é de 100

#### 1.4.1 Desempenho e Escalabilidade

Dois dos principais objetivos no desenvolvimento de aplicações paralelas são o um bom desempenho e uma boa escalabilidade.

O desempenho é a capacidade de reduzir o tempo de resolução do problema à medida que os recursos computacionais aumentam. Já a escalabilidade é a capacidade de aumentar o desempenho à medida que a complexidade do problema aumenta. Então é preciso se preocupar tanto com o desenvolvimento de um bom algoritmo como com a sua paralelização.

Os factores que condicionam o desempenho e a escalabilidade de uma aplicação são os limites arquiteturais e algorítmicos do problema.

- Limites Arquiteturais

Latência e Largura de Banda - são respectivamente a quantidade de tempo necessário para transitar entre dois nós da rede e a quantidade dados que podem ser transmitidos em paralelo ao longo de um caminho.

Coerência dos Dados - diz respeito se o sistema faz uso de barramentos ou redes diferentes.

Capacidade de Memória - o quanto de informação a memória RAM pode armazenar durante a execução de uma programa.

- Limites Algorítmicos

Falta de Paralelismo (código sequencial/concorrência) - alguns algoritmos são naturalmente sequenciais, tendo a sua versão paralela inviável.

Frequência de Comunicação e Sincronização - a quantidade de vezes que é preciso realizar uma comunicação ou sincronização em um algoritmo.

Escalonamento Deficiente (granularidade das tarefas/balanceamento de carga) - o tamanho das tarefas geradas pelo algoritmo pode inviabilizar o desempenho na execução paralela.

#### 1.5 Ambientes para programação paralela

Programar paralelamente varia de acordo com o tipo de arquitetura utilizado e não é uma tarefa trivial. Já existem diversas soluções para a programação eficiente para as mais diversas arquiteturas. Por outro lado, diversos trabalhos tem buscado incorporar mais um nível de paralelismo através de recursos de linguagem de programação, que podem abstrair as diversas camadas de um *cluster* por exemplo.

Existem várias interfaces ou *Application Programming Interfaces* (APIs) de programação paralela, que são bastante específicas em relação a um determinado nível de paralelismo. Neste trabalho irá considerar dois padrões de programação paralela:

- Programação Paralela com *Threads*

Multi-core:

Utiliza-se o padrão *Posix Threads*, como forma de criar e manipular processos leves (*threads*) (ANDREWS, 1999). As bibliotecas que implementam a *Posix threads* são chamadas Pthreads.

Multiprocessadores:

Faz-se uso da biblioteca OpenMP, que permite a chamada de operações paralelas de forma simples, por isso tem-se firmado como a principal forma de se fazer programação paralela com *threads* (CHANDRA, 2001).

- Programação Paralela com Passagem de Mensagens

Multicomputadores:

Para a comunicação inter-processos são adotados os padrões *Message Passing Interface* (MPI), especialmente nas linguagens Fortran, C e C++, além de possuir seus próprios tipos de dados e funções que definem comunicação ponto-a-ponto, síncrona e assíncrona; comunicação coletiva, como barreira, difusão (broadcast), dispersão (scatter), junção (gather) e redução, entre outras; agrupamentos e contextos de processos; e definição de topologia da rede (GROPP et al., 1996).

Para os utilizadores de Java existe a *Java Remote Method Invocation* (Java RMI), o qual permite a chamada remota de métodos, garantindo a execução distribuída de uma aplicação (FARLEY, 1998).

## 2 TEMA SECUNDÁRIO II : PARTICIONAMENTO DE MODELOS PARA GERAÇÃO MALHA EM PARALELO

Todo desenvolvimento de um programa paralelo tem que passar por três fases que são: particionamento da entrada, agrupamento das partições e mapeamento dos agrupamentos. Estes conceitos partem da ideia que para um bom algoritmo paralelo executar, é preciso uma boa estratégia de divisão ou particionamento da entrada para ao final ser feita a junção das várias soluções.

Há ainda a preocupação com a distribuição das tarefas entre os processadores, levando em conta que, ao final, todos eles devam ter realizado uma quantidade similar de processamento, evitando que alguns processadores fiquem ociosos enquanto outros estão sobrecarregados. Se isto acontecer, significa que a carga foi devidamente balanceada entre os processadores. Para isto ocorrer é preciso que tenha sido realizado uma etapa de estimativa de trabalho computacional ou estimativa de carga.

Neste capítulo, são apresentados conceitos necessários para o entendimento de um particionamento de modelos para geração de malha. Isso envolve saber o que é uma malha, como estimar o trabalho computacional ou carga que um modelo geraria para gerar a sua malha, realizar a decomposição ou particionamento de um modelo para a execução em paralelo e estratégias de平衡amento de carga entre processadores. Também será apresentado alguns trabalhos que fazem particionamento de modelos, dando ênfase no modo que é tratado o particionamento dos modelos de entrada e como foi feita a estimativa da carga.

### 2.1 Conceitos e Definições

#### 2.1.1 Malhas Triangulares

Existem diversos tipos de malhas, entre as mais conhecidas e utilizadas estão as malhas que utilizam triângulos. Por ser a menor estrutura geométrica que consegue representar modelos tanto no espaço bidimensional como no tridimensional, as malhas triangulares (Figura 2.1) se tornaram as mais utilizadas nas pesquisas da área de Geometria Computacional. Nos casos tridimensionais estas malhas são chamadas de malhas tetraédricas.

Triangulações ou tetraedralizações são muitas vezes chamadas de malhas ou usadas como malhas, como no caso do método dos elementos finitos (MEF), já que malha é uma união de elementos, que podem ser triângulos, por exemplo. Pode-se definir uma malha  $M$  de uma maneira genérica como:

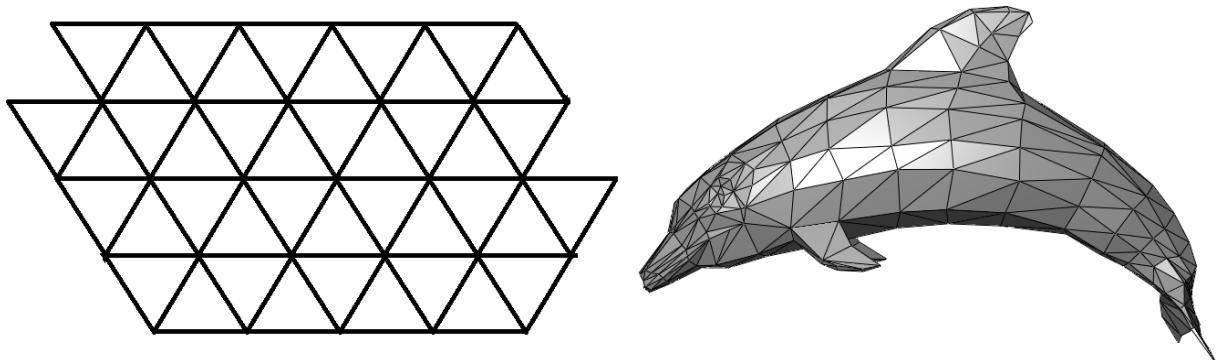


Figura 2.1: Exemplo duas malhas triangulares, uma bidimensional e outra tridimensional.

- $\Omega = \bigcup_{k \in M} k$ , onde  $\Omega$  é um domínio finito limitado,
- O interior de cada elemento  $k$  em  $M$  é não vazio,
- A interseção do interior de dois elementos de  $M$  é vazia.

Uma triangulação também é uma malha, mas nem toda malha é uma triangulação (Figura 2.2). Uma tetraedralização deve respeitar as mesmas regras de uma triangulação, logo uma tetraedralização também é uma malha válida. Neste trabalho quando for mencionado malha, será sempre a malha que respeita as mesmas propriedades de uma triangulação. A definição de triangulação segundo (DæHLEN; HJELLE, 2006) diz que:

- Nenhum triângulo pertencente à triangulação pode ter pontos colineares.
- A interseção do interior de quaisquer dois triângulos pertencentes à triangulação é vazia.
- As bordas de 2 triângulos quaisquer só podem fazer interseção com vértices ou arestas.
- A união de todos os triângulos da triangulação é igual ao domínio.
- O domínio deve ser conectado.
- Não deve existir buracos na triangulação, a menos que eles sejam definidos como entrada.
- Se um triângulo está na borda da triangulação então ele faz interseção por aresta no máximo dois triângulos.

Existem malhas de diferentes geometrias e dimensões. Caso a topologia de elementos e vértices da malha siga alguma regra simples de indexação, essa malha será definida como estruturada, caso contrário, ela será classificada como não-estruturada. Nas malhas estruturadas o conhecimento dos vizinhos de cada elemento não depende do armazenamento ou existência

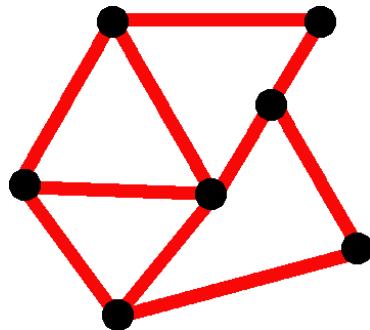


Figura 2.2: Exemplo de malha uma bidimensional que não é uma triangulação válida.

desta informação. Já nas não-estruturadas, para se ter conhecimento dos vizinhos, é necessário armazenar ou calcular estas informações. Existem ainda as malhas mistas, que combinam malhas estruturadas e não-estruturadas (Figura 2.3).

As malhas também podem ser classificadas de acordo com a geometria dos seus elementos, já que não são necessariamente formadas somente de triângulos.. Para malhas bidimensionais, por exemplo, elas podem ser de elementos triangulares ou quadrilaterais, por exemplo, como mostra a Figura 2.3.

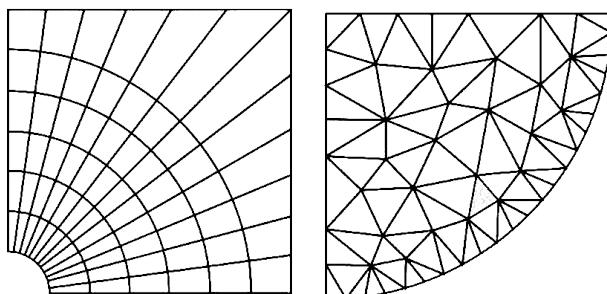


Figura 2.3: Exemplo de malha quadrilateral estruturada e não-estruturada triangular.

Para muitas aplicações, a qualidade dos elementos da malha é muito importante. Para classificar um elemento de uma malha triangular como bom ou ruim pode-se utilizar, dentre outras, uma métrica que é definida como  $\alpha = 2R_i/R_c$ , onde  $R_i$  e  $R_c$  são os raios dos círculos inscrito e circunscrito, respectivamente.

Esta métrica  $\alpha$  tem valor 1,0 para um triângulo equilátero. Quanto pior a qualidade do elemento, mais próximo de 0,0 é o valor de  $\alpha$ . Pode-se dizer que os elementos com  $\alpha \leq 0,1$  são de péssima qualidade e que os elementos com  $\alpha \geq 0,7$  são de boa qualidade, como mostra a Figura 2.4.

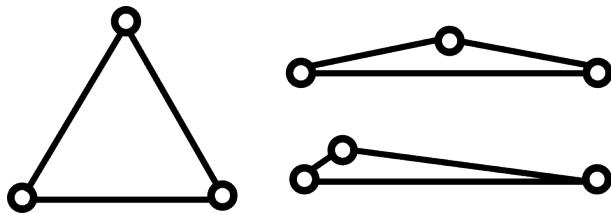


Figura 2.4: Exemplo de um elemento triangular de boa qualidade (esquerda) e dois de péssima qualidade (direita).

### 2.1.2 Geração de Malha

Nesta seção, são apresentadas as técnicas de geração de malhas triangulares mais conhecidas atualmente. Existem diversos algoritmos para geração de malhas, porém eles podem ser enquadrados uma das categorias a seguir:

- Avanço de fronteira, técnica em que a malha é gerada a partir da borda da região;
- Delaunay, técnica em que a malha é gerada procurando-se maximizar o menor ângulo dos triângulos gerados para um dado conjunto de pontos;
- Arbitrária, técnica em que a malha é gerada de maneira diferente das anteriores.

#### 2.1.2.1 Avanço de Fronteira

Este é um dos métodos mais populares de geração de malhas e consiste em criar os elementos no interior do domínio progressivamente a partir de um contorno, especificando a região a ser preenchida (Figura 2.5a). Este contorno é chamado de fronteira inicial ou borda. Os elementos são gerados a partir dessa fronteira dada como entrada. Uma fronteira bidimensional é formada por um conjunto de arestas.

À medida que o algoritmo progride, a fronteira avança em direção ao interior, sempre removendo ou adicionando elementos de fronteira até que todo o domínio seja preenchido. O algoritmo chega ao fim quando não há mais fronteira, ou seja, o domínio foi totalmente triangularizado.

Há casos em que o algoritmo não consegue mais gerar elementos para uma determinada fronteira, isso indica que o algoritmo falhou. O caso de falha ocorre quando todos os possíveis elementos a serem criados se sobreponem a um elemento já existente. Por isso, é importante verificar se elementos se interceptam. Os casos de falha geralmente acontecem em modelos tridimensionais. Entretanto, já existem técnicas para contornar esses problemas e gerar malhas em modelos que falhariam.

Para gerar os novos triângulos no interior do domínio, é necessário criar novos pontos que não pertencem aos dados de entrada. Em geral, são utilizados os pontos de Steiner para isso (RUPPERT, 1999).

Um algoritmo de avanço de fronteira procede da seguinte maneira no caso 2D (Figura 2.5):

1. Selecione uma aresta da fronteira, a aresta base (fig. 2.5b);
2. Encontre um ponto ideal para a formação de um novo triângulo com a aresta base (fig. 2.5c);
3. Crie uma região de busca em torno desse ponto ideal (fig. 2.5d);
4. Selecione o ponto dentro dessa região de busca cujo triângulo (entre esse ponto e a aresta base) seja válido e seja o de melhor qualidade, que pode ser um novo ponto ou um ponto já pertencente à malha;
5. Forme o novo triângulo com o ponto selecionado e adicione-o à malha (fig. 2.5e);
6. Atualize a fronteira, inserindo as arestas que foram criadas e removendo as arestas que já existiam;
7. Se existir aresta na fronteira, volte para o passo 1.

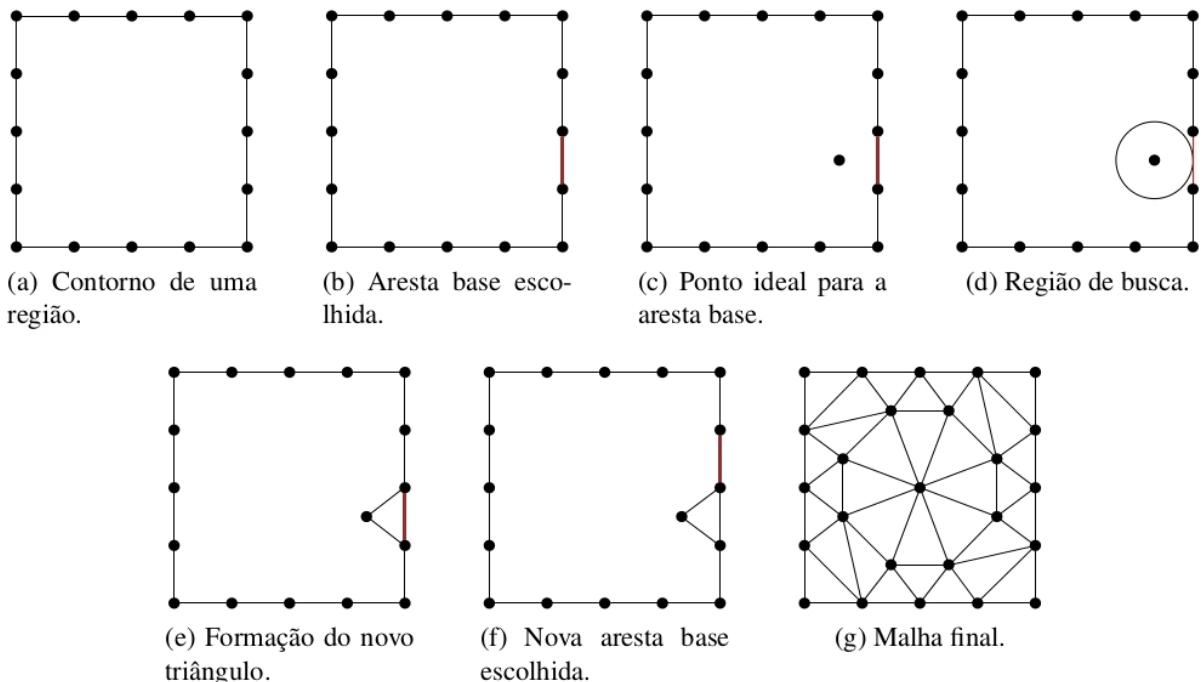


Figura 2.5: Avanço de fronteira (FREITAS, 2010).

Pelo fato da fronteira ser sempre respeitada, os algoritmos de avanço de fronteira têm facilidade em tratar regiões descontínuas, ou por conterem buracos, ou por serem regiões separadas. Como os elementos mais próximos da borda são gerados primeiro, em geral, eles têm uma boa qualidade. A boa qualidade da malha gerada provê estabilidade e precisão à aplicação de métodos numéricos (como os métodos dos elementos finitos).

Porém, nem sempre todos os elementos gerados têm boa qualidade. Ao contrário dos elementos mais próximos da borda, os elementos mais internos à malha nem sempre têm boa qualidade devido à região tornar-se menor à medida que a fronteira avança. Geralmente uma técnica de suavização ou otimização é aplicada na malha resultante do algoritmo para tratar esses casos.

### 2.1.2.2 Delaunay

Esta é uma técnica bastante conhecida na área de geração de malhas, cujo nome é uma homenagem ao matemático russo Boris Delaunay. A entrada para esse problema é um conjunto de pontos e, geralmente, não são utilizados os pontos de Steiner para formar os triângulos.

O critério de Delaunay para a formação dos triângulos é que não exista nenhum outro ponto dentro do círculo que passa pelos três pontos desse triângulo (seu circuncírculo), critério este também chamado de "esfera vazia" (o circuncírculo desse triângulo, Figura 2.6). O critério de Delaunay em si não se constitui num método de geração de malhas, mas é uma forma de saber onde os pontos devem estar localizados no espaço.

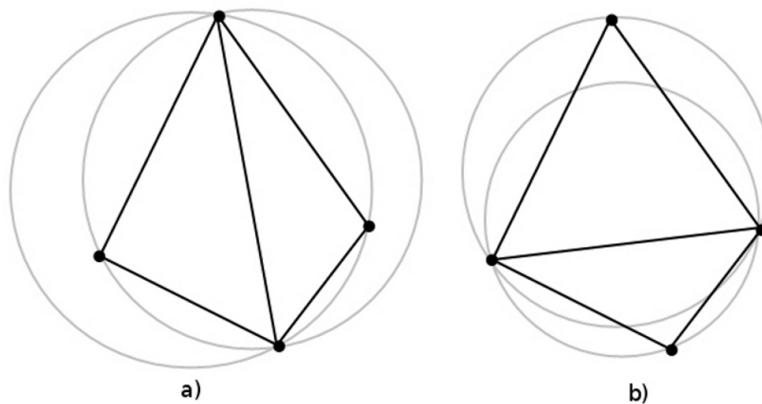


Figura 2.6: a) Critério de Delaunay falhando para os dois triângulos. b) Triangulação válida respeitando o critério de Delaunay.

A malha gerada por Delaunay visa maximizar os ângulos internos dos triângulos gerados, ou seja, dada uma aresta da triangulação de Delaunay, o ponto que forma o maior

ângulo com essa aresta é o ponto que formará um triângulo de Delaunay com ela.

Existem algumas variações de algoritmos de Delaunay. Em uma delas, encontra-se uma aresta que faz parte da triangulação que é, em geral, uma aresta pertencente ao fecho convexo. A partir dela, é encontrado o ponto que formará um triângulo de Delaunay. Assim, com as novas arestas, encontram-se novos triângulos, em um algoritmo parecido com o de avanço de fronteira. Uma outra variação é feita a partir de inserção de pontos. A entrada é uma malha triangular não necessariamente de Delaunay e se modifica essa malha (de apenas um subconjunto de pontos da entrada) pré-existente (Figura 2.7).

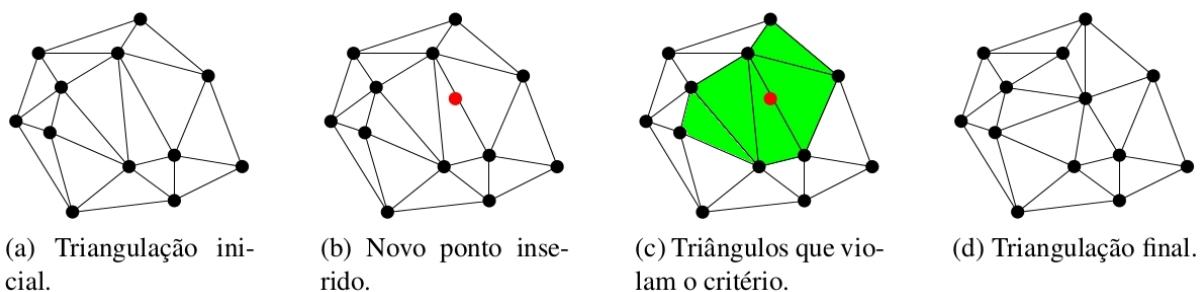


Figura 2.7: Triangulação por inserção de pontos (FREITAS, 2010).

Dependendo da disposição dos pontos da entrada, a triangulação final pode não ter boa qualidade, principalmente em regiões críticas, próximas à borda, gerando instabilidade em métodos numéricos. Uma alternativa para melhorar essa malha é fazer refinamentos e otimizações, que fazem uso de pontos de Steiner.

### 2.1.2.3 Arbitrária

As técnicas de geração de malha arbitrárias são aquelas que não se enquadram nem como Avanço de Fronteira e nem como Delaunay. As malhas são geradas em geral por algoritmos de varredura ou algum outro método.

Outro uso que essas malhas possuem é nas demonstrações de teoremas. O problema de ordenação de pontos pode ser reduzido ao problema de geração de malhas bidimensionais (CARVALHO; FIGUEIREDO, 1991). Prova-se por redução que pode ser gerada uma malha triangular a partir do fecho convexo de um conjunto de pontos em duas dimensões numa complexidade na ordem de  $O(n \log n)$ .

### 2.1.3 Estrutura de Dados

Diversas estruturas de dados que foram criadas na área da computação são muito usadas em problemas de computação gráfica. No contexto desse trabalho, essas estruturas têm o objetivo de fazer uma decomposição espacial do domínio. Com essas decomposições, diversos cálculos são otimizados fazendo uma busca em qual parte da decomposição o objeto de interesse está e limitando os cálculos apenas aos elementos que pertencem a essa decomposição.

Essas estruturas são utilizadas em diversas aplicações como tratamento de colisão e renderização (Figura 2.8). Entre as estruturas de dados bidimensionais, as mais importantes são a *quadtree*, *octree* e a *binary space partitioning* ou simplesmente BSP.

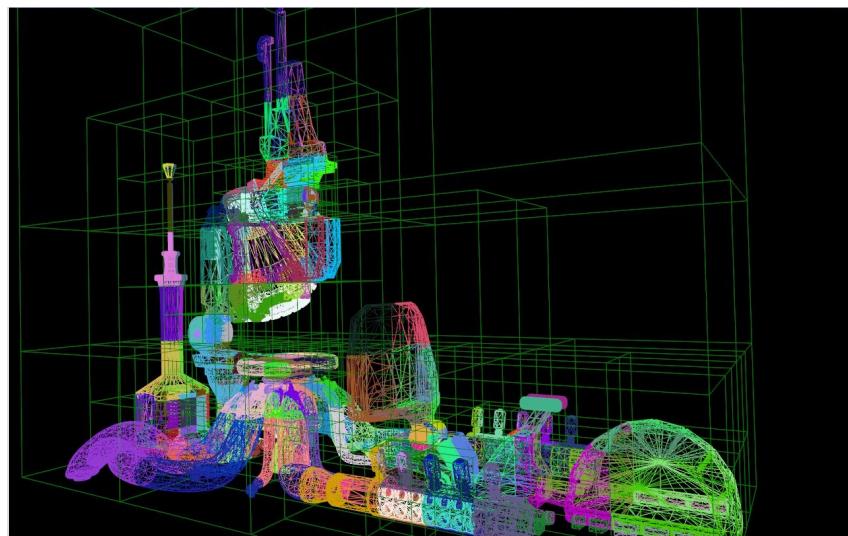


Figura 2.8: Exemplo de uma decomposição espacial feita para renderização e teste de colisão.  
Fonte: <http://togeskov.net/>

#### 2.1.3.1 *Quadtree*

Uma *quadtree* é uma estrutura de dados baseada em árvore em que cada nó possui exatamente quatro filhos (Figura 2.9). Em geral *quadtrees* são utilizadas para decompor domínios bidimensionais recursivamente em quatro regiões de mesmo tamanho. As *quadtrees* podem ser classificadas de acordo com o tipo do dado que elas representam (regiões, pontos, arestas, polígonos), isso vai depender do tipo de aplicação para o qual ele está sendo utilizada. Essa classificação altera o critério de subdivisão da *quadtree*, por exemplo, o critério pode ser a quantidade de pontos internos há um quadrante da *quadtree*, com isto é garantido a quantidade máxima de pontos internos a cada célula desta *quadtree*.

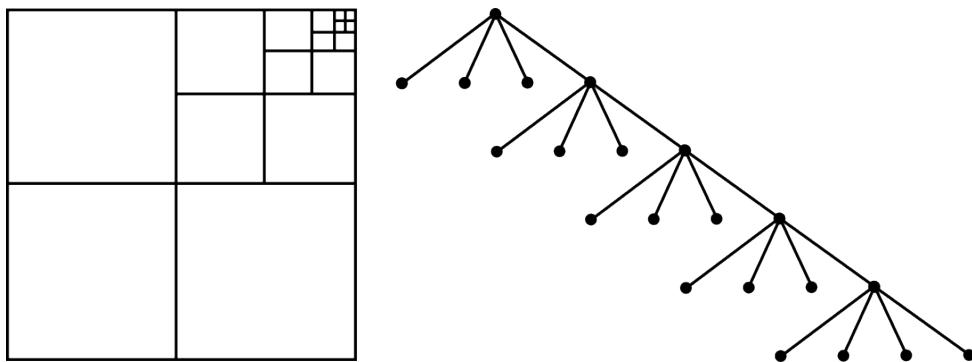


Figura 2.9: Uma subdivisão feita por *quadtree* com cinco níveis e sua representação em árvore.

### 2.1.3.2 Octree

A *octree* é a versão tridimensional da *quadtree*, por isso as mesmas propriedades da *quadtree* se aplicam a ela (Figura 2.10). A diferença é que a entrada será dividida sempre em 8 partes com regiões de mesmo tamanho. Os cortes serão realizados nos eixos X, Y e Z.

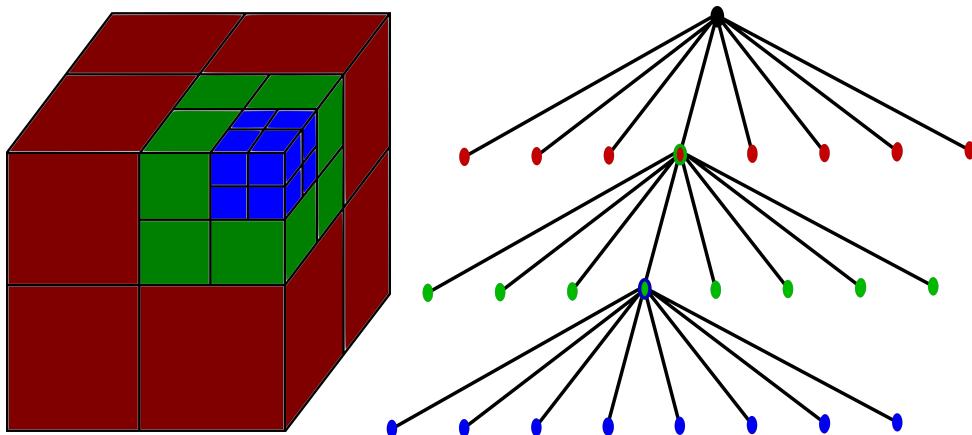


Figura 2.10: Uma subdivisão feita por *octree* com três níveis e sua representação em árvore.

### 2.1.3.3 Binary Space Partitioning (BSP)

BSP (particionamento binário espacial) é um processo genérico que, de forma recursiva, divide um domínio em duas partes, não necessariamente iguais, até que o particionamento do corte satisfaça um ou mais requisitos estabelecidos. Como resultado tem-se dois novos subespaços que podem ainda ser particionados recursivamente. O critério de posicionamento do corte e de parada no particionamento vai depender do objetivo que se deseja ao usar uma BSP.

Pode-se dizer que a BSP é um caso genérico da *quadtree*. A principal diferença entre elas basicamente é a quantidade de partições criadas (quatro para cada subdivisão na *quadtree* e duas na BSP) e a desvantagem está na hora de encontrar o melhor corte para a BSP, que pode ser

bastante custoso se comparado com a *quadtree*.

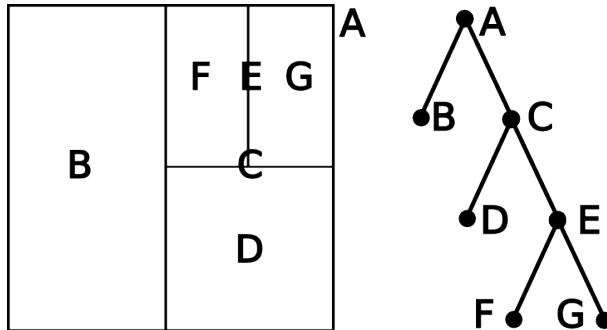


Figura 2.11: Uma subdivisão feita com BSP e sua representação em árvore.

#### 2.1.4 Estimativa de Carga

Neste trabalho as estimativas de carga são classificadas entre três classes:

1. Baseadas em estruturas de decomposição espacial.
2. Baseadas na quantidade de vértices/arestas/faces
3. Baseadas na área/volume.

A primeira delas utiliza estruturas de decomposição espacial, tais como *quadtrees*, *octrees* e BSP's. A vantagem dessas estruturas é que o modelo de entrada é totalmente mapeado em uma estrutura de dados, podendo obter facilmente a informação precisa da quantidade de elementos em uma determinada região da entrada, em geral considera-se a quantidade células da estrutura de decomposição como a carga referente aquele modelo. As principais desvantagens destas estruturas são o tempo de criação e o espaço em memória necessário para guardar as informações.

A segunda estima a carga como sendo a quantidade de vértices, arestas ou faces presentes no modelo de entrada. Quanto maior essa quantidade, maior será a carga computacional necessária para aquele modelo de entrada. A principal desvantagem dessa abordagem é não ter como mensurar a carga que será necessária para realizar a computação interna ao modelo, tendo em vista que foi considerado apenas a quantidade de elementos no exterior do modelo, podendo assim gerar uma estimativa ruim para alguns casos.

A ultima classificação utiliza apenas informações geométricas para estima a carga, considerando assim que quanto maior a área ou volume, maior será a carga computacional necessária para o modelo de entrada. Essa abordagem é útil para modelos de entrada uniforme ou

estruturado, porém, quando um modelo não uniforme ou não estruturado for dado como entrada, a estimativa poderá divergir bastante em relação carga a real.

### **2.1.5 Particionamento de Malhas**

Na geração em paralelo de malhas é necessário dividir a entrada para realizar o processamento em paralelo das diversas partes. Existem duas formas de decompor o domínio, segundo (CHRISOCHOIDES, 2005b). Na primeira forma, uma malha grosseira da região é rapidamente gerada, sequencialmente, e dividida entre os processadores. Essa forma, chamada de decomposição discreta do domínio, envolve ainda o problema de particionamento da malha.

A segunda forma de decompor o domínio envolve dividir a região a partir de funções, segmentos, eixos inerciais, ou estruturas auxiliares, por isso chamada de decomposição contínua do domínio. As regiões criadas serão compostas por parte do contorno de entrada juntamente com uma parte da região interna, criando assim subdomínios. Cada subdomínio é enviado a um processador, onde a malha será gerada.

Uma malha de interface é um conjunto de segmentos ou triângulos para o caso bidimensional ou, no caso tridimensional, um conjunto de triângulos ou tetraedros. Essa malha de interface faz a conexão entre duas partições vizinhas e faz o papel de uma nova fronteira. A forma que ela é criada vai depender da técnica que está sendo utilizada para particionar o domínio.

O particionamento contínuo pode ainda, ser subdividido em duas categorias, dependendo da forma como é gerada a malha entre os subdomínios, chamada de malha de interface. Se essa malha for gerada antes da malha interna ao subdomínio, essa abordagem é chamada de *a priori*. Caso ela seja gerada depois, é chamada de *a posteriori*. A geração da malha de interface *a posteriori* geralmente requer sincronização entre processos. Uma vantagem da decomposição Contínua em relação a Discreta é que a entrada não é modificada. No trabalho de (DECOUGNY; SHEPHARD, 1999) apresenta esta classificação.

### **2.1.6 Balanceamento de carga**

Neste trabalho é considerado a existência de duas categorias para o balanceamento de carga. A primeira irei chamar de abordagem Centralizada, para a utilização desta abordagem é necessário um processo responsável por manter todas as tarefas, os demais processos devem solicitar tarefas diretamente ao processo central.

A segunda abordagem é a Descentralizada, onde as tarefas estão divididas entre

todos os processos disponíveis, a distribuição das tarefas em geral é feita logo no inicio do programa, podendo haver migração de tarefas para outros processos durante a execução.

As duas abordagens necessitam de uma boa estimativa de carga para se obter no final um bom balanceamento de carga entre os processos disponíveis. Técnicas que não possuem estimativa de carga em geral optam por escolher a estratégia Descentralizada. Nestes casos, para compensar a falta de estimativa, é criado muito mais tarefas que o número de processadores disponíveis, fazendo que implicitamente a carga em uma partição fique quase semelhante as outras.

A principal desvantagem da abordagem Centralizada é a quantidade de comunicações que são feitas ao processo central. Isto acaba gerando um gargalo devido a quantidade de requisições simultâneas que possam acontecer.

## 2.2 Particionamento Baseado na Geometria

O Trabalho de (VIDWANS A.; KALLINDERIS, 1994) apresenta uma técnica de particionamento Contínuo *a priori* com balanceamento por divisão e conquista, onde pode ocorrer uma redistribuição das cargas dos processadores. Inicialmente os planos de corte são criados baseados na centroide, utilizando os eixos para orientação do plano de corte, podendo criar os cortes sempre em um dos eixos ou então alinhando o plano em relação a mais de um eixo. É criado então conjuntos de vértices, arestas e faces para cada subdomínio criado. Cada subdomínio é atribuído a um processador e são balanceados trocando elementos desses conjuntos com seus vizinhos. Figura 2.12 mostra um exemplo de balanceamento de carga feito pelo método.

No trabalho de (GAITHER, 1996) traz uma técnica de geração de malha bidimensional por inserção de pontos com particionamento Discreto. A criação das partições é baseada na estimativa da área dos subdomínios e na área dos triângulos que estão sendo gerados, tentando criar assim regiões de áreas iguais. Inicialmente é gerada uma malha grosseira com os vértices de entrada e depois é realizado um agrupamento dos triângulos, de tal forma que ao final a quantidade de regiões e processadores sejam iguais. As novas fronteiras criadas são discretizadas e um algoritmo de Delaunay bidimensional é aplicado. A Figura 2.13 mostra o passo a passo da técnica.

Em (WU POTING; HOUSTIS, 1996), uma técnica de particionamento Discreta é descrita. O particionamento é feito numa malha inicial grosseira. É identificado os subdomínios nesta malha grosseira com base na carga (área da região), no tamanho da interface e conectividade

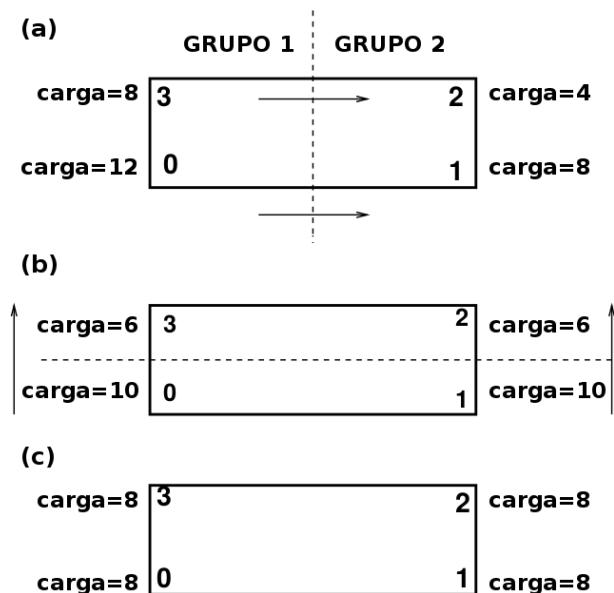


Figura 2.12: Exemplo do método de divisão e conquistar de (VIDWANS A.; KALLINDERIS, 1994) para equilibrar a carga entre quatro processadores. (a) distribuição de carga inicial. (b) Distribuição de carga após o passo 1. (c) Distribuição de carga após o passo 2.



Figura 2.13: Passo a passo da técnica de (GAITHER, 1996).

entre regiões. Após ter os subdomínios definidos, é feito um refinamento na malha inicial grosseira. Ao final os subdomínios são redefinidos com base na nova malha. A Figura 2.14 ilustra o passo a passo da técnica. O trabalho de (BANK RANDOLPH E.; LU, 2005) tem uma metodologia parecida, onde utiliza uma malha grosseira para realizar o particionamento, que é baseado na bisseção.

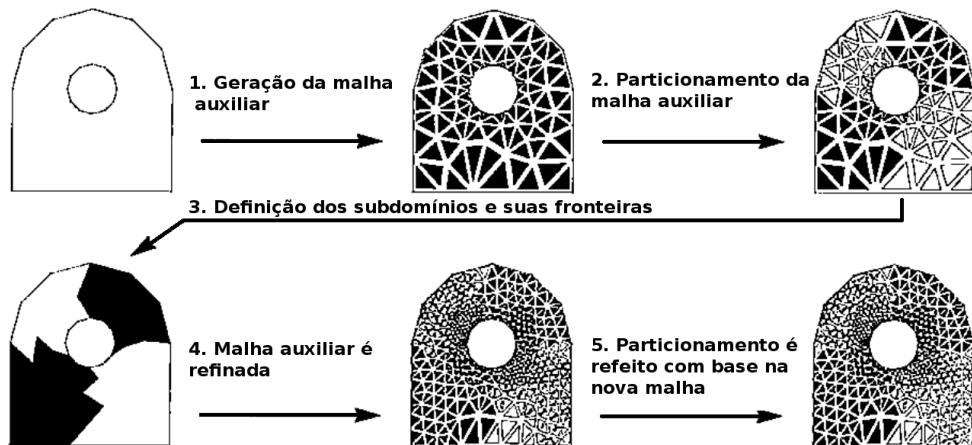


Figura 2.14: O passo a passo da técnica de (WU POTING; HOUSTIS, 1996).

Em (GALTIER Jérôme; GEORGE, 1996) permite utilizar duas abordagens Contínuas *a priori* para particionamento. A primeira é pelo particionamento em um mesmo eixo pela distância entre os planos de corte. A segunda é pelo particionamento recursivo onde os planos de cortes passam pelo momento de inércia. A malha das interfaces são criadas por um grafo de Voronoi, que por sua vez vem de uma triangulação de Delaunay dos vértices iniciais.

Em (SAID, 1999) apresenta uma técnica de particionamento Discreta que utiliza uma grade volumétrica para auxiliar no particionamento. A estimativa de carga é realizada pela quantidade de faces presentes numa região. Tanto a malha quanto a grade volumétrica são geradas por Delaunay. As subdivisões são feitas considerando também o volume das regiões. A Figura 2.15 mostra um exemplo da formação dos subdomínios em

Em (IVANOV; ANDRÄ; KUDRYAVTSEV, 2006), foi desenvolvido um algoritmo com particionamento Contínuo *a priori* baseado em Delaunay para geração de malhas tetraédricas em que o posicionamento do plano de corte é definido pelo centro de massa e pela matriz de inércia. O plano de corte é um plano perpendicular a um eixo que segue uma das três definições:

- Planos criados são equidistantes;
- Volume entre os planos são iguais;
- Passa pelo centro de massa.

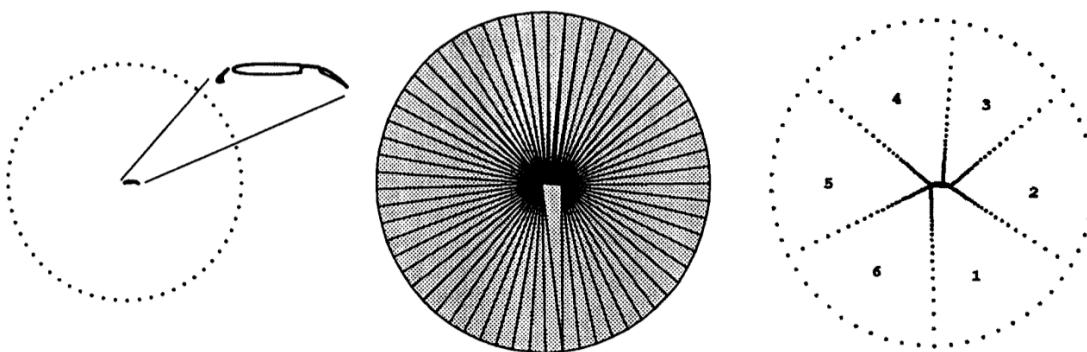


Figura 2.15: Exemplo da formação dos subdomínios em (SAID, 1999). Fronteira de entrada, grade auxiliar inicial e 6 subdomínios gerados juntamente com as suas discretizações respectivamente.

A escolha do critério utilizado para criar os subdomínios depende da geometria da entrada. Assim, dependendo da entrada, um critério pode ser melhor que outro, mas isso depende do conhecimento do usuário. Na Figura 2.16, as três formas de particionamento são apresentadas.

Após ter o plano de corte definido, é feita uma suavização da seção de corte e a sua triangulação por Delaunay para, posteriormente, serem geradas as malhas nos subdomínios. Um problema bem visível nesse método é que para se ter um bom plano de corte é preciso ter um modelo com uma geometria bem comportada, sem forma côncava, alongada ou afinada. Nesse trabalho a quantidade de subdomínios gerados é maior que a de processos para tentar melhorar o balanceamento dinâmico, uma vez que não se tem uma boa precisão na estimativa da carga.

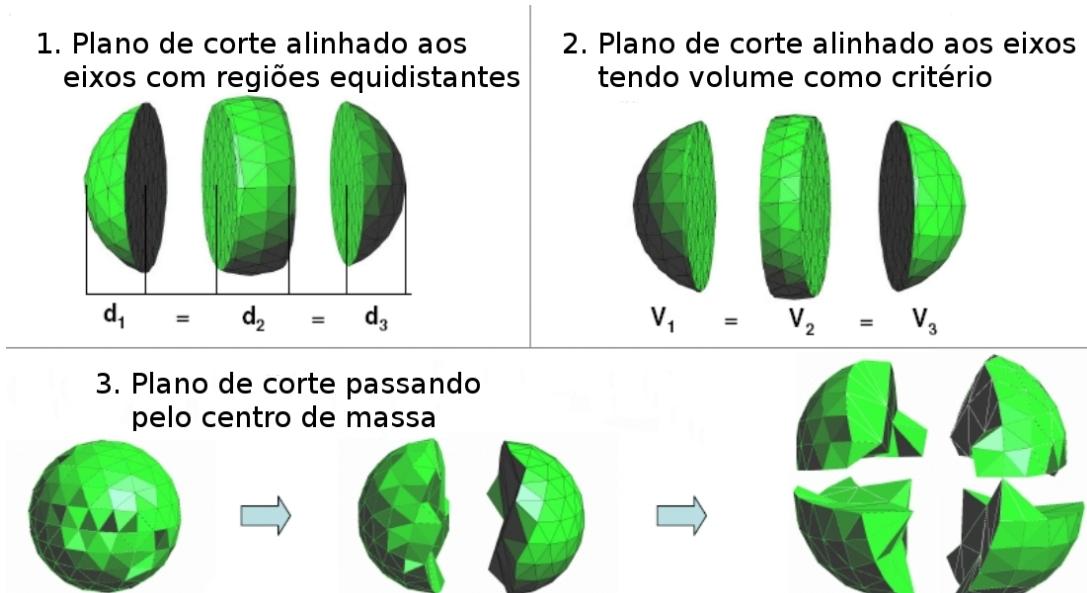


Figura 2.16: As três formas de particionar. 1 - planos equidistantes. 2 - volume dos subdomínios iguais. 3 - centro de massa (IVANOV; ANDRÄ; KUDRYAVTSEV, 2006).

Uma solução parecida é a apresentada em (LÄMMER; BURGHARDT, 2000), onde o plano de corte é traçado no centro de gravidade, porém em duas dimensões, ou seja, apenas um eixo de corte. Este eixo é usado para dividir o domínio recursivamente até que o número de subdomínios seja igual a quantidade de processadores. A partir do eixo, uma aresta é formada, e os valores nos seus pontos extremos são interpolados entre os valores dados como entrada. Uma malha de Delaunay é gerada em cada interior dos subdomínios.

(JURCZYK TOMASZ; GŁUT, 2007) apresenta uma técnica de particionamento *a priori* onde o plano de corte é criado segundo um série de requisitos. Entre os requisitos estão que o volume das regiões geradas devem ser aproximadamente a mesmo, o plano de corte deve ser mínimo, os seja, poucos elementos pertencem ao separador e o ângulo de junção com a fronteira de entrada não deve formar um ângulo agudo. O balanceamento desta técnica é feito pela quantidade de faces na superfície. A Figura 2.17 mostra o processo de particionamento.

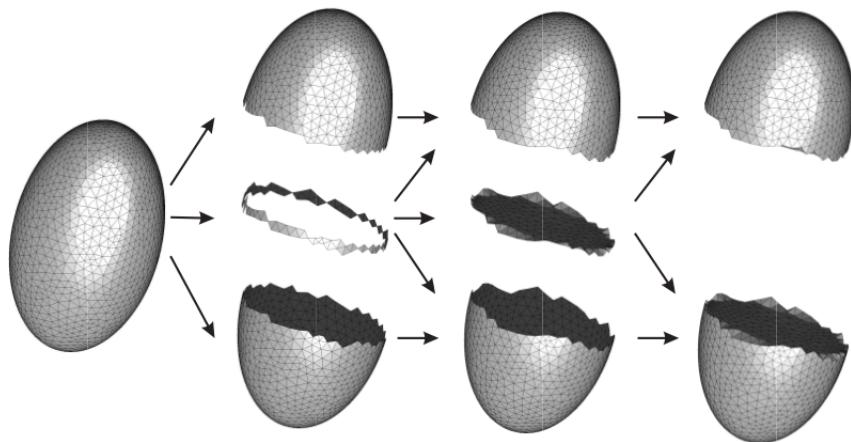


Figura 2.17: Exemplo do processo de particionamento em (JURCZYK TOMASZ; GŁUT, 2007).

Em (ANDRÄ, 2008), é utilizado o centro de massa e o momento de inercia para encontrar os planos de corte do domínio. As interfaces são geradas *a priori* por Delaunay bidimensional e convertidas depois para tridimensional. A Figura 2.18 mostra um exemplo de particionamento feito pela técnica.

Em (PIRZADEH; ZAGARIS, 2009), é descrita uma técnica baseada em Avanço de Fronteiras e Avanço de Camadas com particionamento Contínuo *a priori*.

Inicialmente, é gerada uma malha de superfície nos pontos dados como entrada. Logo em seguida, é feita uma estimativa de carga nos subdomínios utilizando uma *octree* que usa a informação da quantidade de faces para subdividir o domínio. Se necessário, serão criados planos de partições que dividem o domínio em regiões com cargas aproximadamente iguais. As posições destes planos são definidas através do centro de densidade da malha. O centro de densidade indica onde a massa efetiva do sistema está concentrada.

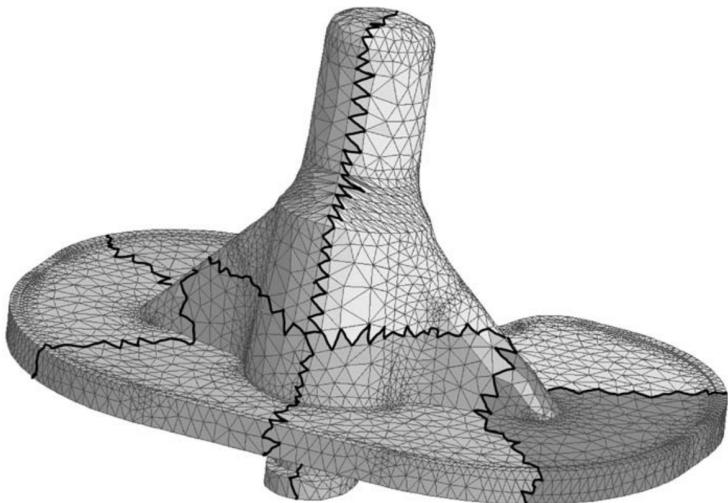


Figura 2.18: Exemplo de um particionamento feito por (ANDRÄ, 2008) para 8 subdomínios.

Em seguida, são identificadas as faces que interceptam o plano de partição e uma malha parcial é gerada na região do plano de corte. Depois disso, para cada lado da partição são agrupadas as faces dos novos subdomínios. Esse processo é repetido até que um número máximo de subdivisões tenha ocorrido. Ao final da execução, tem que ser realizada uma junção de todas as submalhas. A Figura 2.19 ilustra os principais passos dessa técnica para o caso bidimensional. Esta técnica gera malha por Avanço de Fronteira e é uma mistura de *a priori* com *a posteriori*, pois avança uma camada de elementos na interface para depois criar os subdomínios.

Como vantagem desta técnica Contínua pode-se citar que a utilização de avanço de camadas entre as partições faz com que a malha gerada seja praticamente idêntica a uma malha gerada sequencialmente, ou seja, não são gerados padrões entre as partições do domínio. Outra vantagem é que não é necessário nenhum pré-processamento custoso para definir ou construir as partições. Além disso, a construção da *octree* para estimar a carga é automática e de baixo custo.

Uma das desvantagens desse método é que nem sempre é fácil gerar as malhas nas partições, especialmente em três dimensões. Além disso, a qualidade dessas malhas pode ser ruim, prejudicando assim a qualidade da malha gerada no modelo todo. Basear a quantidade de subdivisões num número máximo não é uma boa métrica para controlar a geração dos subdomínios quando não se tem uma boa estimativa de carga, isso pode gerar subdomínios em excesso, aumentando a comunicação entre os processos.

Em (CHEN, 2012) é apresentado uma técnica Contínua *a priori* onde o plano de partição é posicionado usando o centro gravitacional juntamente como o eixo de inercia para dar a normal desse plano. Após isto é encontrado as arestas que fazem interseção com o plano de corte, eliminando aquelas que tenham vértices com vizinhança menor que dois, nas arestas restantes é realizado uma suavização (Figura 2.20). A interface é gerada pela execução de

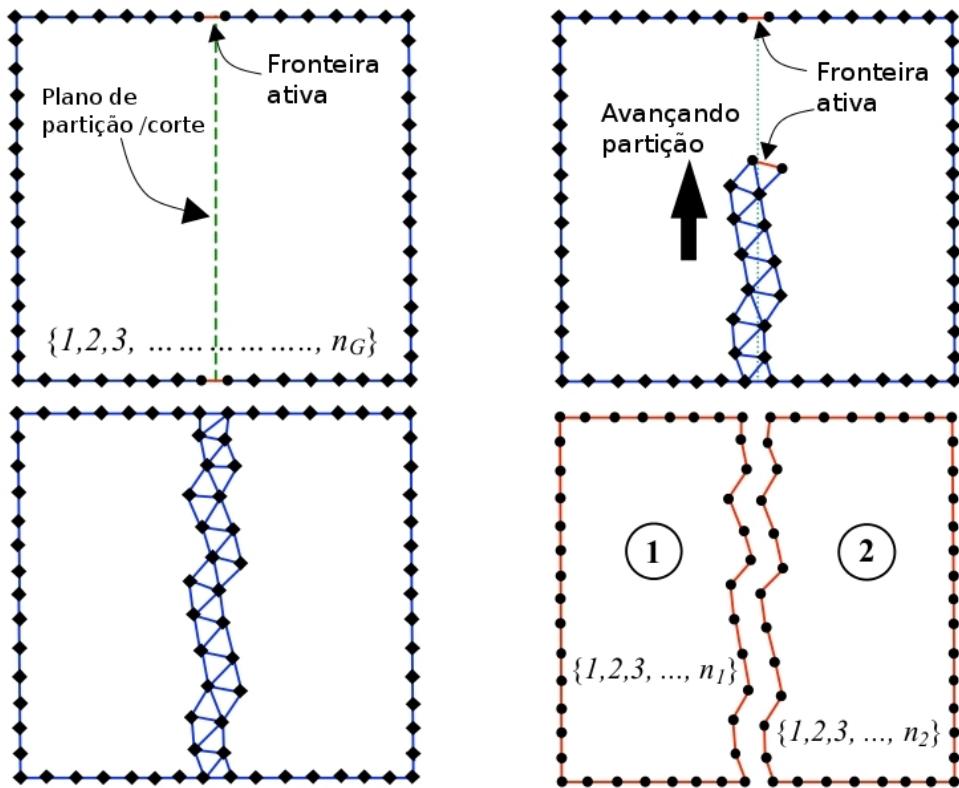


Figura 2.19: Os principais passos da técnica de (PIRZADEH; ZAGARIS, 2009) para gerar os segmentos de interface.

Delaunay nas arestas da fronteira encontrada. Apesar do esforço para a criação de uma boa interface, esta técnica não possui um estimativa de carga clara para os subdomínios, tentando compensar o balanceamento de carga fazendo *over-decomposition* (criação de mais subdomínios que processadores disponíveis). Um trabalho parecido é (ZHENG YAO; CHEN, 2009), onde os planos de cortes devem ser os menores possíveis e que gerem subdomínios de tamanhos praticamente iguais. O posicionamento do corte no principal eixo de inércia.

Em (GLUT; JURCZYK, 2008), é apresentada uma técnica para malhas tridimensionais com uma abordagem baseada no particionamento geométrico onde a entrada é uma malha de superfície. Nesse trabalho são descritas duas técnicas baseadas na *bounding box* gerada a partir da entrada.

A seleção do separador do domínio deve garantir um custo de corte baixo, ou seja, encontrar e posicionar o plano de corte não pode ter um custo computacional alto. Além disso, deve garantir um bom balanceamento de carga e minimizar os elementos conectados por múltiplos subdomínios.

A primeira técnica é baseada na malha de superfície. Para o plano de corte ser criado, é preciso a localização do contorno da malha de superfície e do separador. O contorno é então

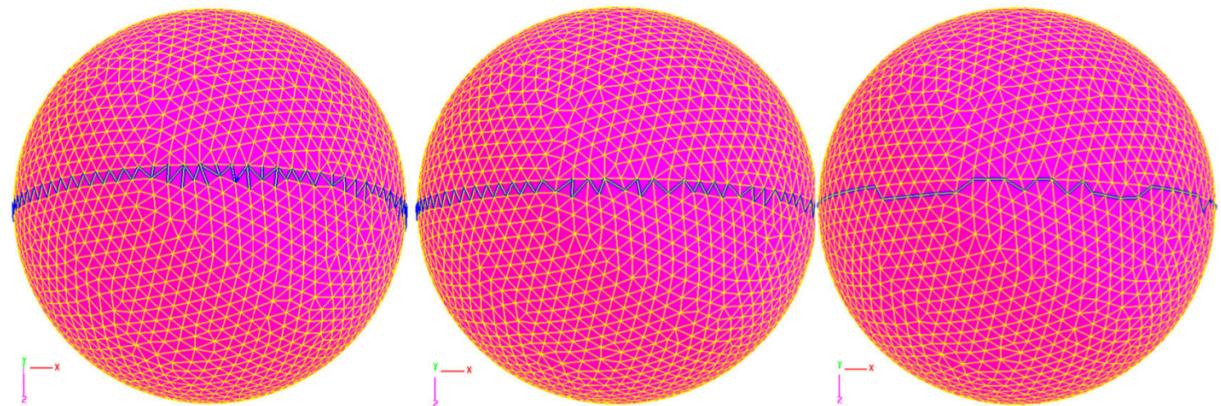


Figura 2.20: Da esquerda para direita: todas as arestas que sofrem interseção, a fronteira inicial, e a fronteira suavizada. Em (CHEN, 2012).

projeto no separador usando uma função 2D de controle espacial baseada no tamanho das arestas (Figura 2.21).

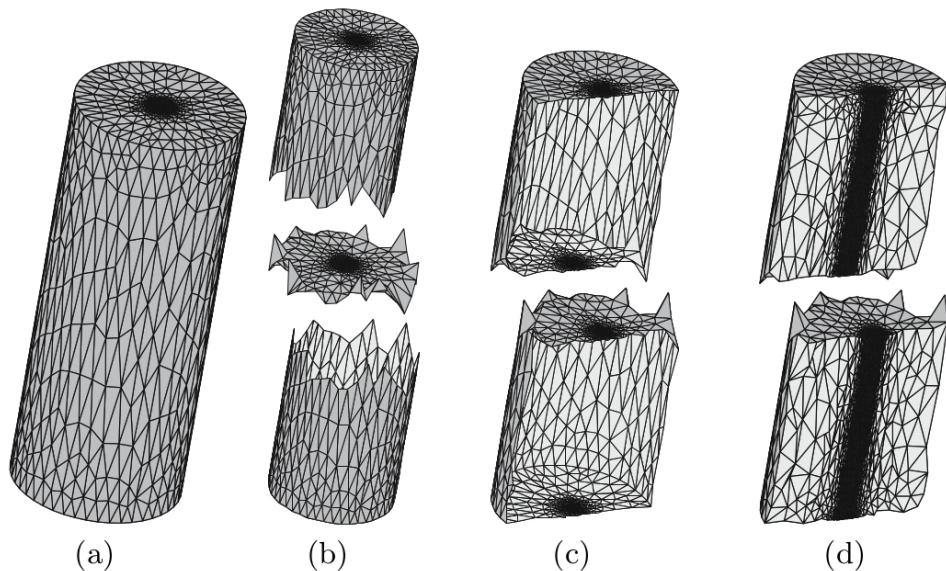


Figura 2.21: Passos da técnica baseada na malha de superfície. (a) malha de superfície; (b) corte; (c) seção transversal; (d) malha final (GLUT; JURCZYK, 2008).

A segunda técnica se baseia numa malha volumétrica grosseira. Primeiramente, é feita a geração de uma malha 3D grosseira utilizando alguma função de controle espacial. O posicionamento do plano de corte é feito parecido com a técnica anterior, porém utilizando a malha volumétrica como função espacial (Figura 2.22).

Esta técnica depende muito da geometria da entrada já que são utilizadas informações da *bounding box* dessa entrada. Isso afeta diretamente a criação dos planos de corte, e por consequência, a malha gerada ao final. Uma das motivações deste trabalho é evitar a criação de subdomínios baseados nos eixos de inércia, pois, segundo o próprio autor, os resultados não são

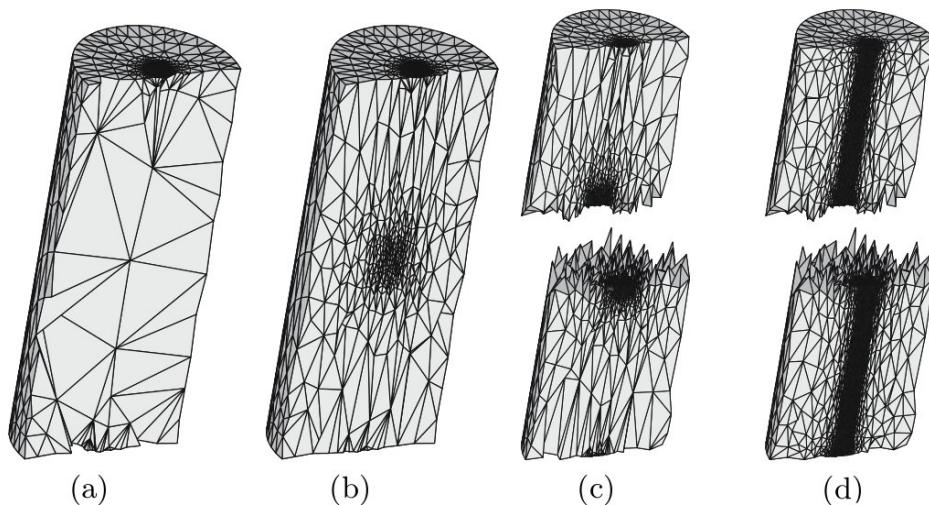


Figura 2.22: Passos da técnica baseada na malha volumétrica grosseira. (a) malha volumétrica grosseira; (b) refinamento da seção transversal; (c) seção transversal; (d) malha final (GLUT; JURCZYK, 2008).

bons.

### 2.3 Particionamento Baseada em Estruturas de Dados

Um dos primeiros trabalhos em partitionamento de domínios foi o de (FARHAT, 1988), que desenvolveu uma técnica para partitionamento de malhas quadrangulares de elementos finitos. Este trabalho subdivide um domínio de acordo com a quantidade de processadores disponíveis, esta técnica utiliza a própria estrutura da malha quadrangular de matriz/grade usando uma estrutura de grade que contém os elementos e realizando o partitionamento e a estimativa de carga em cima desta matriz de elementos. A Figura 2.23 mostra um exemplo de partitionamento.

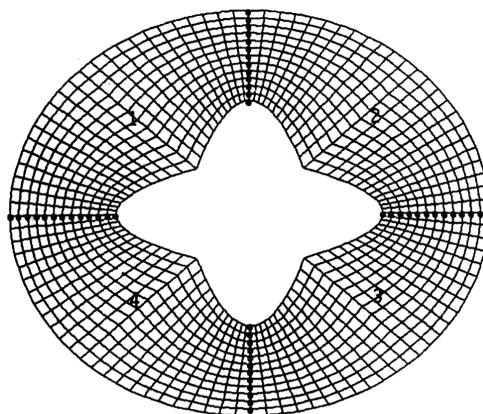


Figura 2.23: Decomposição em 4 subdomínios de uma malha multiconectada na técnica de (FARHAT, 1988).

Alguns trabalhos como o de (BARNARD STEPHEN T.; SIMON, 1994) utilizam grafos para encontrar o corte no domínio. O corte é baseado num grafo criado com as arestas, maximizando a quantidade de vértices nos conjuntos e minimizando a quantidade de arestas cortadas pelo corte. A criação do grafo para o particionamento é ilustrada na Figura 2.24. No trabalho de (SIMON, 1991) além desta forma de particionamento, é mostrada também particionamentos feitos pela bisseção e pelo particionamento recursivo da bisseção espectral (autovetores da matriz Laplaciana de um grafo).

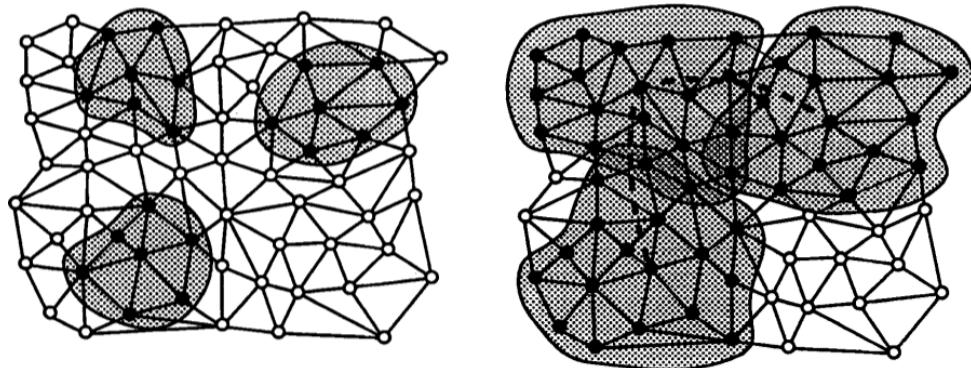


Figura 2.24: Fluxograma da triangulação em paralelo (BARNARD STEPHEN T.; SIMON, 1994).

Em (NIKISHKOV, 1999), um grafo é construído para fazer a estimativa de carga e para realizar o particionamento. O critério de subdivisão é a quantidade de elementos internos a cada subdomínio. A Figura 2.25 mostra um exemplo de particionamento para esta técnica.

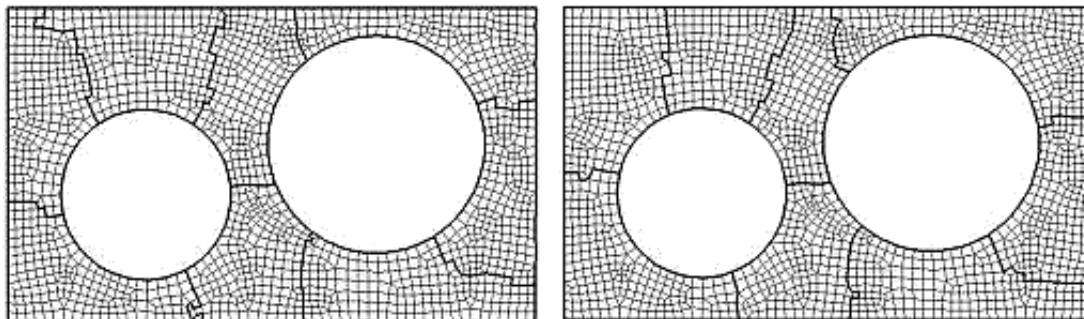


Figura 2.25: Oito subdomínios criados com quantidades iguais de elementos e do lado direito a otimização das partições em (NIKISHKOV, 1999).

Em (DECOURGNY; SHEPHARD, 1999), a entrada do algoritmo é o contorno de um objeto. Cada processador fica com parte de uma *octree* distribuída, que define planos de corte do domínio. A malha das células internas é gerada concorrentemente com *templates*. A região entre o contorno e as células internas é preenchida por uma técnica de Avanço de Fronteira, onde são gerados os elementos internos a uma região delimitada pelos planos de corte. Por último é feita a conexão das malhas dos dois lados de cada plano e de suas intersecções. Essa técnica gera muitas

partições, já que a cada subdivisão oito novos subdomínios são criados, e, por usar *templates*, esta técnica pode gerar uma quantidade excessiva de elementos, além de possivelmente gerar elementos de qualidade ruim nas regiões próximas ao contorno.

Na técnica de (LÖHNER, 2001), é gerada uma *octree* grosseira com relação ao contorno dado como entrada. Esta técnica é classificada como Contínua *a posteriori*. Após essa geração, as células que contêm a parte da fronteira que gerará os menores elementos são identificadas. Assim, partes da malha, correspondentes a cada célula, são geradas simultaneamente por avanço de fronteira, de maneira que cada parte da malha gerada não possa cruzar as extremidades da célula que a contém. Cada octante sofre então um pequeno deslocamento na diagonal com o intuito de gerar mais elementos. Esse deslocamento elimina quase todas as faces entre duas ou mais células e diminui o tamanho da fronteira para o próximo passo. Desse modo a nova fronteira é encontrada e uma nova *octree* é construída para ela, e o procedimento é repetido, até que não seja mais possível gerar malha. Na Figura 2.26, são mostrados os passos do algoritmo e os deslocamentos que são realizados.

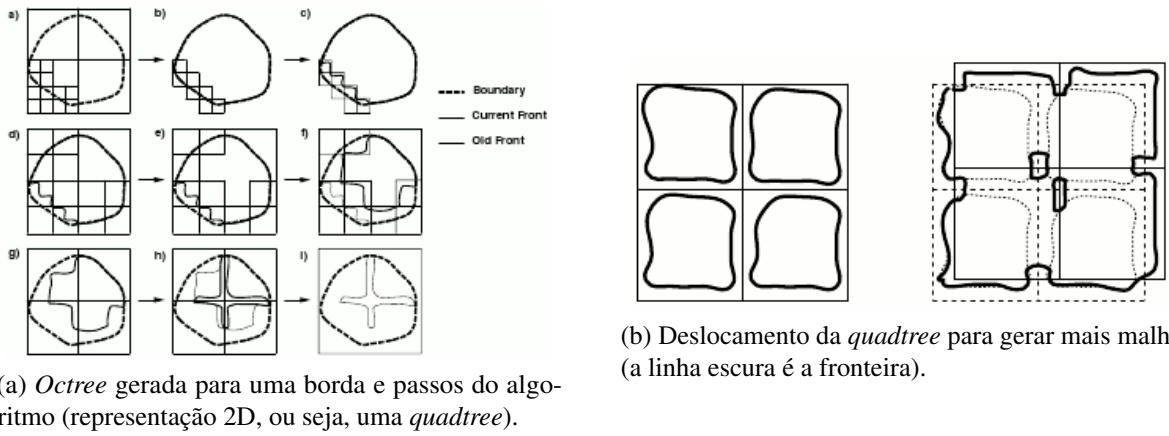


Figura 2.26: Técnica de (LÖHNER, 2001).

O deslocamento de cada octante é feito seguindo-se sempre o mesmo processo, e isso pode não ser o ideal para certos tipos de modelos, onde maneiras distintas de deslocamento poderiam ser mais eficientes (deslocamentos na diagonal, na direção dos eixos principais, entre outros).

No trabalho de (LOHNER, 2014) traz algumas técnicas recentes na área de geração de malhas em paralelo. A principal novidade é a utilização da própria estrutura da árvore de decomposição para unir as interfaces dos subdomínios. A utilização da árvore de decomposição já torna a paralelização mais natural e simples (Figura 2.27).

Em (LARWOOD et al., 2003), é apresentada uma técnica de decomposição de domínio que tem como entrada uma triangulação de borda. Para saber quais subdomínios

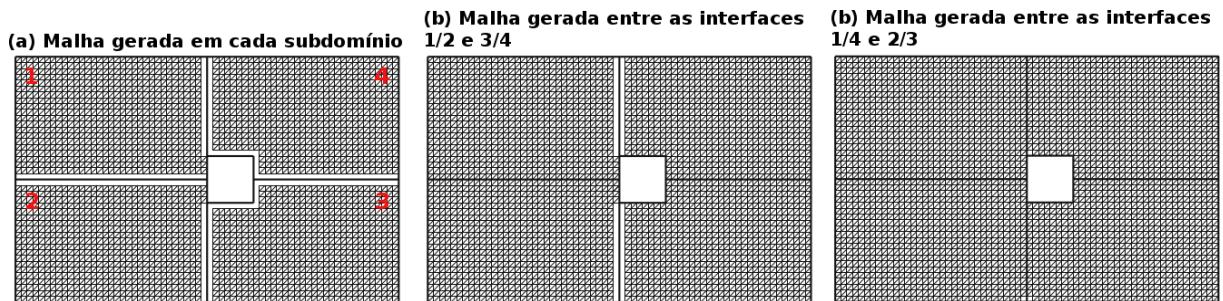


Figura 2.27: Junção das malhas dos subdomínios em (LOHNER, 2014).

devem ser divididos, a técnica usa um critério baseado na quantidade de faces por subdomínio. A decomposição é feita recursivamente usando uma *octree* caso seja tridimensional ou uma *quadtree* caso seja bidimensional, verificando sempre se o número de faces de um subdomínio é menor do que o limite estipulado, e, enquanto a verificação for falsa, a decomposição ocorre. A quantidade máxima de subdivisões está limitada por uma constante maior que o número de processadores disponíveis. Isso evita a criação excessiva de partições e permite que um processador possa receber mais de uma tarefa ao longo da execução. As interfaces dos subdomínios são geradas *a priori* por Delaunay. A qualidade da malha não é garantida neste trabalho, são apresentados apenas resultados de balanceamento de carga entre os processadores que é garantido pela textitover-decomposition.

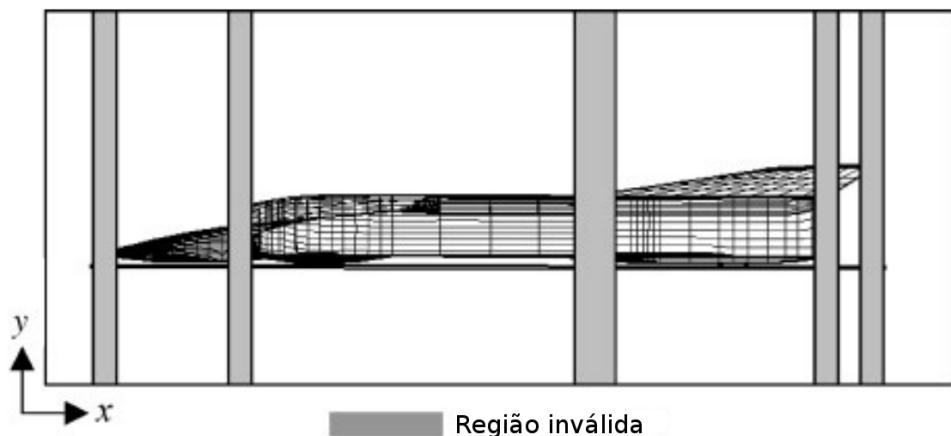


Figura 2.28: Regiões de corte inválidas em cinza (LARWOOD et al., 2003).

Para evitar a criação de elementos ruins, é feita uma verificação no corte baseada no ângulo do vetor normal do plano de corte com a normal dos triângulos, de forma que o plano de corte não possa passar por triângulos com ângulo menor do que uma tolerância. Caso essa verificação falhe, a *octree* (*quadtree*, em 2D) sofre um deslocamento em um dos eixos. A Figura 2.28 mostra um exemplo onde alguns planos de corte falham nos testes.

No trabalho de (CHARMPIS DIMOS C.; PAPADRAKAKIS, 2005) trabalha com

malhas de elementos finitos e é feita uma representação da entrada como um grafo e em seguida é utilizado um particionador de grafos (SCGEN) que procura dividir os pesos dos nós e arestas do grafo de acordo com a quantidade de processadores disponíveis. A imagem 2.29 mostra uma malha de elementos finitos e a sua partição como grafo.

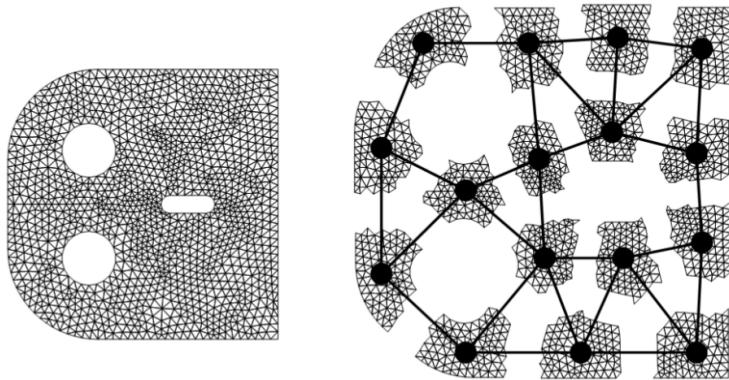


Figura 2.29: Malha de elementos finitos (esquerda) e a partição da malha com sua representação em grafo (à direita) em (CHARMPIS DIMOS C.; PAPADRAKAKIS, 2005).

Em (LINARDAKIS; CHRISOCHOIDES, 2006), é apresentada uma técnica bidimensional que utiliza a triangulação de Delaunay por divisão e conquista para um conjunto de pontos dados como entrada. Primeiramente, é feita uma triangulação utilizando apenas os pontos da borda, que é utilizada para a geração de um grafo ponderado, onde o peso de uma aresta é igual ao raio da circunferência circunscrita do triângulo que a contém. Em seguida é feita uma contração desse grafo, onde os vértices do grafo representam a área a ser triangularizada (futuros subdomínios), e as arestas representam a conexão entre essas áreas.

Através do grafo formado, os planos de corte são posicionados e os subdomínios formados. A Figura 2.30 mostra o resultado da decomposições para quantidades diferentes de subdomínios. Esse processo de subdivisão ocorre até que a quantidade de subdomínios criados seja suficientemente grande. Isso acontece para tentar melhorar o balanceamento de carga. Após isso, a geração da malha interna poderá ser realizada. A desvantagem dessa técnica é a ausência de uma estimativa de carga eficiente, precisando gerar vários subdomínios para melhorar o balanceamento de carga.

Em (ITO et al., 2007) o particionador de malhas METIS também é utilizado. Uma malha tetraédrica grosseira é gerada inicialmente com base nas faces da superfície. O particionador de grafos METIS é utilizado para subdividir o domínio tendo como base a malha grosseira gerada. A Figura 2.31 mostra para uma esfera o processo de particionamento.

Outro trabalho que também utiliza grafos é o de (PANITANARAK THAP; SHONTZ, 2011), que descreve uma técnica de decomposição Discreta que utiliza o programa de partição de

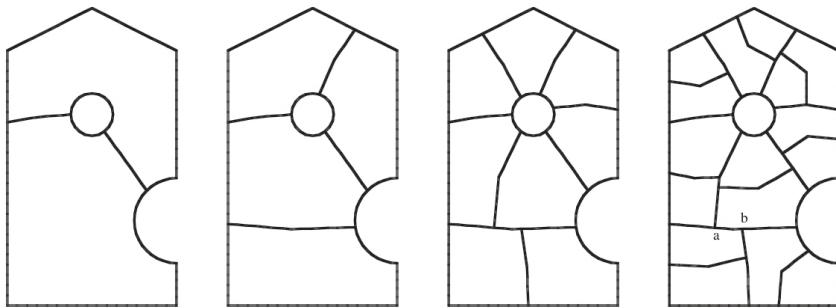


Figura 2.30: Construção do grafo de particionamento de (LINARDAKIS; CHRISOCHOIDES, 2006).

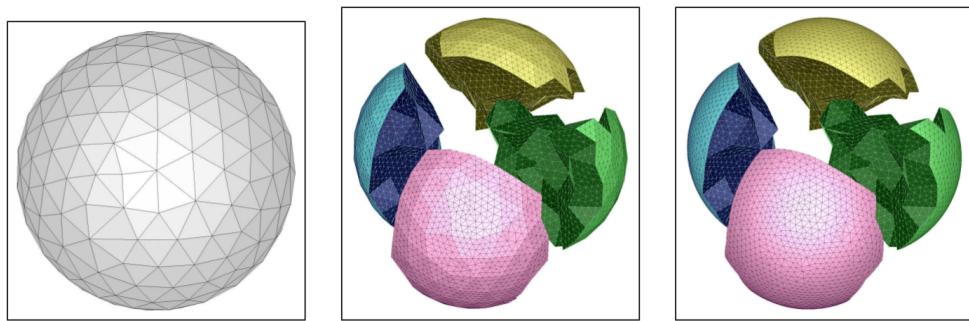


Figura 2.31: Malha tetraédrica inicial, malha de superfície refinada nos subdomínios e melhorias nas faces de superfície no trabalho de (ITO et al., 2007).

grafos METIS ((KARYPIS; KUMAR, 1998)). Primeiramente é construída uma malha grosseira com a restrição dos ângulos formados com a fronteira sejam maiores que  $30^\circ$ . Com a malha devidamente criada, é feita a sua conversão para um grafo, onde cada triângulo é representado como um nó e cada aresta como a vizinhança dos triângulos. O particionamento é feito neste grafo de acordo com a quantidade de subdomínios desejado. A Figura 2.32 mostra um exemplo de particionamento feito utilizando esta técnica.

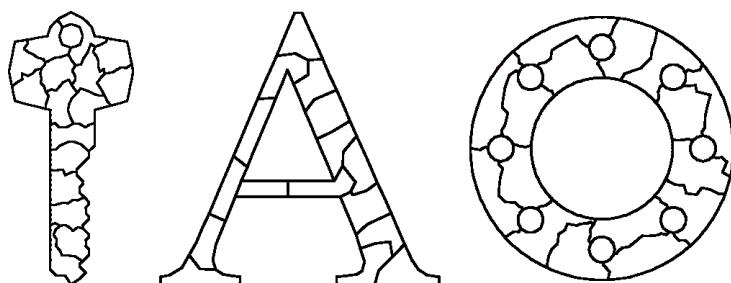


Figura 2.32: Decomposição feita pela técnica de (PANITANARAK THAP; SHONTZ, 2011).

Em (LO, 2012), uma técnica bidimensional para uma triangulação de Delaunay por inserção de pontos é apresentada. Uma *kd-tree* é utilizada para organizar os pontos da entrada em células. Estas células são agrupadas em zonas e distribuídas entre os processadores. A vantagem de usar uma *kd-tree* é que cada célula tem uma quantidade de pontos aproximadamente igual e a

busca espacial é facilitada na hora de fazer a inserção de pontos em uma região. A Figura 2.33 mostra a organização de um conjunto de pontos por partição regular e por *kd-tree*.

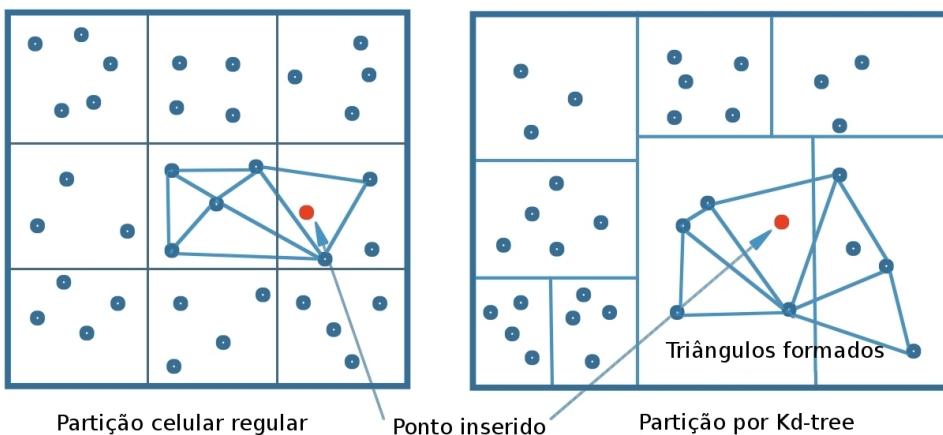


Figura 2.33: Pontos organizados em células. À esquerda por partição regular e à direita por *kd-tree* (LO, 2012).

A quantidade de subdomínios criados é compatível com a quantidade de processadores disponíveis, ou seja, cada processador terá que ficar responsável por uma zona. A inserção dos pontos em cada zona é feita em paralelo, sendo totalmente independente das outras zonas, e a malha gerada em cada zona também será independente.

Os triângulos gerados nas bordas das zonas têm pontos de uma zona vizinha. Isso irá gerar uma camada a mais de triângulos nas zonas, que é necessário para obter uma malha sem buracos entre elas. Ao final, tem de ser feita a junção de todas as malhas geradas em uma só e, para isso, é preciso eliminar as redundâncias (triângulos repetidos entre duas zonas). Essa junção das malhas pode ser um processo complexo em determinados modelos e isso pode prejudicar o desempenho do algoritmo. A Figura 2.34 mostra o fluxograma da triangulação em paralelo.

O trabalho de (FREITAS et al., 2013) apresenta uma técnica bidimensional que recebe como entrada uma fronteira e utiliza uma *quadtree* para particionar e estimar a carga. As células folhas da *quadtree* de particionamento são divididas entre os processadores disponíveis, onde são geradas as malhas internas. Depois de gerar as malhas nos subdomínios iniciais, a fronteira é atualizada e as células da *quadtree* são deslocadas nos eixos cartesianos a fim de gerar mais malha. Esse processo de deslocamento e geração de malha é feito até que não seja mais possível gerar malha. Um processo mestre fica responsável por finalizar a geração da malha e fazer a melhoria na mesma. Este trabalho é classificado como contínuo *a posteriori*. A Figura 2.35 ilustra o passo de deslocamento da *quadtree* no eixo X.

Em (FREITAS; CAVALCANTE-NETO; VIDAL, 2014), é apresentada uma evolução da técnica anterior que pode ser tanto bidimensional como tridimensional, por Avanço de

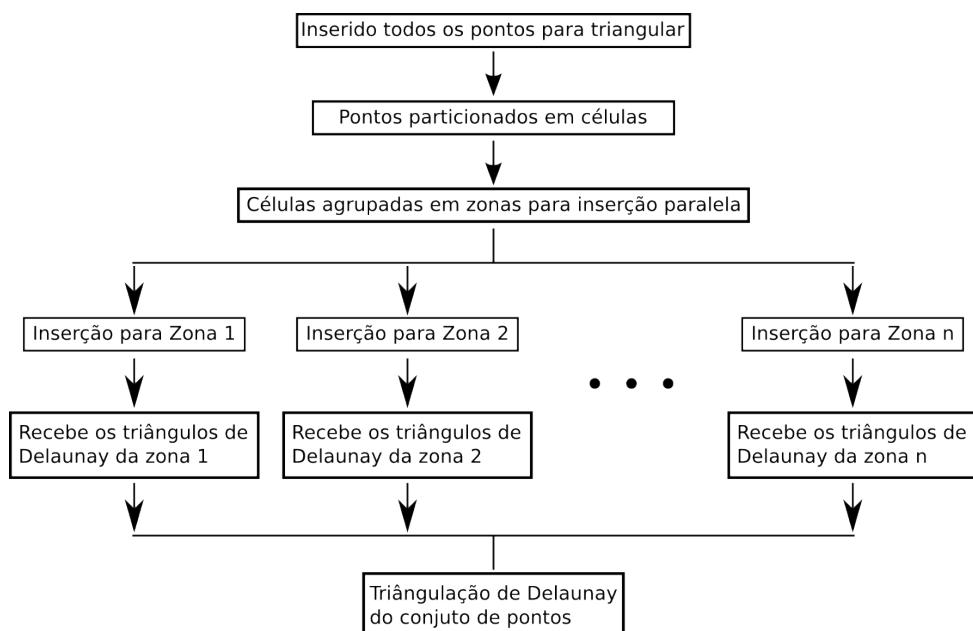


Figura 2.34: Fluxograma da triangulação em paralelo (LO, 2012).

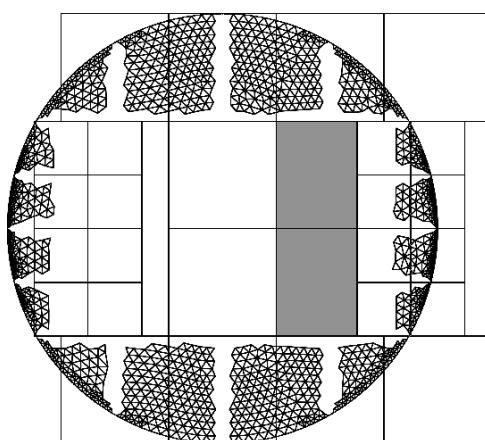


Figura 2.35: Malha gerada, espalhada entre os processos escravos, e as células da *quadtree* de decomposição deslocadas para a direção +X (FREITAS et al., 2013).

Fronteira, com subdivisão baseada em BSP, que recebe como entrada uma superfície de faces triangulares ou uma lista de arestas, para o caso bidimensional. Inicialmente uma *octree* é construída para estimar a carga no domínio de acordo com o tamanho das faces da superfície. As células internas da *octree* têm o tamanho definido de acordo com a maior e a menor face da superfície de entrada. Uma BSP é utilizada para partitionar o domínio de tal forma que a quantidade de subdomínios criados seja igual à quantidade de processadores disponíveis.

Após o particionamento, cada subdomínio pertencente a uma folha da árvore BSP gera sua malha por avanço de fronteira até que não seja mais possível avançar (quando chega no plano criado pela BSP), como mostra a Figura 2.36a. Quando os dois filhos de um nó da BSP terminam de gerar a malha nos seus respectivos subdomínios, o nó pai fica encarregado de juntar as duas malhas e, se necessário, terminar a geração da malha no novo subdomínio criado pela junção dos dois filhos, como mostra as Figuras 2.36b, 2.36c e 2.36d.

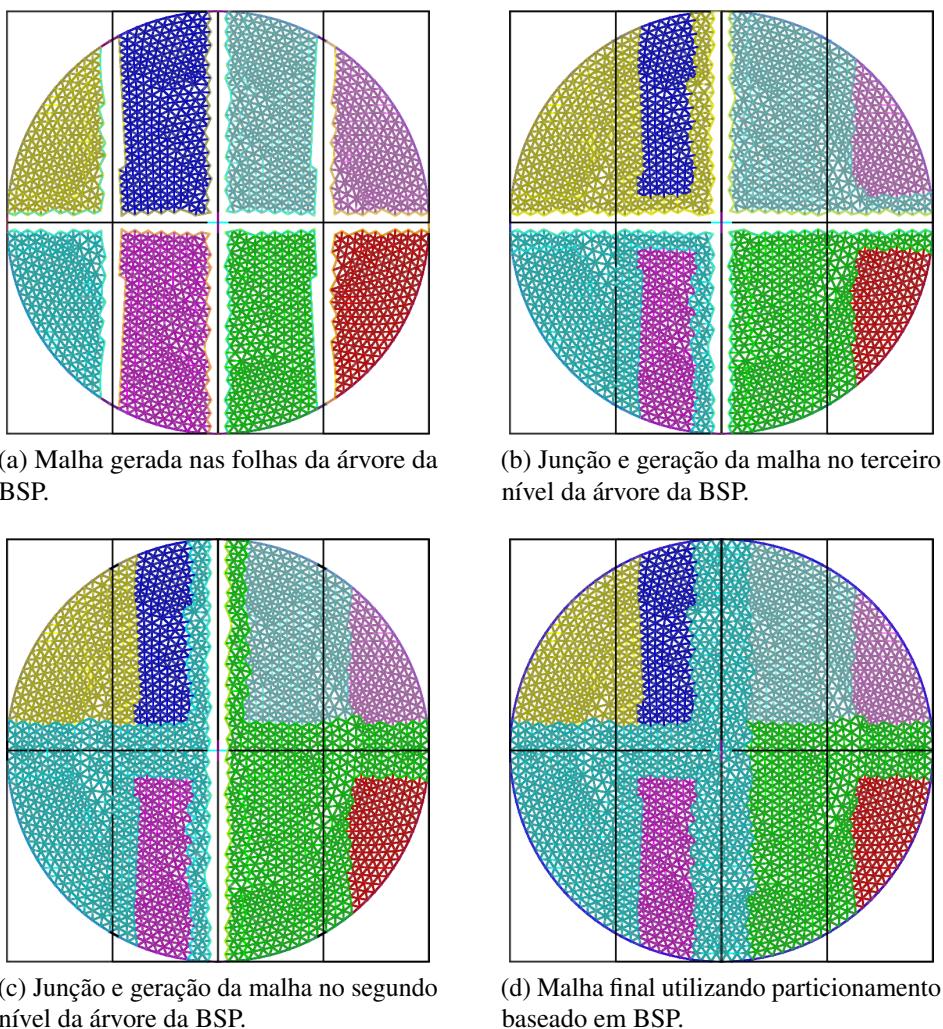


Figura 2.36: Passos da geração da malha no trabalho de (FREITAS; CAVALCANTE-NETO; VIDAL, 2014). Cada cor representa a malha gerada por um processador.

A grande vantagem dessa técnica está na utilização da BSP, que acaba permitindo a geração de subdomínios com cargas mais equilibradas e no ganho de velocidade ao se evitar os deslocamentos que aconteciam anteriormente. Por essa abordagem necessitar de uma sincronização e junção da malha em cada nível da BSP, há uma perda na velocidade apesar dessas junções serem feitas em paralelo por processos diferentes.

## 2.4 Considerações Finais

Este capítulo teve como finalidade apresentar todos os conceitos necessários para o leitor entender os passos básicos para a geração de malha em paralelo. Ao final deste capítulo o leitor terá condições de avaliar e classificar as diversas técnicas de geração de malha em paralelo que existem na literatura.

As técnicas que foram discutidas neste capítulo, em geral, apresentam bons resultados, mas a estimativa de carga na maioria deles é inexistente ou apenas uma métrica para contagem. Sem uma boa estimativa de carga, o balanceamento de carga pode ser um problema, fazendo o desempenho do algoritmo paralelo cair. A forma mais comum adotada pelos trabalhos é de realizar várias subdivisões até que a quantidade de subdomínios criados sejam maior que a quantidade de processadores disponíveis, com isso a falta de estimativa de carga é contornada.

A forma com que os cortes são posicionados nessas técnicas dependem bastante do formato do objeto de entrada. Além disso, algumas técnicas necessitam de intervenção de um usuário, ou seja, o processo de geração da malha não é totalmente automático. Uma boa abordagem de balanceamento e de decomposição de domínio é essencial para algoritmos de subdivisão de domínios, caso contrário, o tempo para geração de malha pode ser prejudicado e a qualidade da malha ser afetada.

## REFERÊNCIAS BIBLIOGRÁFICAS

- ANDREWS, G. R. *Concurrent Programming - Principles and Practice*. 1st. ed. [S.l.]: Addison-Wesley, 1991.
- ANDREWS, G. R. Foundations of parallel and distributed programming. Addison-Wesley Longman Publishing Co., Inc., 1999.
- ANDRÄ, H. e. a. Automatic parallel generation of tetrahedral grids by using a domain decomposition approach. *Computational Mathematics and Mathematical Physics*, v. 48, n. 8, p. 1367–1375, 2008.
- ANGEL, E. *Interactive Computer Graphics - A Top-Down Approach Using OpenGL*. 5th. ed. [S.l.]: Addison Wesley, 2008.
- BANK RANDOLPH E.; LU, S. A domain decomposition solver for a parallel adaptive meshing paradigm. *SIAM journal on scientific computing*, v. 26, n. 1, p. 105–127, 2005.
- BARKER, K. et al. A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems*, v. 15, n. 2, p. 183–192, 2004.
- BARNARD STEPHEN T.; SIMON, H. D. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, v. 6, n. 2, p. 101–117, 1994.
- BEN-ARI, M. *Principles of Concurrent and Distributed Programming*. 2nd. ed. [S.l.]: Addison-Wesley, 2006.
- BERG, M. de et al. *Computational Geometry: Algorithms and Applications*. 2nd. ed. [S.l.]: Springer-Verlag, 2000.
- BISWAS, R. et al. Parallel dynamic load balancing strategies for adaptive irregular applications. *Applied Mathematical Modelling*, v. 25, n. 2, p. 109–122, 2000.
- CARVALHO, P. C. P.; FIGUEIREDO, L. H. de. *Introdução à Geometria Computacional*. 1st. ed. [S.l.]: 18º Colóquio Brasileiro de Matemática, IMPA - Instituto Nacional de Matemática Pura e Aplicada, 1991.
- CAVALCANTE-NETO, J. B. *Simulação Auto-Adaptativa Baseada em Enumeração Espacial Recursiva de Modelos Bidimensionais de Elementos Finitos*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 1994.
- CAVALCANTE-NETO, J. B. *Geração de Malha e Estimativa de Erro para Modelos Tridimensionais de Elementos Finitos com Trincas*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, 1998.
- CAVALCANTE-NETO, J. B. et al. A back-tracking procedure for optimization of simplex meshes. *Communications in Numerical Methods in Engineering*, v. 21, n. 12, p. 711–722, 2005. ISSN 1069-8299.
- CAVALCANTE-NETO, J. B. et al. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *Engineering with Computers*, v. 17, n. 1, p. 75–91, 2001.

- CHANDRA, R. *Parallel programming in OpenMP*. [S.l.]: Morgan kaufmann, 2001.
- CHARMPIΣ DIMOS C.; PAPADRAKAKIS, M. Generation of balanced subdomain clusters with minimum interface for distributed domain decomposition applications. *Domain Decomposition Methods in Science and Engineering*, p. 555–562, 2005.
- CHEN, J. e. a. Improvements in the reliability and element quality of parallel tetrahedral mesh generation. *International Journal for Numerical Methods in Engineering*, v. 92, n. 8, p. 671–693, 2012.
- CHERNIKOV, A.; CHRISOCHOIDES, N. Parallel 2D graded guaranteed quality Delaunay mesh refinement. In: *Proceedings of the 14th International Meshing Roundtable*. [S.l.]: Sandia National Laboratory, 2005.
- CHERNIKOV, A. N.; CHRISOCHOIDES, N. P. Parallel guaranteed quality Delaunay uniform mesh refinement. *SIAM Journal on Scientific Computing*, v. 28, n. 5, p. 1907–1926, 2006.
- CHRISOCHOIDES, N. Parallel mesh generation. *Numerical Solution of Partial Differential Equations on Parallel Computers*, Springer-Verlag, v. 51, p. 237–259, 2005.
- CHRISOCHOIDES, N. A survey of parallel mesh generation methods. Brown University, Providence, 2005.
- CHRISOCHOIDES, N.; NAVÉ, D. Simultaneous mesh generation and partitioning for Delaunay meshes. *Mathematics and Computers in Simulation*, v. 54, n. 4-5, p. 321–339, 2000.
- DAS, S. K.; HARVEY, D. J.; BISWAS, R. Parallel processing of adaptive meshes with load balancing. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n. 12, p. 1269–1280, 2001.
- DECougny, H. L.; SHEPHARD, M. S. Parallel volume meshing using face removals and hierarchical repartitioning. *Computer Methods in Applied Mechanics and Engineering*, v. 174, n. 3-4, p. 275–298, 1999.
- DæHLEN, M.; HJELLE Øyvind. *Triangulations and Applications*. [S.l.]: Springer, 2006.
- EDELSBRUNNER, H. *Geometry and Topology for Mesh Generation*. 1st. ed. [S.l.]: Cambridge University Press, 2006.
- ELGINDY, H. An optimal speed-up parallel algorithm for triangulating simplicial point sets in space. *International Journal of Parallel Programming*, v. 15, n. 5, p. 389–398, 1986.
- FARHAT, C. A simple and efficient automatic fem domain decomposer. *Computers and Structures*, v. 28, n. 5, p. 579–602, 1988.
- FARLEY, J. *Java distributed computing*. [S.l.]: "O'Reilly Media, Inc.", 1998.
- FARRASHKHALVAT, M.; MILES, J. P. *Basic Structured Grid Generation - With an Introduction to Unstructured Grid Generation*. 1st. ed. [S.l.]: Butterworth-Heinemann, 2003.
- FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, v. 21, n. 9, p. 948–960, 1972.

- FOSTER, I. *Designing and building parallel programs*. [S.l.]: Addison Wesley Publishing Company Reading, 1995.
- FREITAS, M. de O.; CAVALCANTE-NETO, J. B.; VIDAL, C. A. *Parallel generation of meshes with cracks using a binary espacial decompositon*. [S.l.], 2014.
- FREITAS, M. O. *Geração em Paralelo de Malhas Triangulares por Avanço de Fronteira com Particionamento por Decomposição Espacial Recursiva*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO CEARÁ, 2010.
- FREITAS, M. O. et al. A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. *Advances in Engineering Software*, v. 59, p. 38–52, 2013.
- FREY, P. J.; GEORGE, P. L. *Mesh Generation - Application to Finite Elements*. 1st. ed. [S.l.]: ISTE Publishing Company, 2000.
- GAITHER, A. e. a. A paradigm for parallel unstructured grid generation. *Numerical Grid Generation in Computational Field Simulations*, p. 731–740, 1996.
- GALTIER Jérôme; GEORGE, P. L. Prepartitioning as a way to mesh subdomains in parallel. *International Meshing Roundtable*, 1996.
- GEORGE, P. L.; BOROUCHAKI, H. *Delaunay Triangulation and Meshing - Application to Finite Elements*. [S.l.]: Hermes, 1998.
- GLOBISCH, G. PARMESH – A parallel mesh generator. *Parallel Computing*, v. 25, p. 509–524, 1995.
- GLUT, B.; JURCZYK, T. Domain decomposition techniques for parallel generation of tetrahedral meshes. *Springer-Verlag Berlin Heidelberg*, p. 641–650, 2008.
- GOLUB, G. H.; LOAN, C. F. V. *Matrix Computations*. 3rd. ed. [S.l.]: The Johns Hopkins University Press, 1996.
- GRAMA, A. et al. *Introduction to Parallel Computing*. 2nd. ed. [S.l.]: Addison Wesley, 2003.
- GROPP, W. et al. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, Elsevier, v. 22, n. 6, p. 789–828, 1996.
- HEARN, D.; BAKER, M. P. *Computer Graphics - C Version*. 2nd. ed. [S.l.]: Prentice Hall, 1996.
- HJELLE, Ø.; DÆHLEN, M. *Triangulation and Applications - Mathematics and Visualizations*. 1st. ed. [S.l.]: Springer, 2006.
- HODGSON, D. C.; JIMACK, P. K. Efficient parallel generation of partitioned, unstructured meshes. *Advances in Engineering Software*, v. 27, n. 1-2, p. 59–70, 1996.
- HWANG, K.; XU, Z. *Scalable parallel computing: technology, architecture, programming*. [S.l.]: McGraw-Hill, Inc., 1998.
- ITO, Y. et al. Parallel unstructured mesh generation by an advancing front method. *Mathematics and Computers in Simulation*, v. 75, n. 5-6, p. 200–209, 2007.

- IVANOV, E.; ANDRÄ, H.; KUDRYAVTSEV, A. N. Domain decomposition approach for automatic parallel generation of tetrahedral grids. *Computational Methods in Applied Mathematics*, v. 6, n. 2, p. 178–193, 2006.
- JONES, M. T.; PLASSMAN, P. E. Unstructured mesh computations on networks of workstations. *Computer-Aided Civil and Infrastructure Engineering*, v. 15, n. 3, p. 196–208, 2000.
- JURCZYK TOMASZ; GŁUT, B. B. P. Parallel 3d mesh generation using geometry decomposition. *Numerical Methods in Industrial Forming Processes*, p. 1579–1584, 2007.
- KARYPIS, G.; KUMAR, V. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*. [S.I.], September 1998.
- KARYPIS, G.; SCHLOEGEL, K.; KUMAR, V. *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.2*. [S.I.], April 2011.
- KOHOUT, J.; KOLINGEROVÁ, I.; ŽÁRA, J. Parallel Delaunay triangulation in  $E^2$  and  $E^3$  for computers with shared memory. *Parallel Computing*, v. 31, n. 5, p. 491–522, 2005.
- LÄMMER, L.; BURGHARDT, M. Parallel generation of triangular and quadrilateral meshes. *Advances in Engineering Software*, v. 31, n. 12, p. 929–936, 2000.
- LARWOOD, B. G. et al. Domain decomposition approach for parallel unstructured mesh generation. *International Journal for Numerical Methods in Engineering*, v. 58, p. 177–188, 2003.
- LAWLOR, O. S. et al. ParFUM: A parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, v. 22, n. 3-4, p. 215–235, 2006.
- LINARDAKIS, L.; CHRISOCHOIDES, N. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing*, v. 27, n. 4, p. 1394–1423, 2006.
- LO, S. Parallel delaunay triangulation — application to two dimensions. *Finite Elements in Analysis and Design*, p. 37–48, 2012.
- LÖHNER, R. A parallel advancing front grid generation scheme. *International Journal for Numerical Methods in Engineering*, v. 51, n. 6, p. 663–678, 2001.
- LOHNER, R. Recent advances in parallel advancing front grid generation. *Archives of Computational Methods in Engineering*, v. 21, n. 2, p. 127–140, 2014.
- LÖHNER, R.; PARikh, P. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids*, v. 8, p. 1135–1149, 1988.
- MERKS, E. An optimal parallel algorithm for triangulating a set of points in the plane. *International Journal of Parallel Programming*, v. 15, n. 5, p. 399–411, 1986.
- MINYARD, T.; KALLINDERIS, Y. Parallel load balancing for dynamic execution environments. *Computer Methods in Applied Mechanics and Engineering*, v. 189, n. 4, p. 1295–1309, 2000.

- MIRANDA, A. C. de O. et al. Surface mesh regeneration considering curvatures. *Engineering with Computers*, Springer-Verlag London Limited, v. 25, n. 2, p. 207–219, 2009. ISSN 0177-0667.
- MIRANDA, A. C. O.; CAVALCANTE-NETO, J. B.; MARTHA, L. F. An algorithm for two-dimensional mesh generation for arbitrary regions with cracks. In: *SIBGRAPI '99: Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing*. [S.I.]: IEEE Computer Society, 1999. p. 29–38.
- MIRANDA, A. C. O. et al. Fatigue life and crack path predictions in generic 2D structural components. *Engineering Fracture Mechanics*, v. 70, p. 1259–1279, 2003.
- MORETTI, C. O. *Um Sistema Computacional Paralelo Aplicado à Simulação de Propagação Tridimensional de Fraturas*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2001.
- MPI Forum. The message passing interface (MPI) standard. Disponível em: <<http://www.mcs.anl.gov/research/projects/mpi>>.
- NIKISHKOV, G. P. e. a. An algorithm for domain partitioning with load balancing. *Engineering Computations*, v. 16, n. 1, p. 120–135, 1999.
- OKUSANYA, T.; PERAIRE, J. Parallel unstructured mesh generation. In: *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*. [S.I.]: Mississippi State University, 1996. p. 719–729.
- OLIKER, L.; BISWAS, R. PLUM: Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, v. 52, n. 2, p. 150–177, 1998.
- OpenMP Architecture Review Board. The OpenMP API specification for parallel programming. Disponível em: <<http://www.openmp.org>>.
- OWEN, S. J. A survey of unstructured mesh generation technology. In: *Proceedings of the 7th International Meshing Roundtable*. Michigan, United States: Sandia National Laboratory, 1998. p. 239–267.
- PANITANARAK THAP; SHONTZ, S. M. Mdec: Metis-based domain decomposition for parallel 2d mesh generation. *Procedia Computer Science*, v. 4, p. 302–311, 2011.
- PÉBAY, P. P. et al. pCAMAL: An embarrassingly parallel hexahedral mesh generator. In: *Proceedings of the 16th International Meshing Roundtable*. Washington, United States: Sandia National Laboratory, 2007.
- PERSSON, P. O. PDE-based gradient limiting for mesh size functions. In: *Proceedings of the 13th International Meshing Roundtable*. [S.I.]: Sandia National Laboratory, 2004. p. 377–387.
- PHONGTHANAPANICH, S.; DECHAUMPHAI, P. Adaptive Delaunay triangulation with object-oriented programming for crack propagation analysis. *Finite Element Analysis and Design*, v. 40, n. 13-14, p. 1753–1771, 2004.
- PIRZADEH, S. Z.; ZAGARIS, G. Domain decomposition by the advancing-partition method for parallel unstructured grid generation. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting*. [S.I.]: AIAA - American Institute of Aeronautics and Astronautics, 2009.

- PREPARATA, F. P.; SHAMOS, M. I. *Computational Geometry: An Introduction*. 2nd. ed. [S.I.]: Springer-Verlag, 1991.
- RIVARA, M. C. et al. Parallel decoupled terminal-edge bisection method for 3D mesh generation. *Engineering with Computers*, v. 22, p. 111–119, 2006.
- RUPPERT, J. A new and simple algorithm for quality 2-dimensional mesh generation. In: *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms*. Texas, United States: SIAM - Society for Industrial and Applied Mathematics, 1999. p. 83–62.
- SAID, R. e. a. Distributed parallel delaunay mesh generation. *Computer methods in applied mechanics and engineering*, v. 177, n. 1, p. 109–125, 1999.
- SEGERLIND, L. J. *Applied Finite Element Analysis*. 2nd. ed. [S.I.]: John Wiley and Sons, 1984.
- SGI - Silicon Graphics International. OpenGL - the industry's foundation for high performance graphics. Disponível em: <<http://www.opengl.org>>.
- SHEWCHUK, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: LIN, M. C.; MANOCHA, D. (Ed.). *Applied Computational Geometry: Towards Geometric Engineering*. [S.I.]: Springer-Verlag, 1996, (Lecture Notes in Computer Science, v. 1148). p. 203–222. From the First ACM Workshop on Applied Computational Geometry.
- SHEWCHUK, J. R. *Delaunay Refinement Mesh Generation*. Tese (Doutorado) — School of Computer Science, Carnegie Mellon University, 1997.
- SHREINER, D. et al. *OpenGL Programming Guide*. 6th. ed. [S.I.]: Addison-Wesley, 2007.
- SIMON, H. D. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, v. 2, n. 2, p. 135–148, 1991.
- SPEAR, A. D. et al. Structural assessment and prognosis using a multi-scale, fracture mechanics-based approach. In: *Proceedings of the 2010 Aircraft Airworthiness and Sustainment Conference*. Texas, United States: [s.n.], 2010.
- TENG, Y. A. et al. A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation. In: *Proceedings of the 1993 Conference on High Performance Network and Computing*. Oregon, United States: Association for Computing Machinery, 1993. p. 112–121.
- VIDWANS A.; KALLINDERIS, Y. V. V. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. *AIAA*, v. 32, n. 3, p. 497–505, 1994.
- WAWRZYNEK, P. A.; INGRAFFEA, A. R. *FRANC2D - A Two-Dimensional Crack Propagation Simulator - User's guide - Version 3.1*. [S.I.], 1993.
- WILSON, J. K.; TOPPING, B. H. V. Parallel adaptive tetrahedral mesh generation by the advancing front technique. *Computers & Structures*, v. 68, n. 1-3, p. 57–78, 1998.
- WU POTING; HOUSTIS, E. N. Parallel adaptive mesh generation and decomposition. *Concurrency: Practice and Experience*, v. 12, n. 3-4, p. 155–167, 1996.
- wxTeam. wxWidgets - cross-platform GUI library. Disponível em: <<http://www.wxwidgets.org>>.

YOSHIMURA, S.; WADA, Y.; YAGAWA, G. Automatic mesh generation of quadrilateral elements using intelligent local approach. *Computer Methods in Applied Mechanics and Engineering*, v. 179, n. 1, p. 125–138, 1999.

ZAGARIS, G.; PIRZADEH, S. Z.; CHRISOCHOIDES, N. A framework for parallel unstructured grid generation for practical aerodynamic simulations. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting*. [S.l.]: AIAA - American Institute of Aeronautics and Astronautics, 2009.

ZHENG YAO; CHEN, J. Unstructured mesh generation and its parallelization. *Computational Mechanics*, p. 22–35, 2009.