UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DANIEL NASCIMENTO TEIXEIRA

UMA TÉCNICA DE DECOMPOSIÇÃO DE DOMÍNIOS *A PRIORI* PARA GERAÇÃO AUTOMÁTICA DE MALHAS TETRAÉDRICAS EM PARALELO

FORTALEZA

2019

DANIEL NASCIMENTO TEIXEIRA

UMA TÉCNICA DE DECOMPOSIÇÃO DE DOMÍNIOS *A PRIORI* PARA GERAÇÃO AUTOMÁTICA DE MALHAS TETRAÉDRICAS EM PARALELO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Computação Gráfica

Orientador: Prof. Dr. Joaquim Bento Cavalcante Neto

Coorientador: Prof. Dr. Creto Augusto Vidal

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

T265t Teixeira, Daniel Nascimento.
Uma técnica de decomposição de domínios a priori para geração automática de malhas tetraédricas em paralelo / Daniel Nascimento Teixeira. – 2019.
123 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2019.
Orientação: Prof. Dr. Joaquim Bento Cavalcante Neto.
Coorientação: Prof. Dr. Creto Augusto Vidal.

1. Decomposição de domínio. 2. Geração em paralelo de malhas. 3. Tetraedralização. I. Título.

CDD 005

DANIEL NASCIMENTO TEIXEIRA

UMA TÉCNICA DE DECOMPOSIÇÃO DE DOMÍNIOS *A PRIORI* PARA GERAÇÃO AUTOMÁTICA DE MALHAS TETRAÉDRICAS EM PARALELO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Computação Gráfica

Aprovada em: 29 de Agosto de 2019

BANCA EXAMINADORA

Prof. Dr. Joaquim Bento Cavalcante Neto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Creto Augusto Vidal (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Markos Oliveira Freitas
Universidade Federal do Ceará (UFC)

Prof. Dr. Evandro Parente Junior
Universidade Federal do Ceará (UFC)

Prof. Dr. Luiz Fernando Campos Ramos Martha
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter-me dado forças e motivação para chegar até onde estou.

Aos meus pais Arimatéa e Doralice que tanto se esforçaram para que eu tivesse uma boa formação. Aos meus irmão Davi e Araceli pela companhia e pela ajuda que me ajudaram da forma que eles podiam para melhorar meu trabalho. Aos meus tios, primos e avós que sempre se importaram comigo e ajudaram na minha criação. E a minha querida namorada Alice Paiva que tanto me ajudou nessa jornada.

Agradeço aos professores orientadores, Joaquim Bento e Creto Vidal, por terem acreditado no meu trabalho e me guiado na minha jornada acadêmica. Aos professores Evandro Parente Junior e Luiz Fernando Martha, pelas observações e contribuições que engrandeceram este trabalho. Em especial agradeço o meu tutor Markos Freitas, que me acompanha desde a graduação e trouxe inúmeras contribuições para o trabalho assim como ajudou bastante no meu crescimento como pesquisador e desenvolvedor.

Aos meus amigos de laboratório Yuri Lenon, Roberto, Rubens, Teófilo, Martha, Rafael Ivo, Jonas, Caio, Suzana, Rafael Siqueira, Laise, Lílian, Arnaldo, Ricado Lenz, Danilo, e aos outros nomes que tenha esquecido, pelas conversas e discussões sobre os mais variados temas. Agradeço também a todos os amigos que tanto torceram por mim e aos que estiveram presente na apresentação deste trabalho em especial. E ao professor Javam que concedeu toda a estrutura do laboratório LSBD para meus estudos e suporte.

Agradeço ao Programa de Mestrado e Doutorado em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC) e ao Grupo de Pesquisa em Computação Gráfica, Realidade Virtual e Animação (CRAb), pela oportunidade dada. Ao NACAD por permitir acesso ao seu *cluster* para executar os testes da minha aplicação.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Nunca deixe alguém dizer que você não pode fazer algo, nem mesmo eu. Tudo bem? Você tem que proteger o seu sonho. Quando as pessoas não podem fazer algo, elas dizem que você também não pode. Se você quer algo, vá buscar"

(Chris Gardner)

RESUMO

Este trabalho propõe uma técnica totalmente automatizada para decomposição de domínios para geração de malhas usando computadores paralelos com memória distribuída. Esta técnica depende de uma estrutura de decomposição que usa planos alinhados aos eixos para decompor o domínio. Esses planos de decomposição são determinados por uma estrutura de partição binária baseada em uma *quadtree* refinada (no caso bidimensional) ou *octree* (no caso tridimensional) que é construída para estimar a quantidade de trabalho computacional necessária para gerar a malha interna. Assim, a quantidade de trabalho computacional em cada subdomínio é aproximadamente a mesma. O nível de refinamento da *quadtree* ou *octree* é usado para guiar a criação da interface de cada subdomínio, definida por sua discretização intercelular. A geração da malha de interface é executada *a priori*, ou seja, cada subdomínio tem sua malha de interface totalmente criada e aprimorada antes da fase de geração da malha interna. Esta técnica gera novos domínios completamente independentes uns dos outros e permite abstrair a técnica de geração de malha aplicada aos subdomínios, que podem combinar, por exemplo, Delaunay e técnicas de Avanço de Fronteira, dentre outras. Além disso, esta técnica de estimativa de carga produz resultados que representam com precisão o número de elementos a serem gerados em cada subdomínio, levando a uma previsão adequada do tempo de execução e um algoritmo bem balanceado, um recurso desejável no processamento paralelo que geralmente é difícil de alcançar. As malhas geradas com a técnica paralela têm uma qualidade similar daquelas geradas serialmente, dentro de limites aceitáveis, o que é desejado de abordagens paralelas.

Palavras-chave: Decomposição de domínio. Geração em paralelo de malhas. Tetraedralização.

ABSTRACT

This work proposes a fully automated technique for domain decomposition to generate meshes using parallel computers with distributed memory. This technique relies on a partitioning structure that uses axis-aligned planes to decompose the domain. These decomposing planes are determined by a binary partitioning structure based on a refined quadtree (in two-dimensional case) or octree (in three-dimensional case) that is built to estimate the amount of work necessary to generate the whole mesh. Thus, the amount of work in each subdomain is approximately the same. The level of refinement of the quadtree or octree is used to guide the creation of each subdomain's interface, defined by its inter-cell discretization. The interface mesh generation is performed *a priori*, i.e., every subdomain has its interface mesh fully created and improved before the internal mesh generation phase. This technique generates new domains completely independent from one another and allows to abstract the mesh generation technique applied to the subdomains, which can combine, for example, Delaunay and Advancing Front Techniques, among others. Also, the load estimation technique produces results that accurately represent the number of elements to be generated in each subdomain, leading to an adequate prediction of execution time and a well-balanced algorithm, a desirable feature in parallel processing that is usually difficult to achieve. The meshes generated with the parallel technique have the same quality as those generated serially, within acceptable limits, which is desired from parallel approaches.

Keywords: Domain decomposition. Parallel mesh generation. Tetrahedralization.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O poder de processamento dos computadores vem crescendo muito nos últimos anos. Já é comum, inclusive, computadores pessoais terem processadores com diversos núcleos. Desenvolver programas ou algoritmos que não utilizam completamente os recursos disponíveis nas máquinas resulta em um desperdício na capacidade de processamento.

Ainda é preciso, entretanto, desenvolver técnicas que tentem alcançar uma boa escalabilidade, ou seja, ter um aumento no seu desempenho quando mais recursos computacionais forem disponíveis, mantendo ou melhorando os resultados obtidos pelo algoritmo sequencial.

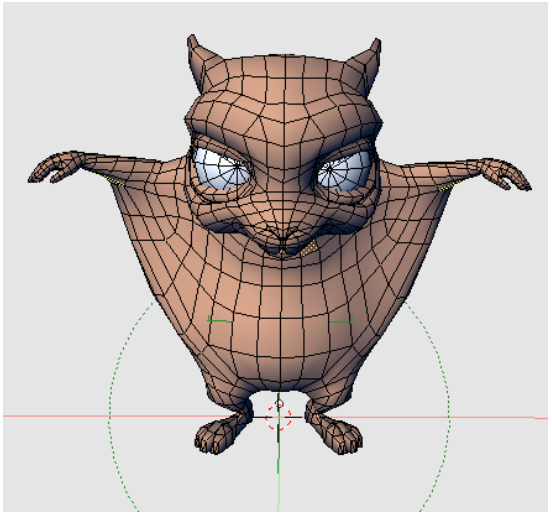
Na área de geração de malhas em paralelo, é possível dividir as técnicas de decomposição em dois grupos, que são as técnicas *a priori* e as *a posteriori*. As técnicas *a priori* criam uma malha de interface para cada partição feita e, após todas as partições estarem criadas, é que a fase de geração de malha nos subdomínios ocorre. Nas técnicas *a posteriori*, não é necessária a criação de uma malha de interface para cada partição pois, normalmente, é definida uma zona entre duas partições onde a malha nessa região é gerada após fase de geração de malha dos subdomínios que compartilham essa zona.

Este trabalho descreve uma técnica *a priori* de decomposição de domínios para geração de malhas triangulares ou tetraédricas usando computadores paralelos com memória distribuída ou compartilhada. Uma das vantagens desse trabalho é que, antes de realizar a decomposição da entrada, são feitos alguns testes para evitar a criação de regiões ruins ou regiões com uma grande diferença de carga.

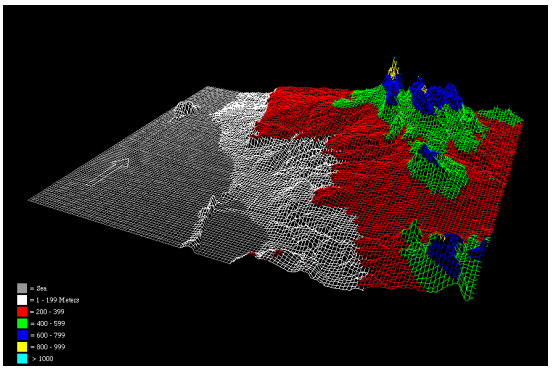
Por ser uma técnica *a priori*, os subdomínios são considerados criados após a geração de suas malhas de interface. Os subdomínios recém-criados são completamente independentes de seus subdomínios vizinhos. Portanto, suas malhas internas podem ser facilmente geradas em computadores paralelos.

Malhas estão presentes no nosso dia a dia e são amplamente utilizadas para representar geometrias e outras informações. Alguns exemplos de aplicações são: em computação gráfica (CG) (Figura 1a), sistemas de informações geográficas (SIG) (Figura 1b), projetos assistidos por computadores (CAD) (Figura 1c) e na engenharia, ajudando na análise e simulação de fenômenos físicos que utilizam o Método dos Elementos Finitos (MEF) (Figura 1d).

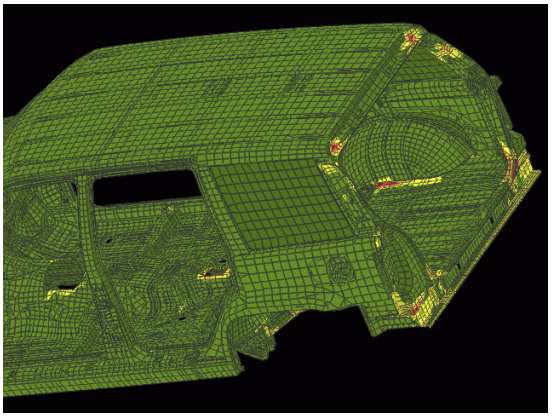
Grandes malhas são necessárias e usadas em aplicações reais para se obter melhores resultados. Além do tamanho, a qualidade dos elementos da malha é fundamental em algumas aplicações, como as da Engenharia. Assim, para um bom gerador de malhas executar em paralelo,



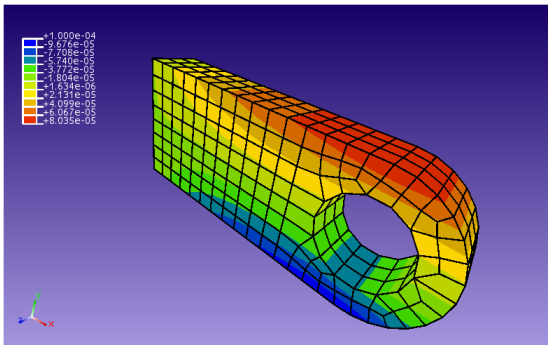
(a) Malha de computação gráfica. Fonte: www.yofrankie.org.



(b) Malha de sistemas de informação geográfica. Fonte: classics.uc.edu.



(c) Malha de projetos assistidos por computadores. Fonte: www.research.ibm.com.



(d) Malha utilizada na engenharia para MEF. Fonte: elaborado pelo autor (2019).

Figura 1 – Exemplos de aplicações para malhas.

é necessário que não haja perda na qualidade da malha gerada, em relação ao gerador de malhas sequencial. No MEF, por exemplo, se uma malha tiver uma grande quantidade de elementos ruins (polígonos, em malhas bidimensionais, e poliedros, em malhas tridimensionais), é possível que este não convirja. Por necessitar de malhas bastante refinadas, é comum que as análises de elementos finitos usem malhas grandes, com dezenas de milhões de elementos.

A geração de malha em paralelo geralmente envolve duas fases distintas: a decomposição (ou particionamento) do domínio em partes menores, chamados de subdomínios (ou partições), que são enviadas para os processadores que estão disponíveis; e a geração de malha em cada uma dessas partes. Em geral se usa a mesma técnica de geração para cada um dos subdomínios usados, como triangulação de Delaunay ou Avanço de Fronteira, por exemplo, o que pode ser uma restrição para a geração. Além disso, deve-se procurar obter a melhor escalabilidade possível, a fim de utilizar eficientemente todos os recursos disponíveis dos computadores.

1.1 Objetivos e Contribuições

O principal objetivo deste trabalho é propor uma técnica *a priori* de decomposição de domínios para geração paralela da malha. Essa técnica gera subdomínios sem modificar o modelo de entrada.

A técnica proposta é dita *a priori* porque a malha de interface entre os subdomínios é gerada antes das suas malhas internas. Assim, os subdomínios são independentes uns dos outros, e seria possível atribuir diferentes técnicas de geração de malhas (Delaunay, Avanço de Fronteira, etc) a diferentes subdomínios. No entanto, na prática, isso seria útil somente se houvesse um processo de inteligência incorporado, capaz de realizar a atribuição automática.

A técnica de decomposição de domínio foi projetada independentemente da técnica de geração de malha escolhida a ser usada nos subdomínios, para atender aos seguintes requisitos:

1. Respeitar os elementos da fronteira de entrada (FE), sem realizar qualquer alteração ou refinamento nela;
2. Produzir elementos de boa qualidade, isto é, elementos gerados devem ter boas proporções;
3. Fornecer boas transições entre regiões com elementos refinados para regiões com elementos grosseiros;
4. Manter a compatibilidade na fronteira compartilhada (elementos de interface) de qualquer par de subdomínios vizinhos;
5. Apresentar uma boa estimativa para o número de elementos gerados em cada subdomínio;
6. Apresentar um bom balanceamento de carga, distribuindo os subdomínios da maneira mais uniforme possível entre os processos paralelos; e
7. Tentar gerar malhas da maneira mais eficiente em termos de tempo de processamento.

O primeiro requisito é muito importante em muitos problemas, como aqueles encontrados em simulações em que o domínio é resultado de alguma digitalização ou que possa conter regiões com diferentes materiais e/ou furos. Nesses problemas, é normalmente desejável que a malha esteja de acordo com a discretização do limite existente das várias regiões. Em relação ao segundo e terceiro requisito, diversos procedimentos e refinamentos são realizados para garantir que os elementos gerados tenham boa qualidade e proporções compatíveis com a região onde se encontram. Assim, para obter uma boa transição, diferenças de magnitude maiores que duas devem ser evitadas.

De acordo com o quarto requisito, é necessário manter a conformidade da malha para facilitar a junção das malhas dos subdomínios. Para que isso aconteça, uma face gerada

em um subdomínio precisa ter uma contraparte simétrica no subdomínio vizinho. Isso garante a total compatibilidade das malhas geradas nos dois lados de uma interface.

O quinto e o sexto requisitos estão intimamente relacionados e afetam diretamente o sétimo requisito. Para se ter um bom equilíbrio entre as cargas de trabalho dos subdomínios, uma boa estimativa de carga é necessária. Para medir essa carga de trabalho, uma estrutura de dados do tipo árvore é usada e, com base nessa estrutura e em outros critérios, os subdomínios são definidos.

O último requisito é bastante importante, pois a velocidade e a qualidade da malha são, na verdade, os dois principais objetivos da geração de malhas paralelas.

1.2 Organização do Trabalho

O restante deste trabalho está organizado em cinco capítulos. No Capítulo 2, são apresentados alguns conceitos que são usados nos capítulos subsequentes. No Capítulo 3,é apresentada uma visão geral das técnicas de decomposição de domínios e de malhas. No Capítulo 4, é apresentada a técnica de decomposição de domínios desenvolvida, detalhando a técnica e a criação das estruturas de dados envolvidas no processo de estimativa da carga e particionamento do domínio. No Capítulo 5, são apresentados os resultados dos testes preparados para demonstrar a eficácia da técnica proposta, e os resultados obtidos são analisados. Por fim, no Capítulo 6, são apresentadas as conclusões sobre o trabalho e são propostas algumas ideias para trabalhos futuros.

2 CONCEITOS PRELIMINARES

Em Computação de Alto Desempenho é conhecido que, para um bom algoritmo paralelo executar, é preciso uma boa estratégia para a divisão da entrada e para a junção das várias soluções ao final.

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 isso acontecer, a carga foi devidamente balanceada entre os processadores.

Neste capítulo, são apresentados inicialmente alguns conceitos necessários para um melhor entendimento dos trabalhos que estão sendo desenvolvidos nos últimos anos na área de decomposição de domínios para geração de malhas em paralelo. Logo em seguida, serão apresentados os trabalhos relacionados a subdivisão de domínios, com ênfase no modo que são tratadas as subdivisões dos domínios e a estimativa de carga em cada técnica apresentada.

2.1 Geometria Computacional

A geometria computacional é a área da computação que estuda estruturas de dados e soluções para problemas geométricos. O seu enfoque é buscar soluções sob o ponto de vista da análise de complexidade de algoritmos. Entre os problemas estudados, estão a construção de fechos convexos, de triangulações, a ordenação de pontos espaciais, a checagem de interseções interseções entre retas/planos, a geração de malhas e outros mais.

2.1.1 Fecho Convexo

O fecho convexo de um conjunto finito de pontos é o menor conjunto convexo que contém tais pontos. Segundo (CARVALHO; FIGUEIREDO, 1991), um conjunto K do $\Re^n$, sendo n um inteiro não negativo, é convexo se quaisquer que sejam $x \in K$, $y \in K$ e $0 \leq \lambda \leq 1$, $\lambda \in \Re$, tem-se $\lambda x + (1-\lambda)y \in K$. Ou seja, todas as combinações convexas dos elementos de K pertencem a K. Um ponto p é dito ser combinação convexa dos pontos $p_i \in K$ se $p = \sum_{i=1}^{|K|} p_i \lambda_i$, sendo $\sum_{i=1}^{|K|} \lambda_i = 1$.

Um ponto $w \in K$ é um ponto extremo de K se não pode ser conectado a um elemento de K por um segmento de reta aberto pertencente a K. O fecho convexo de um conjunto finito

C de pontos no $\Re^n$, sendo n um inteiro não negativo, é o conjunto de todas as combinações convexas de elementos de C. Em outras palavras, pode-se dizer que o fecho convexo é um conjunto finito de pontos que formam um polígono convexo com menor área geométrica que engloba todos os pontos do conjunto (Figura 2).

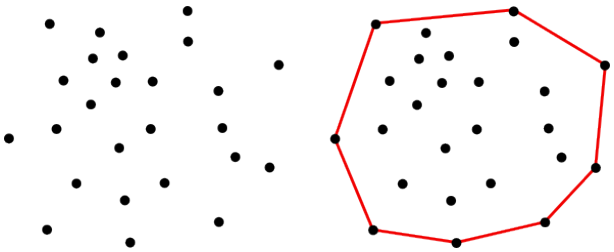


Figura 2 – Conjunto de pontos e o seu fecho convexo.
Fonte: elaborado pelo autor (2019).

2.1.2 Triangulação e Malha

Existem diversos tipos de malhas, dentre as mais conhecidas e utilizadas estão as malhas que utilizam triângulos. Por ser a menor estrutura geométrica que consegue representar modelos tando no espaço bidimensional como no tridimensional (Figura 3), as malhas triangulares se tornaram as mais utilizadas nas pesquisas da área de Geometria Computacional. Nos casos tridimensionais existem as malhas tetraédricas, que são uma extensão natural da malha triangular para o 3D.

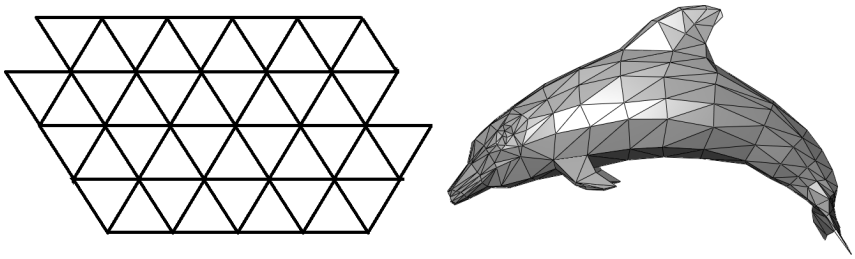


Figura 3 – Exemplo de duas malhas triangulares, uma bidimensional e outra tridimensional.
Fonte: Leapfrog: http://support.lpfrg.com/support/home.

Triangulações ou tetraedralização são casos específicos de malhas, utilizadas em diversas aplicações 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:

- $\Omega = \bigcup_{k \in M} k$, onde Ω 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, logo uma tetraedralização também é uma malha válida. A Figura 4 mostra uma malha bidimensional inválida. Neste trabalho quando for mencionado malha, será sempre a malha que respeita as mesmas propriedades de uma triangulação. A definição de triangulação segundo (HJELLE; DÆHLEN, 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 dois 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 devem 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 com, no máximo dois triângulos.

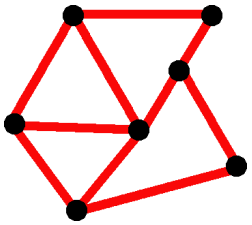


Figura 4 – Exemplo de uma malha bidimensional que não é uma triangulação válida.
Fonte: elaborado pelo autor (2019).

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 é definida como não-estruturada. Nas malhas estruturadas, o conhecimento dos vizinhos de cada elemento não depende do armazenamento ou existência desta informação, pois a vizinhança pode ser facilmente descoberta pela indexação. Já nas não-estruturadas, para se ter conhecimento dos vizinhos, é necessário armazenar ou calcular estas informações (Figura 5). Existem ainda as malhas mistas, que combinam elementos de tipos diferentes, tais como triângulos, quadriláteros, tetraedros, hexaedros, pirâmides, prismas dentre outros.

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 5.

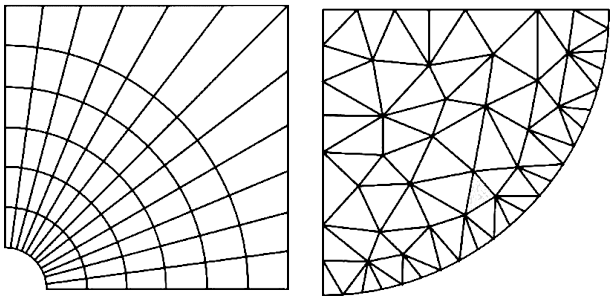


Figura 5 – Exemplo de malha estruturada quadrilateral e não-estruturada triangular.
Fonte: elaborado pelo autor (2019).

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. Em 3D, a métrica é $\alpha = 3R_i/R_c$, onde R_i e R_c são os raios das esferas inscrita e circunscrita, respectivamente.

Esta métrica α tem valor $1,0$ para um triângulo/tetraedro equilátero. Quanto pior a qualidade do elemento, mais próximo de $0,0$ é o valor de α. 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 6.

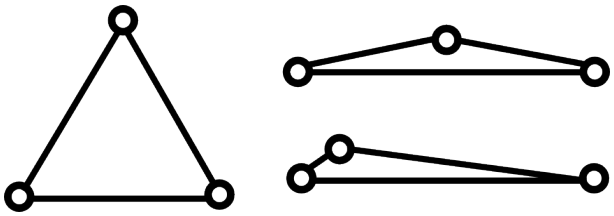


Figura 6 – Exemplo de um elemento triangular de boa qualidade (esquerda) e dois de péssima qualidade (direita).
Fonte: elaborado pelo autor (2019).

2.2 Geração de Malha

Nesta seção, são apresentadas algumas técnicas de geração de malha conhecidas. Existem diversos algoritmos para geração de malhas, porém eles podem ser enquadrados em 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.

2.2.1 Avanço de Fronteira

Esse 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 7a). 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.

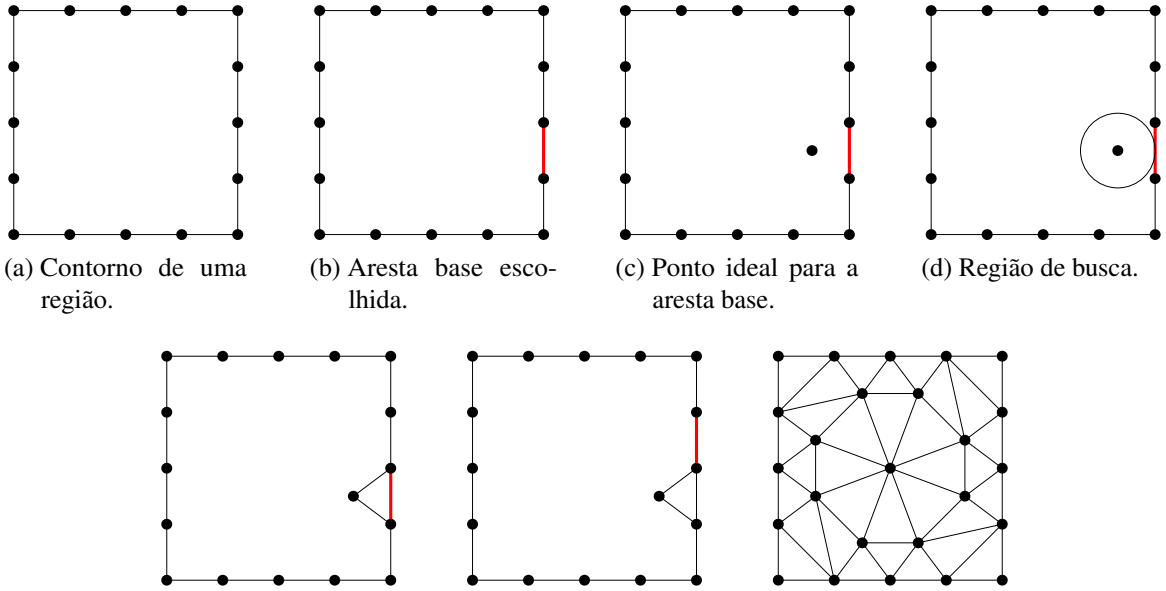
Há casos em que o algoritmo não consegue mais gerar elementos para uma determinada fronteira, isso indica que o algoritmo falhou. O caso de falha só ocorre quando todos os possíveis elementos a serem criados se sobrepõem a um elemento já existente. Por isso, é importante verificar se os elementos se interceptam. Os casos de falha só acontecem em modelos tridimensionais. Entretanto, existem técnicas para contornar esses problemas e gerar malhas em modelos cuja malha normalmente não seria gerada (CAVALCANTE-NETO *et al.*, 2005).

Para gerar os novos triângulos no interior do domínio, é necessário criar novos pontos que não pertencem aos dados de entrada. Em alguns casos, os pontos de Steiner são utilizados para isso, como mostrado em Erten e Üngör (2009).

Um algoritmo de avanço de fronteira procede da seguinte maneira no caso 2D (Figura 7):

1. Selecione uma aresta da fronteira, a aresta base (Figura 7b);
2. Encontre um ponto ideal para a formação de um novo triângulo com a aresta base (Figura 7c);
3. Crie uma região de busca em torno desse ponto ideal (Figura 7d);
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 (Figura 7e);
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.



(a) Contorno de uma região. (b) Aresta base escolhida. (c) Ponto ideal para a aresta base. (d) Região de busca.

(e) Formação do novo triângulo. (f) Nova aresta base escolhida. (g) Malha final.

Figura 7 – Avanço de fronteira (FREITAS, 2010).

Pelo fato de a 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 MEF).

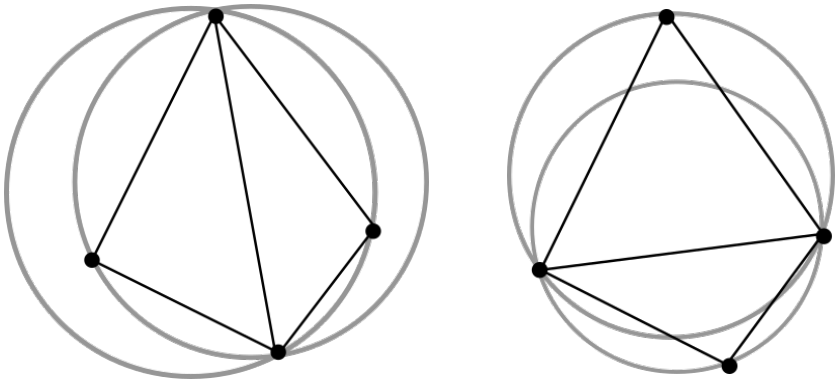
Porém, ao contrário dos elementos mais próximos da borda, os elementos mais internos à malha nem sempre têm boa qualidade, devido a região tornar-se menor à medida que a fronteira avança. Geralmente, uma técnica de suavização ou otimização é aplicada na malha resultante do algoritmo para melhorar a qualidade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

2.2.2 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, é feita uma busca pelo melhor ponto, entre os já existentes, para formar

um novo triangulo, caso não seja encontrado, um ponto deve ser criado.

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 8). 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.



(a) Critério de Delaunay falhando para os dois triângulos.

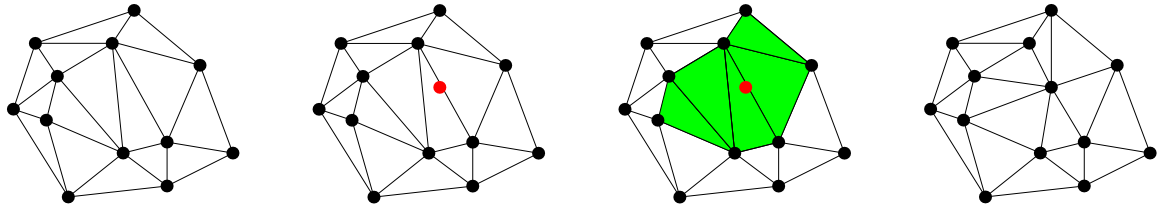
(b) Triangulação válida respeitando o critério de Delaunay.

Figura 8 – Critério Delaunay aplicado nas possíveis triangulações de 4 pontos.
Fonte: elaborado pelo autor (2019).

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 (para garantir que a final seja de Delaunay, a inicial já tem que ser de Delaunay) e se modifica essa malha (de apenas um subconjunto de pontos da entrada) pré-existente (Figura 9).

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 (RUPPERT, 1999).



(a) Triangulação inicial. (b) Novo vértice inserido. (c) Triângulos que violam o critério. (d) Triangulação final.

Figura 9 – Triangulação por inserção de vértices (FREITAS, 2010).

2.2.3 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 em uma complexidade na ordem de $O(n \log n)$.

2.3 Geração de Malha em Paralelo

Na geração em paralelo, é necessário dividir a entrada para realizar o processamento em paralelo dos diversos subdomínios. Existem duas formas de decompor o domínio. Na primeira forma, é criada sequencialmente uma malha grosseira da região e dividida entre os processadores. Essa forma, chamada de decomposição discreta do domínio, envolve ainda o problema de decompor a 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. 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 um conjunto de triângulos ou tetraedros, no caso tridimensional. Essa malha de interface faz a conexão entre dois subdomínios vizinhos, e faz o papel de uma nova fronteira. A forma que ela é criada depende da técnica que está sendo utilizada para decompor o domínio.

A decomposição contínua pode ainda, ser subdividida em duas categorias, depen-

dendo 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. O trabalho de deCougny e Shephard (1999) apresenta uma classificação de algoritmos de subdivisão de malhas.

Existe outra classificação apresentada por Chrisochoides (2005) que classifica as técnicas de acordo com a maneira que o domínio é decomposto. Podemos ter então técnicas classificadas como Discretas ou Contínuas.

As técnicas de decomposição Discreta geralmente utilizam uma malha grosseira baseada na entrada dada. Primeiramente, são definidas as regiões e em seguida as bordas destas regiões são refinadas (tanto as interfaces entre dois subdomínios quanto o contorno dado como entrada). Após isso, as regiões também chamadas de sub-malhas, estarão prontas para a geração da malha.

Já as técnicas de decomposição Contínua não utilizam um malha grosseira. As regiões criadas serão compostas por parte do contorno de entrada juntamente com uma parte da região interna, criando assim subdomínios. A região interna pode necessitar da criação de uma interface antes da geração da malha (*a priori*), assim como pode ser gerada depois (*a posteriori*). Uma vantagem da decomposição contínua em relação a discreta é que a entrada não é modificada.

2.4 Estrutura de Dados

Diversas estruturas de dados, que foram criadas na área da computação, são 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 se localiza, 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 10). Entre as estruturas de dados espaciais, as mais importantes são a *quadtree*, *octree* e a *binary space partitioning* (BSP).

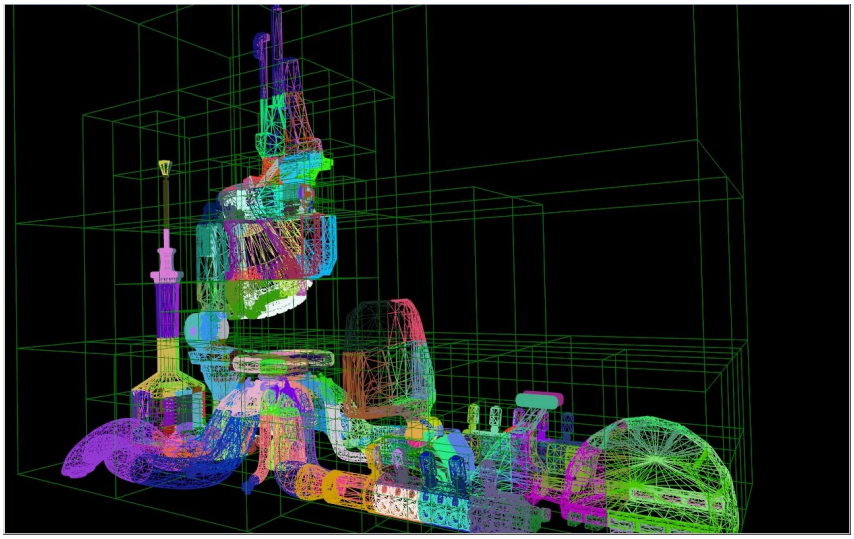


Figura 10 – Exemplo de uma decomposição espacial feita para renderização e teste de colisão.
Fonte: http://togeskov.net/

2.4.1 Quadtree

Uma *quadtree* é uma estrutura de dados baseada em árvore em que cada nó interno possui exatamente quatro filhos (Figura 11). 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), a 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 que existem um quadrante da *quadtree*, com isto é garantida a quantidade máxima de pontos internos a cada célula desta *quadtree*.

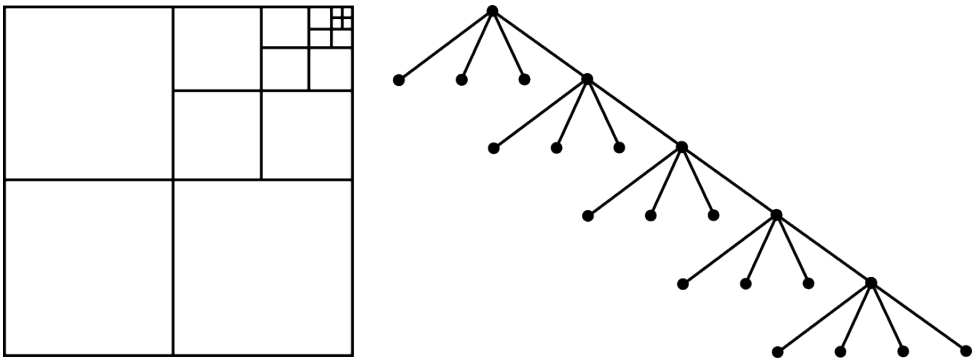


Figura 11 – Uma subdivisão feita por *quadtree* com cinco níveis e sua representação em árvore.
Fonte: elaborado pelo autor (2019).

2.4.2 Octree

A *octree* é a versão tridimensional da *quadtree*, por isso as mesmas propriedades da *quadtree*se aplicam a ela (Figura 12). A diferença é que a entrada será divida sempre em 8 partes com regiões de mesmo tamanho. Os cortes serão realizados nos eixos X, Y e Z.

Esta estrutura é bastante utilizada na subdivisão espacial de modelos tridimensionais. Uma das principais aplicações práticas desta estrutura é em renderização de jogos. A estratégia de dividir o espaço reduz a quantidade de testes de colisões, reduzindo assim o tempo de execução.

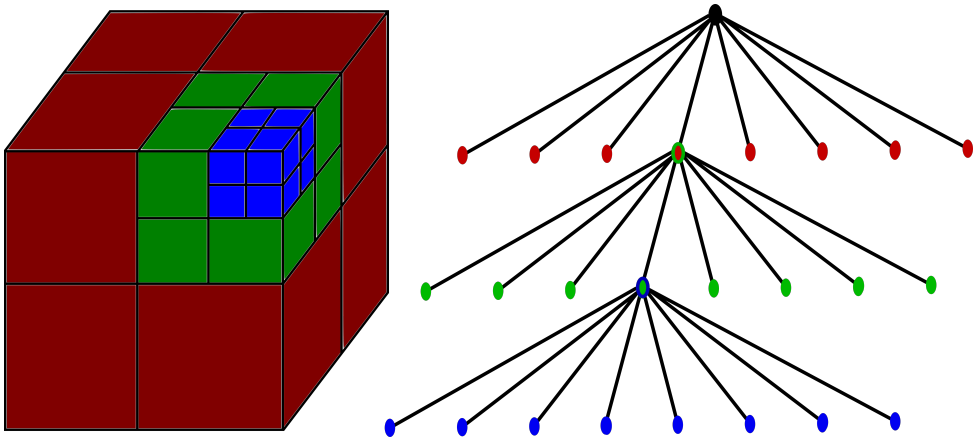


Figura 12 – Uma subdivisão feita por *octree* com três níveis e sua representação em árvore.
Fonte: elaborado pelo autor (2019).

2.4.3 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.

Uma BSP cuja subdivisão acontece sempre alinha aos eixos pode ser vista como 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*.

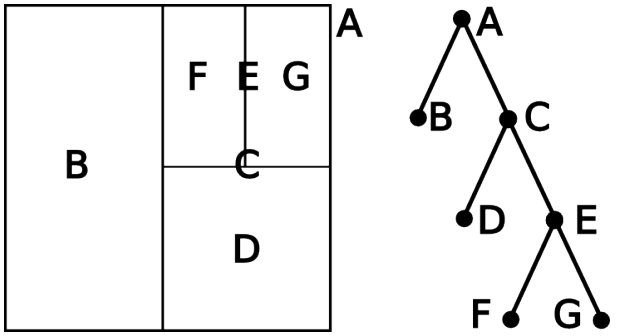


Figura 13 – Uma subdivisão feita com BSP e sua representação em árvore.
Fonte: elaborado pelo autor (2019).

2.5 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 múltiplos processadores 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 dessa 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, dentre outros.

2.5.1 Classes de Arquiteturas Existentes

Uma aplicação paralela pode ser composta por uma ou mais tarefas. As tarefas que compõem uma aplicação paralela podem ser executadas em vários processadores, caracterizando, dessa 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.

Pode-se 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 14 mostra um exemplo de uma SMPs.
- MPPs (*Massively parallel processing* ou processadores massivamente paralelos)
  São compostos por vários nós (processador e memória) independentes, interconectados por redes dedicadas e de alta velocidade. Ou seja, existe apenas uma máquina com centenas

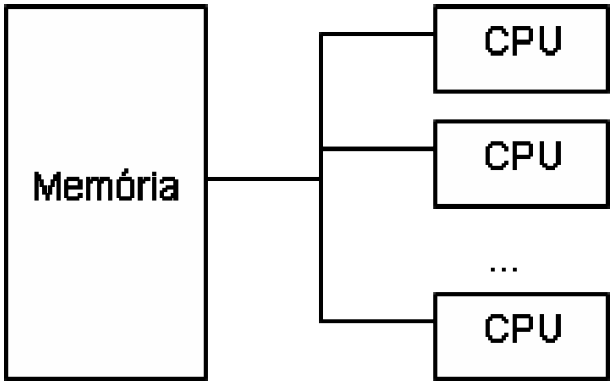


Figura 14 – Arquitetura de um Multiprocessador Simétrico (SMP).
Fonte: elaborado pelo autor (2019).

de CPU's interconectadas. A Figura 15 mostra um exemplo de uma MPPs.

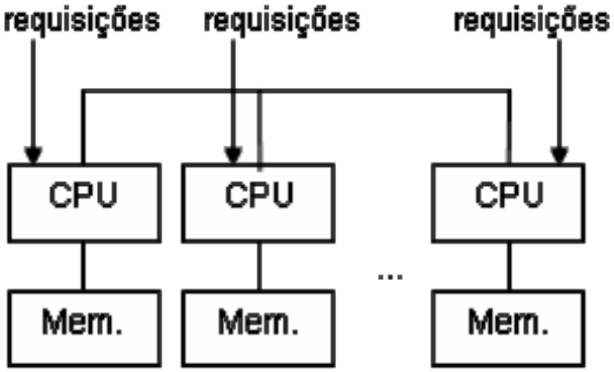


Figura 15 – Arquitetura de um MPP.
Fonte: elaborado pelo autor (2019).

- 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 é que os nós que compõem uma MPP tipicamente são conectados por redes desenvolvidas especificamente para o MPP, enquanto uma NOW é composto por equipamentos de rede e processadores comuns. A Figura 16 mostra um exemplo de uma NOWs.
- 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 17 mostra um exemplo de uma grade computacional.
- *Clusters*

  Diversos processadores que estão fisicamente organizados em uma mesma máquina (ou,

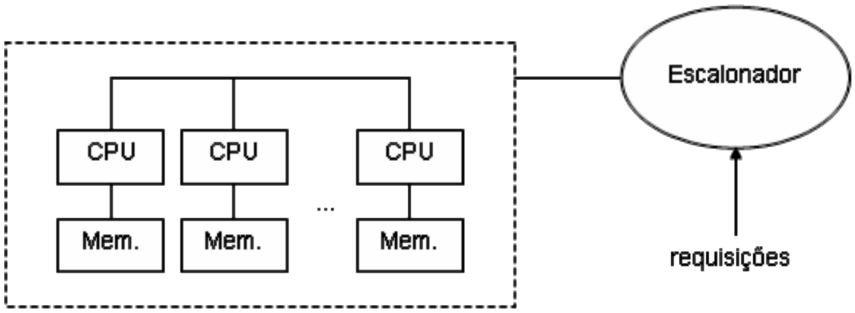


Figura 16 – Arquitetura de uma NOW ou aglomerado de computadores.
Fonte: elaborado pelo autor (2019).

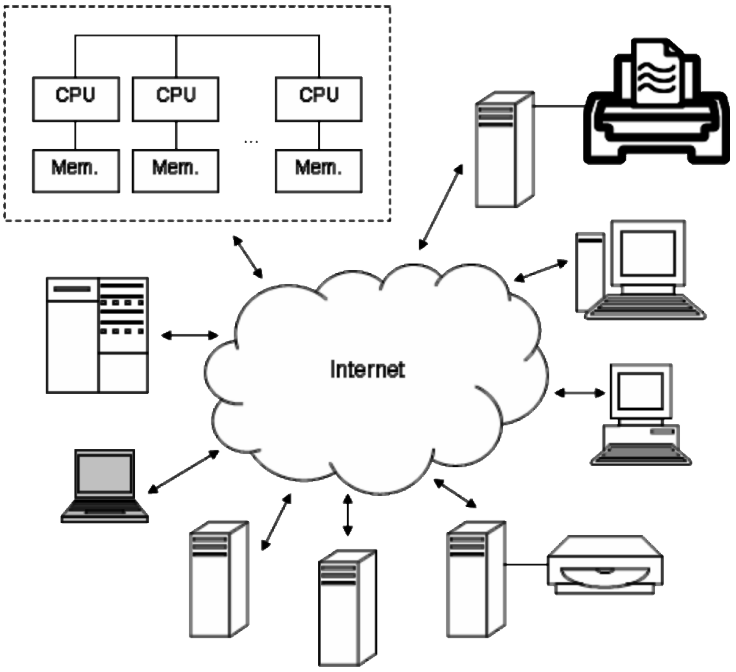


Figura 17 – Arquitetura de uma grade computacional.

pelo menos, em uma mesma sala) e conectados através de uma rede comum ou de alta velocidade (*ethernet* ou *infinband* são as mais difundidas atualmente). Ou seja, cada máquina é independente das outras em termos de memória, disco e processamento. Essas máquinas são interconectadas por uma rede de alta velocidade. Sendo assim, trabalhar com *clusters* torna a distribuição de carga mais evidente para o programador. A Figura 18 mostra um exemplo de *cluster*.

2.5.2 Modelos de Paradigmas de Programação Paralela

Em programas sequenciais, não existe a preocupação que uma posição de memória seja alterada ao mesmo tempo que ela esteja sendo lida. Em computação paralela, há essa preocupação em manter o estado da memória, existindo várias técnicas para manter a ordem de leitura e escrita na memória. Basicamente existem três tipos de paradigmas de programação

Figura 18 – *Cluster* Solaris da Sun Microsystems.
Fonte: *https://en.wikipedia.org/wiki/Solaris_Cluster*.

paralela:

- Memória Compartilhada

  Engloba basicamente os sistemas UMA (*Uniform Memory Acess*), 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 esquerda da Figura 19).

- 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 direita da Figura 19). A comunicação entre os processos ocorre através de troca de mensagens.

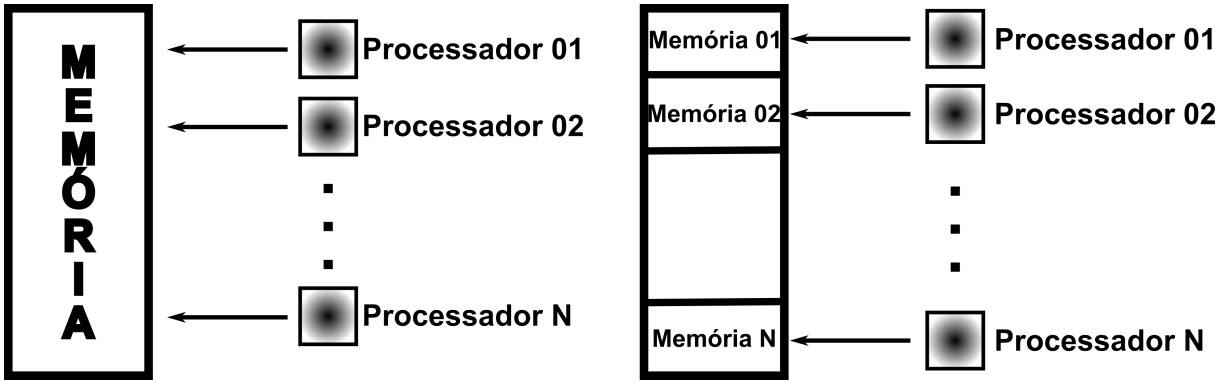


Figura 19 – Arquiteturas paralelas: memória compartilhada à esquerda e memória distribuída à direita.
Fonte: elaborado pelo autor (2019).

- Memória Mista ou Híbrida

  Ocorre, por exemplo, 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 20 ilustra esse tipo de paradigma.

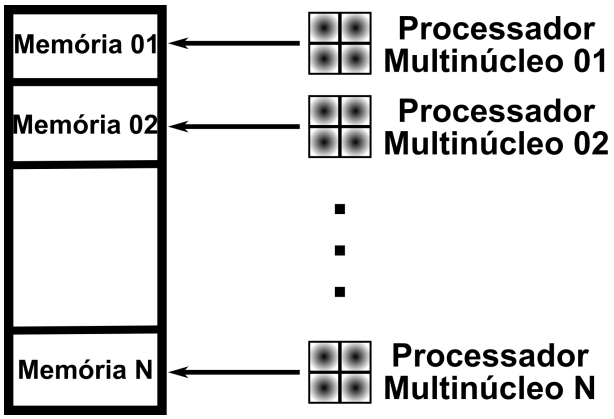


Figura 20 – Arquiteturas paralela mista ou híbrida.
Fonte: elaborado pelo autor (2019).

Um modelo teórico não restringe a arquitetura de uma máquina. Por exemplo, um programa escrito para o modelo de memória compartilhada pode ser executado em uma máquina de arquitetura distribuída, através de uma emulação via *software*. Essa emulação, entretanto, pode acarretar um *overhead* no tempo de execução. O contrário também pode acontecer, ou seja, um programa escrito para o modelo de memória distribuída pode ser executado em uma máquina de arquitetura compartilhada.

2.5.3 Desempenho e Escalabilidade

Dois dos principais objetivos no desenvolvimento de aplicações paralelas são 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 fatores 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 um pacote de dados transitar entre dois nós da rede e a quantidade de 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 um 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.

2.5.4 Balanceamento de Carga

Uma aplicação que executa em paralelo cria várias novas tarefas que devem ser distribuídas entre os processadores existentes. Quando a quantidade de tarefas se torna maior que a quantidade de processadores disponíveis, torna-se necessário um balanceamento de carga, ou seja, distribuir o processamento entre os processadores de modo a obter a maior velocidade possível de execução. O principal objetivo é manter os processadores ocupados o máximo de tempo possível, evitando que alguns deles fiquem ociosos enquanto outros estão ocupados.

Esse balanceamento pode ser estático (o balanceamento ocorre antes da execução de qualquer processo) ou dinâmico (é realizado durante a execução do processo). O principal problema de um balanceamento estático é a dificuldade em estimar com precisão os tempos de execução das várias partes do programa. O modelo de balanceamento dinâmico possui duas classificações:

1. Centralizado (Figura 21)
   - Todas as tarefas são manipuladas a partir de uma localização central.
   - Exemplo: Modelo mestre-escravo - o processo mestre mantém a coleção de tarefas a serem executadas e os processos escravos solicitam essas tarefas.
2. Não-centralizado (Figura 22)
   - Uma coleção de processos trabalhadores opera sobre um problema, e esses trabalhadores interagem entre si, reportando o resultado final a um único processo.
   - Exemplo: Algoritmo de *polling* aleatório - o processo P_i pede tarefas ao processo P_x,

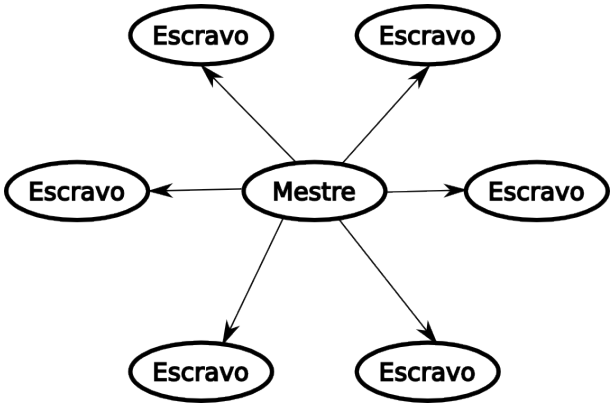


Figura 21 – Estratégia de balanceamento centralizado mestre/escravo onde um processo controla todas as tarefas e os escravos solicitam e executam as mesmas.

Fonte: elaborado pelo autor (2019).

onde x é um número selecionado aleatoriamente entre 0 e $n-1$ (excluindo i).

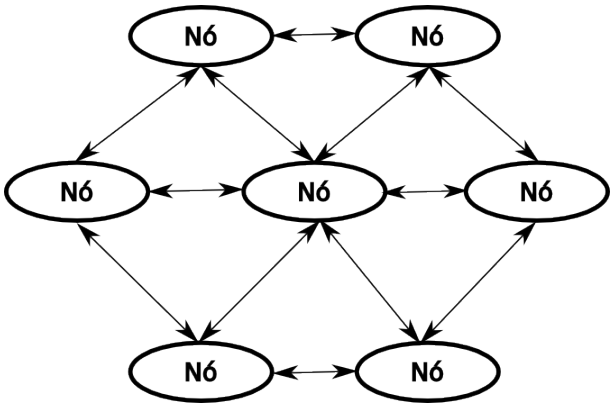


Figura 22 – Estratégia de balanceamento não-centralizado onde os nós podem se comunicar entre si.

Fonte: elaborado pelo autor (2019).

2.5.5 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. Assim, afirma-se que um sistema é escalável quando ele resolve um problema de magnitude γ 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 o número de processadores é multiplicado por x. Já um *speed-up* super linear seria obter um ganho maior que x quando o número de processadores é multiplicado por x.

  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 geralmente aumenta e isso faz o tempo de execução cair, degradando 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%.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos mais relevantes relacionados com a decomposição de domínios e decomposição de malhas. Os trabalhos foram divididos em duas classes, os que possuem decomposição baseadas na geometria das entradas e os que possuem decomposição baseada em algum estrutura de dados. Ao final, tecem-se algumas considerações finais.

3.1 Decomposição Baseada em Geometria

O trabalho de Vidwans *et al.* (1994) apresenta uma técnica de decomposição contínua *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 no centroide, utilizando os eixos para orientação do plano de corte, podendo criar os cortes sempre em um dos eixos cartesianos ou então alinhando o plano em relação a mais de um eixo. São criados, então, conjuntos de vértices, arestas e faces, para cada subdomínio criado. Cada subdomínio é atribuído a um processador. O balanceamento ocorre com a troca de elementos desses conjuntos de um processador para seus vizinhos. A Figura 23 mostra um exemplo de balanceamento de carga feito pelo método.

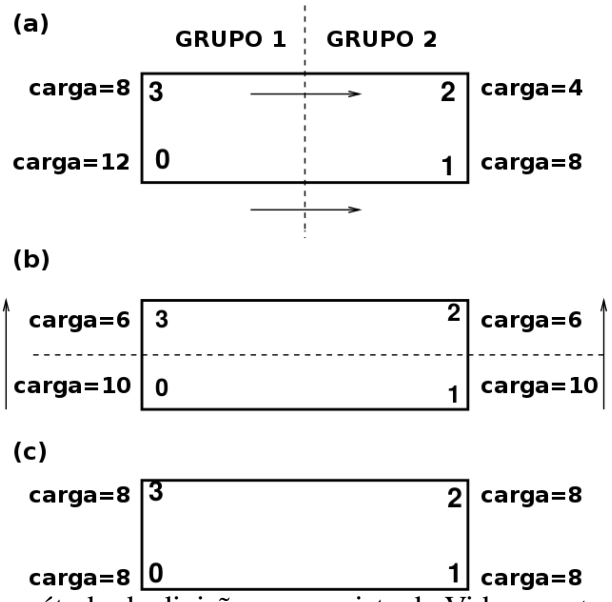


Figura 23 – Exemplo do método de divisão e conquista de Vidwans *et al.* (1994) para balancear a carga entre quatro processadores. (a) Distribuição de carga inicial. (b) Distribuição de carga após o passo 1. (c) Distribuição de carga após o passo 2.

O trabalho de GAITHER (1996) traz uma técnica de geração de malha bidimensional por inserção de pontos com decomposição discreta. Ela é baseada na área dos sub-domínios e na

área dos triângulos que estão sendo gerados, não possuindo, assim, nenhuma estimativa de carga. 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 fronteiras das regiões criadas são discretizadas e um algoritmo de Delaunay bidimensional é aplicado. A Figura 24 mostra o passo a passo da técnica.

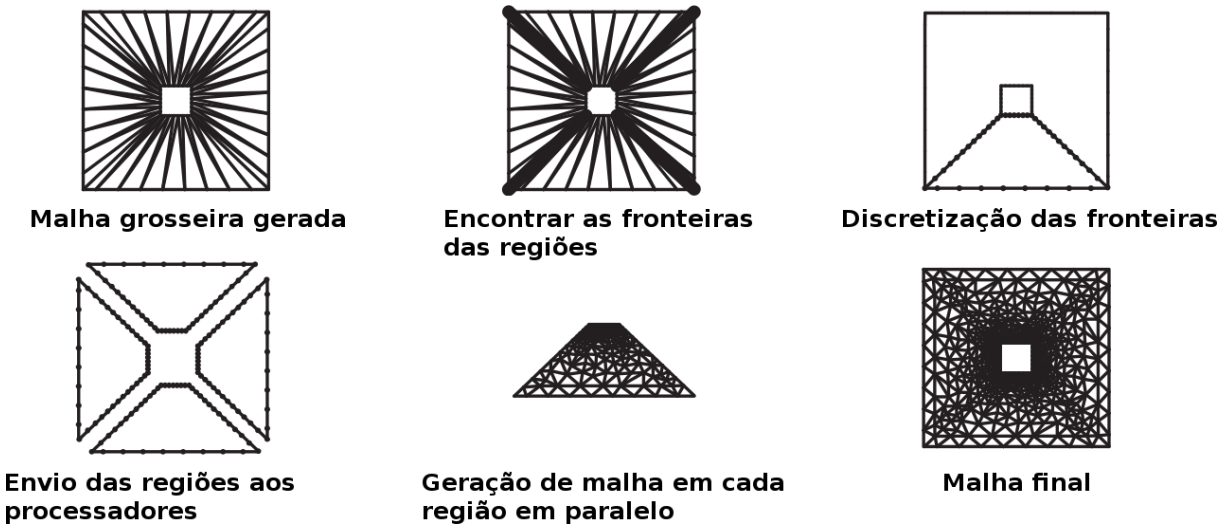


Figura 24 – Passo a passo da técnica de GAITHER (1996).

Em Wu e Houstis (1996), uma técnica de decomposição discreta é descrita. O particionamento é feito numa malha inicial grosseira. São identificados os sub-domínios nesta malha grosseira com base na carga (área da região), no tamanho da interface e na conectividade 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 25 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.

Em Galtier e George (1996), é permitido utilizar duas abordagens contínuas *a priori* para decomposição. A primeira, é pela decomposição em um mesmo eixo pela distância entre dos planos de corte. A segunda, é pela decomposição recursiva, onde os planos de cortes passam pelo eixo do 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), é apresentada uma técnica de decomposição discreta que utiliza uma grade volumétrica para auxiliar na decomposição. A estimativa de carga é realizada pela quantidade de faces presentes numa região. Tanto a malha quanto a grade volumétrica são

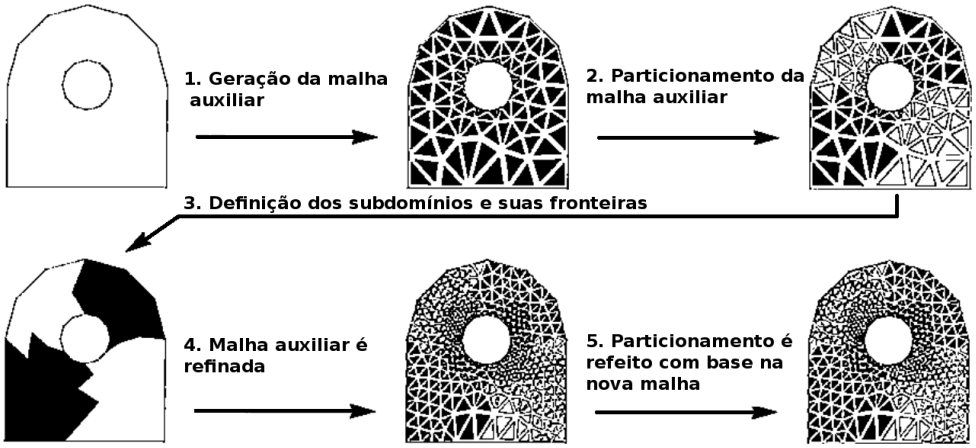


Figura 25 – O passo a passo da técnica de Wu e Houstis (1996).

geradas por Delaunay. As subdivisões são feitas considerando também o volume das regiões. A Figura 26 mostra um exemplo da formação dos subdomínios.

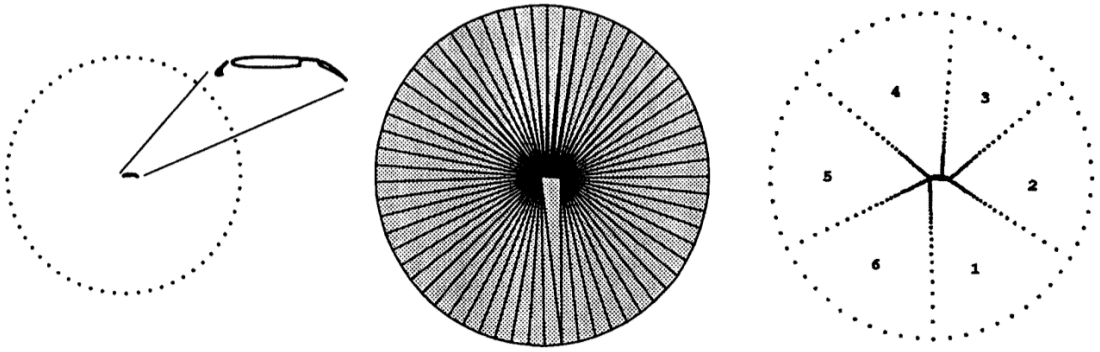


Figura 26 – Exemplo da formação dos subdomínios em SAID (1999). Fronteira de entrada, grade auxiliar inicial e seis subdomínios gerados juntamente com as suas discretizações.

Em Ivanov *et al.* (2006), foi desenvolvido um algoritmo 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;
- Plano passa pelo centro de massa.

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 27, as três formas de decomposição são apresentadas.

Após ter o plano de corte definido, é feita uma suavização da seção de corte e a sua triangulação 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.

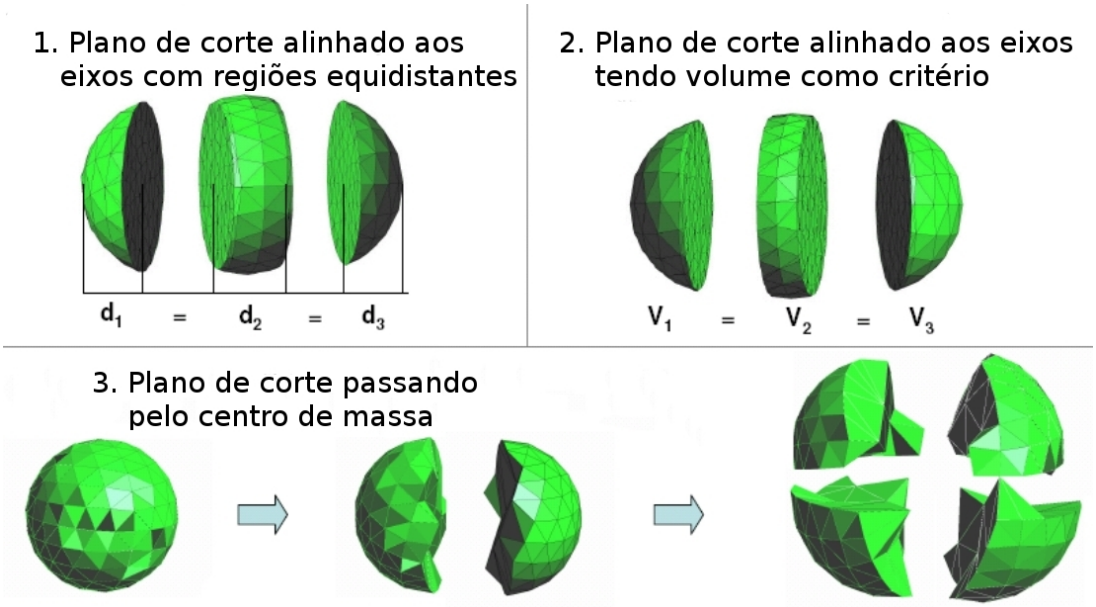


Figura 27 – As três formas de subdividir em Ivanov *et al.* (2006). 1 - Planos equidistantes. 2 - Volume dos subdomínios iguais. 3 - Centro de massa.

Uma solução parecida é a apresentada em Lämmer e Burghardt (2000), em que o plano de corte é traçado da mesma maneira, porém em duas dimensões, ou seja, um eixo de corte. Este eixo é usado para dividir o domínio recursivamente. A partir do eixo, uma aresta é formada, e os valores nos seus pontos extremos são interpolados dos valores dados como entrada. Quando o número de subdomínios for igual ao número de processadores, uma malha de Delaunay é gerada em cada interior concorrentemente. Essa técnica não apresenta uma boa escalabilidade nos seus resultados.

O trabalho de Jurczyk *et al.* (2007) apresenta uma técnica de decomposição *a priori* onde o plano de corte é criado segundo um série de requisitos. Os requisitos são: o volume das regiões geradas devem ser aproximadamente o mesmo, o plano de corte deve ser mínimo, ou 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 28 mostra o processo de decomposição.

Em ANDRÄ (2008), são utilizados o centro de massa e o momento de inércia para se encontrar os planos de corte do domínio. As interfaces são geradas *a priori* por Delaunay

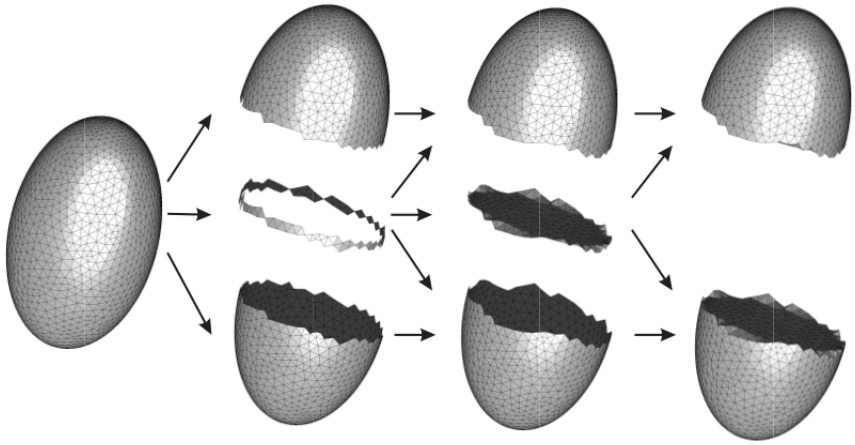


Figura 28 – Exemplo do processo de decomposição em Jurczyk *et al.* (2007).

bidimensional e convertidas depois para tridimensional. A Figura 29 mostra um exemplo de particionamento feito pela técnica.

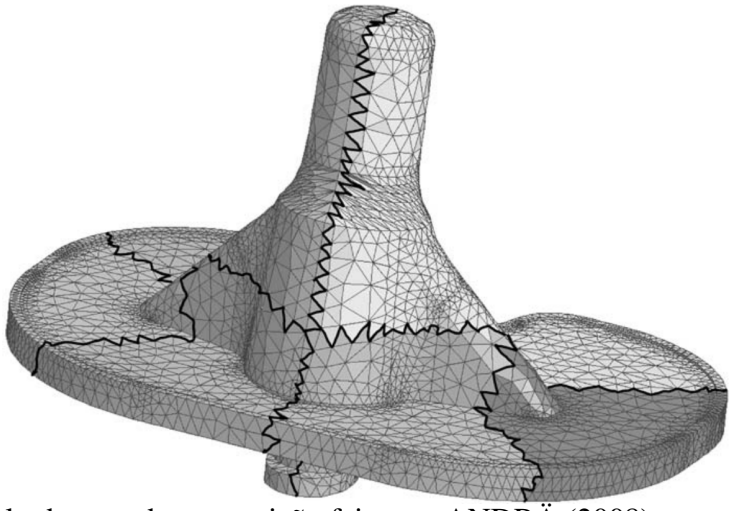


Figura 29 – Exemplo de uma decomposição feita por ANDRÄ (2008) para oito subdomínios.

Em Pirzadeh e Zagaris (2009), é descrita uma técnica baseada em avanço de fronteiras e avanço de camadas com decomposição contínua *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, são criados planos de partição que dividem o domínio em regiões com cargas aproximadamente iguais. As posições destes planos são definidas através do centro de densidade da malha. Esse centro indica onde a massa efetiva do sistema está concentrada.

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, é realizada uma junção de todas as submalhas. A Figura 30 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.

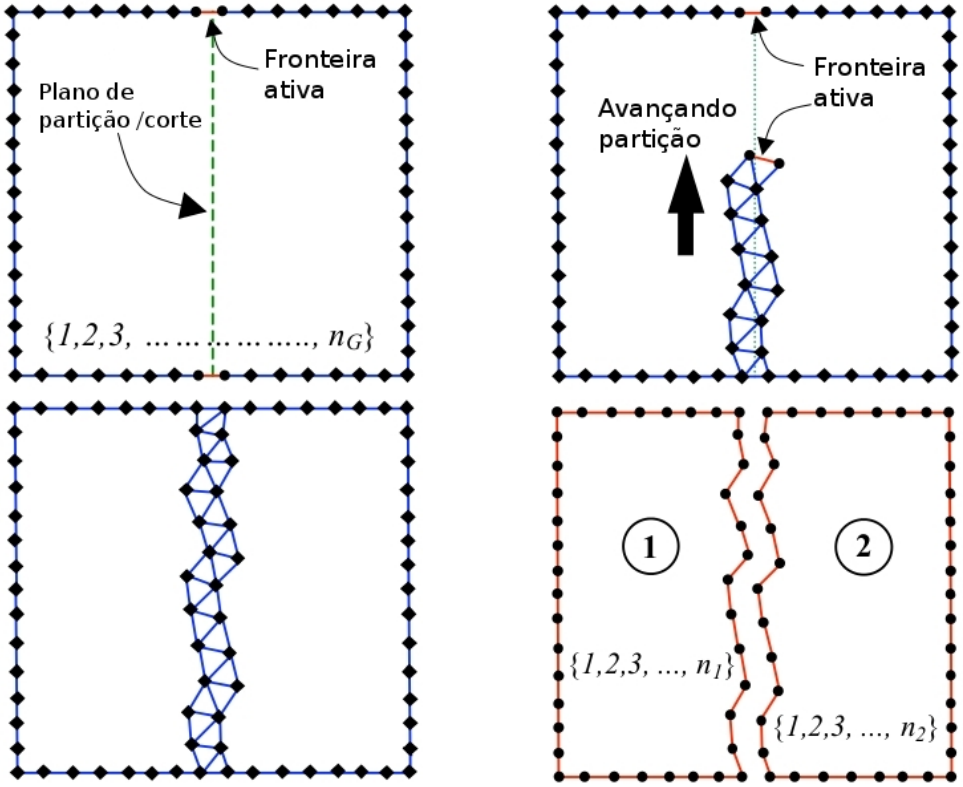


Figura 30 – Os principais passos da técnica de Pirzadeh e Zagaris (2009) para gerar os segmentos de interface.

Como vantagens desta técnica contínua, pode-se citar que a utilização do 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 em um 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 apresentada uma técnica contínua *a priori* em que o plano de partição é posicionado usando o centro gravitacional para posicionar o plano de partição, juntamente como eixo de inércia, como normal do plano. Após isso, são encontradas as arestas que fazem interseção com o plano de corte, eliminando aquelas que tenham vértices com vizinhança menor que dois. Nas arestas restantes é realizada uma suavização (Figura 31). A interface é gerada por Delaunay nas arestas da fronteira encontrada. Apesar do esforço para a criação de uma boa interface, esta técnica não possui uma estimativa de carga, criando mais subdomínios que processadores disponíveis (*over-decomposition*). Em um trabalho parecido, Zheng e Chen (2007) os planos de cortes devem ser os menores possíveis e que gerem subdomínios de tamanhos praticamente iguais. O plano de corte é posicionado no eixo principal de inércia.

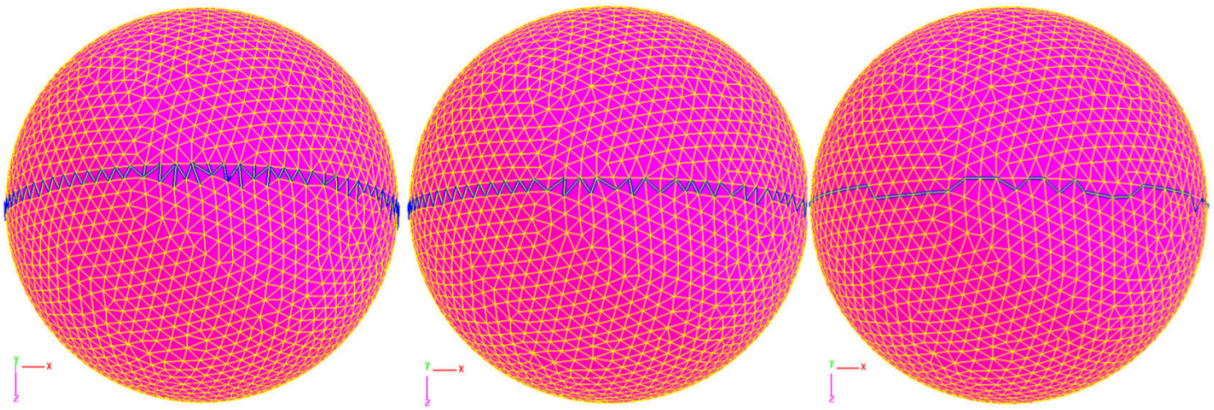


Figura 31 – Da esquerda para direita: todas as arestas que sofrem interseção, a fronteira inicial, e a fronteira suavizada. Em (CHEN, 2012).

Em Zhang *et al.* (2013), é apresentada uma técnica bidimensional para geração de malhas em geometrias complexas, ou seja, que possuem vários vértices, formando ângulos côncavos. Inicialmente, esta técnica gera uma malha de Delaunay grosseira usando apenas os vértices do modelo de entrada. Utilizando as informações dessa malha é aplicada uma estratégia de criação de novas regiões (*Shortest-virtual-edge rule*) usando a distância entre vértices e os ângulos formados por este vértice na malha de Delaunay grosseira. Ao final, após aplicar esta estratégia, será possível criar sub-regiões no modelo inicial, reduzindo assim a complexidade da sua geometria (Figura 32).

O trabalho de Yilmaz e Ozturan (2015) apresenta duas técnicas para decomposição de modelo para geração de malha em paralelo, usando o NETGEN (SCHÖBERL, 1997). A primeira delas é uma técnica *a priori* baseada na geometria do modelo. Ela gera inicialmente um malha grosseira para servir como estimativa de carga. O cálculo da carga é feito usando a

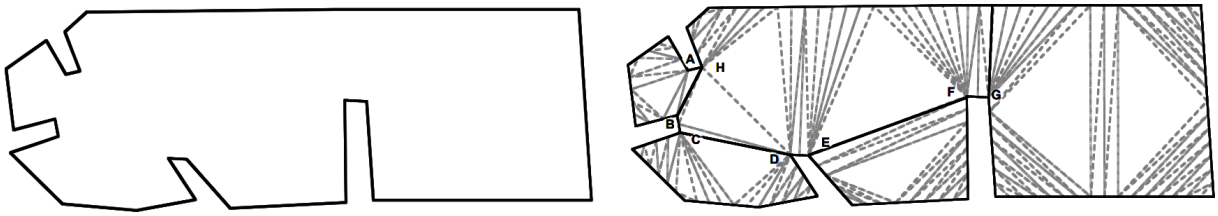


Figura 32 – Exemplo de um modelo de entrada (a esquerda) da técnica de Zhang *et al.* (2013), e regiões criadas após a decomposição (a direita).

quantidade de elementos tetraédricos, e a decomposição é feita utilizando uma BSP paralela aos eixos cartesianos. O processo 0 é o responsável por fazer toda a decomposição e criação das malhas de interfaces e, somente após todos os subdomínios estarem criados, é que eles serão divididos entre os outros processos disponíveis. Cada processo deve sincronizar os id's dos vértices e elementos criados com o restante dos processos.

A segunda técnica de decomposição apresentada gera uma malha grosseira e utiliza o METIS (KARYPIS; KUMAR, 1998) para particionar a malha entre os outros processos disponíveis. Cada processo aplica um refinamento na sua malha até que o tamanho da malha alcance um valor pré-estabelecido e, para finalizar, os identificadores dos vértices e elementos criados são sincronizados com os outros processos. A Figura 33 mostra uma decomposição usando a segunda técnica desse trabalho.

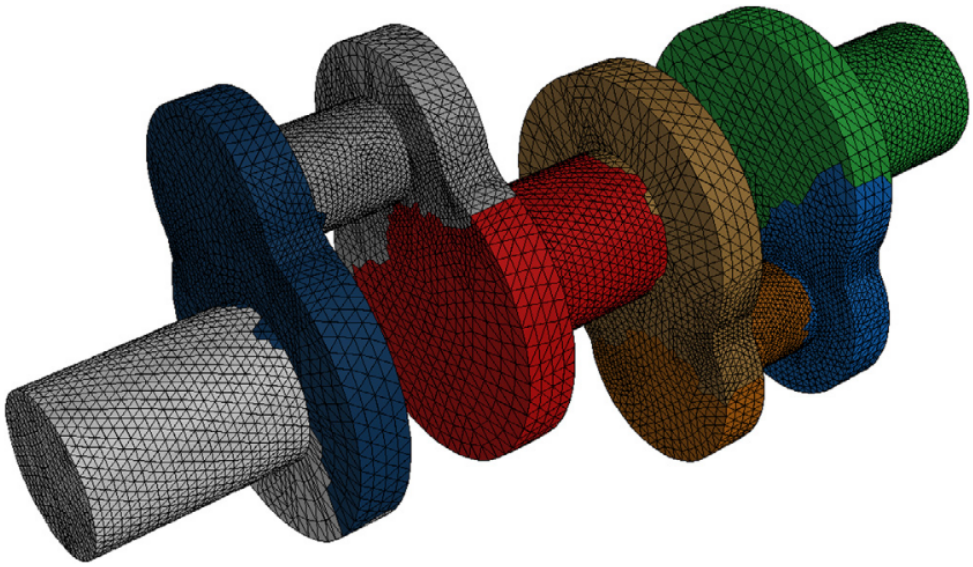


Figura 33 – Malha gerada para o modelo Shaft no trabalho de Yilmaz e Ozturan (2015) utilizando a segunda técnica.

A primeira técnica do trabalho de Yilmaz e Ozturan (2015) é bastante parecida com a técnica proposta nesta tese. As maiores diferenças são: em Yilmaz e Ozturan (2015) é necessário uma grande quantidade de sincronização de id's entre os processos, enquanto neste trabalho não é

necessário essa sincronização; e que em Yilmaz e Ozturan (2015) não existe nenhum tratamento para os planos definidos pela BSP de decomposição, podendo inviabilizar as decomposições a serem feitas. Por conta disso, os modelos apresentados por ele nos resultados são de geometria simplificada (uma esfera, um torus e um cubo).

3.2 Decomposição Baseada em Estruturas de Dados

Um dos primeiros trabalhos em decomposição de domínios foi o de FARHAT (1988), que desenvolveu uma técnica para decomposição de malhas de elementos finitos. Essa técnica subdivide a malha de acordo com a quantidade de processadores disponíveis e utiliza a própria estrutura da malha quadrangular de matriz/grade para realizar a subdivisão e a estimativa de carga. A Figura 34 mostra um exemplo de decomposição de malha.

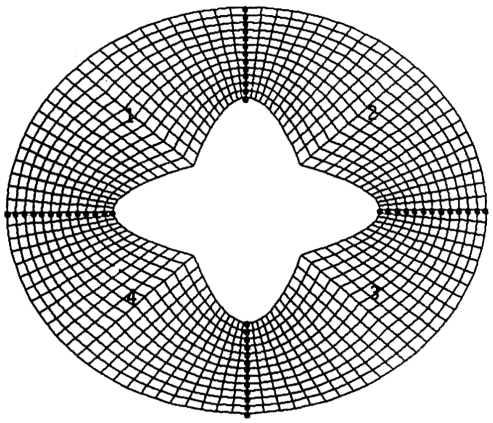


Figura 34 – Decomposição em 4 subdomínios de uma malha multiconectada na técnica de FARHAT (1988).

Alguns trabalhos como o de Barnard e Simon (1994) utilizam grafos para encontrar o melhor plano de corte para decomposição de malhas. O corte é baseado em um grafo criado com as arestas da malha, maximizando a quantidade de vértices nos conjuntos e minimizando a quantidade de arestas cortadas pelo plano de corte. A criação do grafo para a subdivisão é ilustrada na Figura 35. No trabalho de Simon (1991), além desta forma de subdividir, são mostradas também subdivisões feitas pela bisseção e pela subdivisão recursiva da bisseção espectral (autovetores da matriz laplaciana de um grafo).

Em NIKISHKOV (1999), um grafo é construído para fazer a estimativa de carga e para realizar a subdivisão. O critério de subdivisão é a quantidade de elementos internos a cada subdomínio. A Figura 36 mostra um exemplo de subdivisão para esta técnica.

Em deCougny e Shephard (1999), a entrada do algoritmo é o contorno de um objeto.

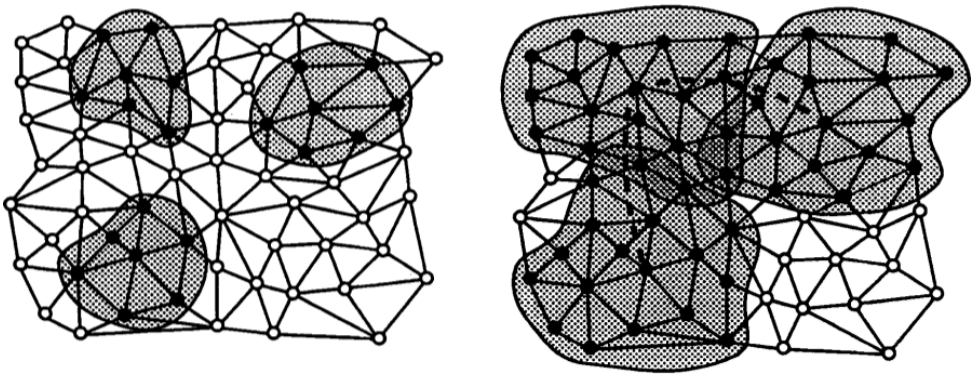


Figura 35 – Criação de três regiões pelo trabalho de Barnard e Simon (1994).

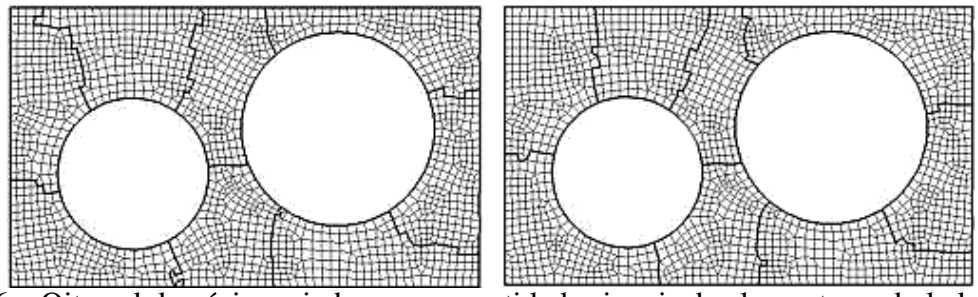
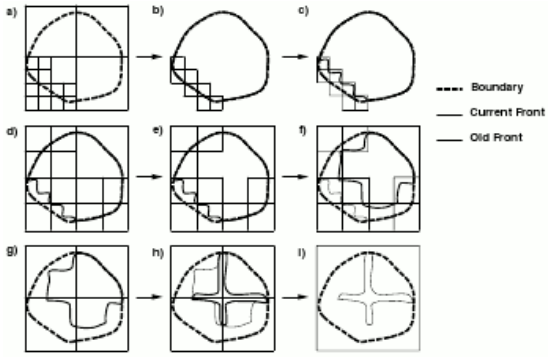


Figura 36 – Oito subdomínios criados com quantidades iguais de elementos e do lado direito a otimização dos subdomínios em NIKISHKOV (1999).

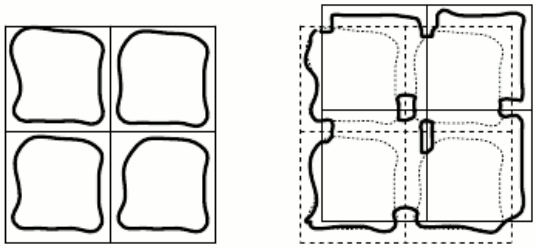
Cada processador fica com parte de uma *octree* distribuída, que define planos de corte do domínio. A malha das células internas é gerada concorrentemente com *templates*. A região entre o contorno e as células internas é preenchida por uma técnica de avanço de fronteira, onde são gerados os elementos 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 interseções. Essa técnica gera muitos subdomínios, 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. 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 37, são mostrados os passos do algoritmo e os deslocamentos que são realizados. Essa técnica é

classificada como contínua *a posteriori*.



(a) *Octree* gerada para uma borda e passos do algoritmo (representação 2D, ou seja, uma *quadtree*).



(b) Deslocamento da *quadtree* para gerar mais malha (a linha escura é a fronteira).

Figura 37 – 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 (o deslocamento dele é na diagonal, na direção dos eixos principais, entre outros).

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 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 feito por *over-decomposition*.

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, o plano de corte sofre um deslocamento em um dos eixos. A Figura 38 mostra um exemplo onde alguns planos de corte falham nos testes.

O trabalho de Charmpis e Papadrakakis (2005) apresenta uma técnica para subdivisão de malhas de elementos finitos. É feita uma representação da entrada como um grafo e, em

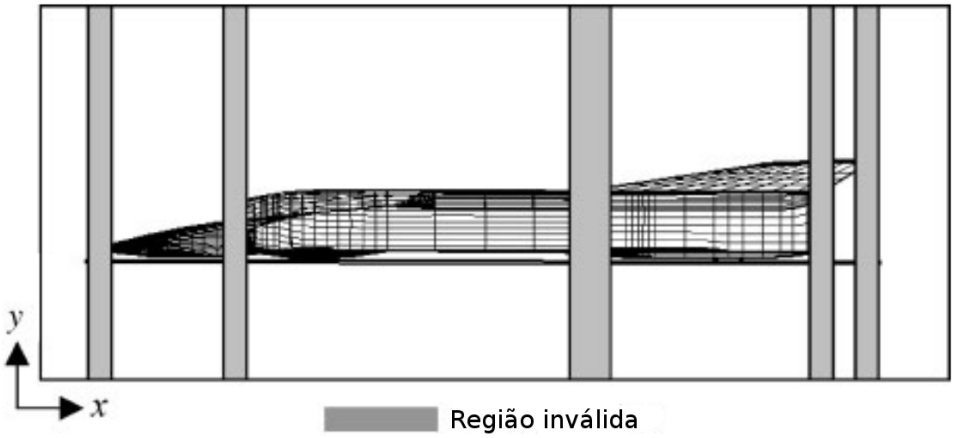


Figura 38 – Regiões de corte inválidas em cinza (LARWOOD *et al.*, 2003).

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 Figura 39 mostra uma malha de elementos finitos e a sua partição como grafo.

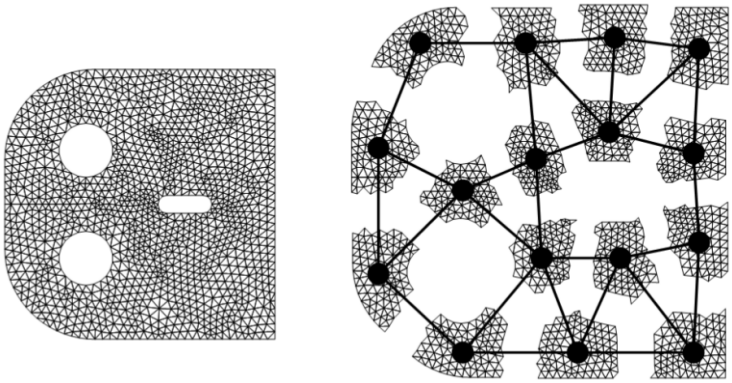


Figura 39 – Malha de elementos finitos (esquerda) e a partição da malha com sua representação em grafo (à direita) em Charmpis e Papadrakakis (2005).

Em LINARDAKIS e 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 40 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 (*over-decomposition*). 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.

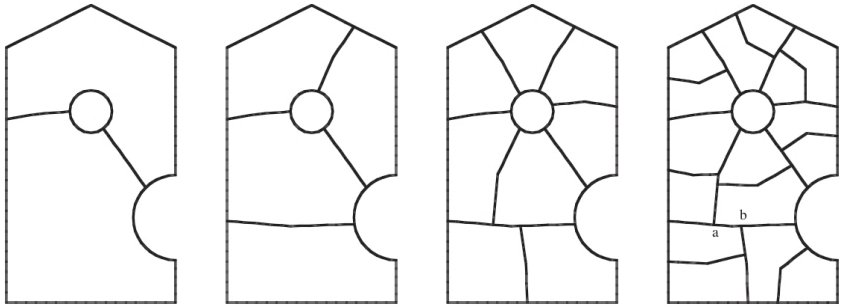


Figura 40 – Construção do grafo de subdivisão de LINARDAKIS e CHRISOCHOIDES (2006).

Em Ito *et al.* (2007), uma malha tetraédrica grosseira é gerada inicialmente com base nas faces da superfície. O particionador de grafos METIS (KARYPIS; KUMAR, 1998) é utilizado para subdividir o domínio, tendo como base a malha grosseira gerada. A Figura 41 mostra para uma esfera o processo de particionamento.

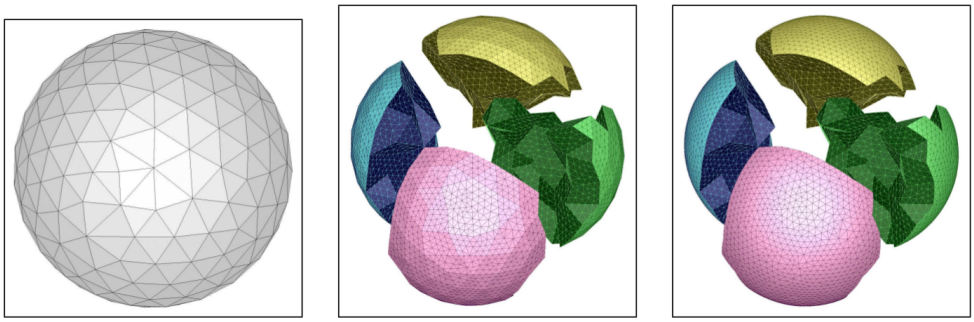


Figura 41 – Malha tetraédrica inicial, malha de superfície refinada nos subdomínios e melhorias nas faces de superfície no trabalho de Ito *et al.* (2007).

Em Glut e Jurczyk (2008), é apresentada uma técnica para malhas tridimensionais com uma abordagem baseada na decomposição geométrica onde a entrada é uma malha de superfície. Uma das motivações deste trabalho é evitar a criação de subdomínios baseados nos eixos de inércia, pois, segundo o autor, os resultados não são bons. 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

projetado no separador usando uma função 2D de controle espacial baseada no tamanho das arestas (Figura 42).

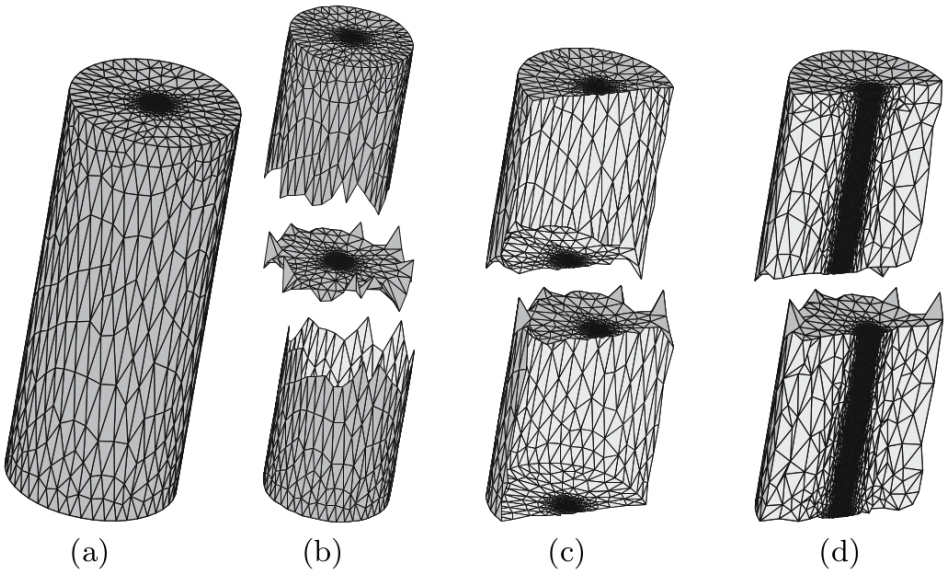


Figura 42 – Passos da técnica baseada na malha de superfície. (a) Malha de superfície; (b) corte; (c) Seção transversal; (d) Malha final em (GLUT; JURCZYK, 2008).

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 43).

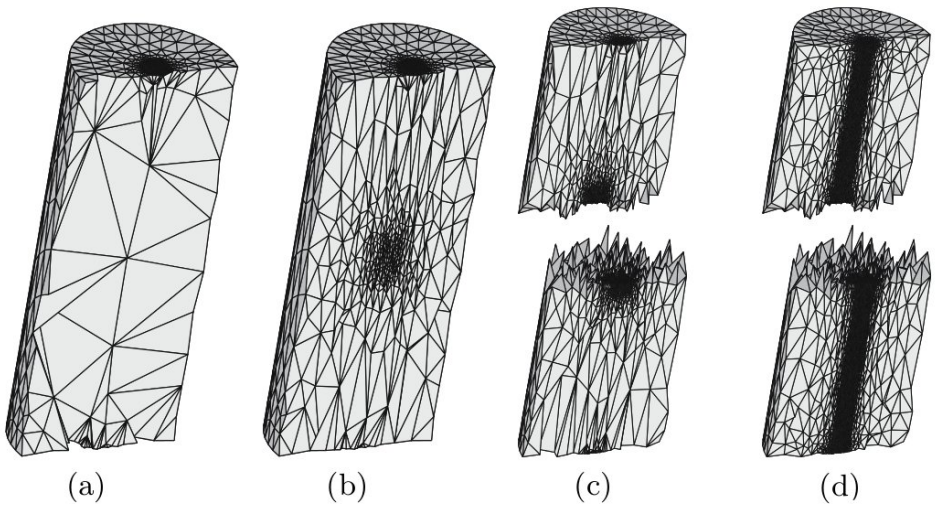


Figura 43 – Passos da técnica baseada na malha volumétrica grosseira. (a) Malha volumétrica grosseira; (b) Refinamento da seção transversal; (c) Seção transversal; (d) Malha final em (GLUT; JURCZYK, 2008).

Essa 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.

Outro trabalho que também utiliza grafos é o de Panitanarak e Shontz (2011), que descreve uma técnica de decomposição discreta que também utiliza o programa de partição de grafos METIS. Primeiramente, é construída uma malha grosseira, com a restrição dos ângulos formados com a fronteira sejam maiores que 30^o. Com a malha devidamente criada, é feita a sua conversão para que seja um grafo, onde cada triângulo é representado como um nó e cada adjacência entre triângulos, como uma aresta. A decomposição é feita neste grafo de acordo com a quantidade de subdomínios desejado. A Figura 44 mostra exemplos de decomposições feitas utilizando esta técnica.

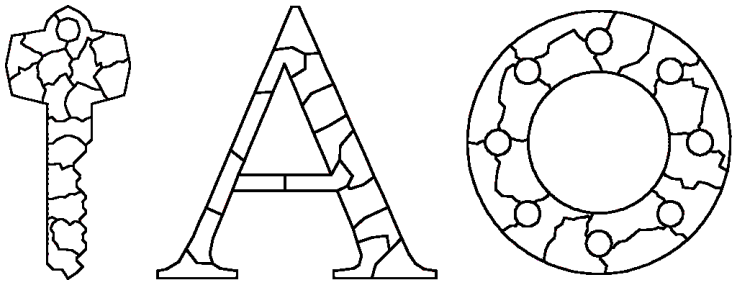


Figura 44 – Decomposição feita pela técnica de Panitanarak e Shontz (2011).

Em Lo (2012b), é apresentada uma técnica bidimensional e, em Lo (2012a) a sua versão tridimensional, para uma triangulação de Delaunay por inserção de pontos. 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 45 mostra a organização de um conjunto de pontos por partição regular e por *kd-tree* e a Figura 46, a sua versão tridimensional da decomposição para 24 zonas.

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 podem ter pontos de uma zona vizinha. Isso pode gerar uma camada a mais de triângulos nas zonas, que é necessária para obter uma

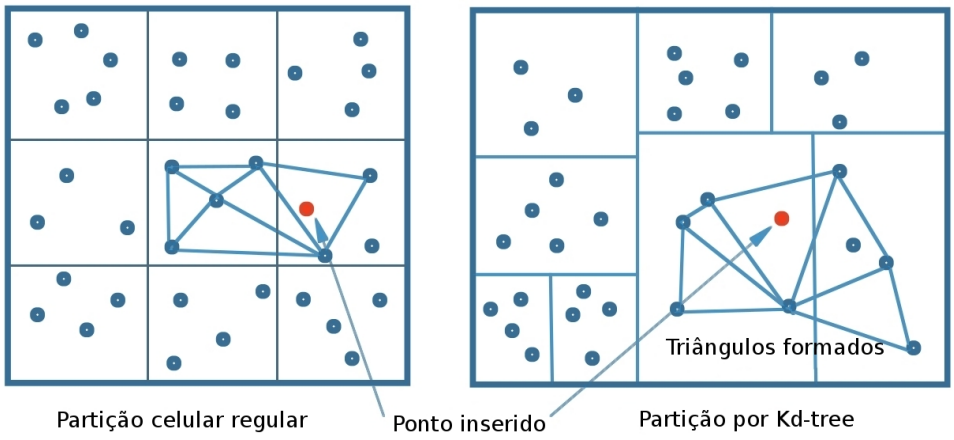


Figura 45 – Pontos organizados em células. À esquerda por partição regular e à direita por *kd-tree* em (LO, 2012b).

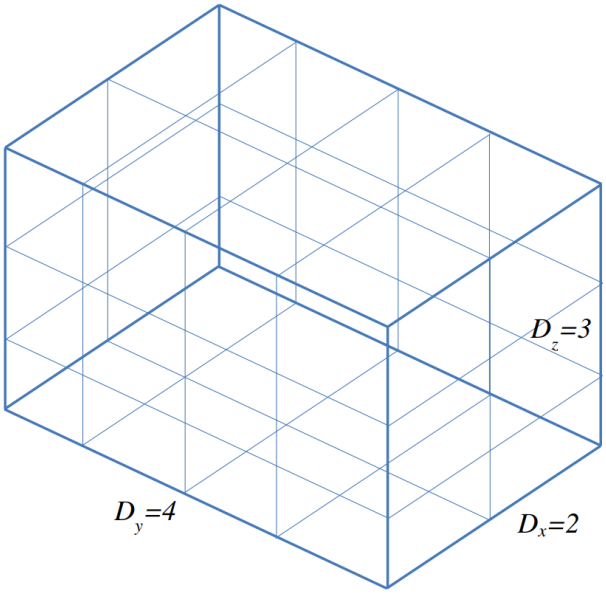


Figura 46 – Decomposição tridimensional feito para 2x4x3 = 24 zonas em (LO, 2012a).

malha sem buracos entre elas. Ao final, é feita a junção de todas as malhas geradas em uma só, eliminando possíveis 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 47 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

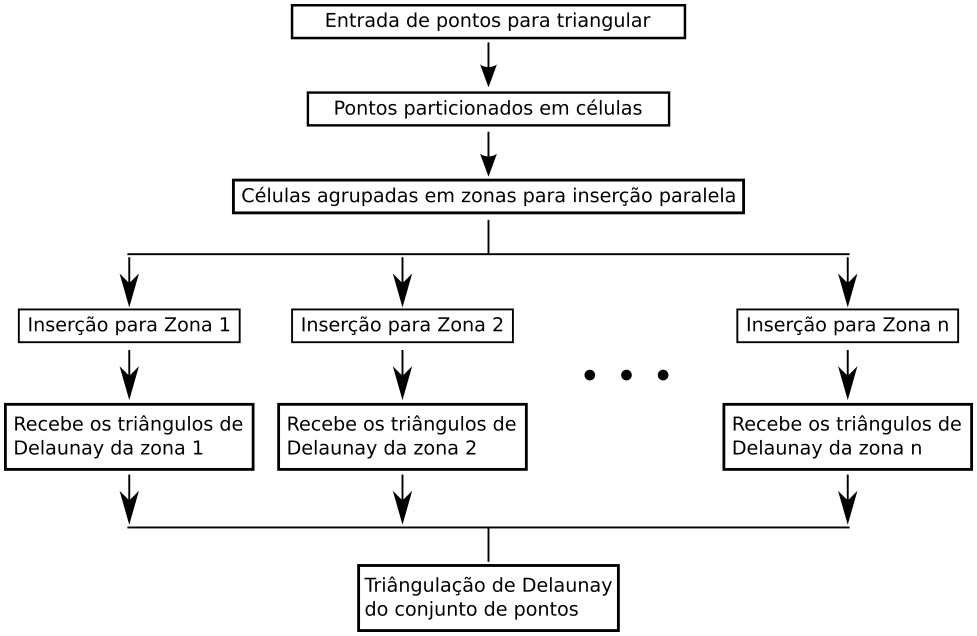


Figura 47 – Fluxograma da triangulação em paralelo em (LO, 2012b).

melhoria nela. Esse trabalho é classificado como contínuo *a posteriori*. A Figura 48 ilustra o passo de deslocamento da *quadtree* no eixo X.

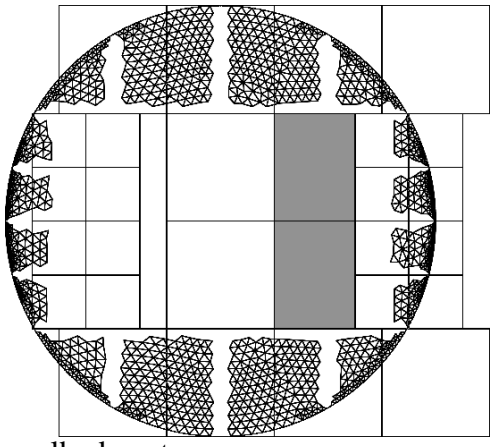


Figura 48 – Malha gerada, espalhada entre os processos escravos, e as células da *quadtree* de decomposição deslocadas para a direção +X em (FREITAS *et al.*, 2013).

O 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 49).

O trabalho de Loseille *et al.* (2015) apresenta uma técnica de geração paralela de malha volumétrica que faz uso de particionamento de malha. A entrada dessa técnica é uma malha volumétrica grosseira, onde a carga é considerada como a quantidade de vértices desta malha. O particionamento pode ser feito usando três critérios (algoritmo de Hilbert, busca em

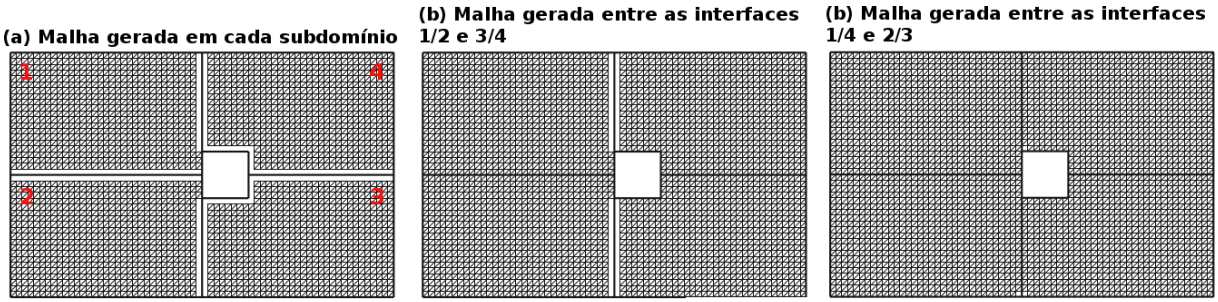


Figura 49 – Junção das malhas dos subdomínios em (LOHNER, 2014).

largura e busca em largura com reinício), todos os critérios se baseiam nas adjacências dos vértices, criando aglomerados de vértices como um subdomínio. Cada subdomínio necessita gerar a sua malha refinada e tratar a comunicação com sua vizinhança (Figura 50).

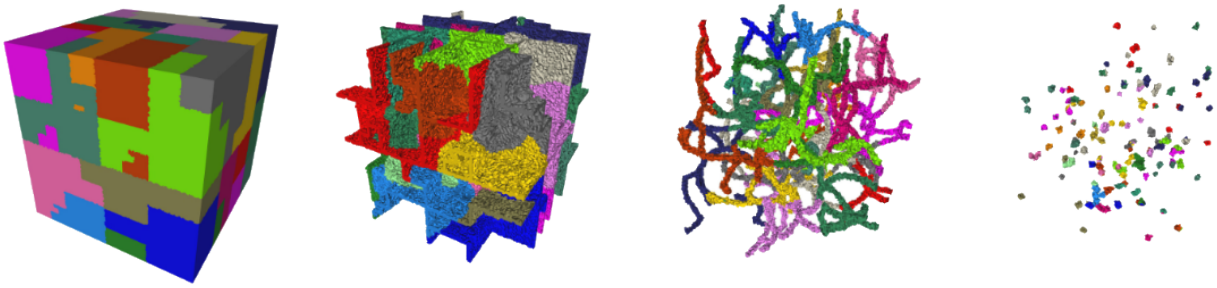


Figura 50 – Particionamento recursivo para 32 subdomínios em um cubo. Da esquerda para direita são mostradas as interfaces de comunicação necessárias em cada nível em (LOSEILLE *et al.*, 2015).

Em Smith *et al.* (2015) e Zhou *et al.* (2010) é apresentada uma técnica de particionamento de malhas chamado de ParMA (*Partitioning using Mesh Adjacencies*), que usa as adjacências da malha de entrada para criar uma grafo, afim de particioná-la (Figura 51). Além do particionamento da malha, também são mostradas técnicas de melhorias das partições, visando um melhor balanceamento de elementos e vértices entre as partições. Essa melhoria é feita migrando um pequeno número de elementos da partição com carga mais pesada para partições vizinhas com menos carga. Posteriormente os mesmos autores criaram o ambiente do PUMI (*Parallel Unstructured Mesh Infrastructure*) em (IBANEZ *et al.*, 2016), para trabalhar com geração de malha e simulações de análise adaptativa.

Em Freitas *et al.* (2016), é apresentada uma evolução da técnica anterior que pode ser tanto bidimensional como tridimensional, por avanço de fronteira, com subdivisão baseada em BSP, que recebe como entrada uma superfície de faces triangulares ou uma lista de arestas, para o caso bidimensional. Para estimar a carga dos subdomínios, é utilizada a *octree* previamente gerada para estimar a carga no domínio de acordo com o tamanho das faces da superfície. As

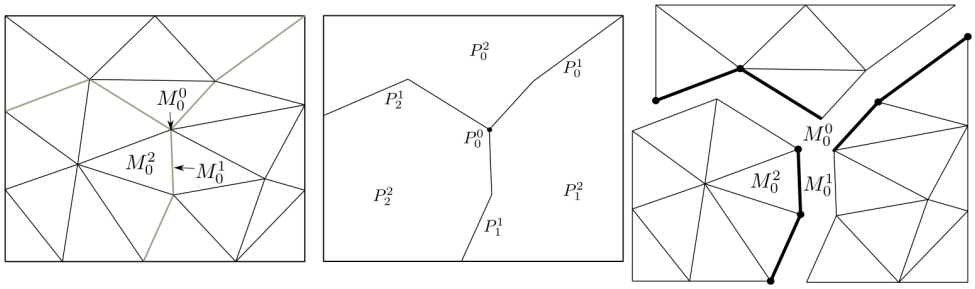


Figura 51 – Exemplo de uma malha (esquerda), seu modelo de partição (centro) e os elementos de cada partição (direita) em (SMITH *et al.*, 2015).

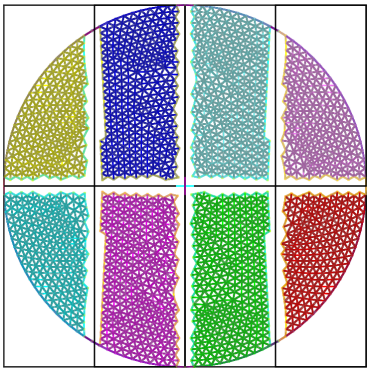
células internas da *octree* têm o tamanho definido de acordo com a maior e a menor face da superfície de entrada. Uma BSP é utilizada para particionar o domínio de tal forma que a quantidade de subdomínios criados seja igual à quantidade de processadores disponíveis.

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 52a. 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 mostram as Figuras 52b, 52c e 52d.

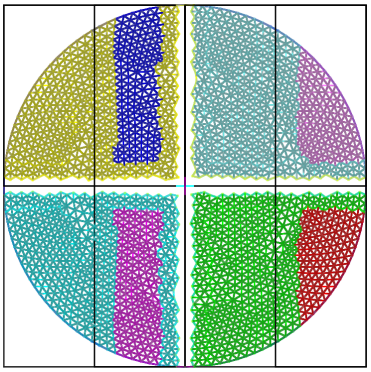
A grande vantagem dessa técnica está na utilização da BSP, que acaba permitindo a geração de subdomínios com cargas mais equilibradas e no ganho de velocidade ao se evitar os deslocamentos que aconteciam anteriormente. Por essa abordagem necessitar de uma sincronização e junção da malha em cada nível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

Em Wang e Jin (2016) é apresentada uma técnica para geração paralela de malhas com bilhões elementos, usando o METIS (KARYPIS; KUMAR, 1998) para o particionamento da malha. A entrada dessa técnica é uma malha grosseira, que será particionada em duas etapas (Figura 53). Na primeira etapa, é feito um particionamento da malha inicial para gerar regiões de acordo com a quantidade de nós computacionais disponíveis, na sequência pode ser usado, em cada partição, uma técnica de multiplicação de malha (KABELIKOVA *et al.*, 2011). Na segunda etapa, é repetido o processo de particionamento da malha, mas desta vez, a quantidade de partições geradas deverá ser igual a quantidade de núcleos de computação de cada nó computacional. Tendo todas as partições devidamente criadas, é aplicada uma técnica de multiplicação de malha até que a quantidade de elementos alcancem o objetivo estabelecido.

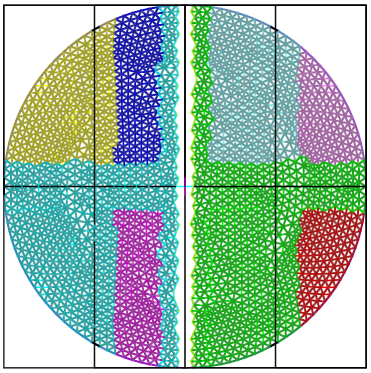
Outro trabalho que usa o particionador de grafos e de malhas ParMETIS (KARYPIS,



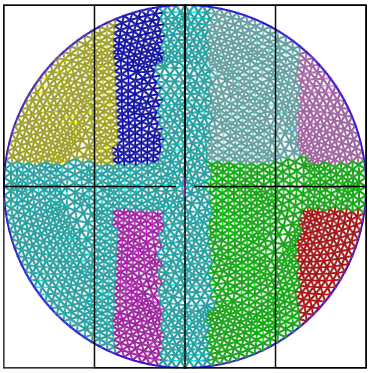
(a) Malha gerada nas folhas da árvore da BSP.



(b) Junção e geração da malha no terceiro nível da árvore da BSP.



(c) Junção e geração da malha no segundo nível da árvore da BSP.



(d) Malha final utilizando particionamento baseado em BSP.

Figura 52 – Passos da geração da malha no trabalho de Freitas *et al.* (2016). Cada cor representa a malha gerada por um processador.

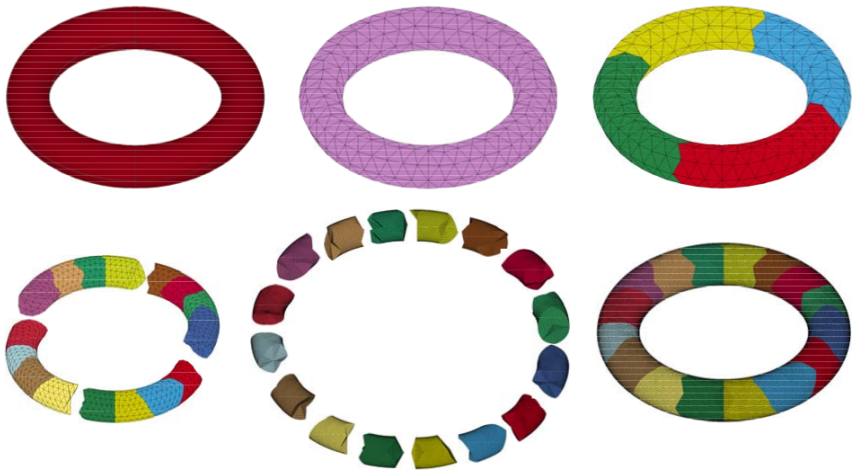


Figura 53 – Passos para a geração de uma malha com bilhões de elementos de Wang e Jin (2016).

2011) para realizar a decomposição do domínio é o Chen *et al.* (2018). Nesse trabalho a superfície de entrada pode ser redimensionada logo no início do processo e, após esse possível de redimensionamento, são criadas uma malha de superfície grosseira e uma malha volumétrica grosseira. Essas malhas são utilizadas como base pelo ParMETIS para criar os subdomínios e repassá-los para outros processadores. Cada processador irá, primeiro, regerar a sua malha de

superfície, para deixá-la mais refinada, para depois gerar a sua malha volumétrica. Ao final, os processos devem verificar as possíveis interseções de faces de sua superfície com as de seus vizinhos. Caso tenham interseções, deverá ser feito uma compatibilização das faces para deixar a malha final válida. A Figura 54 mostra as etapas envolvidas nessa técnica.

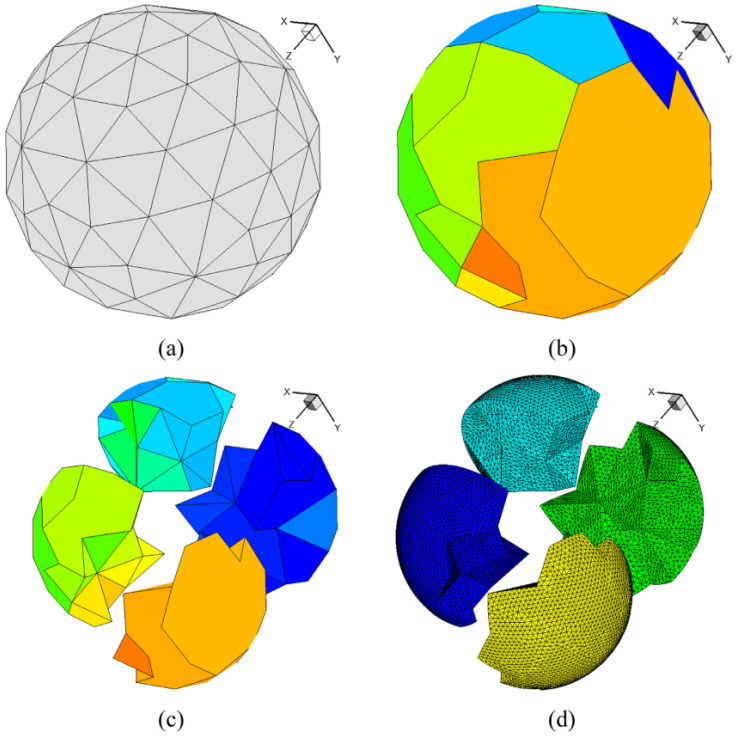


Figura 54 – Um exemplo que ilustra a abordagem de decomposição de Chen *et al.* (2018). (a) A malha de superfície. (b) A malha grosseira. (c) O resultado da decomposição do domínio. (d) Os subdomínios depois de estarem refinados.

3.3 Considerações

As técnicas de decomposição que foram discutidas nessa seção, em geral, apresentam bons resultados na qualidade das malhas, mas a estimativa de carga na maioria deles é inexistente ou é 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 muito maior que a quantidade de processadores disponíveis (*over-decomposition*), com isso, a falta de estimativa de carga é contornada.

Alguns dos trabalhos citados aqui mostram, em seus resultados, que é possível gerar até 512 subdomínios para um determinado modelo de entrada. Porém, os *speed-up* dessas técnicas não apresentam uma escalabilidade que justifique uma decomposição exagerada. Um

exemplo disso acontece no trabalho de Chen *et al.* (2018), que em um certo modelo, que gera 1 bilhão de elementos, reduz o tempo de geração de malha de 30 minutos para 4 minutos, utilizando 512 núcleos computacionais. A redução de tempo nesse exemplo foi de $7,5$ vezes, então, caso fosse utilizado uma técnica de decomposição escalável, bastaria criar 8 subdomínios e distribuí-los entre 8 núcleos computacionais para se obter o mesmo ganho.

Um ponto limitante, que ocorre em algumas técnicas, é que a superfície de entrada pode sofrer modificações ou refinamentos, caso comum em técnicas feitas para a geração de bilhões de elementos. Isso pode ser um problema para modelos que possuem restrições em suas fronteiras, não podendo assim aplicar essas técnicas.

A forma com que os cortes são posicionados nessas técnicas dependem bastante do formato do modelo 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.

Outro fator importante que deve ser considerado é o tamanho (quantidade de elementos) da malha gerada em paralelo. Algumas técnicas podem alterar drasticamente a quantidade de elementos de uma malha, fazendo que a malha gerada sequencialmente seja bastante diferente da gerada em paralelo.

No próximo capítulo, será descrita a técnica proposta nesse trabalho. Essa técnica é uma evolução do trabalho de Freitas *et al.* (2016), incorporando elementos presentes nos trabalhos de Chen *et al.* (2018), Glut e Jurczyk (2008), Larwood *et al.* (2003), e Lämmer e Burghardt (2000), e, ainda, trazendo novas contribuições e soluções a certos problemas enfrentados.

4 TÉCNICA PROPOSTA

A técnica proposta neste trabalho funciona para modelos bidimensionais e tridimensionais. A fronteira de entrada (FE) desta técnica (uma malha de superfície para o caso 3D ou uma lista de segmentos para o caso 2D) define o limite de um ou mais modelos. Esses modelos podem conter buracos, o que é uma vantagem sobre alguns trabalhos da literatura que não tratam esses casos ou que não lidam de maneira natural ou adequada. Além disso, ter todos os elementos da FE como entrada não é uma limitação, uma vez que em muitos problemas as FE's podem ser resultados de digitalizações feitas por *scanners* e esses dados não podem ser alterados e/ou exigem compatibilidade entre regiões vizinhas (Figura 55).

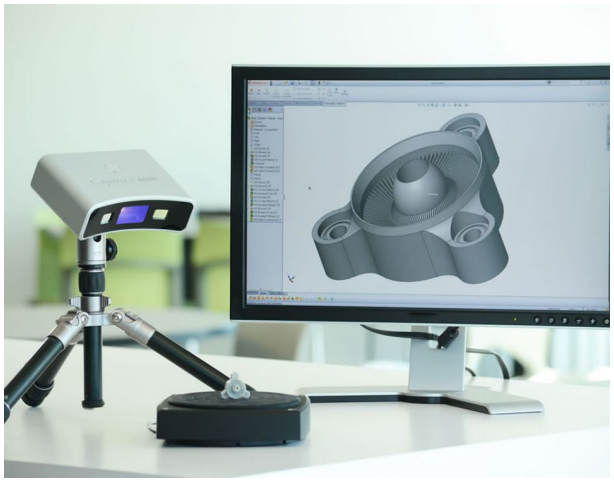


Figura 55 – *Scanner* 3D usado para engenharia reversa da Versus Design, o resultado desta digitalização será uma malha de superfície.
Fonte: www.versusdesign.com.br

Esta técnica proposta é baseada em Freitas *et al.* (2014), que é uma técnica paralela *a posteriori* para a geração de malha descrita em Cavalcante-Neto *et al.* (2001). A principal diferença entre os trabalhos é que este é uma técnica paralela *a priori*, sendo necessário a geração de malhas de interface, que possibilita a independência dos subdomínios, bem como o melhor posicionamento do plano de decomposição (para possibilitar a geração da malha de interface de qualidade) e a melhoria/finalização das malhas de interface após a geração das malhas internas.

A técnica deste trabalho pode ser dividida em cinco etapas (a Figura 56 mostra as etapas para a geração de malha em paralelo). A primeira etapa, apresentada na Seção 4.1, é receber o modelo de entrada e, em seguida, é gerada uma estrutura de dados do tipo árvore (*quadtree* em 2d e *octree* em 3d) que será refinada de acordo com os tamanhos dos elementos da FE. A criação dessa árvore é feita levando-se em consideração que o tamanho de uma célula

interna não pode ser maior que a maior célula que esteja fazendo interseção com a fronteira do modelo. Além disso, um balanceamento de 2:1 é aplicado, garantindo uma transição suave entre regiões refinadas e grosseiras do modelo.

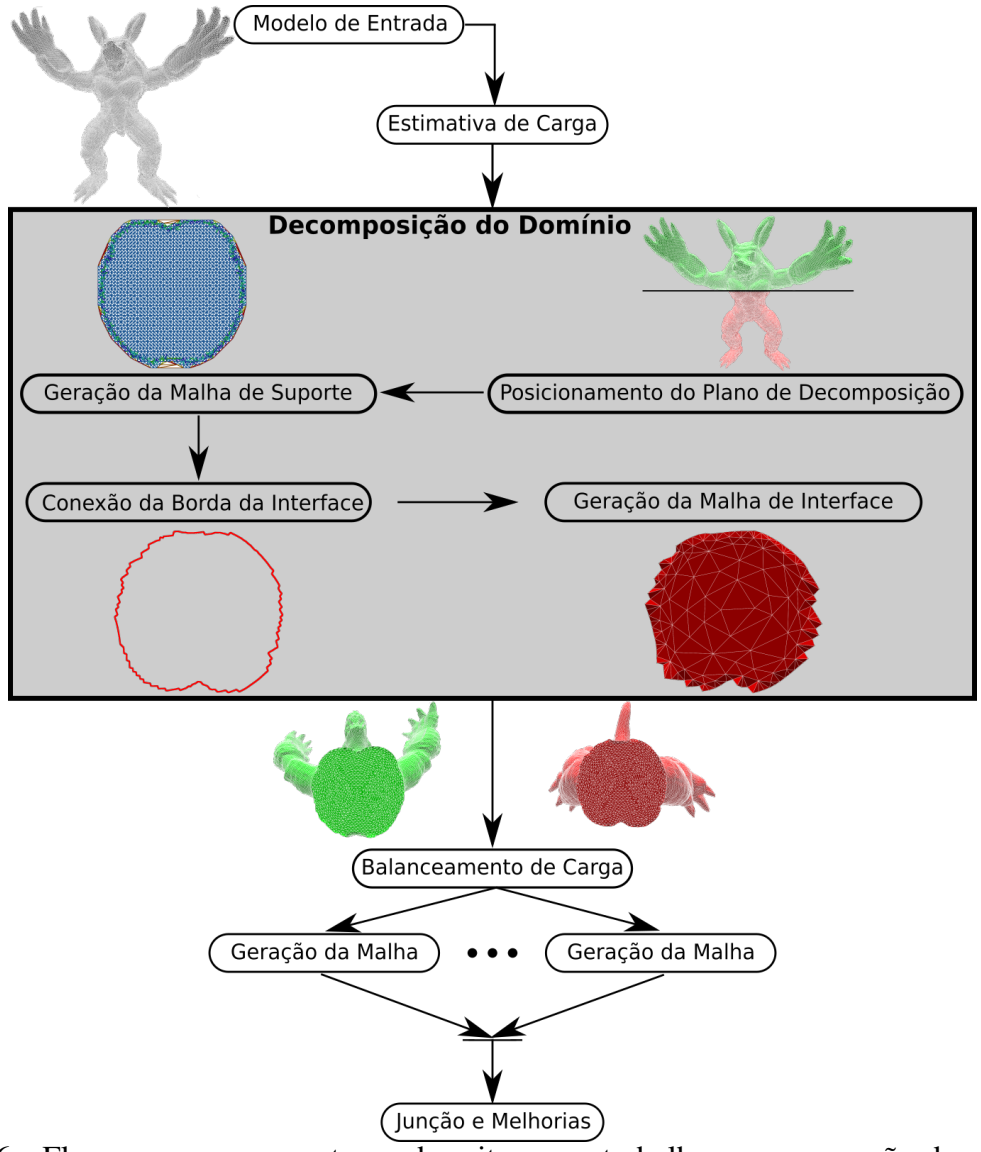


Figura 56 – Fluxograma com as etapas descrita nesse trabalho para a geração da malha em paralelo.

Fonte: elaborado pelo autor (2019).

Através dessa estrutura, é possível ter uma boa representação da distribuição de densidade dos elementos que serão gerados para uma determinada entrada. Assim, essa estrutura fornece uma boa aproximação da quantidade de trabalho computacional necessária para gerar a malha em cada subdomínio. Portanto, ela pode ser usada como uma estrutura de estimativa de carga de trabalho. Essa estimativa de carga computacional é baseada na quantidade de células que são internas ou estão sobre os elementos da FE.

A segunda etapa, apresentada na Seção 4.2, é decompor a entrada usando uma BSP (*Binary Spatial Partition*), que é uma estrutura de dados do tipo árvore binária. Apesar de, teoricamente, uma BSP poder ser construída em qualquer eixo, neste trabalho, a BSP deve ser alinhada aos eixos.

A construção da BSP é guiada pela estrutura de estimativa de carga, de forma que a quantidade de trabalho esteja distribuída o mais uniforme possível entre os subdomínios.

A construção da BSP define eixos (2D) ou planos (3D) que dividem o domínio. Esses eixos ou planos, por sua vez, não podem ser posicionados em uma região que prejudique o procedimento geração nem a qualidade da malha.

Depois disso, uma malha de interface é gerada para cada plano de decomposição (PD) que foi definido pela BSP. Após a criação da malha de interface, o subdomínio estará criado, estando totalmente desconectado de todos os outros, porém compatível com seus subdomínios vizinhos. O tamanho dos elementos das malhas de interface também são orientados pela *quadtree*/*octree* pois, como descrito anteriormente, esta funciona como uma função de densidade para o tamanho dos elementos em uma região.

Ao final do procedimento, os nós internos da árvore BSP indicam os eixos/planos de partição, e as folhas indicam os subdomínios (ver Figura 57). É possível, ainda, criar um ou mais subdomínios por processo. Esses processos geram suas malhas em paralelo, independentemente dos outros processos (terceira etapa, apresentada na Seção 4.4). Após esse procedimento paralelo de geração de malha (quarta etapa, apresentada na Seção 4.5), os processos responsáveis pelos nós internos da BSP finalizam a malha (quinta etapa, apresentada na Seção 4.6), que consiste em fazer a junção das malhas de interface de subdomínios vizinhos e melhorar a qualidade da malha em regiões próximas a ela. A Figura 57 mostra os passos para a geração de três subdomínios em uma esfera (vermelho, azul e verde). É possível notar que a decomposição segue uma estrutura do tipo árvore binária.

Nas próximas seções, serão detalhadas todas as etapas da técnica proposta. Apesar de o foco e de os testes estarem em casos 3D, ilustrações 2D serão usadas, algumas vezes, com o intuito de facilitar as explicações.

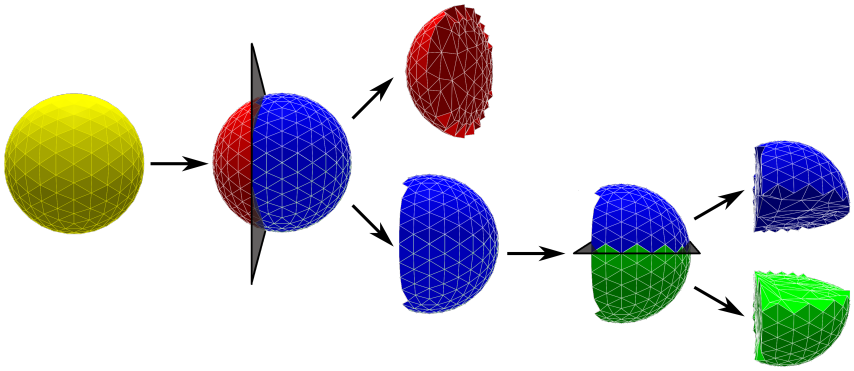


Figura 57 – Visão geral da técnica paralela para a criação de três subdomínios.
Fonte: elaborado pelo autor (2019).

4.1 Estimativa de Carga

Em Computação de Alto Desempenho (CAD), a carga é uma medida da quantidade de trabalho a ser realizado em um subdomínio ou um conjunto de subdomínios. Em problemas de geração de malha, a carga está relacionada com o número de elementos que serão gerados em cada subdomínio. A técnica de estimativa de carga usada nesse trabalho é descrita no trabalho de Freitas *et al.* (2016), que leva em consideração os seguintes pontos:

- A estimativa de carga depende do nível de discretização da malha, que é normalmente especificado pelo usuário ou por outro software, através de parâmetros de entrada, juntamente com a fronteira de entrada. A carga é maior em subdomínios com níveis de discretização mais altos (as Figuras 58 e 59 mostram regiões geometricamente iguais com discretizações diferentes).
- Em regiões com o mesmo nível de discretização, a estimativa de carga depende do tamanho do subdomínio. A carga é maior em subdomínios maiores (à esquerda da Figura 60).
- Em subdomínios de tamanhos iguais com diferentes níveis de discretização, é possível gerar um número diferente de elementos, dependendo do nível de refinamento de cada região. Assim, quanto mais refinada for a malha na região, maior será a carga (as Figuras 60 e 61 mostram regiões com discretização uniforme e não-uniforme, respectivamente)

As malhas à direita das Figuras 58, 59 e 61 sugerem uma carga maior que as malhas à esquerda das mesmas figuras. Na Figura 60, a carga associada às regiões cercadas por linhas contínuas é maior que a carga associada às regiões cercadas por linhas tracejadas.

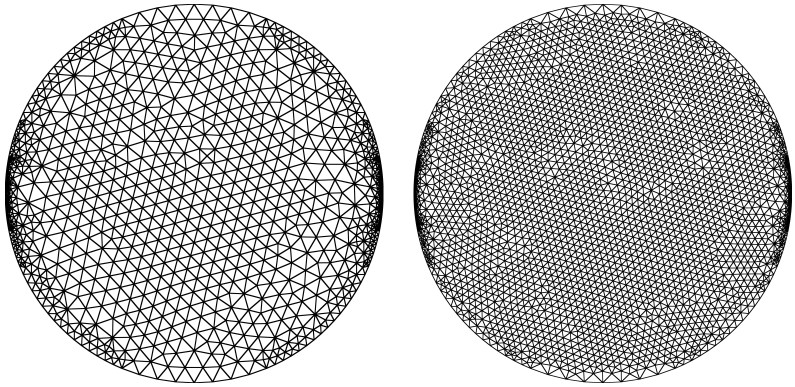


Figura 58 – Estimativa de carga para o caso bidimensional: refinamento menor (esquerda) e maior (direita)

Fonte: Freitas (2010).

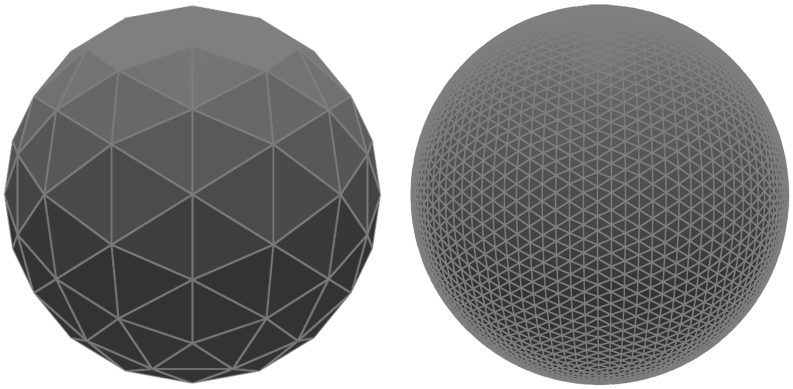


Figura 59 – Estimativa de carga para o caso tridimensional: refinamento menor (esquerda) e maior (direita).

Fonte: elaborado pelo autor (2019).

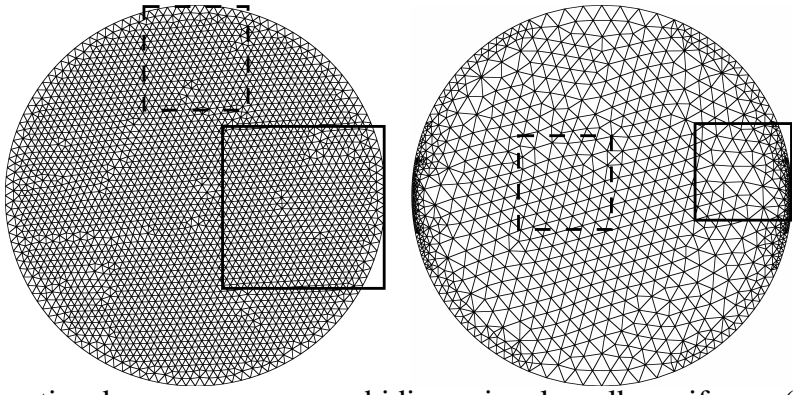


Figura 60 – Estimativa de carga para o caso bidimensional: malha uniforme (esquerda) e não-uniforme (direita).

Fonte: Freitas (2010).

4.1.1 Estrutura de Estimativa de Carga

Para obter uma boa estimativa de carga, uma ideia geral da malha desejada deve ser conhecida desde o início. Assim, uma estrutura de dados do tipo árvore (uma *quadtree*, caso seja bidimensional, ou uma *octree*, caso seja tridimensional) é construída para refletir o tamanho

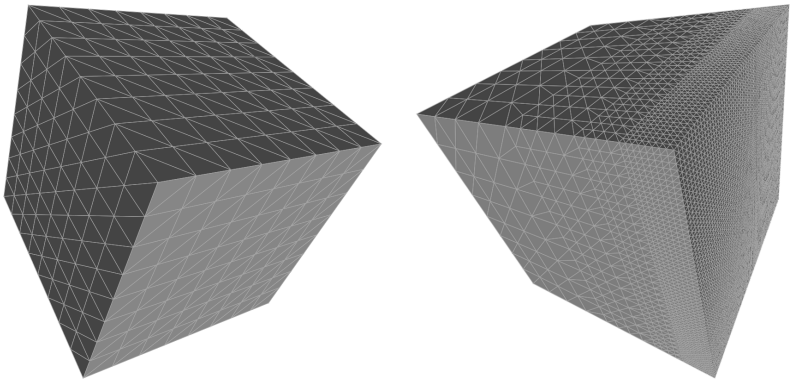


Figura 61 – Estimativa de carga para o caso tridimensional: faces de um cubo com malha uniforme (esquerda) e um cubo com malha não-uniforme (direita).
Fonte: elaborado pelo autor (2019).

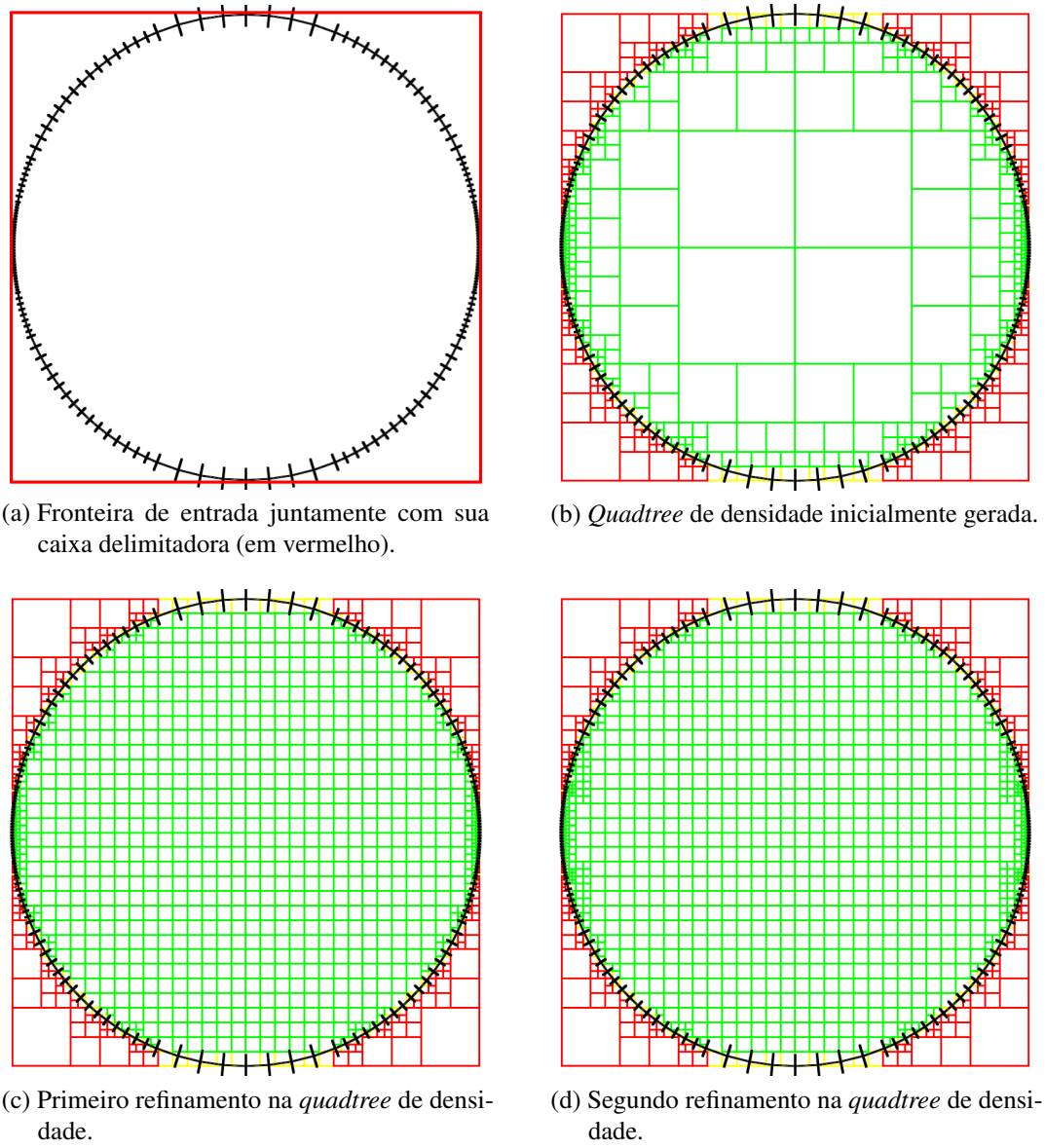
dos elementos a serem gerados. Essa estrutura de dados do tipo árvore representa uma função de densidade que ajuda no balanceamento de carga, na geração das malhas de interface dos subdomínios que serão criados, e na geração das malhas internas ao subdomínio.

A criação da *octree* obedece as mesmas regras de criação da *quadtree*. Assim, por ser mais simples de se representar e de se entender, todo o procedimento será ilustrado usando o caso bidimensional.

A *quadtree* de estimativa de carga, ou *quadtree* de densidade, é utilizada tanto para estimar a carga como para auxiliar a geração das malhas de interface. Sua criação é iniciada com a construção de um quadrado de tamanho mínimo que engloba toda a fronteira dada como entrada (Figura 62a); este quadrado é a célula-raiz da *quadtree*. Essa *quadtree* é subdividida até que todos os pontos médios das arestas da FE estejam dentro de uma célula da *quadtree* de densidade (Figura 62b). O tamanho destas células tem de ser menor que o comprimento da aresta à qual o ponto médio pertence, multiplicada por uma constante. Essa constante tem valor $\sqrt{3}/2$, que equivale a 0,85, que é uma aproximação da altura de um triângulo equilátero de lado unitário. Para o caso tridimensional, compara-se a raiz quadrada da área de uma face triangular com o comprimento da célula da *octree*, multiplicada por 0.4 (valor encontrado empiricamente).

Dois refinamentos são, então, aplicados. O primeiro garante que todas as células internas da *quadtree* de densidade não sejam maiores que a maior célula que contém um ponto médio da borda de entrada (Figura 62c). Com esse refinamento, garante-se um tamanho máximo para as células de acordo com as arestas da fronteira. Assim, os elementos que serão gerados no interior do domínio terão proporções semelhantes aos maiores elementos na borda.

O segundo refinamento, também conhecido na literatura como refinamento 2:1, tem como objetivo garantir que a diferença entre os níveis de células vizinhas (que compartilham um



(a) Fronteira de entrada juntamente com sua caixa delimitadora (em vermelho).

(b) *Quadtree* de densidade inicialmente gerada.

(c) Primeiro refinamento na *quadtree* de densidade.

(d) Segundo refinamento na *quadtree* de densidade.

Figura 62 – Passos da geração da *quadtree* de densidade.
Fonte: elaborado pelo autor (2019).

lado) da *quadtree* não seja maior que um, como mostrado na Figura 62d. Este refinamento é feito para garantir uma transição suave entre elementos grandes e pequenos. Mais detalhes acerca da construção dessa *quadtree* podem ser encontrados em Miranda *et al.* (1999) e a construção de sua versão tridimensional, uma *octree*, pode ser encontrada em Cavalcante-Neto *et al.* (2001). A Figura 63 mostra em detalhes o resultado nas células da *quadtree* quando o refinamento 2 : 1 é aplicado na Figura 62c.

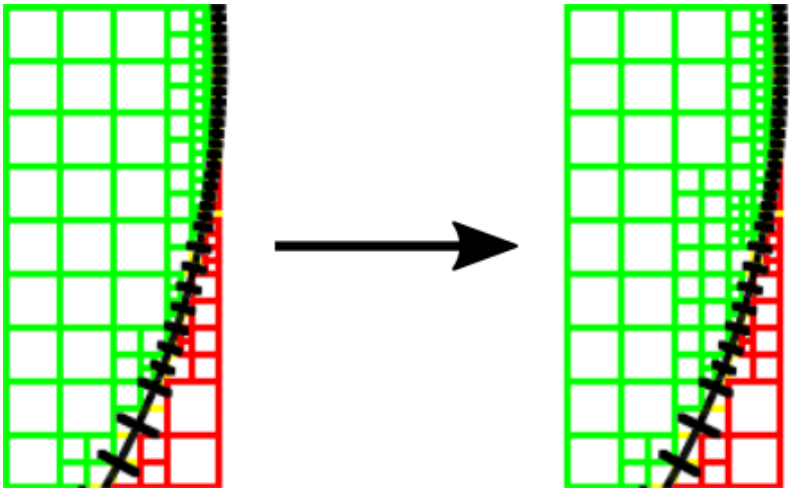


Figura 63 – *Zoom* ilustrando o segundo refinamento (refinamento 2:1) sendo aplicado em uma região da *quadtree*.
Fonte: elaborado pelo autor (2019).

4.1.2 Classificação das Células

Depois que a estrutura de estimativa de carga é construída para um modelo, uma classificação das suas células é utilizada para evitar que uma região externa ao modelo seja contabilizada para o cálculo da estimativa de carga.

Assim, as células são classificadas como: interna (caso a célula esteja totalmente interna ao domínio), externa (caso a célula esteja totalmente externa ao domínio) ou sobre a fronteira do domínio (caso a célula faça interseção com algum vértice, aresta ou face do domínio). O algoritmo de classificação é descrito em (FREITAS, 2010). A Figura 64 mostra as células classificadas de uma *quadtree* de densidade para um dado domínio circular, onde as cores atribuídas às células indicam a sua classificação (células verdes - dentro do domínio, células vermelhas - fora do domínio, células amarelas - sobre a fronteira).

4.1.3 Cálculo da Carga

A *quadtree* de densidade é utilizada para fazer uma estimativa da carga neste trabalho. Com a sua construção, é possível ter noção do tamanho dos elementos pertencentes ao domínio do objeto. Assim, uma ideia geral sobre a malha desejada é descoberta através da *quadtree*. A classificação das células folhas da *quadtree* de densidade é utilizada para selecionar apenas as células que não estão fora do domínio de entrada, ou seja, as células que estão no interior do domínio e as que interceptam a borda (células verdes e amarelas na Figura 64). A quantidade dessas folhas não externas é considerada como a carga total para o domínio e seus subdomínios.

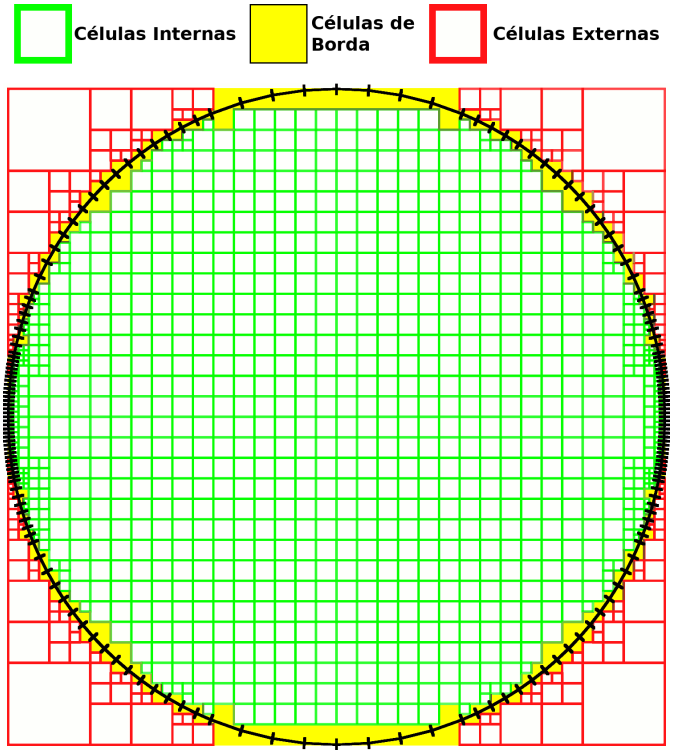


Figura 64 – Quadtree de densidade com as células devidamente classificadas (células com bordas verdes - dentro do domínio, células com bordas vermelhas - fora do domínio, células amarelas - sobre a fronteira)

Fonte: elaborado pelo autor (2019).

4.2 Decomposição do Domínio

A técnica proposta neste trabalho tem apenas um pré-requisito quanto à estrutura de decomposição espacial para decompor o domínio dado como entrada: esta deve gerar regiões que sejam paralelas às células da *quadtree* de densidade, ou seja, devem ser alinhada aos eixos cartesianos. Estruturas baseadas em árvore como *quadtree*, *octree*, BSP (*Binary Space Partition*) e *Kd-tree* (*k-dimensional tree*) são algumas das opções, pois geram regiões que possibilitam a perfeita execução do algoritmo de geração dos subdomínios.

Essa restrição tem a finalidade de facilitar a estimativa de carga dos subdomínios, após a definição do plano de decomposição (PD), alinhado aos eixos globais. Isso acontece porque classificar uma célula dessas estruturas espaciais como interna ou externa a uma região é muito mais simples e exato que computar interseções entre elas. De outra maneira, calcular interseções de um PD em um eixo/plano qualquer com as células da estrutura de estimativa de carga para descobrir a fração de carga que estaria interna ou externa ao subdomínio seria pesado computacionalmente.

Além disso, é importante ressaltar que, para uma boa decomposição, é preciso

uma boa estimativa de carga e, para se obter uma malha de boa qualidade, é preciso uma boa decomposição do domínio. Esta seção descreve como é feita a criação de uma BSP alinhada aos eixos para decompor a entrada. Dentre todas as estruturas de dados a BSP (*Binary Space Partition*) alinhada aos eixos é a estrutura que dá mais liberdade na criação dos subdomínios.

4.2.1 Decomposição Utilizando BSP

Uma excelente estrutura de decomposição espacial é a BSP, aqui chamada de BSP de decomposição, tendo a sua criação guiada pela estrutura de estimativa de carga descrita anteriormente, na Seção 4.1. A utilização desta estrutura foi baseada no trabalho de (FREITAS *et al.*, 2014), onde mais detalhes da construção e implementação podem ser encontrados. A BSP de decomposição pode ser construída de diversas formas, dependendo apenas do critério de decomposição. Pode-se, então, dividir as formas de decomposição como baseadas e não baseadas na carga.

Uma das formas de decomposição baseadas na carga é utilizar o critério que cada subdomínio deva ter idealemente uma carga próxima a carga média (L/P, sendo L a carga total da entrada e P a quantidade de processadores disponíveis ou a quantidade de partições desejada). Essa é uma das formas mais eficientes de gerar a BSP de decomposição, pois é possível obter $N = P$, sendo N a quantidade de subdomínios criados, com uma carga muito próxima a L/P para cada partição.

Os modos de decomposição que não se baseiam na carga devem se basear em algum outro critério para realizar a decomposição, como, por exemplo, a geometria do modelo na região da partição. O eixo mediano é um critério que se baseia apenas na geometria do modelo de entrada para realizar a decomposição.

Esse trabalho utiliza uma combinação dos dois modos de decomposição, pois a BSP de decomposição, utilizada nesse trabalho, é construída de tal forma que a diferença de carga associada a cada dois novos subdomínios deve ser a menor possível e os ângulos entre o plano de decomposição (PD) e os vetores normais às faces (elementos da FE) que ele toca devem ser maiores que um *threshold* pré-especificado.

4.2.2 Posicionamento do Plano de Decomposição

O posicionamento de um plano de decomposição do modelo envolve três passos. O primeiro passo consiste em encontrar a posição ideal para cada eixo global. O segundo passo

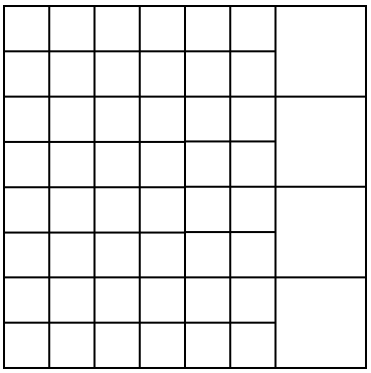
consiste em verificar se algum dos planos em decomposição definidos na etapa anterior formam ângulos ruins com os elementos da fronteira, nesse caso, o plano é realocado. O terceiro passo consiste em selecionar o plano que melhor divide a carga entre as opções de partições. O primeiro e o terceiro passos são descritos detalhadamente em Freitas *et al.* (2016). O segundo passo, entretanto, foi adicionado nesse trabalho, por se tratar de uma técnica *a priori*.

O primeiro passo começa com a raiz da BSP de decomposição sendo definida como um quadrado, em 2D, ou um cubo, em 3D, que envolve todo o domínio. A subdivisão é executada posicionando o plano de decomposição (PD) no centro geométrico dessa célula no eixo X. Esse plano divide as células filhas da raiz da *octree*, deixando 4 células filhas à esquerda e as outras quatro à direita, no caso 3D (duas células filhas de cada lado, em 2D). A Figura 65 mostra os passos para encontrar o melhor PD no eixo X em um modelo bidimensional.

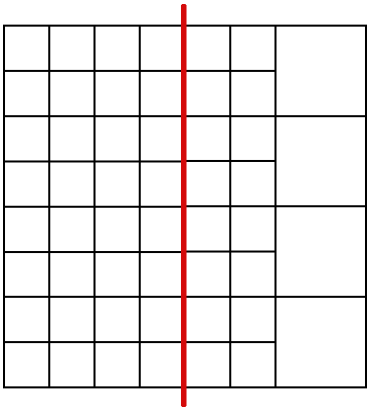
Se esse posicionamento gerar cargas iguais para as duas partições, essa posição será a melhor para o eixo X. Caso contrário, a célula BSP mais pesada é selecionada e o PD é reposicionado no centro dessa célula, no mesmo eixo. Este procedimento é realizado recursivamente até que as cargas dos dois subdomínios sejam iguais ou quando as células-folhas da estrutura de estimativa de carga sejam atingidas, ou seja, quando não for mais possível descer na estrutura da árvore quadtree/octree. Dentre todas as posições testadas, é selecionada aquela que melhor balanceia a carga entre os dois subdomínios, ou seja, aquela que minimiza o valor absoluto da diferença entre as cargas dos dois subdomínios. Este procedimento é executado para cada eixo, havendo assim três posições ideais de PD, uma para cada eixo cartesiano (duas posições ideais, no caso 2D).

Outros critérios, como considerar a carga referente a interface, podem ser aplicados juntos na primeira etapa, para encontrar a melhor posição do PD. Esse critério indicaria uma carga trabalho de geração de malha na interface, necessária em técnicas *a posteriori*, como é o caso de Freitas *et al.* (2016). Entretanto, como esse trabalho se trata de uma técnica *a priori*, logo este critério não é necessário.

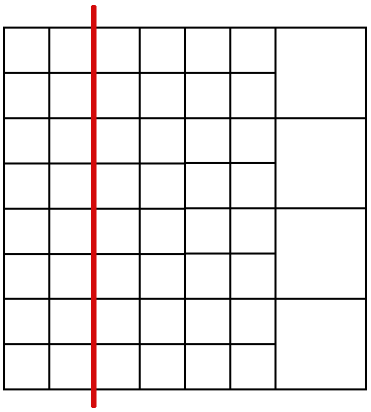
O segundo passo é verificar e corrigir, se necessário, a posição de cada PD definido no primeiro passo. O objetivo é evitar regiões finas ou ruins, como mostra a Figura 66. Uma região é dita ruim quando os elementos da FE próximos ao PD estão praticamente paralelos a ele. Nesse caso, seria criada uma região muito fina, ou estreita, entre o modelo e o PD, levando à geração de elementos de má qualidade nessa região. Isso é feito verificando o ângulo formado entre o PD e os elementos da fronteira de entrada (FE) do modelo (Figura 68a). Para o caso



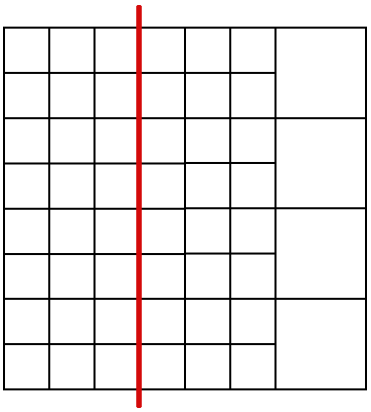
(a) *Quadtree* de estimativa de carga. Carga total: 52. Carga ideal de cada subdomínio: 26.



(b) Posição inicial do plano de partição (em vermelho). Cargas dos subdomínios: 32 e 20. Diferença: 12.



(c) Deslocamento do plano de partição para o próximo nível da esquerda. Cargas dos subdomínios: 16 e 36. Diferença: 20.



(d) Deslocamento do plano de partição para o próximo nível da direita. Cargas dos subdomínios: 24 e 28. Diferença: 4.

Figura 65 – Passos de uma decomposição bidimensional de um domínio por BSP no eixo X para dois processadores, buscando a diferença mínima de carga total.

Fonte: elaborado pelo autor (2019).

tridimensional, esse ângulo é calculado utilizando as normais (do PD e a do elemento da FE), como mostra a Figura 67.

Nesse trabalho, definimos que o ângulo deve ser maior que 17°, o que resulta em decomposições de boa qualidade e evita falhas no algoritmo de geração de malha. Esse ângulo está dentro da faixa de valores mencionada por Shewchuk (1996a), onde diz que se o ângulo mínimo for 20,7° ou menor, teoricamente é garantido que o algoritmo de triangulação seja finalizado. Se a verificação falhar, o PD deve ser movido para outra posição. O fator de deslocamento é definido como o lado da menor célula da estrutura de estimativa de carga.

A direção do deslocamento do PD é sempre alternada, ou seja, varia entre os deslocamentos positivos e negativos. No caso de um certo deslocamento ser aplicado e o teste do

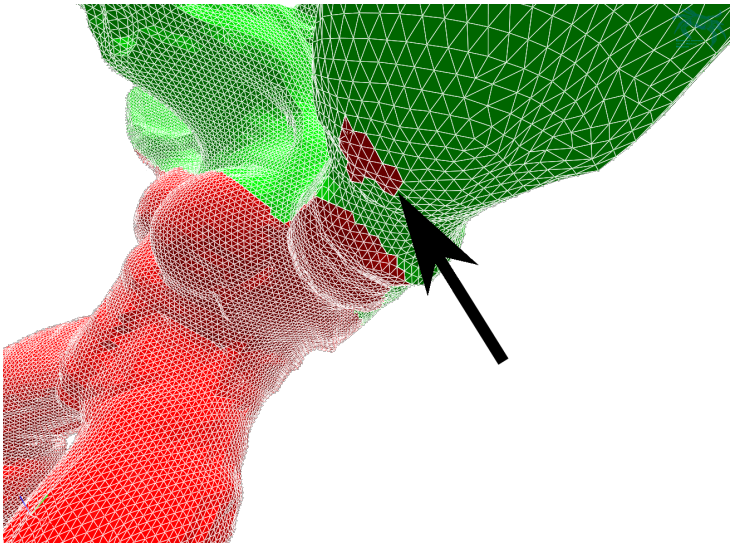


Figura 66 – Vista lateral de uma decomposição realizada no eixo Y no modelo Armadillo. Observe a região ruim no braço do modelo (seta).

Fonte: elaborado pelo autor (2019).

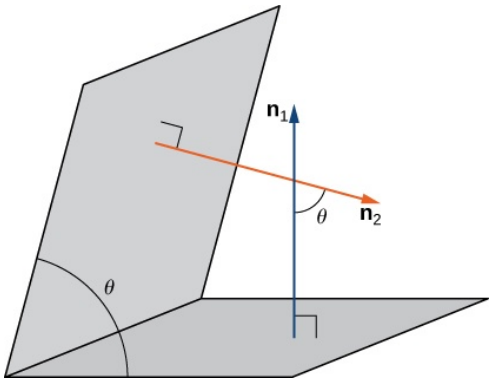
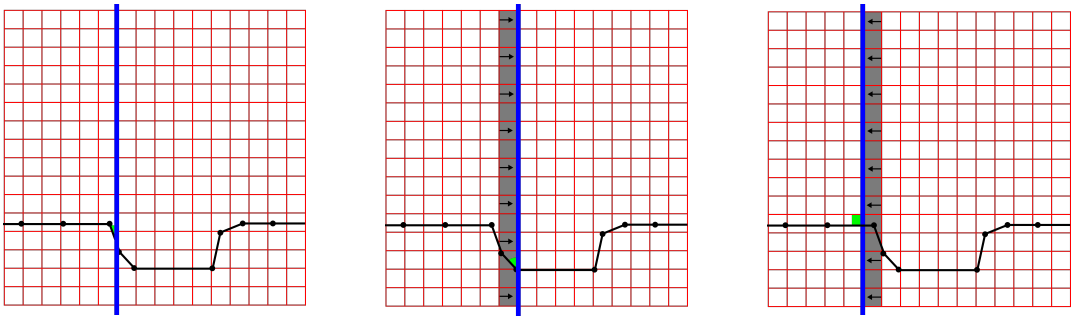


Figura 67 – Calculo do ângulo entre dois planos.

ângulo falhar, o próximo deslocamento deve ser na direção oposta (Figuras 68b e 68c). Após duas tentativas de deslocamento com falha (uma com sentido positivo e outra, negativo), o fator de deslocamento é aumentado pelo tamanho da menor célula da estrutura de estimativa de carga. Este procedimento só termina quando a verificação do ângulo é satisfeita ou quando o deslocamento atingir um limite pré-estabelecido.

Nesse trabalho, o número máximo de deslocamentos para cada direção é definido como $\lceil \sqrt[d]{L} * p \rceil$, onde L é a carga do subdomínio que está sendo particionado, d é a dimensão da entrada e p foi definido nesse trabalho como 10%. A quantidade de deslocamentos não pode ser um valor muito alto para que a carga dos subdomínios não seja significativamente afetada por essa verificação de ângulo e a qualidade da malha gerada possa ser melhor. O Algoritmo 1 mostra como a verificação do ângulo e o possível deslocamento do PD são feitos para o eixo X. No caso em que número máximo de deslocamentos for atingido, o PD será posicionado na região em que o ângulo que falha é o máximo, quando comparado aos outros ruins.



(a) Posição inicial do PD em um eixo. O ângulo entre o PD e o elemento da fronteira forma um ângulo menor do que o limite estipulado.

(b) O PD é movido para a direita, mas o teste do ângulo continua a falhar para o ângulo limite estipulado.

(c) O PD é, então, movido para a esquerda (em relação à posição inicial) e passa no teste do ângulo.

Figura 68 – Etapas do teste do ângulo para validar uma posição do PD.
Fonte: elaborado pelo autor (2019).

No final dos dois primeiros passos, haverá um PD para cada eixo, estando eles na melhor posição possível dentro dos critérios estabelecidos. O terceiro passo é selecionar o melhor PD entre os eixos disponíveis. Neste trabalho, o PD selecionado é o que proporciona a menor diferença de carga entre dois subdomínios vizinhos.

4.2.3 Decomposição do Domínio em Paralelo

O procedimento de criação do plano de decomposição (PD) descrito anteriormente é executado até que a quantidade de subdomínios criada seja igual ao valor desejado. Esse procedimento de decomposição é realizado em paralelo, por todos os processos disponíveis. Uma vantagem de utilizar uma subdivisão baseada em BSP é que cada processo precisa decompor apenas as partes do modelo necessárias para alcançar seu próprio subdomínio. Ou seja, cada processo trabalha apenas nas regiões necessárias para alcançar seus subdomínios sob sua responsabilidade.

A Figura 69 mostra as etapas para criar o BSP de decomposição. Neste exemplo, uma esfera é subdividida em 8 subdomínios, atribuídos a 8 processos. Observe que, para evitar a comunicação excessiva entre processos, mais de um processo pode trabalhar em uma cópia de uma região. A Figura 70 mostra apenas as regiões que o Processo 4 precisa decompor para chegar em seu subdomínio e quais os processos também processam essas regiões. É importante observar que, dentre todos os processos que passam por uma determinada região, o processo definitivamente responsável por ela é o de menor identificador. A distribuição dos identificadores e dos subdomínios são feitos utilizando a própria estrutura da BSP de decomposição.

Algoritmo 1: Verificação do Plano de Decomposição (PD) para o eixo X

```
deslocamento = estruturaArvore.getTamFolha();
posInicial = PD.getX();
posAngMax = PD.getX();
planoEncontrado = False;
melhorAngulo = 0.0;
p = 10%;
for tentativa in range(1, 2 * ⌈ dim√(subdominio.getCarga() * p) ⌉) do
    if tentativa é par then
        PD.setX(posInicial + deslocamento);
    else
        PD.setX(posInicial − deslocamento);
        deslocamento+ = estruturaArvore.getTamFolha();
    end
    if anguloEstaOk(listElemBorda, PD) then
        planoEncontrado = True;
        break;
    else
        angulo = encontrarPiorAngulo(listElemBorda, PD);
        if angulo > melhorAngulo then
            melhorAngulo = angulo;
            posAngMax = PD.getX();
        end
    end
end
if planoEncontrado == False then
    PD.setX(posAngMax);
end
```

Quando o número desejado de subdomínios não é potência de dois, é necessário adaptar o processo de criação da BSP. Por exemplo, suponha um caso em que a carga total de um modelo de entrada seja 1000 e deseja-se criar 3 subdomínios. Caso os PDs sejam sempre perfeitos, a primeira divisão geraria duas regiões de carga 500. Uma dessa regiões deverá ser dividida novamente para se obter os 3 subdomínios, resultando, assim, em três regiões com cargas 500, 250 e 250. A carga ideal para este exemplo seria 333 para cada subdomínio, porém a carga obtida está visivelmente desbalanceada e longe da ideal.

Para corrigir esse problema, são aplicados fatores de proporcionalidade durante a construção da BSP, para indicar que a carga de uma região deve ser X vezes maior que a carga da outra região. Ao aplicar esses fatores de proporcionalidade, é possível obter qualquer número desejado de subdomínios, com cargas menos discrepantes.

A Figura 71 mostra o processo de decomposição do modelo Fertility para três

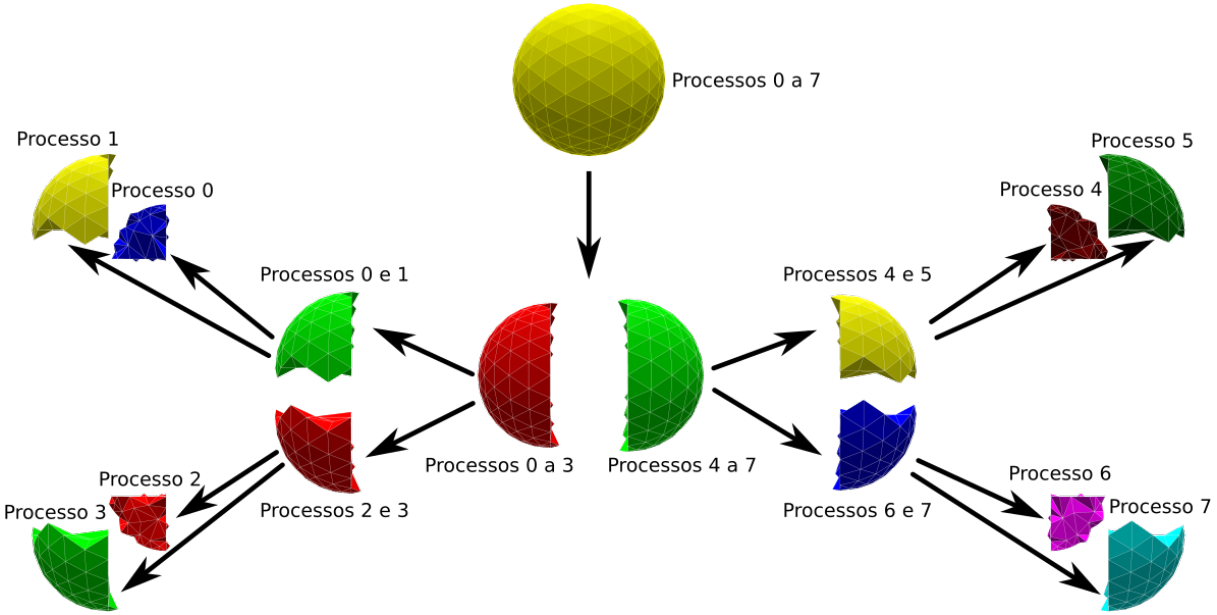


Figura 69 – Visão global da criação de uma BSP para uma esfera e os subdomínios resultantes para cada processo. Cada célula-folha da BSP representa um processo diferente.

Fonte: elaborado pelo autor (2019).

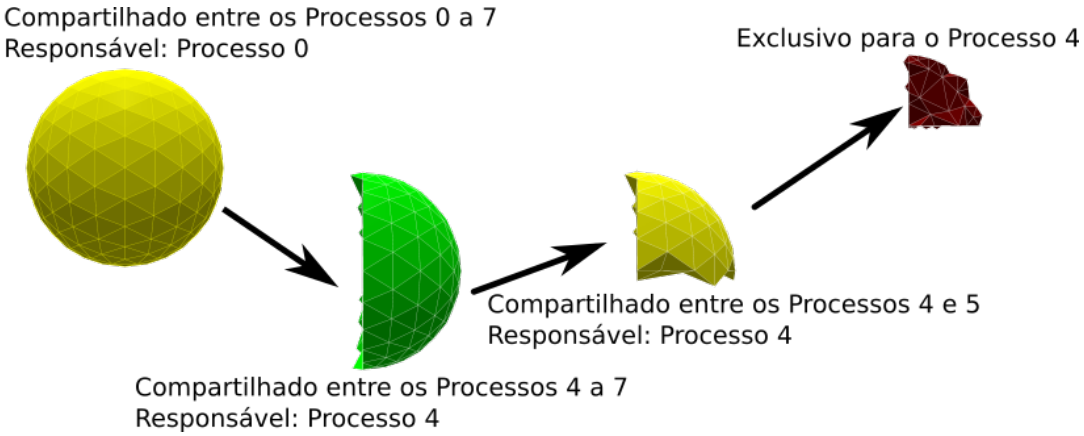


Figura 70 – Processos responsáveis por realizar a decomposição em cada região até chegar no subdomínio do Processo 4. Observe que outros processos podem funcionar em uma cópia da mesma região que o Processo 4, mas, na última região encontrada, o subdomínio é exclusivamente seu.

Fonte: elaborado pelo autor (2019).

regiões, cujo fator de proporcionalidade usado para a raiz da BSP é $1:2$. Esse fator indica que, na raiz da BSP, a carga de trabalho de um dos subdomínios deve ser duas vezes a carga de trabalho do subdomínio vizinho. Na sequencia, o subdomínio de maior carga é subdividido em dois subdomínios com fator de proporcionalidade $1:1$. Portanto, com o uso do fator de proporcionalidade, é possível obter três subdomínios com estimativas de carga semelhantes.

Na Figura 69, todos os fatores de proporcionalidades são $1:1$, pois o valor 8 é uma potência de dois. Mais detalhes sobre a decomposição do domínio em paralelo e sobre o fator de proporcionalidade podem ser encontrados em Freitas *et al.* (2016).

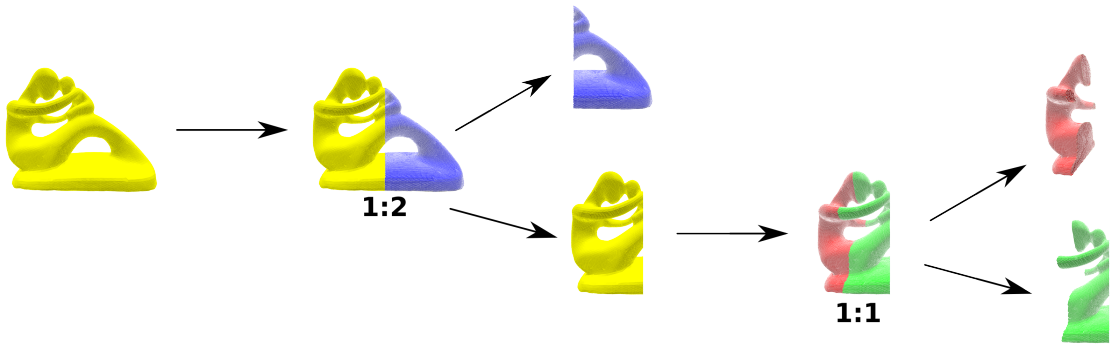


Figura 71 – Exemplo da criação de uma BSP para três subdomínios para o modelo Fertility. Os fatores de proporcionalidade são indicados em cada nó da BSP.

Fonte: elaborado pelo autor (2019).

4.3 Geração das Interfaces dos Subdomínios

Na geração das interfaces dos subdomínios, é necessário ter a estrutura de estimativa de carga (uma *quadtree* no caso bidimensional ou uma *octree* no caso tridimensional) e a estrutura de decomposição (a BSP) devidamente criadas, ambas apresentadas nas Seções 4.1 e 4.2. Qualquer outra estrutura de decomposição espacial que gere regiões paralelas aos eixos pode ser utilizada nesta técnica de geração de subdomínios, que é uma vantagem desta técnica de decomposição.

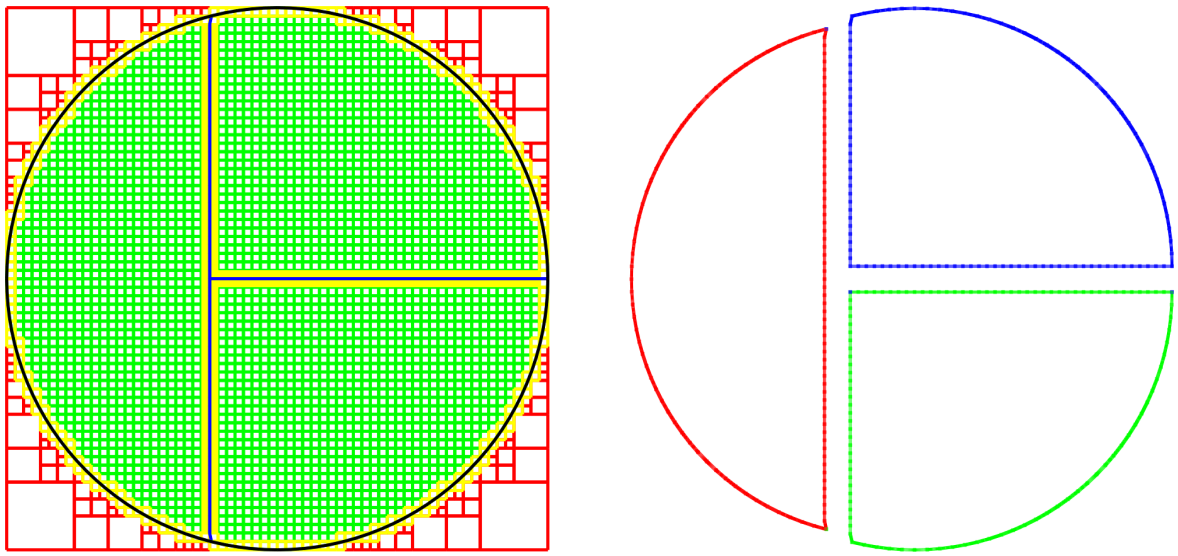
As malhas de interface dos subdomínios são geradas à medida que a BSP está sendo criada recursivamente. Isso significa que as malhas de interface são construídas antes que a malha interna dos subdomínios sejam geradas. Por esse motivo, essa técnica é chamada a *priori*.

Para gerar a malha de interface dos subdomínios, é necessário ter uma função de tamanho (2D) ou uma malha de suporte (3D) e os elementos da FE do modelo que o PD toca. A função de tamanho ou a malha de suporte, que são geradas automaticamente, conforme explicado na próxima seção, orientam a geração dos elementos da malha de interface e são responsáveis pelo tamanho e o posicionamento deles. Os elementos da FE, segmentos em 2D ou faces em 3D, que cruzam com PD, são necessários para fazer a malha de interface se unir ao modelo, de modo que não haja buracos ou falhas no novo subdomínio. Isso também é feito automaticamente pela técnica.

4.3.1 Grade de Suporte

Para cada PD criado, é descoberta a sua interseção com as células folhas da estrutura de estimativa de carga (Figura 72a para o caso bidimensional e Figura 73a para o caso

tridimensional). Como resultado dessa interseção, é obtido um conjunto de células da estrutura de estimativa de carga que cruzam ou apenas tangenciam a partição que se deseja criar. Essas células guiarão a criação das fronteiras dos seus respectivos subdomínios (as fronteiras criadas para o caso bidimensional e tridimensional, respectivamente, estão nas Figuras 72b e 73b). Com a estrutura de estimativa de carga criada, é possível obter informações de tamanho, que são baseadas nas arestas ou faces do modelo de entrada, e essas informações ajudam na criação da malha de interface compatível com a discretização do domínio.



(a) Interseção entre a *quadtree* de estimativa de carga e a BSP de decomposição.

(b) Interfaces do modelo geradas para três subdomínios.

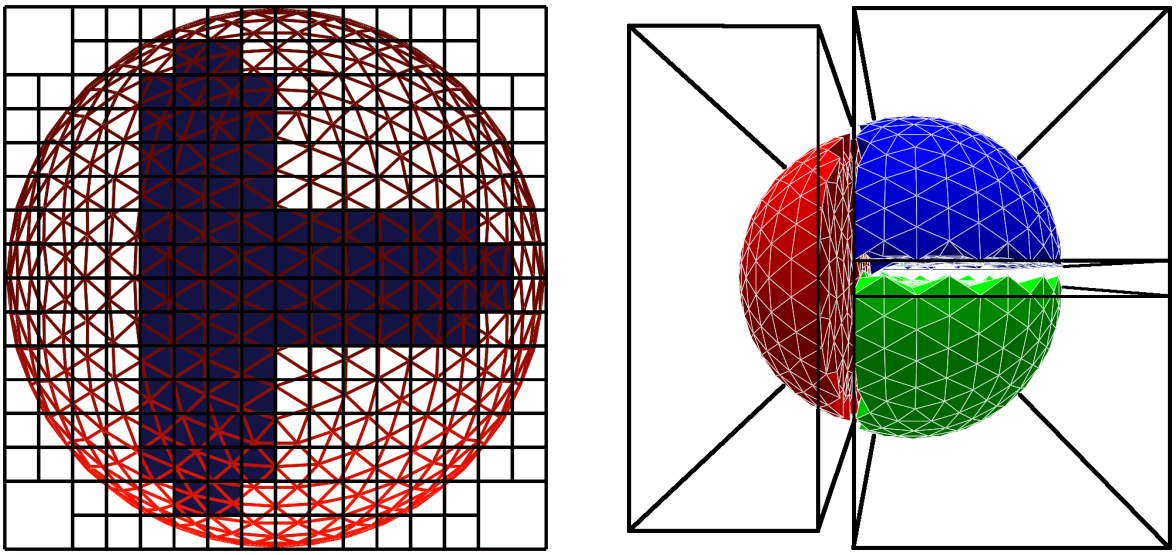
Figura 72 – Células da *quadtree* de estimativa de carga usadas para guiar a geração da interface entre os subdomínios.

Fonte: elaborado pelo autor (2019).

A Figura 72 mostra um exemplo do caso bidimensional da junção da *quadtree* de estimativa de carga com uma BSP de particionamento e a sua interface gerada, já a Figura 73 mostra um exemplo semelhante, porém para o caso tridimensional com uma *octree*.

Dentre todas as células da estrutura de estimativa de carga, deve-se selecionar um conjunto de células que fazem interseção com o PD. As células que não fazem nenhum tipo de interseção com o PD não estão nesse conjunto.

Células que estão totalmente fora da fronteira do domínio, ou seja, que não estão dentro ou sobre a fronteira do domínio, mas que também fazem interseção com a estrutura de particionamento, não devem fazer parte desse conjunto. Como essas células estão totalmente fora do domínio, não haverá criação de interfaces nessas regiões; logo, elas devem ser retiradas desse conjunto.



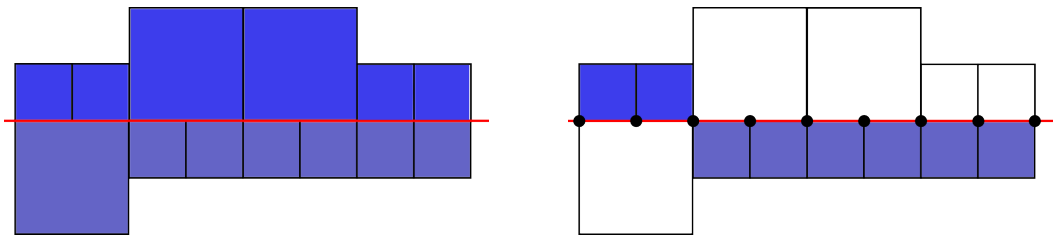
(a) Interseção entre a *octree* de estimativa de carga e a BSP de decomposição. As células da *octree* de estimativa de carga que foram selecionadas estão em azul.

(b) Interfaces do modelo geradas para três subdomínios.

Figura 73 – Células da *octree* de estimativa de carga usadas para guiar a geração da interface entre subdomínios.

Fonte: elaborado pelo autor (2019).

Após a criação do conjunto de células internas que interceptam o PD, é realizada a construção da grade de suporte, que tem como base as menores células deste conjunto de células. Na Figura 74a as célula em azul escuro são externas ao subdomínio e as em azul claro são internas. Depois que as menores células são selecionadas, os vértices são criados usando as suas informações, resultando nos vértices mostrados na Figura 74b. O mesmo procedimento acontece para as células da *octree*, porém, como a visualização bidimensional é mais didática do que a tridimensional, optou-se por mostrar apenas um exemplo bidimensional. A Figura 75 mostra os vértices criados para um caso tridimensional.



(a) Exemplo onde a *quadtree* de densidade possui células de tamanhos diferentes em subdomínios vizinhos.

(b) Células em azul foram as selecionadas para geração da grade de suporte.

Figura 74 – Processo de criação da grade de suporte para o caso bidimensional.

Fonte: elaborado pelo autor (2019).

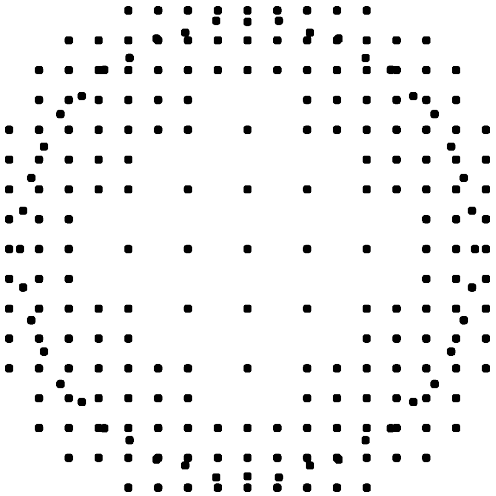
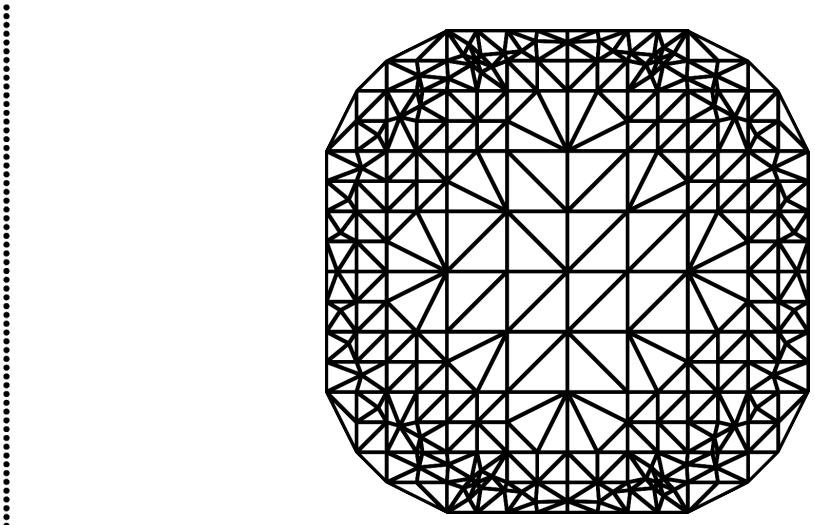


Figura 75 – Vértices criados com base nas menores células de uma *octree*.
Fonte: elaborado pelo autor (2019).

No caso bidimensional, a grade de suporte é apenas uma sequência de vértices. No caso tridimensional, uma triangulação bidimensional deve ser aplicada usando os vértices previamente criados. A Figura 76 ilustra duas grades de suporte, que foram geradas para um modelo bidimensional e outro tridimensional. Neste trabalho, uma técnica de Delaunay descrita em Shewchuk (1996b) foi empregada para gerar a triangulação.



(a) Grade de suporte resultante do PD que passa entre a partição vermelha e a junção da partição azul com a verde da Figura 72.

(b) Grade de suporte resultante do PD que passa entre a partição vermelha e a junção da partição azul com a verde da Figura 73

Figura 76 – Grades de suporte para os casos bidimensional e tridimensional.
Fonte: elaborado pelo autor (2019).

4.3.2 Elementos de Interseção

Para conectar a malha de interface com a borda do modelo de entrada, alguns processamentos devem ser feitos para que não fiquem falhas ou buracos no subdomínio criado. O caso bidimensional, em particular, é mais simples que o tridimensional.

No caso bidimensional é feita a interseção das arestas da fronteira de entrada (FE) do modelo com o PD feito pela BSP de decomposição. As arestas de interseção resultantes serão os candidatos a fazerem parte da malha de interface. A Figura 77 mostra o resultado obtido para o caso bidimensional.

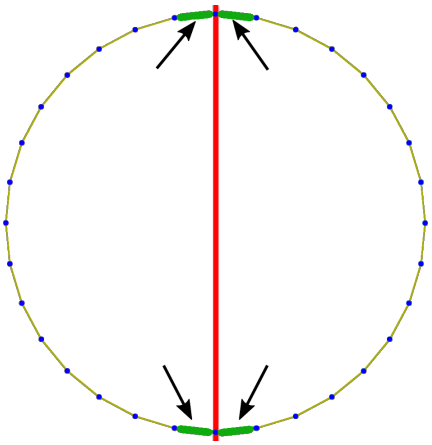
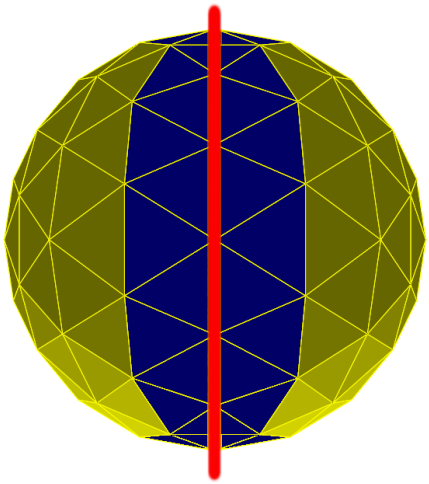


Figura 77 – Arestas, em verde, resultantes da interseção do PD com as arestas da FE, no caso bidimensional.
Fonte: elaborado pelo autor (2019).

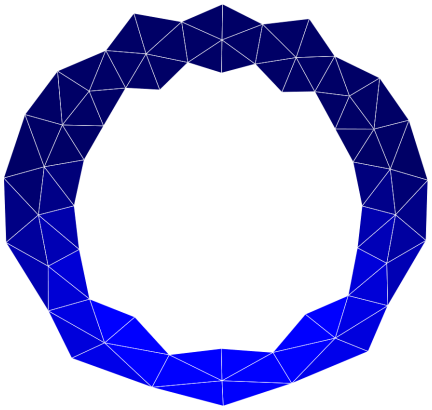
Já no caso tridimensional, a interseção das faces da FE do modelo com o PD da BSP de decomposição irá resultar em um conjunto de faces triangulares. A Figura 78 ilustra o resultado obtido para o caso tridimensional. Usando as faces selecionadas, é feita uma busca pelo melhor ciclo de arestas para criar a malha de interface na próxima etapa.

O critério de seleção inicial das arestas é obter todas as arestas que interceptam o plano de partição, eliminando aquelas que ficarem soltas (vértices com menos de dois segmentos adjacentes) ou que formem ciclos desnecessários, como em CHEN (2012). Ao final, é obtido um ciclo, que não é necessariamente o melhor, e, por isso, é necessário realizar mais um passo de melhoria. As Figuras 79a e 79c ilustram em vermelho o ciclo inicial de arestas encontrado, tendo como base as faces do modelo que interceptaram o PD.

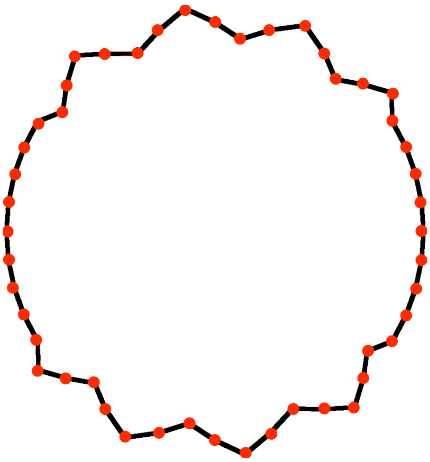
A melhoria do ciclo de arestas é feita visando melhorar o ângulo entre as arestas que, consequentemente, melhorará a malha de interface que será gerada. Para cada par de arestas vizinhas, é feita uma verificação se existe uma face que contém estas duas arestas; se existir e



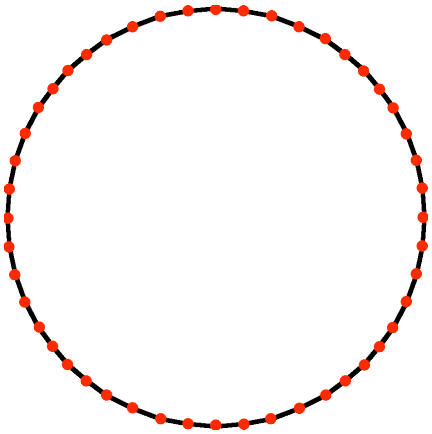
(a) Interseção do PD com o modelo de entrada. As faces em azul fazem insterseção com o PD.



(b) Visão lateral das faces que fazem insterseção com o PD.



(c) Seleção das melhores arestas que formam um ciclo em torno da partição.

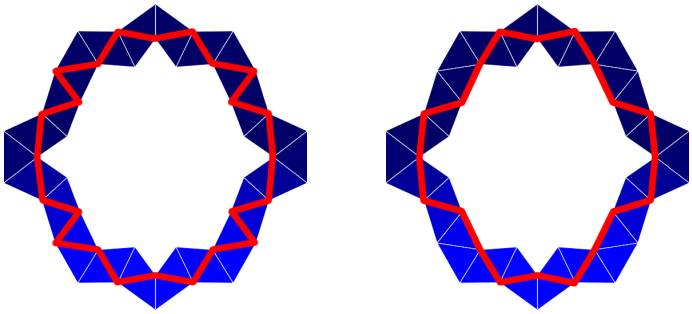


(d) Projeção das arestas selecionados para o plano criado pelo PD.

Figura 78 – Seleção das melhores arestas que interceptam o PD, no caso tridimensional.
Fonte: elaborado pelo autor (2019).

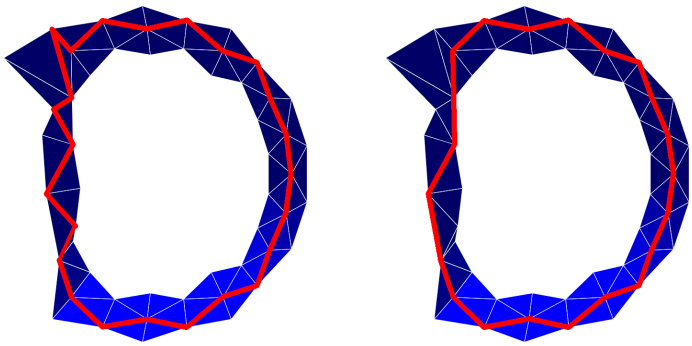
o ângulo entre as arestas melhorar, estas duas arestas são removidas do ciclo e é adicionada a outra aresta da face encontrada. Um exemplo pode ser visto da Figura 79 onde é possível ver as mudanças que ocorreram quando a melhoria foi realizada. A Figura 78c mostra os segmentos selecionados para a decomposição de uma esfera.

O último passo é projetar os segmentos selecionados no mesmo plano que o PD. Deve-se tomar cuidado para que dois vértices não sejam projetados nas mesmas coordenadas cartesianas. Se isso acontecer, os dois vértices coincidentes são movidos em direções opostas, respeitando a direção de seus segmentos adjacentes. A Figura 78d mostra o resultado da projeção dos segmentos mostrados na Figura 78c.



(a) Modelo 01: Arestas selecionadas inicialmente.

(b) Modelo 01: Melhoria realizada na arestas previamente selecionadas.



(c) Modelo 02: Arestas selecionadas inicialmente.

(d) Modelo 02: Melhoria realizada na arestas previamente selecionadas.

Figura 79 – Seleção do ciclo de arestas (em vermelho) com e sem melhoria.
Fonte: elaborado pelo autor (2019).

4.3.3 Geração da Malha de Interface - Caso Bidimensional

No caso bidimensional, as arestas da interface são criadas seguindo a sequência de vértices encontrada anteriormente, na Seção 4.3.1. É necessário realizar a junção dessas arestas com a FE do modelo e, para isso, é feita uma busca nos vértices que fazem interseção com o PD. O vértice que formar o maior ângulo com a interface será o candidato para realizar a junção da interface com o modelo. Ao final desse processo cada partição terá sua interface construída e estará pronta para geração da malha, como mostra a Figura 80.

4.3.3.1 Teste de Proximidade

Um dos problemas de se utilizar estruturas que geram regiões paralelas aos eixos para decompor domínios, é a possibilidade de os PD estarem posicionados em regiões onde os elementos a serem gerados sejam de má qualidade ou até mesmo em posições em que não é possível gerar elementos (a Figura 81a mostra um caso em que o PD está muito próximo da fronteira do modelo dado como entrada).

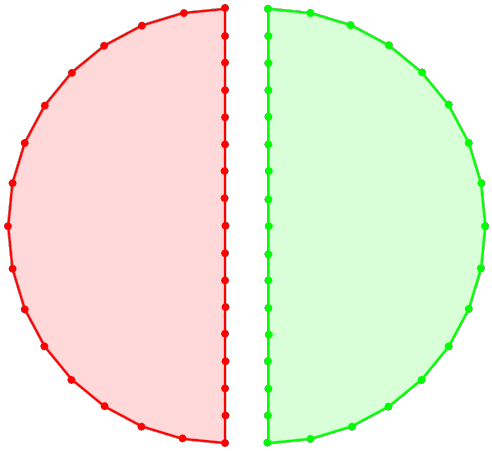
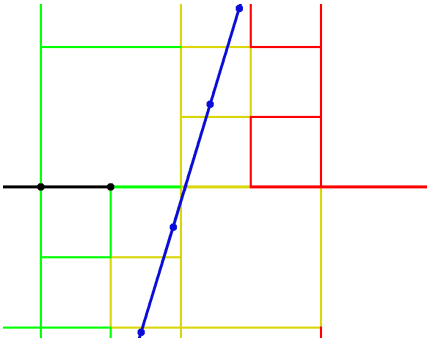
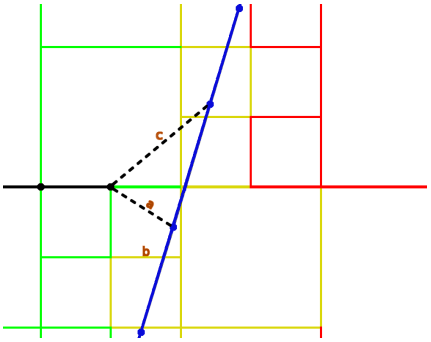


Figura 80 – Dois subdomínios bidimensionais totalmente criados e prontos para a fase de geração de malha.

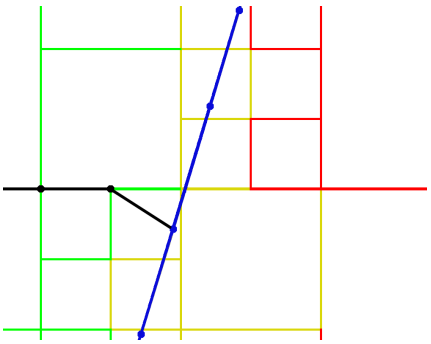
Fonte: elaborado pelo autor (2019).



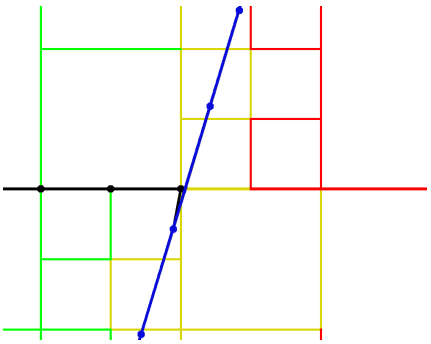
(a) Caso em que o tratamento de proximidade é realizado.



(b) Busca pelo vértice da fronteira que forma o maior ângulo entre os candidatos.



(c) Aresta criada pelo teste de proximidade.



(d) Aresta que seria criada caso o teste de proximidade não fosse realizado.

Figura 81 – Possíveis casos no teste de proximidade.

Fonte: elaborado pelo autor (2019).

A busca pelo vértice da fronteira mais próximo que intercepta a partição não garante a qualidade do subdomínio criado pois, dependendo do refinamento da entrada e do posicionamento do PD, o vértice mais próximo pode ter uma posição ruim para gerar a malha. Se esses casos não forem tratados, o algoritmo de geração da malha possivelmente falhará ou gerará elementos de baixa qualidade.

Para tratar os casos citados, são realizados testes de proximidade que utilizam as informações das células da estrutura de estimativa de carga, a *quadtree*. Com esses testes, a qualidade da malha melhora, além de se evitar possíveis erros na geração da malha.

O teste de proximidade é realizado quando a interface que está sendo criada utiliza uma célula da *quadtree* que intercepta a FE. Se isso ocorrer, é feita uma verificação se a célula da *quadtree*, que está sendo testada é classificada como *sobre* a fronteira do domínio. Em caso afirmativo, é realizada uma busca pelo vértice da fronteira que faz o maior ângulo (Figura 81b, onde, dentre as arestas *a* e *c*, a escolhida é a aresta *a*). Se o comprimento dessa aresta for menor que o lado da célula da estrutura estimativa de carga que está sendo utilizada (*b*, na Figura 81b), é gerada a aresta que liga os dois vértices (Figura 81c). Caso contrário, será criado um novo vértice segundo as informações da grade de suporte. Esse teste do tamanho evita que sejam criados elementos desproporcionalmente grandes ligando a interface com a FE.

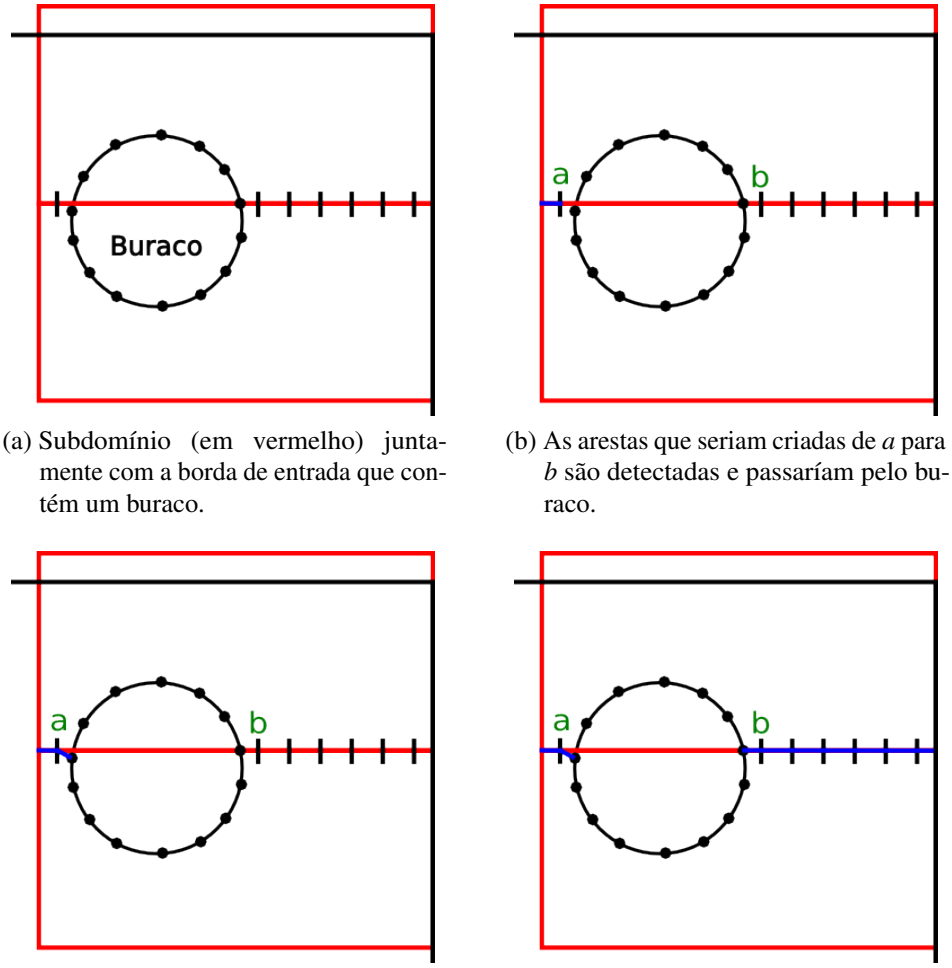
Esse teste é baseado na ideia de que, para um bom elemento ser gerado, ele precisa de uma área mínima. Nesse trabalho, é assumido que essa distância mínima deve ser o tamanho da célula da *quadtree* de estimativa de carga, para o caso bidimensional. Com esse teste, evita-se a criação de arestas com ângulos muito pequenos (Figura 81d) e melhora a qualidade dos elementos que serão gerados.

4.3.3.2 Tratamento de Buracos

Alguns modelos possuem buracos em seus domínios e, nesses casos, é preciso realizar um teste simples para evitar que sejam criadas arestas que atravessem a fronteira. Esse tratamento é realizado apenas quando o PD intercepta algum buraco do modelo.

Quando se tem um PD fazendo interseção com um buraco, existirão células classificadas como fora da partição, entre as que estão classificadas como dentro ou que cruzam a FE. O algoritmo tentaria gerar uma aresta entre duas células classificadas como internas, mas essa aresta cruzaria células classificadas como externas ao domínio.

Quando é detectada uma possível colisão de uma aresta com células externas, são criadas duas arestas em vez de uma. A primeira conecta o último vértice criado ao vértice mais próximo da primeira interseção do buraco (Figura 82c), e a segunda liga o vértice mais próximo da segunda interseção com o buraco a um vértice que será criado de acordo com a grade de suporte (Figura 82d). Essas conexões são feitas utilizado os mesmos testes de proximidade descritos na Seção 4.3.3.1.



(a) Subdomínio (em vermelho) juntamente com a borda de entrada que contém um buraco.

(b) As arestas que seriam criadas de *a* para *b* são detectadas e passaríam pelo buraco.

(c) Teste feito para selecionar a melhor aresta para *a*.

(d) Teste feito para selecionar a melhor aresta para *b*.

Figura 82 – Passos feitos nos tratamentos de buracos.
Fonte: elaborado pelo autor (2019).

4.3.4 Geração da Malha de Interface - Caso Tridimensional

A interface tridimensional corresponde a uma malha de superfície, cuja fronteira corresponde aos elementos de interseção descobertos na Seção 4.3.2. Outra vantagem do modo proposto é que qualquer técnica de geração de malha de superfície, que possa gerar elementos usando uma malha ou grade de suporte como função de tamanho, pode ser aplicada. Nesse trabalho, a malha de interface é gerada pelo algoritmo descrito em (MIRANDA *et al.*, 2009), cuja entrada compreende: os segmentos selecionados na Seção 4.3.2; e uma malha de suporte para orientar a criação dos elementos de superfície, conforme descrito na Seção 4.3.1.

A malha da interface é gerada por uma técnica de avanço de fronteira, que é guiada pela malha de suporte fornecida como entrada. Essa malha de suporte ajudará na criação de novos elementos, indicando a posição e o tamanho ideal para os novos vértices e elementos.

O resultado da geração de malha de interface é uma malha triangular bidimensional. Para finalizar o procedimento de geração da malha de interface, todos os vértices dos elementos de interseção, que foram projetados no PD anteriormente, retornam às suas posições originais (a Figura 83 mostra uma visão geral de como o algoritmo funciona), gerando uma malha de interface tridimensional.

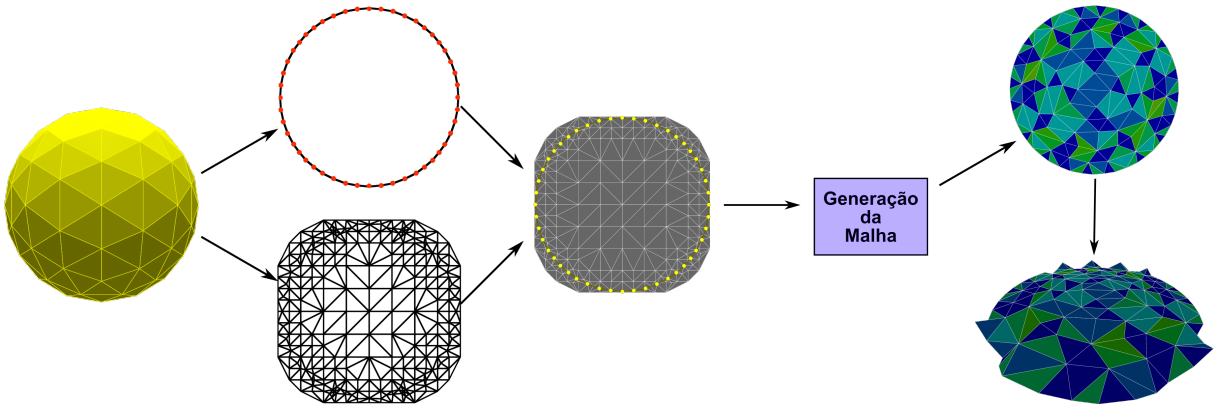


Figura 83 – Dois subdomínios tridimensionais totalmente criados e prontos para a fase de geração de malha.

Fonte: elaborado pelo autor (2019).

Ao final desse procedimento, a malha de interface para cada subdomínio está completamente criada, e cada subdomínio está pronto para gerar sua malha interna. A Figura 84 mostra um modelo tridimensional decomposto em dois subdomínios. Observe que os dois subdomínios são complementares entre si, pois sua união resulta na superfície originalmente dada como entrada, sem furos ou espaços vazios.

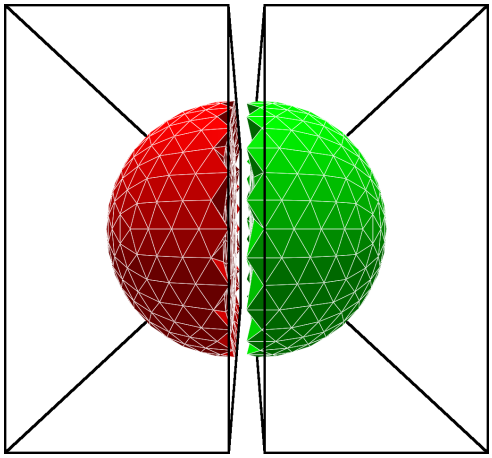


Figura 84 – Dois subdomínios tridimensionais totalmente criados e prontos para a etapa de geração de malha.

Fonte: elaborado pelo autor (2019).

4.4 Balanceamento da Carga

Após todos os subdomínios estarem devidamente criados, eles devem ser distribuídos entre os processadores disponíveis para que as submalhas sejam geradas por alguma técnica de geração de malha. Essa distribuição das tarefas deve ser feita de tal forma que todos os processadores gastem a mesma quantidade de tempo, evitando que algum processador fique sobrecarregado ou ocioso.

De acordo com o modo como o domínio é decomposto, é possível ter como resultado apenas uma tarefa por processador ou várias tarefas por processador. O número de tarefas é o mesmo número de folhas da BSP de decomposição, por isso uma estratégia de balanceamento não-centralizado que utiliza a própria BSP para balancear a carga é utilizada neste trabalho.

Todos os processos geram sua própria versão da BSP. No entanto, cada processo gera apenas a parte da árvore BSP necessária para sua execução, ou seja, somente o caminho da raiz da BSP para seu próprio subdomínio. Assim, os nós internos pertencem a muitos processos ao mesmo tempo, porém os nós folha pertencem apenas a um processo. Para um nó interno, para processos que o geram, o que tem o identificador mais baixo é considerado seu proprietário, e os outros processos têm cópias desse nó interno. A Figura 85 mostra a BSP resultante para a decomposição do modelo mostrado na Figura 71, juntamente com os proprietários de seus nós,

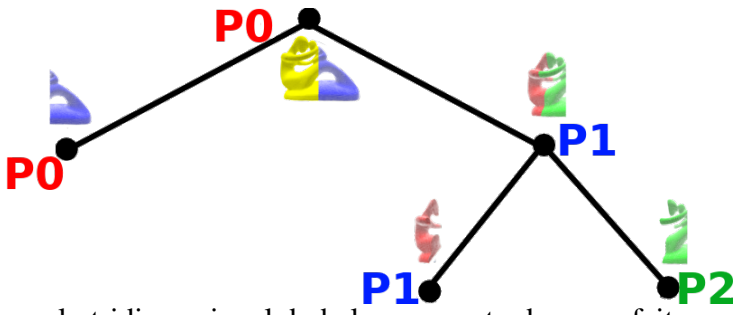


Figura 85 – Um exemplo tridimensional do balanceamento de carga feito para três subdomínios da Figura 71.

Fonte: elaborado pelo autor (2019).

Se desejado, a BSP pode continuar sendo subdividida por um único processo, fazendo uso de paralelismo de memória compartilhada. Com esse recurso, é possível gerar um subdomínio por *thread* em um processo. Quando um processo (ou uma *thread* em um processo) atinge um nó folha da árvore BSP, ou seja, em seu próprio subdomínio, ele gera sua malha interna.

4.5 Geração da Malha

Cada processador tem a possibilidade de aplicar algoritmos diferentes, se desejado, podendo ser aplicada qualquer técnica de geração de malha que respeite os pré-requisitos citados no Capítulo 1. Foi selecionada uma técnica de Avanço de Fronteira para realizar os testes.

O algoritmo de Avanço de Fronteira utilizado foi desenvolvido em Miranda *et al.* (1999) e Cavalcante-Neto *et al.* (2001). Foram utilizadas estruturas de dados geométricas para acelerar a busca por vértices candidatos para a geração de um novo elementos e para a busca de possíveis arestas intersectantes, garantindo uma rápida execução do procedimento de geração de malha.

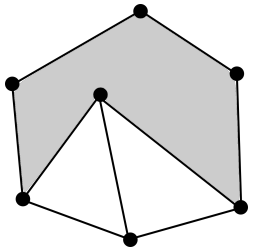
O processo de geração de malha consiste em três fases. A primeira é baseada na geometria, tentando gerar elementos de boa qualidade mantendo a validade da malha. A segunda é baseada na topologia, buscando apenas gerar elementos válidos, mesmo que não sejam de boa qualidade. A última fase é a de geração por retrocesso (*back-tracking*), que tenta finalizar a geração da malha nas áreas onde não foi possível realizar o avanço de fronteira. Mais detalhes deste procedimento podem ser encontrados em Cavalcante-Neto *et al.* (2001).

Na versão bidimensional da técnica as duas primeiras fases são suficientes para gerar a malha triangular. Na versão tridimensional nem sempre as duas primeiras fases conseguem finalizar a malha, por isso é necessário o passo de geração por retrocesso (*back-tracking*). A Figura 86 ilustra o processo de retrocesso aplicado a uma cavidade. Vale lembrar que, nos casos tridimensionais, nem sempre existe uma solução para a geração da malha.

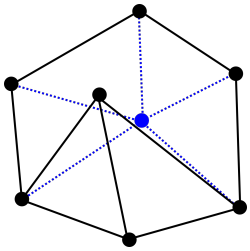
O processo de geração de malha é complementado com algumas melhorias. Ao todo, são três fases de melhorias, duas delas são modificações na topologia e a outra, na geometria. As fases topológicas são a troca de faces (*face swapping*) e a troca de arestas (*edge swapping*), representadas nas Figuras 87 e 88 respectivamente.

Na fase geométrica é aplicada uma suavização de Laplace, onde os vértices internos da malha são movidos caso os elementos suavizados sejam melhores que o pior elemento antes da suavização. Esta suavização consiste em mover um vértice interno da malha de tal forma que ele se aproxime do centroide do polígono definido pelos seus vértices adjacentes, seguindo a Equação 4.1.

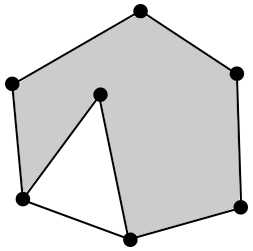
$$X_V^{n+1} = X_V^n + \phi \frac{\sum_{i=1}^{m} (X_i^n - X_V^n)}{m} \tag{4.1}$$



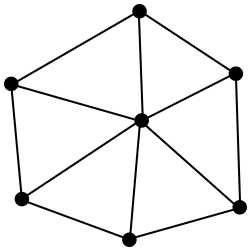
(a) Cavidade em cinza.



(b) Teste de visibilidade, sendo feito com o centroide (em azul) e suas arestas de teste.

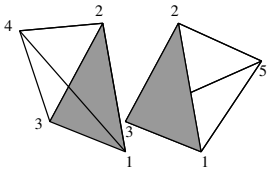


(c) Cavidade após remoção de um elemento.

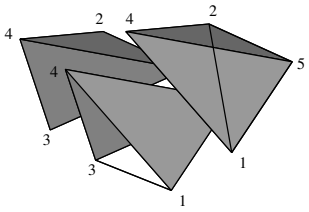


(d) Malha final, ligada ao centroide, após o retrocesso ser aplicado.

Figura 86 – Geração de malhas por retrocesso.
Fonte: Freitas *et al.* (2013).

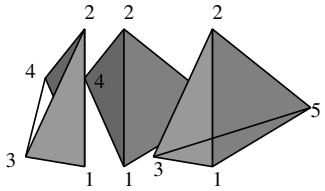


(a) Tetraedros adjacentes à face (1, 2, 3).

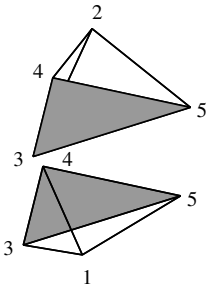


(b) Tetraedros após a troca de faces ser realizada.

Figura 87 – Troca de faces.
Fonte: Freitas *et al.* (2013).



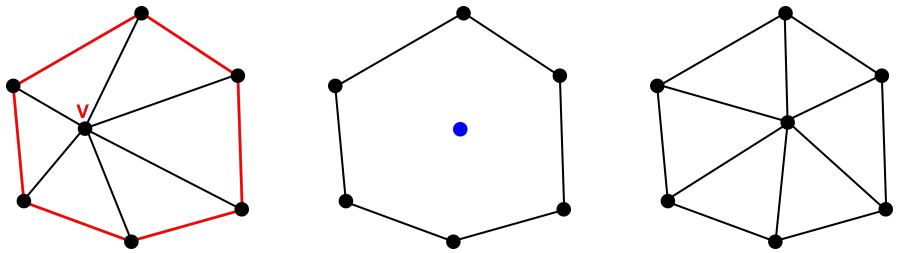
(a) Tetraedros adjacentes à aresta (1, 2).



(b) Tetraedros após a troca.

Figura 88 – Troca de arestas, com três tetraedros adjacentes (caso inverso à troca de faces).
Fonte: Freitas *et al.* (2013).

Nessa equação, X_V^n é a posição do vértice V na iteração de suavização n, m é o número de vértices adjacentes a V por algum tetraedro. i corresponde o i-ésimo vértice adjacente a V, X_V^n é a posição do i-ésimo vértice adjacente a V na iteração de suavização n, e ϕ é o parâmetro de relaxamento, normalmente ajustado para um valor entre 0 e 1 (neste trabalho, o valor de 1 foi utilizado para ϕ). A Figura 89 ilustra como a suavização é feita para $\phi = 1,0$.



(a) Polígono formado pelos vértices adjacentes a V (em vermelho).

(b) Centroide do polígono em azul.

(c) Resultado final da suavização de Laplace.

Figura 89 – Suavização de Laplace feita para $\phi = 1,0$.
Fonte: Freitas *et al.* (2013).

4.6 Junção e Finalização da Malha

A junção da malha é realizada seguindo a estrutura da BSP de decomposição de baixo para cima. Cada nó da árvore da BSP corresponde a um processador. A junção das malhas é realizada sempre que dois nós irmãos acabam a fase de geração de malha, um dos processadores ficará responsável pelo processamento e o outro será liberado. A alocação das tarefas foram previamente feitas, conforme explicado na Seção 4.4. A Figura 90 mostra a árvore BSP para 8 subdomínios, cada subdomínio é atribuído a um processador.

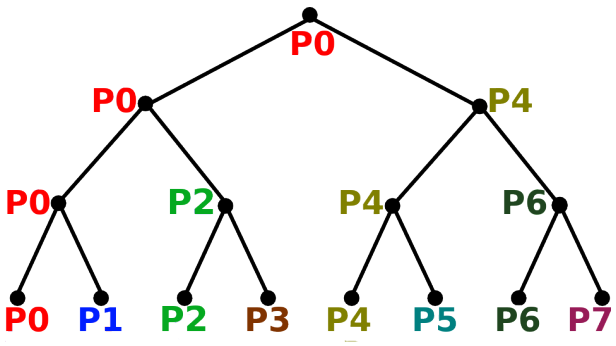
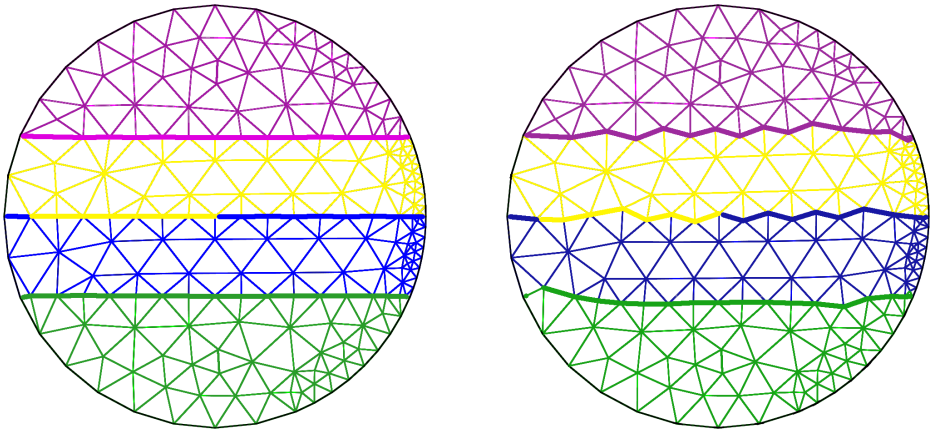


Figura 90 – Árvore da BSP de decomposição usada na junção dos subdomínios. Em cada nó é mostrado o processador responsável pela geração da malha ou da sua junção e melhoria.
Fonte: elaborado pelo autor (2019).

A finalização da malha é realizada logo após a junção dos subdomínios e consiste em melhorias como as descritas na Seção 4.5. As melhorias são aplicadas nas interfaces dos subdomínios juntamente com certas camadas de elementos adjacentes a elas. A Figura 91 mostra um mesmo modelo bidimensional com e sem as melhorias nas interfaces.



(a) Junção das malhas geradas nos diversos processadores.

(b) Malha resultante depois do último refinamento.

Figura 91 – Finalização da malha: junção das malhas geradas pelos processadores (à esquerda) e malha refinada final (à direita).

Fonte: elaborado pelo autor (2019).

A melhoria da malha é feita na região onde a malha de interface foi construída, juntamente com algumas camadas de elementos adjacentes. Essas camadas são os elementos adjacentes à malhas de interface. Foi verificado em (ITO *et al.*, 2007) que duas camadas de elementos são suficientes para se obter uma boa malha ao final. A camada 0 consiste dos próprios segmentos ou faces da malha de interface, a camada 1 consiste nos elementos adjacentes aos segmentos ou faces da malha de interface. Assim, a camada N compreende os elementos presentes na camada $N-1$ mais seus elementos adjacentes. Depois dessa etapa, a malha está completamente gerada em todo o modelo.

4.7 Considerações

Esse capítulo apresentou uma técnica *a priori* para decomposição de modelos para geração em paralelo de malhas. Todas as etapas necessárias para o entendimento da técnica foram descritos e ilustrados. Este trabalho foi desenvolvido para aceitar entradas bidimensionais ou tridimensionais, podendo-se ainda utilizar qualquer algoritmo para a geração da malha em paralelo.

Usando uma decomposição baseada nos eixos e aplicando testes de validação, o PD criado é aquele que melhor divide a carga entre as partes geradas. Além disso, o modelo de criação das interfaces *a priori* que gera subdomínios totalmente independentes possibilita assim a utilização de diferentes técnicas de geração de malha com uma baixa comunicação entre as tarefas.

Um ponto bastante importante deste trabalho é que a criação da malha de interface leva em consideração a discretização do modelo de entrada. Dessa forma, a malha gerada em paralelo deve tem as mesmas proporções que a malha gerada sequencialmente, não gerando assim refinamentos artificiais na tentativa de elevar a qualidade da malha.

5 EXEMPLOS E RESULTADOS

Este capítulo apresenta os resultados obtidos pela técnica proposta nesse trabalho. Entre os resultados mostrados, constam a qualidade da malha, a diferença entre os elementos gerados sequencialmente e em paralelo, o tempo de execução e o *speed-up* da implementação.

Como dito anteriormente, a técnica *a priori* descrita nesse trabalho é uma evolução do trabalho de Freitas *et al.* (2016), que é uma técnica *a posteriori*. Assim, serão apresentadas, também, algumas comparações entre as duas técnicas, a fim de mostras as vantagens e as desvantagens de cada uma.

Para uma comparação justa entre os resultados, foi utilizado o mesmo algoritmo de geração de malha (Seção 4.5) em todos os casos de teste.

A técnica descrita neste trabalho foi implementada em C++ (The C++ Standards Committee, 2015), utilizando a biblioteca de passagem de mensagens MPI (*Message Passing Interface*) (MPI Forum, 2019) para paralelismo de memória distribuída. As versões com interface foram ainda implementadas com a biblioteca de renderização OpenGL (GRAPHICS; OPENGL, 1998) (com suas bibliotecas glu, glut (GLUT; JURCZYK, 2008) e glew (IKITS; MAGALLON, 2015)), e a biblioteca de interfaces wxWidgets (wxTeam, 2019).

O computador utilizado para rodar os testes da implementação foi o Lobo Carneiro (LoboC) do Núcleo Avançado em Computação de Alto Desempenho (NACAD) da Coppe/UFRJ, que possui 252 nós, cada um configurado com um Intel Xeon E5-2670v3 (Haswell) e 64 GB de RAM, totalizando 6048 núcleos de processamento e 16 TB de RAM. Os testes foram feitos utilizando até 16 nós com memória distribuída, para a versão de memória distribuída.

5.1 Modelos

Ao todo, foram executados cinco modelos, um de geometria simples (Beam) e os outros quatro com geometrias complexas, para avaliar o comportamento da técnica em diferentes entradas. A Figura 92 mostra os modelos utilizados pelo procedimento de geração em paralelo de malhas. Na Figura 93, são apresentadas as decomposições dos 5 modelos de teste para 16 subdomínios.

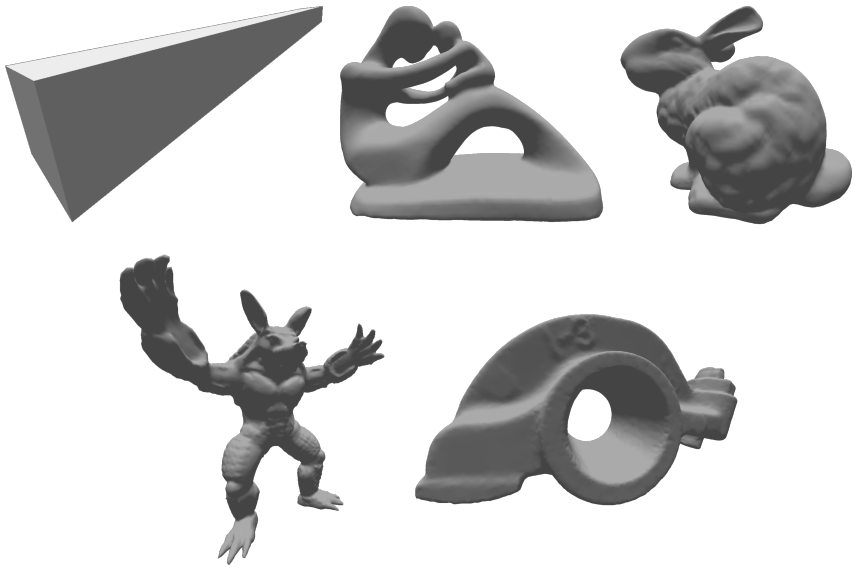


Figura 92 – Modelos utilizados para testes: Beam, Fertility, BunnyBotsch, Armadillo e RockerArm.

Fonte: elaborado pelo autor (2019).

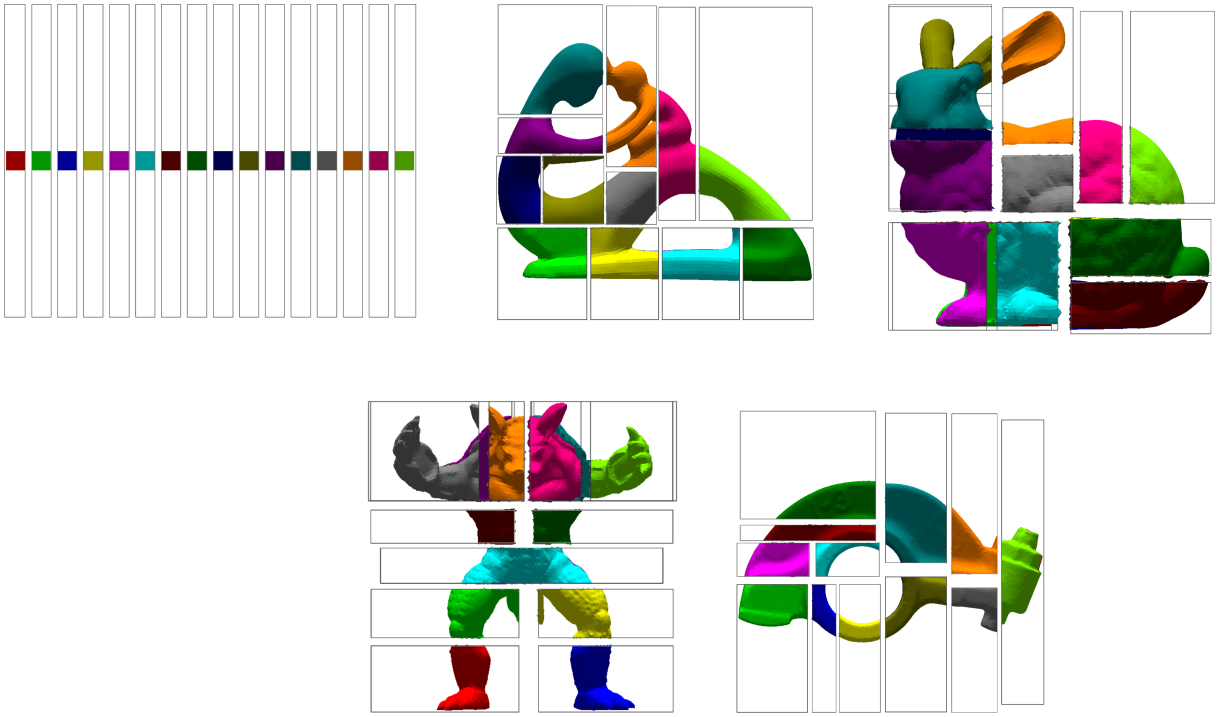


Figura 93 – Visão frontal da decomposição para 16 subdomínios dos modelos de testes: Beam, Fertility, BunnyBotsch, Armadillo e RockerArm. Cada cor representa um processo diferente que será responsável pela geração da malha volumétrica do subdomínio.

Fonte: elaborado pelo autor (2019).

5.2 Tamanho das Malhas

A Figura 94 mostra os tamanhos das malhas geradas sequencialmente e em paralelo (para 2, 4, 8 e 16 processos) pela técnica proposta. Como pode ser visto, os tamanhos das malhas

variam aproximadamente de 12 milhões a 22 milhões de elementos. Os tamanhos das malhas e a quantidade de processadores utilizados são razoáveis para o tamanhos das malhas e para os teste aqui realizados, ilustrando bem o comportamento da técnica nesses modelos.

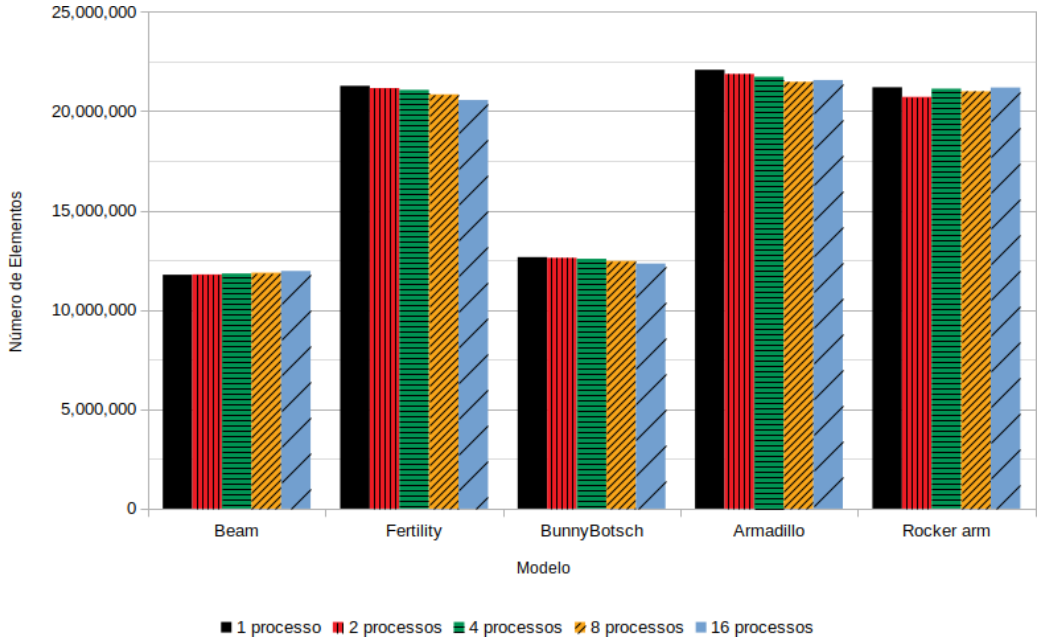


Figura 94 – Quantidade de elementos gerados nos modelos.
Fonte: elaborado pelo autor (2019).

Um ponto importante dessa técnica é que os tamanhos das malhas geradas em paralelo estão no mesmo intervalo de valores ligeiramente inferior que suas malhas geradas sequencialmente. Como pode ser visto, nenhum refinamento excessivo é aplicado nas interfaces, na malha ou na decomposição do domínio como uma tentativa de melhorar a qualidade da malha artificialmente.

5.3 Qualidade

Para avaliar a qualidade das malhas geradas é utilizada a métrica $\alpha = 3R_i/R_c$, onde R_i e R_c são os raios das esferas inscrita e circunscrita, respectivamente (LEWIS *et al.*, 1996; PARTHASARATHY *et al.*, 1993). Esta métrica α tem valor 1,0 para um tetraedro equilátero. Quanto pior for a qualidade de um elemento, mais próximo de 0,0 é o valor de α. Um elemento com $\alpha \leq 0{,}1$ é dito ter uma péssima qualidade, enquanto que elementos com $\alpha \geq 0{,}7$ têm boa qualidade.

A qualidade da malha de cada um dos cinco modelos é mostrada na Figura 95. Nessa figura, a qualidade dos elementos gerados é agrupada de 0,1 em 0,1. Pode-se ver, pelos gráficos

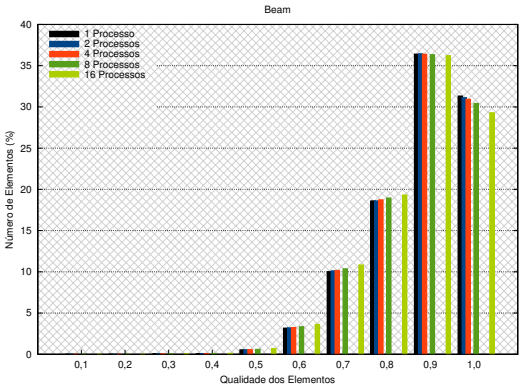
à esquerda, que a quantidade de elementos de baixa qualidade, com $\alpha \leq 0{,}4$, é bem pequena, quando comparada com a quantidade de elementos de boa qualidade, com $\alpha \geq 0{,}7$. É possível ver que em torno de 80% dos elementos gerados se encontram na faixa de valor entre 0,8 e 1,0 (máxima qualidade) e que a quantidade de elementos nessa faixa é praticamente a mesma das malhas geradas de forma sequencial. No geral, esses gráficos mostram que, para a técnica proposta com decomposição por BSP, a malha resultante é bastante boa.

Os gráficos de diferença de qualidade ajudam a comparar as malhas geradas em paralelo com as sequenciais. Nesses gráficos, barras acima de zero indicam que mais elementos naquela faixa de valor da métrica de qualidade foram gerados na versão paralela do que na sequencial; barras abaixo de zero indicam que existem mais elementos gerados sequencialmente naquela determinada faixa de valor do que na versão paralela. Algo interessante de se observar é que à medida que mais processos participam da geração da malha, mais essa diferença aumenta. Isso ocorre porque quando se adiciona um novo processo na geração da malha, se adiciona também um novo subdomínio para esse processo. Quanto mais subdivisões forem feitas, mais a malha final vai divergir da malha sequencial.

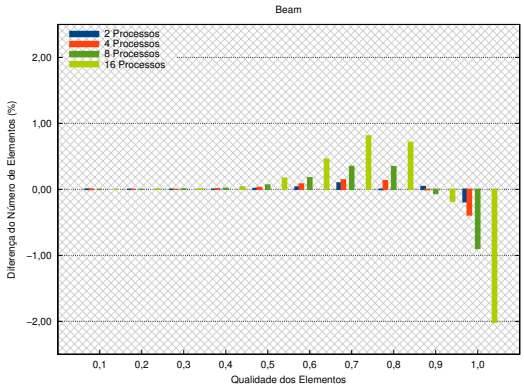
Os gráficos à direita da Figura 95 mostram a diferença, em percentual, entre a qualidade da malha gerada sequencialmente e a qualidade das malhas geradas em paralelo. É possível ver que houve um crescimento de até 1,0% na quantidade de elementos gerados em paralelo e classificados na faixa 0,5 a 0,8 e um decaimento de menos de 2,5% na faixa de 0,9 a 1,0. Isso pode ser entendido como se, alguns elementos de alta qualidade, com α entre 0,9 e 1,0, reduzissem sua qualidade e passassem a ser de qualidade mediana a boa, com alfa de 0,5 a 0,8. Como observado na figura, a variação de qualidade para melhor ou para pior não ultrapassa de 2,5% da quantidade de elementos da malha.

Em técnicas de particionamento *a priori*, é esperado que a inserção artificial da malha interface leve a uma diminuição considerável na qualidade da malha, em especial, em elementos próximos à interface. Isso não é observado em técnicas *a posteriori*, uma vez que a malha de interface é gerada pelo próprio gerador de malha volumétrico, o que torna a malha mais natural nessas regiões.

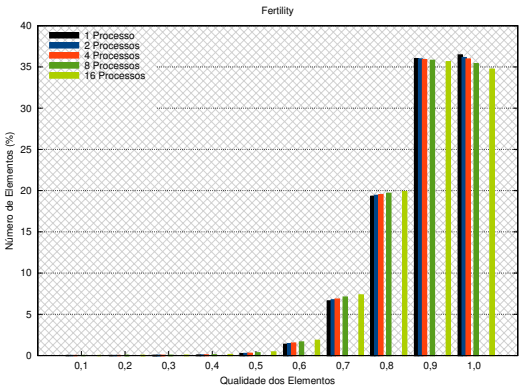
Para contornar esse problema, a técnica descrita nesse trabalho tem a precaução de somente gerar uma malha de interface em regiões que tenham bons ângulos entre o plano de decomposição e as faces do modelo, e o cuidado de realizar uma melhoria na malha ao redor da malha de interface. As malhas de interfaces geradas neste trabalho refletem o tamanho dos



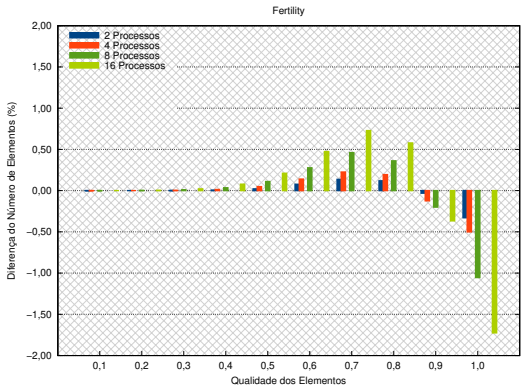
(a) Qualidade das malhas do modelo Beam.



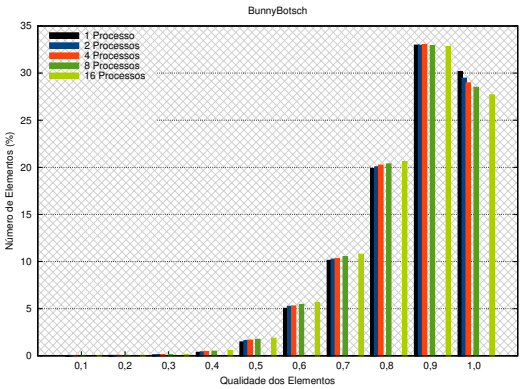
(b) Diferença na qualidade das malhas do modelo Beam.



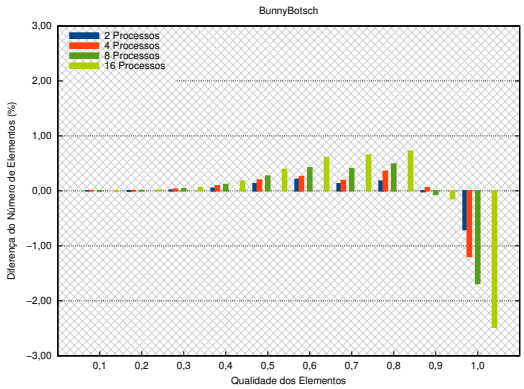
(c) Qualidade das malhas do modelo Fertility.



(d) Diferença na qualidade das malhas do modelo Fertility.



(e) Qualidade das malhas do modelo BunnyBotsch.

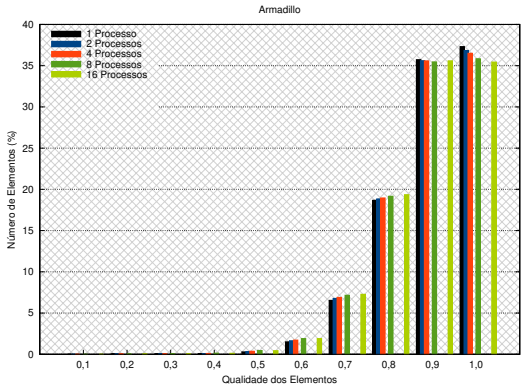


(f) Diferença na qualidade das malhas do modelo Bunny-Botsch.

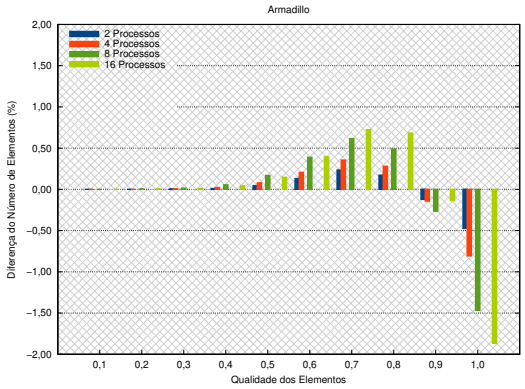
Figura 95 – Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença, em porcentagem, da qualidade das malhas geradas (coluna da direita).

Fonte: elaborado pelo autor (2019).

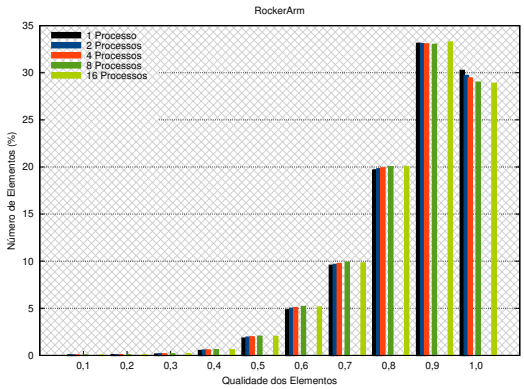
elementos internos dos modelos, não sendo aplicado nenhum tipo de refinamento, a fim de melhorar a qualidade da malha interna. Por isso, a malha gerada em paralelo tem uma boa



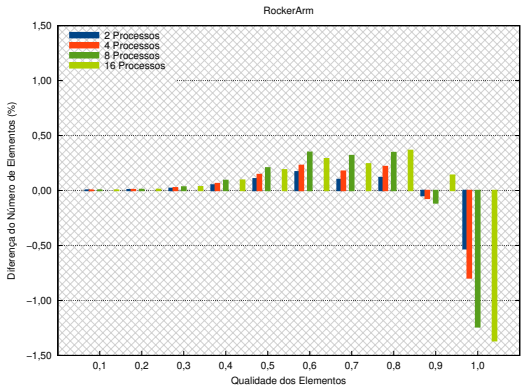
(g) Qualidade das malhas do modelo Armadillo.



(h) Diferença na qualidade das malhas do modelo Armadillo.



(i) Qualidade das malhas do modelo RockerArm.



(j) Diferença na qualidade das malhas do modelo RockerArm.

Figura 95 – Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença, em porcentagem, da qualidade das malhas geradas (coluna da direita) (continuação).

Fonte: elaborado pelo autor (2019).

qualidade que se aproxima da malha gerada sequencialmente.

Como pode ser visto nos gráficos desta seção, em todos os casos, as malhas geradas apresentam boa qualidade. Somente uma pequena porcentagem de elementos (nos piores casos essa porcentagem fica próxima de 20%) não estão na faixa de qualidade de 0,8 a 1,0.

A técnica paralela descrita neste trabalho gera uma malha de qualidade aproximadamente igual à da malha gerada sequencialmente. Nota-se que um número maior de subdomínios criados geralmente leva à diminuição da qualidade da malha na faixa entre 0,9 e 1,0, e ao aumento de elementos na faixa entre 0,5 e 0,8. A Figura 96 mostra a diferença total entre as malhas sequencial e paralela. A diferença total é calculada como o somatório dos valores percentuais absolutos da faixa de diferença de qualidade 0,0 a 1,0. No pior caso, utilizando 16 processos, a malha do modelo BunnyBotsch difere cerca de 5,25% da malha gerada em sequencial.

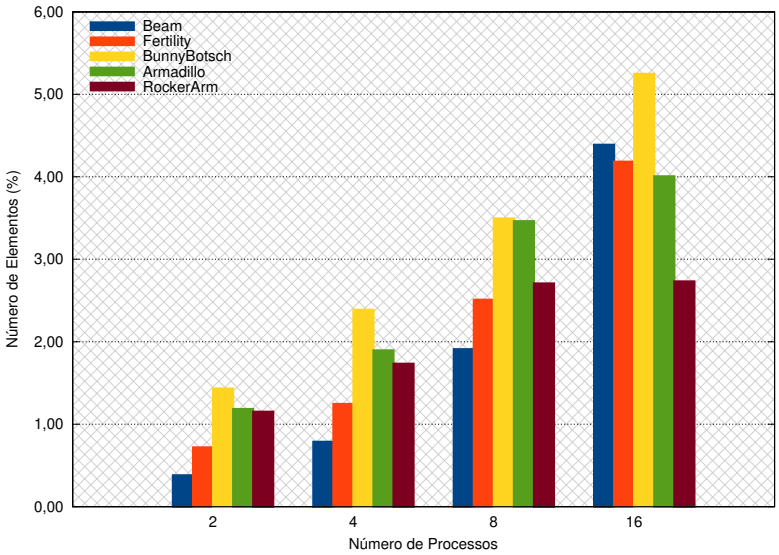


Figura 96 – Diferença total na qualidade das malhas quando comparada com as sequenciais.
Fonte: elaborado pelo autor (2019).

5.4 Tempo de execução e *speed-up*

A Figura 97 mostra o *speed-up* atingido pela implementação da técnica proposta. Do lado esquerdo da figura, no eixo Y, é mostrado o valor do *speed-up*. No eixo X é mostrada a quantidade de processos utilizados na geração da malha. Como dito anteriormente na Seção 2.5.5, o *speed-up* indica quantas vezes a implementação paralela foi mais rápida que a sequencial quando uma determinada quantidade de processos foi utilizada. Idealmente, uma implementação paralela teria um *speed-up* linear, significando que n processos a tornam n vezes mais rápida para uma mesma carga computacional. Na prática, isso é difícil de ser obtido, por causa das inevitáveis porções sequenciais presentes no algoritmo paralelo, além do tempo de comunicação entre os processos, que não existe no caso sequencial.

Como pode-se ver no gráfico de *speed-up*, é alcançado um *speed-up* superlinear para todos os testes feitos. Um *speed-up* superlinear significa que o ganho da implementação paralela com P processos foi maior do que P.

Na Figura 98, é mostrado o tempo de execução obtido pela implementação da técnica proposta. Do lado esquerdo do gráfico, no eixo Y, é mostrado o tempo de execução em segundos.

Pelo gráfico de tempo de execução é possível ver que o tempo de geração de malha do modelo RockerArm reduziu de 5,2 horas para apenas 13,5 minutos utilizando apenas 16 processos. Outra constatação observada a partir do gráfico é que, inicialmente, as curvas são

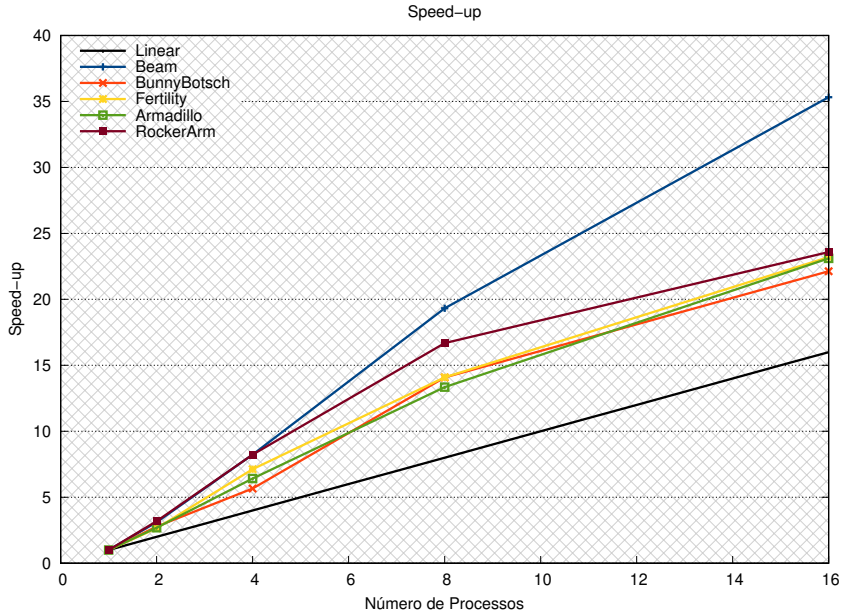


Figura 97 – *Speed-up* para todos os modelos de testes.
Fonte: elaborado pelo autor (2019).

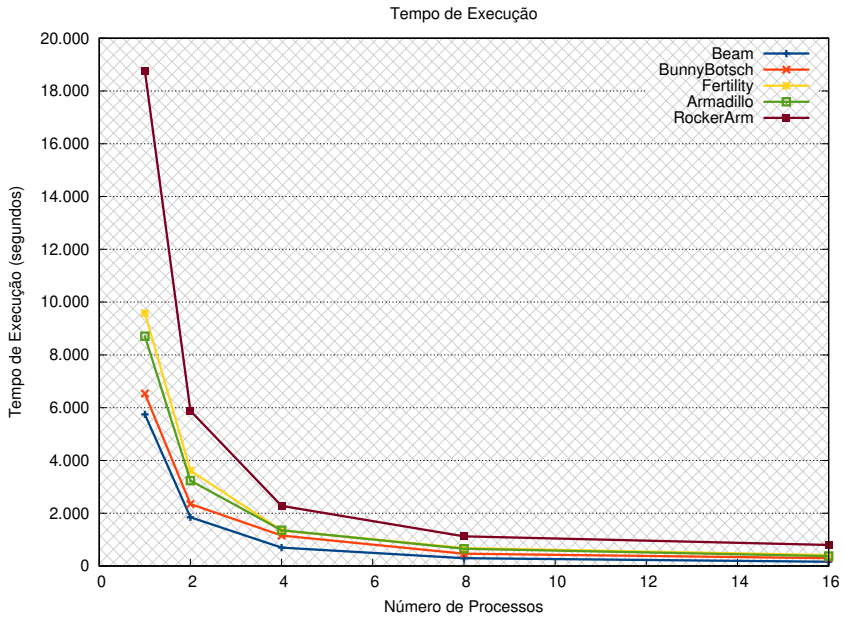


Figura 98 – Tempo de execução para todos os modelos de testes.
Fonte: elaborado pelo autor (2019).

bem acentuadas mas, com o aumento no número de processos, as inclinações diminuem, fazendo com que as curvas fiquem praticamente horizontais no final. Isso indica que não compensa mais aumentar o número de processos pois, a partir desse ponto, a curva do gráfico de tempo de execução mostrará pequenas reduções ou prováveis aumentos no tempo de execução. Como dito

anteriormente, para o tamanho dos modelos escolhidos, 16 processos são o ideal para se mostrar os ganhos de velocidade.

O *speed-up* superlinear pode ser justificado da seguinte maneira. O algoritmo utilizado para gerar a malha, descrito em Cavalcante-Neto *et al.* (2001), tem complexidade assintótica de $O(N^p logN)$, onde N é o tamanho da malha gerada e p é um número um pouco maior que 1.

Quando utiliza-se 2 processos, por exemplo, e assumindo que o balanceamento foi bem feito e que, portanto, cada processo gera metade da malha, o tempo de execução total é aproximadamente $(N/2)^p log(N/2)$. Esse tempo de execução, que é menor que a $(N^p logN)/2$, pois $(N/2)^p < N^p/2$ e $log(N/2) < logN$. Assim, com dois processos, o tempo de execução cai para menos da metade do tempo da implementação sequencial.

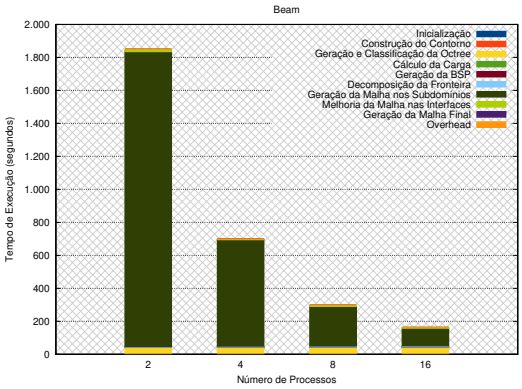
Esse ganho é ainda mais amplificado quando aumenta-se o número de processos. Portanto, um comportamento super-linear é esperado, desde que o tempo de processamento sequencial inerente ao algoritmo paralelo não se sobressaia a esse ganho.

5.5 Detalhamento do tempo de execução

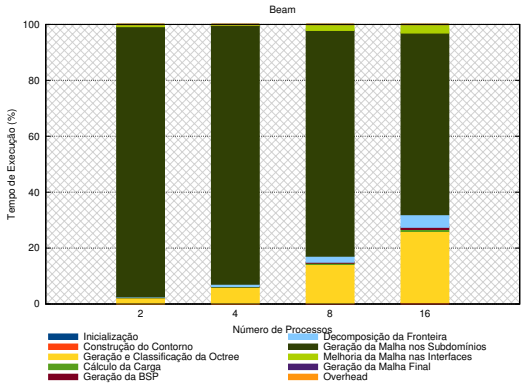
As duas colunas da Figura 99 mostram o detalhamento do tempo de execução do processo mestre, em termos absolutos e relativos, respectivamente, para os modelos de testes apresentados nesta seção. Nos gráficos, constam todos os passos envolvidos na técnica proposta, que são: inicialização, construção das estruturas de dados do contorno, geração e classificação da *octree*, cálculo da estimativa de carga, geração da BSP, decomposição da fronteira, geração de malha nos subdomínios, melhoria da malha nas interfaces e *overhead*, que contabiliza o tempo extra gasto fazendo trabalho improdutivo.

Utilizando apenas dois processos, o tempo total gasto na fase de geração da malha consome do tempo total cerca de 90%. À medida que mais processos são adicionados, esse percentual vai reduzindo. Utilizando 16 processos, o tempo de geração da malha consome, em média, de 40% do tempo total, enquanto o tempo para criar as estruturas de dados e de decomposição chega a 50%. Esse fato mostra que, para os modelos utilizados, a criação de 16 subdomínios é o máximo que se pode criar para que o tempo de geração da malha paralela não seja menor que o tempo para gerar as estruturas de dados e realizar as decomposições. Isso é um dos fatores que fazem essa técnica não precisar realizar *over-decomposition*.

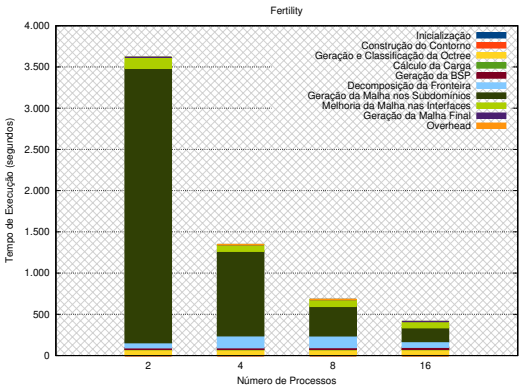
É esperado que o tempo de decomposição da fronteira, de melhoria das malhas



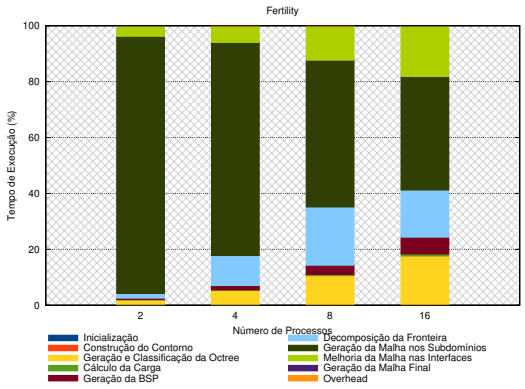
(a) Tempo absoluto do modelo Beam.



(b) Tempo em porcentagem do modelo Beam.



(c) Tempo absoluto do modelo Fertility.



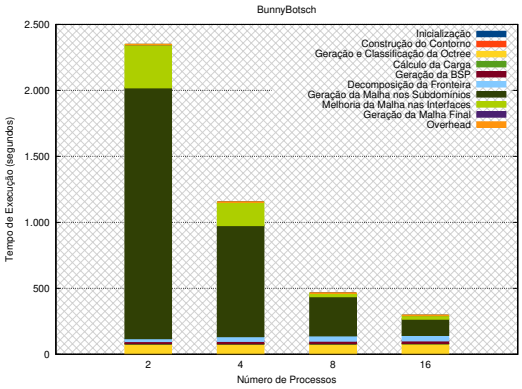
(d) Tempo em porcentagem do modelo Fertility.

Figura 99 – Detalhamento do tempo de execução absoluto (coluna da esquerda) e em porcentagem (coluna da direita) para todos os modelos de teste.

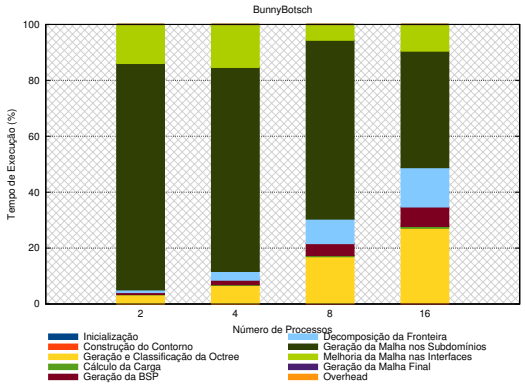
Fonte: elaborado pelo autor (2019).

nas interfaces e de *overhead* aumentem com o número de processos, enquanto que o tempo de geração de malha nos subdomínios diminua com o aumento do número de processos.

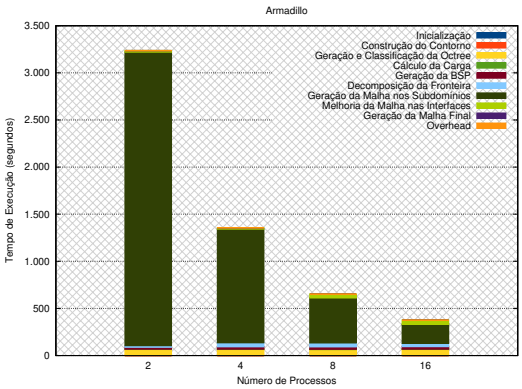
Assim, conforme aumenta-se o número de processos, os passos que têm tempo absoluto constante acabam ganhando mais importância, relativamente ao tempo total de execução.



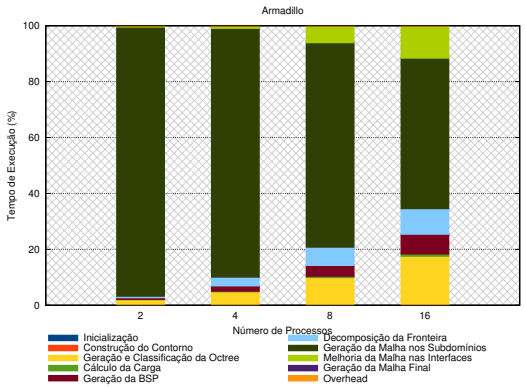
(e) Tempo absoluto do modelo BunnyBotsch.



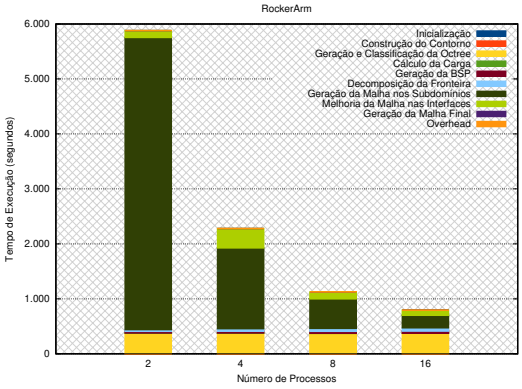
(f) Tempo em porcentagem do modelo BunnyBotsch.



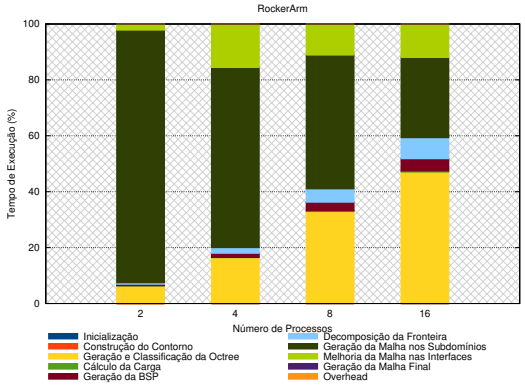
(g) Tempo absoluto do modelo Armadillo.



(h) Tempo em porcentagem do modelo Armadillo.



(i) Tempo absoluto do modelo RockerArm.



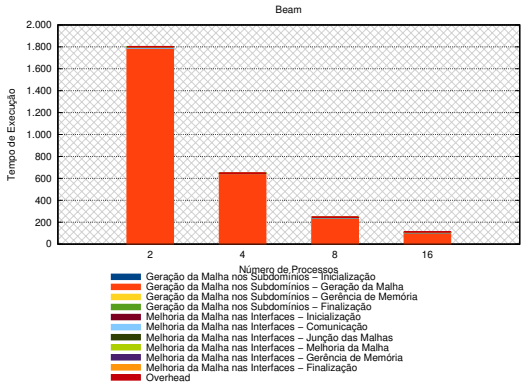
(j) Tempo em porcentagem do modelo RockerArm.

Figura 99 – Detalhamento do tempo de execução absoluto (coluna da esquerda) e em porcentagem (coluna da direita) para todos os modelos de teste. (continuação).

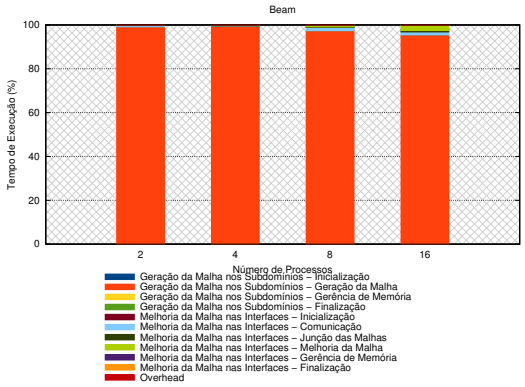
Fonte: elaborado pelo autor (2019).

A Figura 100 detalha o tempo de execução dos procedimentos de geração de malha, em termos absolutos (à esquerda) e relativos (à direita). Nos gráficos referentes ao modelo de teste Beam, as fases de Comunicação, Junção das Malhas e Melhoria da Malha nas Interfaces não chegam a utilizar 10% do tempo total quando utilizados 16 processos, enquanto outros modelos,

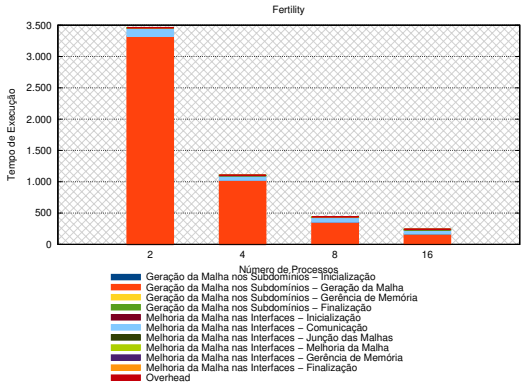
como o Fertility e o RockerArm, utilizam cerca de 30%. A explicação para esse fato é que a geometria do modelo Beam é simples, bem como a sua decomposição, e esse modelo é uniforme, com carga igualmente distribuída. Isso faz com que praticamente não exista desbalanceamento de carga entre os processos. Já para os modelos Fertility e RockerArm, a geometria complexa se reflete na dificuldade de realizar uma decomposição que crie subdomínios bem balanceados, gerando assim altos tempos para comunicação entre os processos, pois para dois subdomínio se comunicarem, eles já devem ter finalizado a etapa de Geração da Malha.



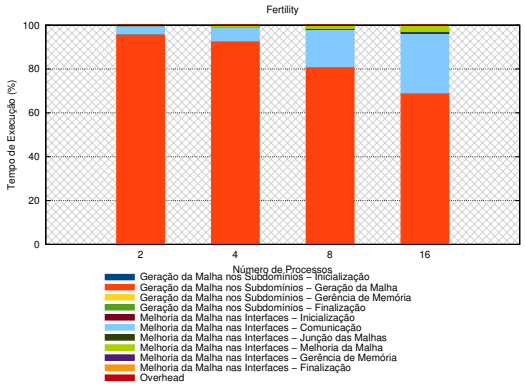
(a) Tempo absoluto do modelo Beam.



(b) Tempo em porcentagem do modelo Beam.



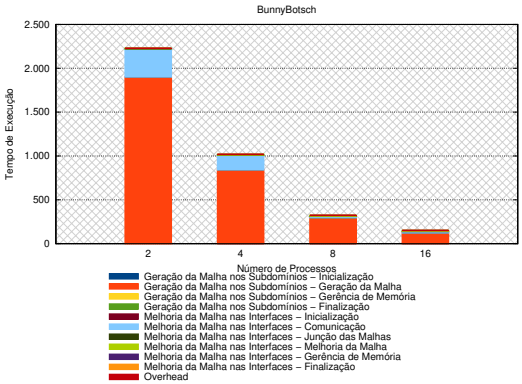
(c) Tempo absoluto do modelo Fertility.



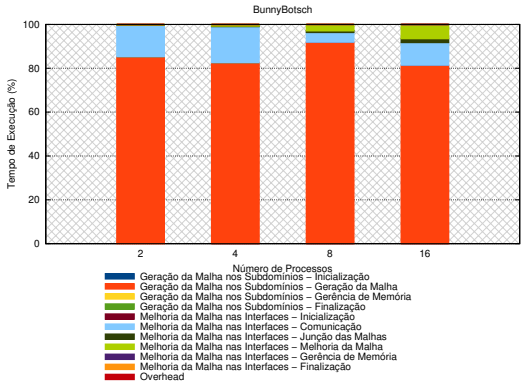
(d) Tempo em porcentagem do modelo Fertility.

Figura 100 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita).

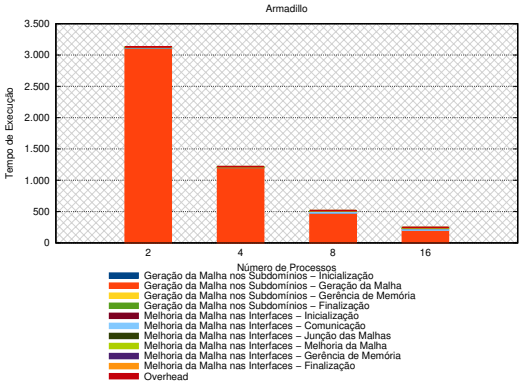
Fonte: elaborado pelo autor (2019).



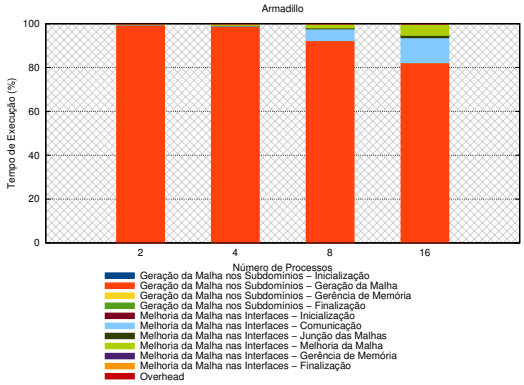
(e) Tempo absoluto do modelo BunnyBotsch.



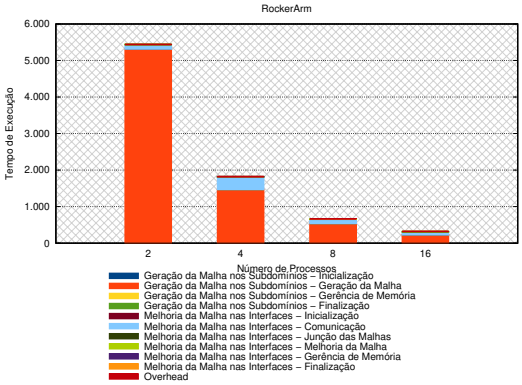
(f) Tempo em porcentagem do modelo BunnyBotsch.



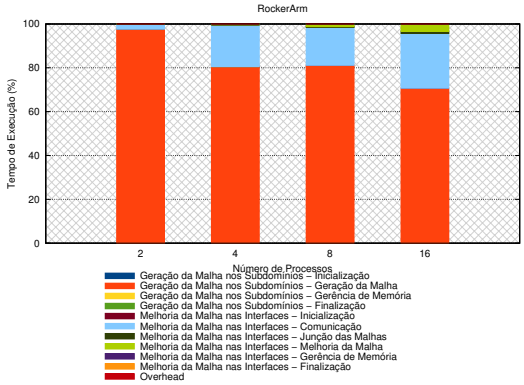
(g) Tempo absoluto do modelo Armadillo.



(h) Tempo em porcentagem do modelo Armadillo.



(i) Tempo absoluto do modelo RockerArm.



(j) Tempo em porcentagem do modelo RockerArm.

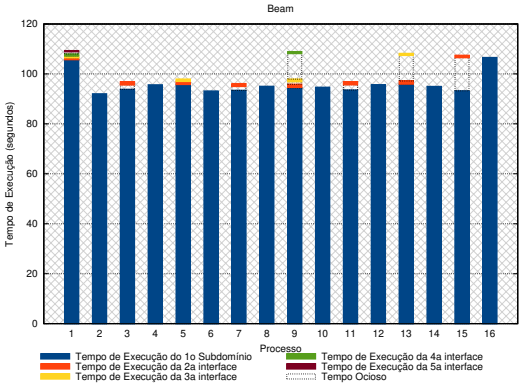
Figura 100 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita) (continuação).

Fonte: elaborado pelo autor (2019).

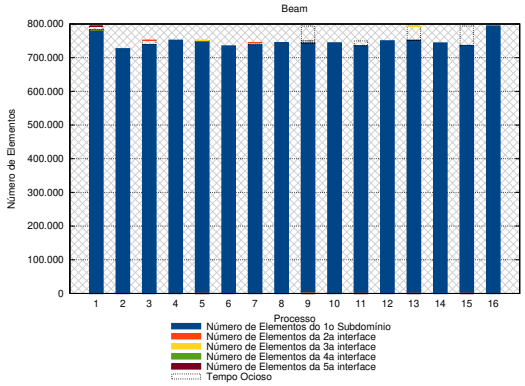
5.6 Balanceamento da carga

Nesta seção, é mostrado o balanceamento da carga, levando em conta o número de elementos gerados em cada subdomínio, que é considerado como carga nesse trabalho. Os

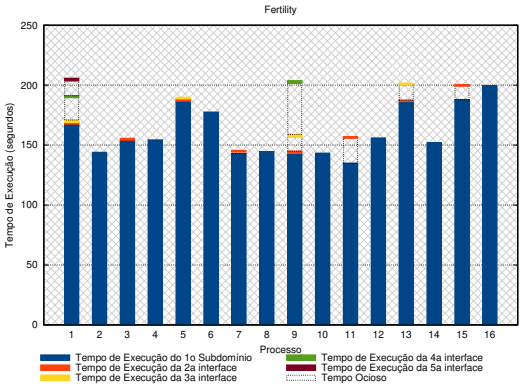
gráficos da Figura 101 mostram que, para os todos os modelos de teste, a técnica proposta tem uma carga bem distribuída entre os processos. A coluna da esquerda mostra o tempo de execução de cada subdomínio em cada processo e a coluna da direita da mesma figura mostra a quantidade de elementos gerados em cada subdomínio e em cada interface (elementos podem ser gerados na fase de melhoria da malha na interface, por conta da etapa de melhoria por retrocesso, como descrito na Seção 4.5). O eixo Y do gráfico representa o tempo de execução ou a quantidade de elementos gerados, enquanto que o eixo X indica os processos utilizados. Cada barra de valores de cada cor diferente representa um certo subdomínio de responsabilidade daquele processo. A primeira barra (cor azul) corresponde ao tempo de execução ou a quantidade de elementos gerados no subdomínio do processo indicado. As outras barras correspondem ao tempo de execução ou à quantidade de elementos gerados durante a melhoria da malha nas interfaces. O tempo em que um processo está em espera é indicado por uma barra transparente.



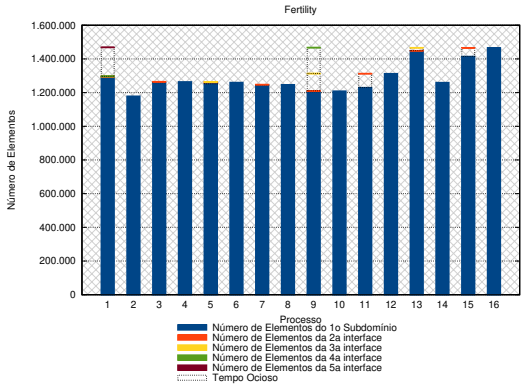
(a) Tempo de execução do modelo Beam.



(b) Número de elementos do modelo Beam.



(c) Tempo de execução do modelo Fertility.

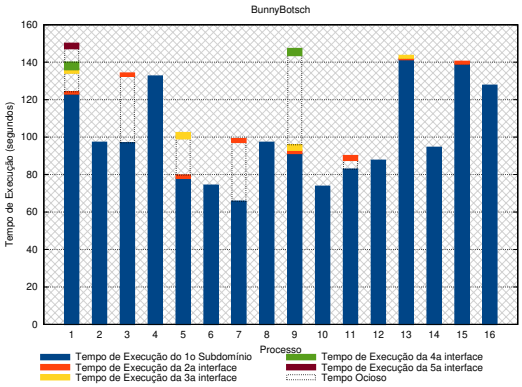


(d) Número de elementos do modelo Fertility.

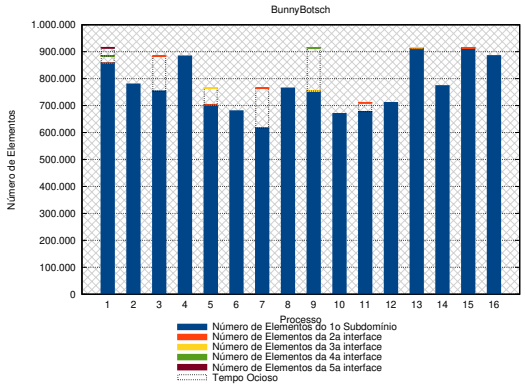
Figura 101 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, com 16 processos.

Fonte: elaborado pelo autor (2019).

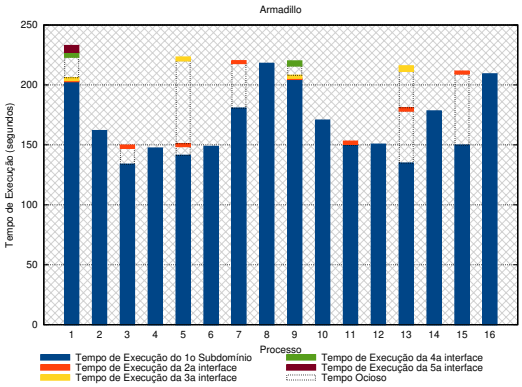
Como dito anteriormente, o modelo Beam tem geometria simples e sua carga está distribuída de maneira uniforme, refletido na carga dos subdomínios, e gerando um balanceamento de ótima qualidade. Já nos outros modelos, por terem geometria complexas, é possível ver que existem desbalanceamentos entre os subdomínios. Um dos motivos desse desbalanceamento é o teste aplicado no PD para evitar a criação de subdomínios em regiões ruins, como mostrado na Seção 4.2.2. Mesmo existindo um desbalanceamento entre os subdomínios, os gráficos de *speed-up* mostram que esse grau de desbalanceamento não afeta drasticamente os resultados, sendo ainda possível ter resultados de *speed-up* superlineares.



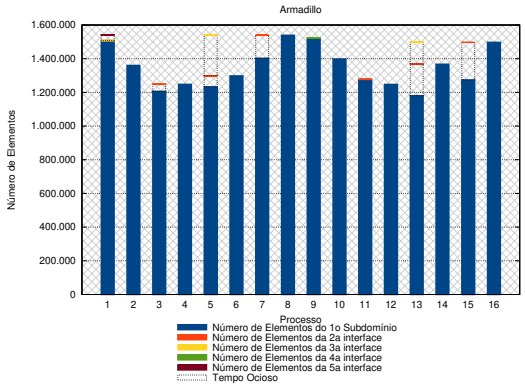
(e) Tempo de execução do modelo BunnyBotsch.



(f) Número de elementos do modelo BunnyBotsch.



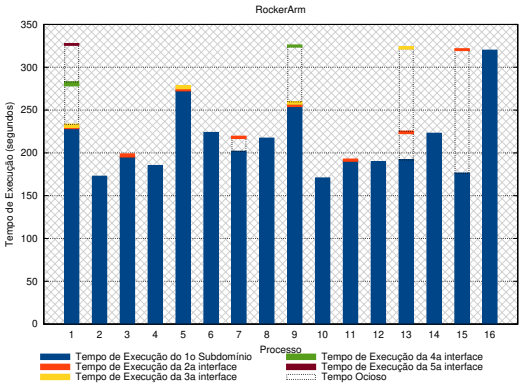
(g) Tempo de execução do modelo Armadillo.



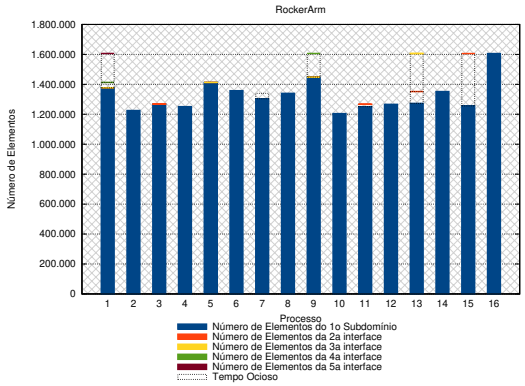
(h) Número de elementos do modelo Armadillo.

Figura 101 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, para 16 processos (continuação).

Fonte: elaborado pelo autor (2019).



(i) Tempo de execução do modelo RockerArm.



(j) Número de elementos do modelo RockerArm.

Figura 101 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, para 16 processos (continuação).

Fonte: elaborado pelo autor (2019).

5.7 Comparação com Abordagem *a Posteriori*

Como dito anteriormente, a técnica *a priori* aqui proposta é uma extensão do trabalho *a posteriori* de Freitas *et al.* (2016). Para entender melhor as diferenças entre as duas abordagens, são mostrados aqui nessa seção os resultados de tamanho da malha, tempo de execução, qualidade e balanceamento para o modelo de teste Armadillo. Esse foi o modelo escolhido para exemplificar as diferenças entre as duas técnicas por conseguir mostrar bem o comportamento delas.

Para comparar o tempo de execução de duas técnicas de geração em paralelo de malhas é preciso saber primeiro a quantidade de elementos que cada técnica está gerando para que a comparação seja a mais justa possível. A Figura 102 mostra os tamanhos das malhas geradas nas duas abordagens. É possível ver que a diferença nos tamanhos das malhas aumentam à medida que mais partições são feitas no modelo, isso é uma característica das técnicas *a priori*, pois ela insere artificialmente novas fronteiras no modelo para criar subdomínios independentes. Em técnicas *a posteriori* esse fato não acontece, resultando numa malha gerada em paralelo com praticamente a mesma quantidade de elementos que a malha sequencial.

Na Figura 103, é mostrada a qualidade das malhas geradas do modelo Armadillo pela abordagem *a posteriori*. A diferença de qualidade para a abordagem *a priori* (Figuras 95g e 95h) é pequena quando comparada. É possível notar que a diferença total entre a malha gerada sequencialmente e em paralelo é 4,0% na abordagem *a priori* e na *a posteriori* fica em torno de 3,0%. Esse fato ocorre por conta da inserção das fronteiras artificiais da abordagem *a priori*. A Figura 104 mostra a diferença total na qualidade da malha quando comparada com a gerada em

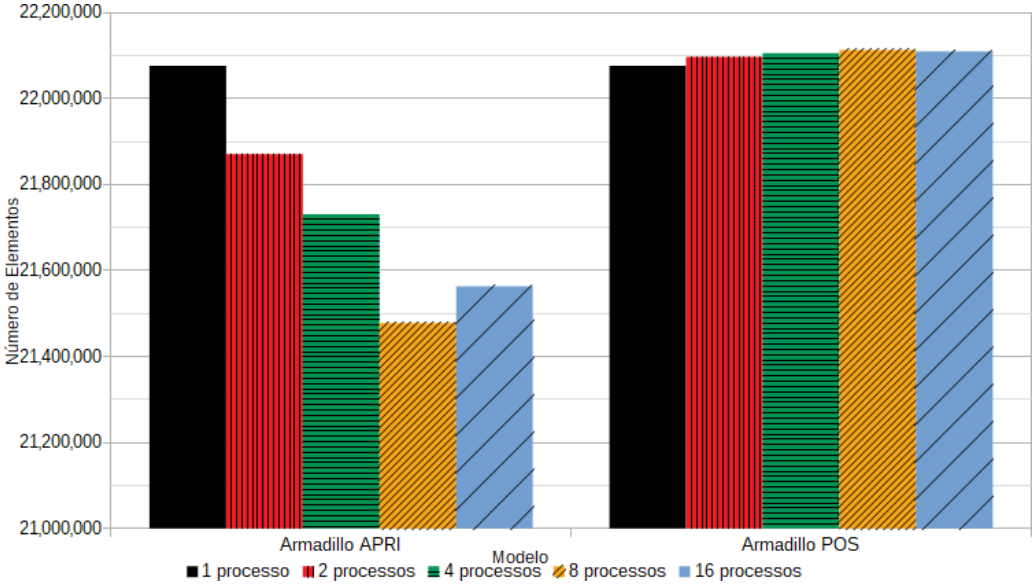
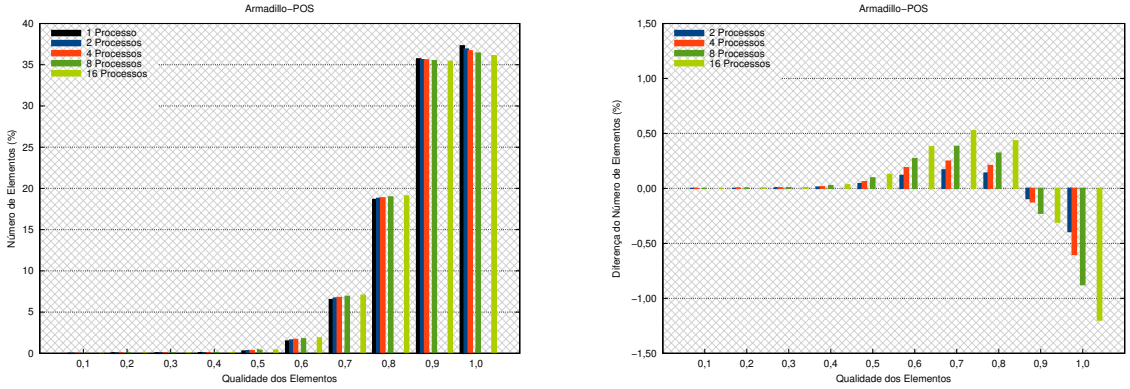


Figura 102 – Quantidade de elementos gerados no modelo Armadillo para as técnicas *a priori* e *a posteriori* na faixa de 21 milhões a 22,2 milhões.

Fonte: elaborado pelo autor (2019).

sequencial para as duas abordagens.



(a) Qualidade das malhas do modelo Armadillo.

(b) Diferença na qualidade das malhas do modelo Armadillo.

Figura 103 – Qualidade, em porcentagem, das malhas geradas sequencialmente e em paralelo (coluna da esquerda) e diferença na porcentagem da qualidade das malhas geradas (coluna da direita) para a abordagem *a posteriori*.

Fonte: elaborado pelo autor (2019).

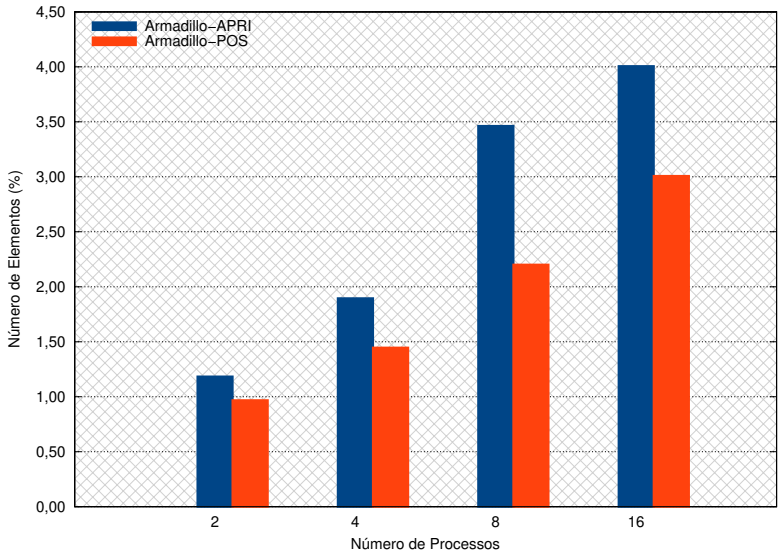


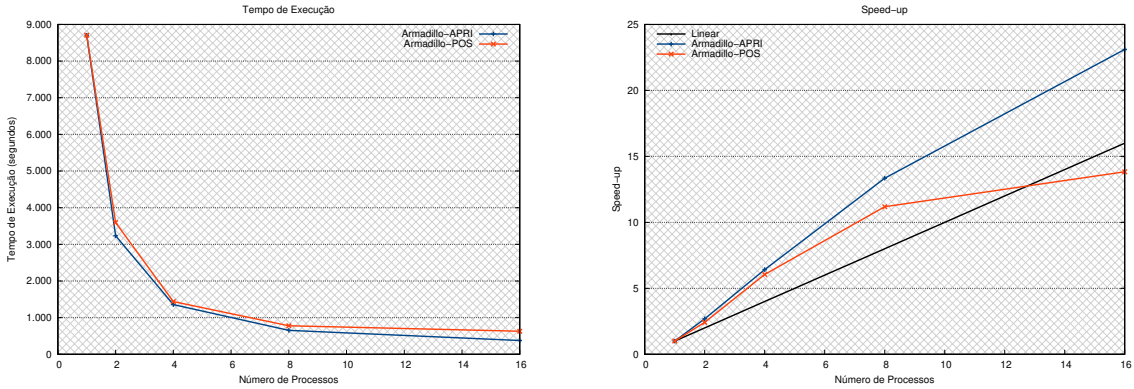
Figura 104 – Diferença total na qualidade da malha quando comparada a sequencial para as técnicas *a priori* e *a posteriori*.

Fonte: elaborado pelo autor (2019).

É possível notar que as duas abordagens têm o mesmo comportamento com relação à quantidade de elementos em cada faixa de qualidade: poucos elementos de baixa qualidade e muitos elementos de alta qualidade. Essa característica advém da técnica sequencial utilizada, que também tem esse comportamento.

A Figura 105 mostra o tempo de execução e o *speed-up* do modelo Armadillo para as abordagens *a priori* e *a posteriori*. Na abordagem *a posteriori*, é possível ver uma queda brusca no *speed-up* quando aumenta-se de 8 para 16 processos. Quando esse resultado é comparado com a abordagem *a priori*, nota-se que existe uma grande perda da escalabilidade. A explicação para isso é que, em abordagens *a posteriori*, é preciso fazer a geração e junção das malhas de interface de subdomínios vizinhos. Logo, quanto maior a quantidade de subdomínios, maior o tempo de execução.

A Figura 106 detalha os tempos envolvidos na geração da malha em paralelo do modelo Armadillo pela abordagem *a posteriori*. Enquanto que na abordagem *a priori*, havia apenas o tempo de melhoria da malha nas interfaces, na abordagem *a posterior*, essa fase é substituída por uma de geração de malhas entre subdomínios, a malha de interface. O tempo de execução dessa fase cresce a medida que mais processos são adicionados, podendo atingir cerca de 40% do tempo total quando são utilizados 16 processos. Além do tempo para a geração da malha entre subdomínios, é possível notar um desbalanceamento maior por conta da comunicação



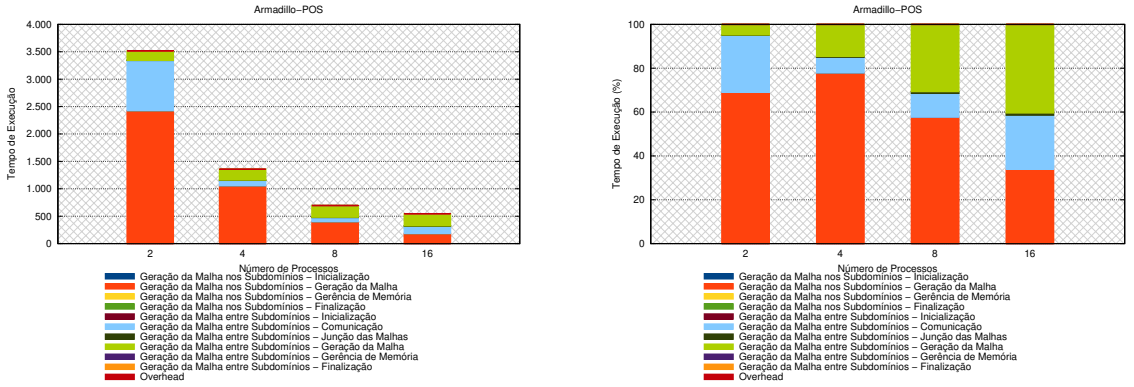
(a) Tempo de execução para o modelo Armadillo nas abordagens *a priori* e *a posteriori*.

(b) *Speed-up* para o modelo Armadillo nas abordagens *a priori* e *a posteriori*.

Figura 105 – Tempo de execução e *speed-up* do modelo Armadillo para as abordagens *a priori* e *a posteriori*.

Fonte: elaborado pelo autor (2019).

entre processos.



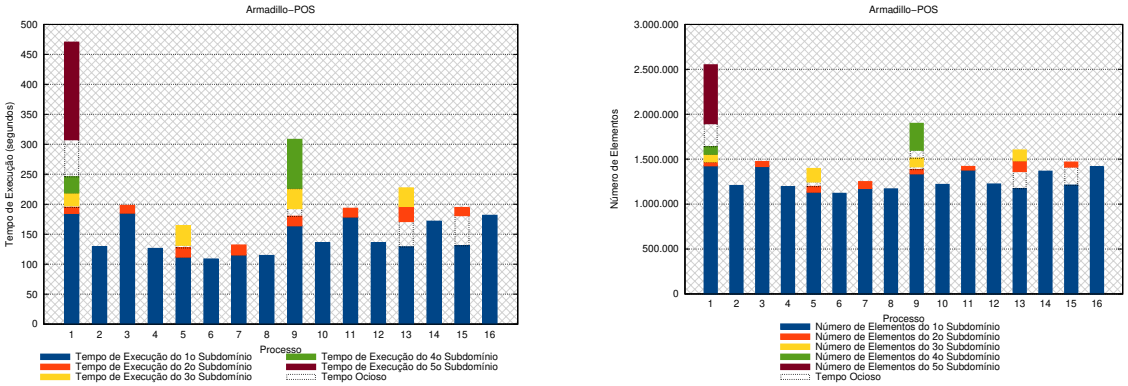
(a) Tempo absoluto do modelo Armadillo.

(b) Tempo em porcentagem do modelo Armadillo.

Figura 106 – Detalhamento do tempo de execução absoluto de geração de malha (coluna da esquerda) e em porcentagem (coluna da direita) para a abordagem *a posteriori*.

Fonte: elaborado pelo autor (2019).

A Figura 107 mostra o balanceamento de carga entre os 16 processos na abordagem *a posteriori*. Pela característica dessa técnica, é necessário fazer a geração da malha entre os subdomínios a cada junção realizada, isso acaba gerando um tempo extra, elevando assim o seu tempo de execução. Assim, o balanceamento de carga reflete diretamente no tempo de execução dos processos, que reflete no tempo de execução total da abordagem. Por essa razão, a técnica *a posteriori* apresenta um *speed-up* menor que a técnica a *priori*.



(a) Tempo de execução do modelo Armadillo. (b) Número de elementos do modelo Armadillo.

Figura 107 – Tempo de execução (coluna da esquerda) e número de elementos (coluna da direita) por processo, para 16 processos, para a abordagem *a posteriori*.

Fonte: elaborado pelo autor (2019).

5.8 Considerações

Nesse capítulo, foram apresentados os resultados da técnica proposta para cinco modelos tridimensionais, um deles de geometria simplificada e quatro deles com geometria complexa. Apesar dessa técnica funcionar para entradas bidimensionais e tridimensionais, foram escolhidos apenas os modelos tridimensionais complexos para realização dos testes. Resultados usando esta técnica de decomposição para entradas bidimensionais podem ser encontradas em Teixeira *et al.* (2011). Além disso, foram mostradas comparações da técnica proposta, que usa uma abordagem *a priori*, com a técnica de decomposição de Freitas *et al.* (2016), que utiliza uma abordagem *a posteriori*.

A técnica proposta neste trabalho consegue gerar uma malha de qualidade, com níveis bem próximos aos da malha gerada de forma sequencial. Com relação à comparação entre as duas técnicas (*a priori* e *a posteriori*), é possível constatar que a técnica *a posteriori* possui uma qualidade mais próxima à da malha gerada sequencialmente, tendo uma diferença menor que a técnica descrita neste trabalho. Esse fato já era esperado, pois abordagens contínuas de particionamento *a posteriori* não criam fronteiras artificiais, limitando a geração da malha naquelas regiões onde foi feito a decomposição, fato que ocorre em abordagens *a priori*. Isso, entretanto, não é um problema sério, pois a qualidade da malha não varia mais que 2% na faixa mais discrepante, como mostrado. Com relação à questão de velocidade, a técnica apresentada neste trabalho apresenta um *speed-up* superlinear para os modelos apresentados, sendo melhor quando comparada com a do trabalho de Freitas *et al.* (2016).

6 CONCLUSÃO

Esse trabalho apresentou uma técnica *a priori* para decomposição de domínios, bidimensionais ou tridimensionais, para a geração paralela da malha. Apesar de os resultados estarem focados em computadores de memória distribuída, esta técnica também funciona em computadores de memória compartilhada. A técnica proposta é dita *a priori* porque a malha de interface entre os subdomínios é gerada antes das suas malhas internas. Essa técnica gera novos domínios independentes entre si, permitindo abstrair a técnica de geração de malha aplicada em cada subdomínios, podendo-se combinar, por exemplo, as técnicas de Delaunay e Avanço de Fronteira, dentre outras.

Ao final deste trabalho, foram apresentados os resultados obtidos pela técnica proposta. Os resultados foram comparados com os resultados para os mesmos modelos no trabalho de Freitas *et al.* (2016). A abordagem *a posteriori*, que serviu de base para esse trabalho, tem esse nome por que gera a malha na interface entre subdomínios após a geração da malha interna aos subdomínios.

Os métodos de decomposição de domínios que criam subdomínios independentes *a priori* podem não ser adequados quando executados em grandes números de processadores, mesmo que o processo de decomposição de domínios seja automático, e que utilize técnicas para uma decomposição apropriada. Portanto, como já detectado por Yilmaz e Ozturan (2015), técnicas de decomposição *a priori* não foram feitas para a utilização de *over-decomposition* em suas entradas.

A aplicação de testes para validar os Planos de Decomposição (PD) criados permitem que esta técnica funcione para modelos de geometrias bastante complexas, mantendo a qualidade de malha próxima à malha sequencial e preservando o balanceamento da carga, mantendo assim a escalabilidade da técnica. Graças aos testes aplicados, esta técnica de decomposição pôde reduzir o tempo de geração de uma malha de 21 milhões de elementos de 2,6 horas para menos de 3 minutos, usando 16 processos.

Uma vantagem de utilizar uma *Binary Space Partitioning* (BSP) para decompor os domínios nessa técnica *a priori* é que os subdomínios criados tendem a ficar com a mesma proporção de carga. Utilizando o fator de proporcionalidade na BSP para decomposição, a quantidade de subdomínios criados é a mesma quantidade de processos disponíveis, evitando o excesso de tarefas e de comunicação, maximizando assim o *speed-up*.

Essa técnica aborda a decomposição de domínios para geração de malha em paralelo

e não a geração de malha propriamente dita. Apesar de qualquer algoritmo poder ser utilizado, em particular, um algoritmo de avanço de fronteira foi utilizado para gerar malha e está descrito em Miranda *et al.* (1999) e Cavalcante-Neto *et al.* (2001).

A principal contribuição deste trabalho é a apresentação de uma nova técnica de decomposição de domínios para geração de malha em paralelo que apresenta bons resultados com a utilização de uma BSP para auxiliar a decomposição, mostrando que abordagens *a priori* possuem *speed-up* melhores que abordagens *a posteriori*, apesar de gerar uma malha levemente mais diferente que essa outra abordagem. Essa técnica mantém do tamanho da malha gerada em paralelo praticamente na mesma proporção que a malha gerada sequencialmente e a qualidade dos elementos gerados mantém a qualidade da malha gerada sequencialmente, não chegando a 5,5% (nos testes realizados) a diferença quando são utilizados 16 processos.

6.1 Trabalhos Futuros

Alguns pontos podem ser melhorados neste trabalho. O primeiro ponto seria a aplicação de transformações lineares, como rotações, no modelo de entrada para que os PD posicionados inicialmente não estejam em posições ruins para a decomposição e geração da malha, sendo assim não haveria a necessidade dos testes de ângulo PD nas primeiras decomposições, ajudando o balanceamento da carga.

O segundo ponto seria permitir que os PD cruzassem as células cheias da *octree*. Assim, seria possível posicionar o PD em qualquer eixo, que poderiam ser eixos diagonais em vez de somente os eixos globais (X, Y e Z). Com essa melhoria, os testes de ângulo poderiam aplicar rotações e translações no PD.

O terceiro ponto seria a criação de uma técnica de decomposição híbrida, incorporando a técnica de decomposição *a posteriori* de Freitas *et al.* (2016) a este trabalho. A ideia é combinar as vantagens de cada método, onde a técnica *a posteriori* seja responsável pela criação da malha de interface próxima da borda, e que a malha de interface no interior do modelo, em regiões afastadas da borda, seja criada *a priori*. Com isso seria possível um melhor balanceamento da carga computacional, uma vantagem desta técnica *a priori*, e seria agora possível a utilização de *overdecomposition*, uma vantagem de técnicas *a posteriori*.

O quarto ponto seria a paralelização da criação e classificação da *octree*, que são feitos sequencialmente. À medida que mais processos são adicionados esse tempo começa a crescer proporcionalmente, chegando a consumir 50% do tempo total no modelo RockerArm.

Essa melhoria permitiria maiores ganhos de tempo e *speed-up*.

O quinto ponto seria a permitir que o Plano de Decomposição (PD) fosse posicionado na melhor posição possível, de acordo com algum critério. Um possível critério seria, por exemplo, a melhor posição do PD com relação aos ângulos com as faces. Somente depois de posicionar o PD é que seria feito a estimativa de carga e o balanceamento, aplicando um fator de proporcionalidade para compensar o possível desbalanceamento causado pelo PD.

Por último, a geração de malhas em domínios com fraturas não foi testada, mas espera-se fazer uma versão deste trabalho que suporte modelos com fraturas existentes na entrada. A técnica sequencial tem suporte a fratura e, por não gerar malha de interface, a técnica paralela *a posteriori* também possui essa característica. Nessas malhas, a superfície da fratura é representada como uma região de área nula formada por elementos da FE geometricamente coincidentes, cujas normais têm sentidos opostos.

REFERÊNCIAS

ANDRÄ, H. e. a. Automatic parallel generation of tetrahedral grids by using a domain decomposition approach. **Computational Mathematics and Mathematical Physics**, [*s.l.*], v. 48, n. 8, p. 1367–1375, 2008.

BANK RANDOLPH E.; LU, S. A domain decomposition solver for a parallel adaptive meshing paradigm. **SIAM journal on scientific computing**, [*s.l.*], v. 26, n. 1, p. 105–127, 2005.

BARNARD, S. T.; SIMON, H. D. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. **Concurrency: Practice and experience**, Wiley Online Library, [*s.l.*], v. 6, n. 2, p. 101–117, 1994.

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.; MARTHA, L. F.; WAWRZYNEK, P. A.; INGRAFFEA, A. R. A back-tracking procedure for optimization of simplex meshes. **Communications in Numerical Methods in Engineering**, [*s.l.*], v. 21, n. 12, p. 711–722, 2005. ISSN 1069-8299.

CAVALCANTE-NETO, J. B.; WAWRZYNEK, P. A.; CARVALHO, M. T. M.; MARTHA, L. F.; INGRAFFEA, A. R. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. **Engineering with Computers**, [*s.l.*], v. 17, n. 1, p. 75–91, 2001.

CHARMPIS, D. C.; PAPADRAKAKIS, M. Generation of balanced subdomain clusters with minimum interface for distributed domain decomposition applications. Springer, [*s.l.*], p. 555–562, 2005.

CHEN, J.; XIAO, Z.; ZHENG, Y.; ZOU, J.; ZHAO, D.; YAO, Y. Scalable generation of large-scale unstructured meshes by a novel domain decomposition approach. **Advances in Engineering Software**, Elsevier, v. 121, p. 131–146, 2018.

CHEN, J. e. a. Improvements in the reliability and element quality of parallel tetrahedral mesh generation. **International Journal for Numerical Methods in Engineering**, [*s.l.*], v. 92, n. 8, p. 671–693, 2012.

CHRISOCHOIDES, N. A survey of parallel mesh generation methods. Brown University, Providence, [*s.l.*], 2005.

DECOUGNY, H. L.; SHEPHARD, M. S. Parallel volume meshing using face removals and hierarchical repartitioning. **Computer Methods in Applied Mechanics and Engineering**, [*s.l.*], v. 174, n. 3-4, p. 275–298, 1999.

ERTEN, H.; ÜNGÖR, A. Quality triangulations with locally optimal steiner points. **SIAM Journal on Scientific Computing**, SIAM, [*s.l.*], v. 31, n. 3, p. 2103–2130, 2009.

FARHAT, C. A simple and efficient automatic fem domain decomposer. **Computers and Structures**, v. 28, n. 5, p. 579–602, 1988.

FREITAS, M. de O.; CAVALCANTE-NETO, J. B.; VIDAL, C. A. **Parallel generation of meshes with cracks using a binary espacial decomposision**. [*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, [*s.l.*], 2010.

FREITAS, M. O.; WAWRZYNEK, P. A.; CAVALCANTE-NETO, J. B.; VIDAL, C. A.; MARTHA, L. F.; INGRAFFEA, A. R. A distributed-memory parallel technique for two-dimensional mesh generation for arbitrary domains. **Advances in Engineering Software**, [*s.l.*], v. 59, p. 38–52, 2013.

FREITAS, M. O.; WAWRZYNEK, P. A.; CAVALCANTE-NETO, J. B.; VIDAL, C. A.; CARTER, B. J.; MARTHA, L. F.; INGRAFFEA, A. R. Parallel generation of meshes with cracks using binary spatial decomposition. **Engineering with Computers**, Springer, [*s.l.*], v. 32, n. 4, p. 655–674, 2016.

GAITHER, A. e. a. A paradigm for parallel unstructured grid generation. **Numerical Grid Generation in Computational Field Simulations**, [*s.l.*], p. 731–740, 1996.

GALTIER, J.; GEORGE, P. L. Prepartitioning as a way to mesh subdomains in parallel. [*s.l.*], 1996.

GLUT, B.; JURCZYK, T. Domain decomposition techniques for parallel generation of tetrahedral meshes. **Springer-Verlag Berlin Heidelberg**, [*s.l.*], p. 641–650, 2008.

GRAPHICS, S.; OPENGL, I. **The Industry's Foundation for High-Performancs Graphics**. 1998. Disponível em: <http://www.opengl.org>. Acesso em: 06 set. 2019.

HJELLE, Ø.; DÆHLEN, M. **Triangulations and applications**. [*s.l.*]: Springer Science & Business Media, 2006.

HWANG, K.; XU, Z. **Scalable parallel computing: technology, architecture, programming**. [*s.l.*]: McGraw-Hill, Inc., 1998.

IBANEZ, D. A.; SEOL, E. S.; SMITH, C. W.; SHEPHARD, M. S. Pumi: Parallel unstructured mesh infrastructure. **ACM Transactions on Mathematical Software (TOMS)**, ACM, [*s.l.*], v. 42, n. 3, p. 17, 2016.

IKITS, M.; MAGALLON, M. **The OpenGL Extension Wrangler Library**. 2015. Disponível em: <http://glew.sourceforge.net>.

ITO, Y.; SHIH, A. M.; ERUKALA, A. K.; SONI, B. K.; CHERNIKOV, A.; CHRISOCHOIDES, N. P.; NAKAHASHI, K. Parallel unstructured mesh generation by an advancing front method. **Mathematics and Computers in Simulation**, [*s.l.*], 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**, [*s.l.*], v. 6, n. 2, p. 178–193, 2006.

JURCZYK, T.; GŁUT, B.; BREITKOPF, P. Parallel 3d mesh generation using geometry decomposition. [*s.l.*], v. 908, n. 1, p. 1579–1584, 2007.

KABELIKOVA, P.; RONOVSKY, A.; VONDRAK, V. Parallel mesh multiplication for code_saturne. **PRACE project (FP7/2007-2013)**, [*s.l.*], p. 1–8, 2011.

KARYPIS, G. Metis and parmetis. **Encyclopedia of parallel computing**, Springer, [*s.l.*], p. 1117–1124, 2011.

KARYPIS, G.; KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. **SIAM Journal on scientific Computing**, SIAM, [*s.l.*], v. 20, n. 1, p. 359–392, 1998.

LÄMMER, L.; BURGHARDT, M. Parallel generation of triangular and quadrilateral meshes. **Advances in Engineering Software**, [*s.l.*], v. 31, n. 12, p. 929–936, 2000.

LARWOOD, B. G.; WEATHERILL, N. P.; HASSAN, O.; MORGAN, K. Domain decomposition approach for parallel unstructured mesh generation. **International Journal for Numerical Methods in Engineering**, [*s.l.*], v. 58, p. 177–188, 2003.

LEWIS, R. W.; ZHENG, Y.; GETHIN, D. T. Three-dimensional unstructured mesh generation: Part 3 - volume meshes. [*s.l.*], v. 134, p. 285–310, 1996.

LINARDAKIS, L.; CHRISOCHOIDES, N. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. **SIAM Journal on Scientific Computing**, [*s.l.*], v. 27, n. 4, p. 1394–1423, 2006.

LO, S. Parallel delaunay triangulation in three dimensions. **Computer Methods in Applied Mechanics and Engineering**, Elsevier, [*s.l.*], v. 237, p. 88–106, 2012.

LO, S. Parallel delaunay triangulation—application to two dimensions. **Finite Elements in Analysis and Design**, Elsevier, [*s.l.*], v. 62, p. 37–48, 2012.

LÖHNER, R. A parallel advancing front grid generation scheme. **International Journal for Numerical Methods in Engineering**, [*s.l.*], v. 51, n. 6, p. 663–678, 2001.

LOHNER, R. Recent advances in parallel advancing front grid generation. **Archives of Computational Methods in Engineering**, [*s.l.*], v. 21, n. 2, p. 127–140, 2014.

LOSEILLE, A.; MENIER, V.; ALAUZET, F. Parallel generation of large-size adapted meshes. **Procedia Engineering**, Elsevier, [*s.l.*], v. 124, p. 57–69, 2015.

MIRANDA, A. C. de O.; MARTHA, L. F.; WAWRZYNEK, P. A.; INGRAFFEA, A. R. Surface mesh regeneration considering curvatures. **Engineering with Computers**, Springer-Verlag London Limited, [*s.l.*], 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.l.*]: IEEE Computer Society, 1999. p. 29–38.

MPI Forum. **The Message Passing Interface (MPI) Standard**. 2019. Disponível em: <http://www.mcs.anl.gov/research/projects/mpi>. Acesso em: 06 set. 2019.

NIKISHKOV, G. P. e. a. An algorithm for domain partitioning with load balancing. **Engineering Computations**, [*s.l.*], v. 16, n. 1, p. 120–135, 1999.

PANITANARAK, T.; SHONTZ, S. M. Mdec: Metis-based domain decomposition for parallel 2d mesh generation. **Procedia Computer Science**, Elsevier, [*s.l.*], v. 4, p. 302–311, 2011.

PARTHASARATHY, V. N.; GRAICHEN, C. M.; HATHAWAY, A. F. A comparison of tetrahedron quality measures. [*s.l.*], v. 15, p. 255–261, 1993.

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.l.*]: AIAA - American Institute of Aeronautics and Astronautics, 2009.

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**, [*s.l.*], v. 177, n. 1, p. 109–125, 1999.

SCHÖBERL, J. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. **Computing and visualization in science**, Springer, [*s.l.*], v. 1, n. 1, p. 41–52, 1997.

SHEWCHUK, J. R. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: **Workshop on Applied Computational Geometry**. [*s.l.*]: [s.n.], 1996. p. 203–222.

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.l.*]: Springer-Verlag, 1996, (Lecture Notes in Computer Science, v. 1148). p. 203–222. From the First ACM Workshop on Applied Computational Geometry.

SIMON, H. D. Partitioning of unstructured problems for parallel processing. **Computing systems in engineering**, Elsevier, [*s.l.*], v. 2, n. 2-3, p. 135–148, 1991.

SMITH, C. W.; RASQUIN, M.; IBANEZ, D.; JANSEN, K. E.; SHEPHARD, M. S. Application specific mesh partition improvement. **SIAM Journal on Scientific Computing**, [*s.l.*], 2015.

TEIXEIRA, D. N.; FREITAS, M. O.; CAVALCANTE-NETO, J. B.; VIDAL, C. A. A technique for parallel mesh generation using a priori quadtree's inter-cell discretization. In: **XXXII CILAMCE-Congresso Ibero-Latino-Americano de Méto-do Computacionais em Engenharia**. [*s.l.*]: [s.n.], 2011.

The C++ Standards Committee. **JTC1/SC22/WG21 - The C++ Standards Committee - ISOCPP**. 2015. Disponível em: <http://www.open-std.org/jtc1/sc22/wg21>. Acesso em: 06 set. 2019.

VIDWANS, A.; KALLINDERIS, Y.; VENKATAKRISHNAN, V. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. **AIAA journal**, [*s.l.*], v. 32, n. 3, p. 497–505, 1994.

WANG, X.; JIN, X. A distributed-memory parallel approach for the generation of multibillion element tetrahedral meshes. In: IEEE. **2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)**. [*s.l.*], 2016. p. 301–306.

WU, P.; HOUSTIS, E. N. Parallel adaptive mesh generation and decomposition. **Engineering with Computers**, Springer, [*s.l.*], v. 12, n. 3-4, p. 155–167, 1996.

wxTeam. **wxWidgets - Cross-Platform GUI Library**. [*s.l.*]: [s.n.], 2019. Disponível em: <http://www.wxwidgets.org>. Acesso em: 06 set. 2019.

YILMAZ, Y.; OZTURAN, C. Using sequential netgen as a component for a parallel mesh generator. **Advances in Engineering Software**, Elsevier, [*s.l.*], v. 84, p. 3–12, 2015.

ZHANG, Y.; JIA, Y.; WANG, S. S.; ALTINAKAR, M. Composite structured mesh generation with automatic domain decomposition in complex geometries. **Engineering Applications of Computational Fluid Mechanics**, Taylor & Francis, [*s.l.*], v. 7, n. 1, p. 90–102, 2013.

ZHENG, Y.; CHEN, J. Unstructured mesh generation and its parallelization. Springer, [*s.l.*], p. 22–35, 2007.

ZHOU, M.; SAHNI, O.; DEVINE, K. D.; SHEPHARD, M. S.; JANSEN, K. E. Controlling unstructured mesh partitions for massively parallel simulations. **SIAM Journal on Scientific Computing**, SIAM, [*s.l.*], v. 32, n. 6, p. 3201–3227, 2010.