UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

FRANCISCO WALLISON CARLOS ROCHA

ALGORITMOS PARA CONSTRUÇÃO DE *QUADTREES* COM BASE EM PARTIÇÃO BINÁRIA DE ESPAÇO

RUSSAS

2018

FRANCISCO WALLISON CARLOS ROCHA

ALGORITMOS PARA CONSTRUÇÃO DE *QUADTREES* COM BASE EM PARTIÇÃO BINÁRIA DE ESPAÇO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Markos Oliveira Freitas

RUSSAS

2018

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)

R573a Rocha, Francisco Wallison.
Algoritmos para construção de quadtrees com base em partição binária de espaço / Francisco Wallison Rocha. – 2018.
75 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2018.
Orientação: Prof. Dr. Markos Oliveira Freitas.

1. Quadtree. 2. Programação paralela. 3. Binary Space Partitioning. I. Título.

CDD 005.1

FRANCISCO WALLISON CARLOS ROCHA

ALGORITMOS PARA CONSTRUÇÃO DE *QUADTREES* COM BASE EM PARTIÇÃO BINÁRIA DE ESPAÇO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

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

Prof. Ms. Daniel Márcio Batista de Siqueira
Universidade Federal do Ceará (UFC)

Prof. Ms. Rafael Fernandes Ivo
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar e investir em mim. Mãe, seu cuidado e dedicação foi que deram a esperança para seguir. Pai, sua presença em minha mente e coração significou segurança e certeza de que não estou sozinho nessa caminhada. Álamo, seu carinho e carisma me deram entusiasmo nessa caminhada. Avó e Avô, o apoio e dedicação de vocês me deram

AGRADECIMENTOS

A Deus, por ter me dado saúde e força para superar as dificuldades.

Aos meus pais e meu irmão, por todo amor, carinho e incentivo a mim destinados, principalmente à minha mãe, por todo o seu empenho e apoio na minha trajetória.

Aos meus familiares, em especial aos meus avós, tias e tio por todo amor, ajuda e apoio destinados a mim.

Ao Dr. Markos Oliveira Freitas, meu professor, orientador e amigo, por ter acreditado em meu potencial, pela oportunidade, paciência e conhecimentos passados. Suas contribuições foram essenciais para a realização deste trabalho. Agradeço também por todos os momentos descontraídos e alegres.

Ao Laboratório Interdisciplinar de Computação e Engenharia de Software (LINCE), em especial aos meu colegas da Computação Gráfica, Ângelo Matheus, Gilberto Lima e Davison Alves, junto com o nosso orientador Prof. Dr. Markos Oliveira Freitas, que realizaram pesquisas ao meu lado e contribuíram de algum modo para este trabalho.

À bela Flor (Estefânia Estevam), por todo apoio, carinho, incentivo e momentos felizes no decorrer dessa trajetória. Agradeço também por sempre ter tentado me fazer uma pessoa melhor.

À senhora Aparecida Estevam, ao senhor Paulo Estevam e familiares, a segunda família que ganhei durante esse período, pela consideração, carinho e vibrações positivas.

Aos meus amigos, em especial Ângelo Matheus, Eduardo Santos, Davidson Alves, Gilberto Lima, Leiliane Santos, Luis Felipe, Matheus Rocha, Raimundo Mota e Vinícius Falcão, pelas alegrias, tristezas e dores compartilhas.

Ao meu ex-professor e amigo Manoel Carvalho (NEO ou Manel), por me apresentar o mundo maravilhoso da T.I, me ajudar e incentivar durante todo o período dos Ensinos Médio e Superior.

Ao Ms. Marcos Vinicius de Andrade Lima, meu professor e amigo, por sempre ter acreditado em mim, me motivando e mostrando os caminhos para que eu pudesse alcançar meus objetivos.

Aos meus professores, por todo conhecimento compartilhado durante esses quatro anos de curso, e por todo empenho e dedicação na minha formação técnico-humana.

Ao Prof. Dr. Lindberg Lima Gonçalves, pelo esforço e empenho para implantar e manter o Campus da UFC - Campus Russas, o que foi muito importante para que eu e muitos

outros tivéssemos acesso ao ensino superior.

Aos meus colegas de curso, pelos momentos compartilhados no decorrer da graduação.

E agradaço a todos que direta ou indiretamente contribuiram para que momentos como este acontecessem.

"Não te preocupes com os que não te conhecem, mas esforça-te por seres digno de ser conhecido."

(Confúcio)

RESUMO

Técnicas de subdivisão de espaço, como quadtrees e octrees, são amplamente utilizadas como estruturas auxiliares para diversas aplicações. Em particular, elas são utilizadas na geração de malhas de elementos finitos. Para geração de grandes malhas refinadas, faz-se necessária a utilização de programação paralela. Assim, este trabalho apresenta uma técnica para construção paralela de *quadtrees* com base em Partição Binária do Espaço, do inglês *Binary Space Partitioning* - BSP. Dado um modelo discretizado em vértices e arestas, é possível extrair informações para realizar uma estimativa de carga que serve como guia para o particionamento do modelo entre os processos usando a BSP. Este particionamento visa equilibrar o trabalho de geração e de armazenamento da *quadtree* realizado por cada processo. Essa técnica foi aplicada em três modelos para 2 a 16 processos, apresentando bons resultados para modelos uniformemente distribuídos.

Palavras-chave: *Quadtree*. Programação paralela. *Binary Space Partitioning*.

ABSTRACT

Space subdivision techniques, such as quadtrees and octrees, are widely used as auxiliary structures for various applications. In particular, they are used in the generation of finite element meshes. For generation of large refined meshes, it is necessary to use parallel programming. Thus, this work presents a technique for parallel construction of quadtrees based on Binary Space Partitioning (BSP). Given a discrete model in vertices and edges, it is possible to extract information to perform a load estimate that serves as a guide for partitioning the model between processes using BSP. This partitioning aims to balance the work of generation and storage of the quadtree performed by each process. This technique was applied in three models for 2 to 16 processes, presenting good results for uniformly distributed models.

Keywords: Quadtree. Parallel programming. Binary Space Partitioning.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, a computação tem tido uma participação cada vez maior na resolução de problemas científicos e de engenharia, através dos Métodos de Elementos Finitos (MEFs) (CAMATA, 2011). Esses métodos são utilizados para representar os problemas do mundo real mantendo suas propriedades no domínio computacional. Ultimamente, as instâncias dos problemas que têm tido um crescimento cada vez maior, necessitando assim de mais e mais poder de processamento. Com o surgimento dos supercomputadores, tornou-se mais fácil a resolução de tais problemas, tanto no aspecto de processamento quanto no de armazenamento (CAMATA, 2011).

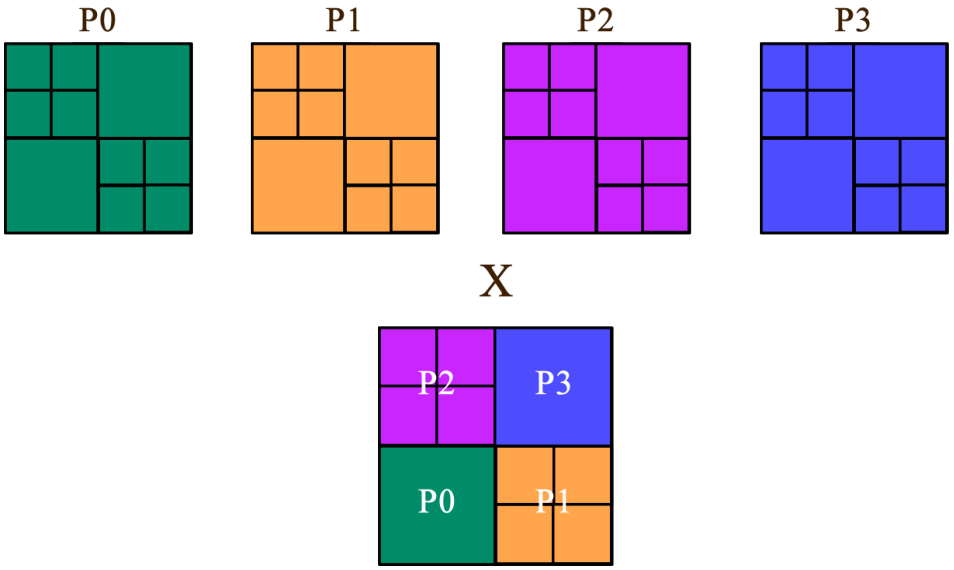
Camata (2011) afirma que, para acompanhar o crescimento da complexidade dos problemas e dos avanços computacionais, faz-se necessária a utilização de algoritmos paralelos eficientes, ou seja, que apresentem uma boa escalabilidade. Porém, houve um atraso na paralelização das técnicas de geração de malha com relação aos MEFs. Muitas técnicas de geração de malhas fazem uso de estruturas de dados espaciais, como a estrutura de árvore *octree*. Assim, para que a técnica de geração de malhas como um todo seja paralelizada eficientemente, é necessário paralelizar também a geração da *octree*.

Nos últimos anos, têm sidos obtidos resultados significativos em algoritmos paralelos para a construção de *octrees* em sistemas de memória distribuída. Algumas bibliotecas como o Dendro, proposta por Sundar *et al.* (2008), e o p4est, proposta por Burstedde *et al.* (2011), apresentam diversos algoritmos para construir e balancear estas estruturas em paralelo, utilizando por trás uma técnica chamada de curva de preenchimento de espaço.

Atualmente, existe um algoritmo paralelo proposto em Freitas *et al.* (2016), que apresenta um bom *speed-up* e uma boa estimativa de carga para a geração de malhas tetraédricas. No entanto, esse algoritmo depende da construção da *octree*, que é feita de forma sequencial com replicações da árvore em todos os processos (Figura 1). Esse procedimento sequencial limita a aplicação prática em modelos grandes, tanto em termos de tempo de processamento como de armazenamento, pois conforme o número de processos aumenta, esse tempo se torna proporcionalmente grande, podendo chegar a 25% do tempo total de execução.

Diante disso, uma boa alternativa é a construção da *octree* de maneira paralela. Como visto anteriormente, existem várias abordagens para esse fim, tais como Camata (2011), que são apresentadas algumas estratégias computacionais envolvendo MEFs. Entretanto, o trabalho só apresenta uma solução para a construção de *octrees* em paralelo, fazendo-se necessário abordar

Figura 1 – Atual x ideal. O trabalho apresentado por Freitas *et al.* (2016) encontra-se atuamente na situação em que a *quadtree*/*octree* é replicada em todos os processos. No entanto, o ideal é que uma única *quadtree*/*octree* seja divida entre todos os processos.



Fonte: Produzida pelo autor (2018).

mais soluções e realizar comparativos entre as abordagens, permitindo assim que se possa apresentar uma técnica adequada para resolução do problema de replicação da *octree* apresentado em Freitas *et al.* (2016).

1.1 Objetivos

O principal objetivo desse trabalho é apresentar uma técnica para a construção de *quadtrees*/*octrees* em sistemas de memória distribuída que possa servir de solução para o problema da replicação da *quadtree*/*octree* apresentado em Freitas *et al.* (2016).

Os objetivos específicos desse trabalho são:

- Apresentar um algoritmo para construção paralela de uma *quadtree* baseado no uso de uma BSP.
- Medir o balanceamento da técnica proposta.
- Medir a acurácia da estimativa de carga em relação ao número de células da árvore gerada nos subdomínios.
- Realizar comparativos entre técnicas para estimativa de carga e particionamento usando uma BSP.

1.2 Organização

Este trabalho está organizado nas seguintes capítulos. No Capítulo 2, apresenta-se as terminologias usadas no restante desse trabalho. No Capítulo 3, discute-se sobre os trabalhos relacionados. No Capítulo 4, discute-se a solução proposta para a resolução do problema abordado. No Capítulo 5 são discutidos os resultados obtidos nesse trabalho. Por fim, no Capítulo 6 são apresentadas as principais contribuições e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

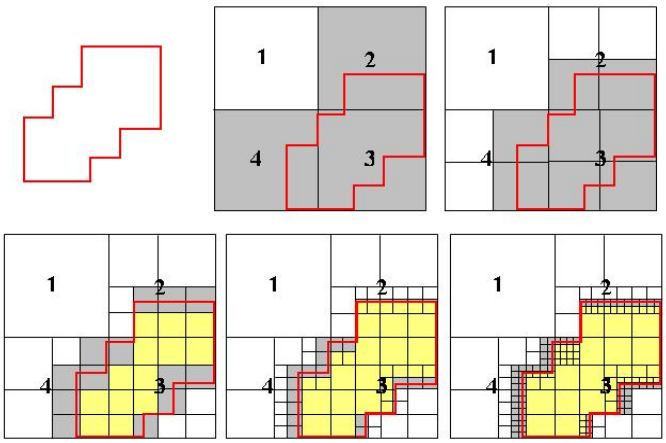
2.1 Introdução

Uma *quadtree* é uma estrutura de dados espacial muito utilizada na computação gráfica e em aplicações baseadas no método de elementos finitos. Muitas aplicações paralelas utilizam de uma curva de preenchimento de espaço para linearizar uma *quadtree* para obter um melhor particionamento. Neste trabalho usamos também de uma outra estrutura de dados espacial, a *Binary Space Partition* – BSP, que é usada para particionar o domínio (a *quadtree*) e dividi-la entre os processos.

2.2 *Quadtree*

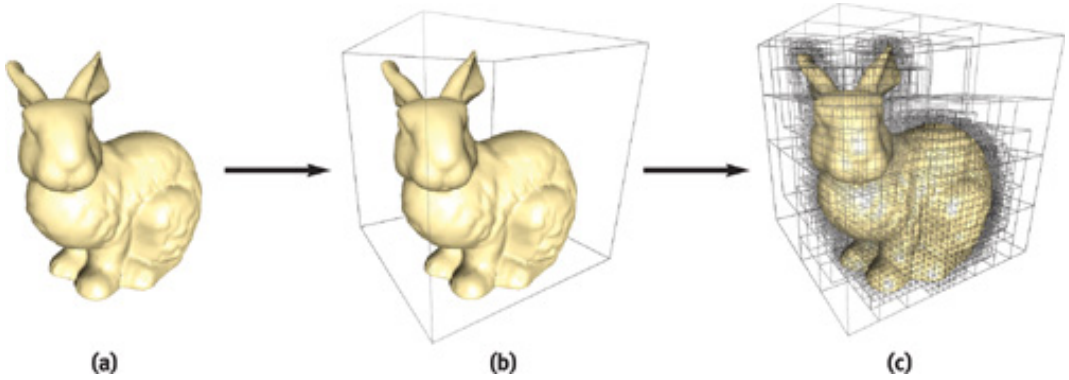
Quadtree é uma estrutura de dados espacial utilizada para subdividir um domínio 2D. Consiste em uma árvore que possui nós internos contendo quatro nós filhos (quadrantes) cada, e os nós que não possuem nenhum filho são denominados nós folhas (Figura 2). O seu análogo 3D, a *octree*, possui oito filhos para cada nó interno e nenhum para os seus nós folhas (Figura 3). *Quadtrees* e *octrees* possuem aplicações na ciência e na engenharia, geração de malhas, computação gráfica, processamento de imagens, tratamento de colisões, motores de jogos, etc. (SUNDAR *et al.*, 2008). Essas estruturas, por sua vez, requerem um particionamento hierárquico de uma região alinhada aos eixos cartesianos (CAMATA; COUTINHO, 2010). O nó raiz é a caixa delimitadora do domínio, cada nó filho é menor que seu pai e está em um nível acima (TU *et al.*, 2005).

Figura 2 – Exemplo de uma *quadtree* subdividindo um domínio 2D



Fonte: <http://www.lcad.icmc.usp.br/~nonato/ED/Quadtree/quadtree.htm>.

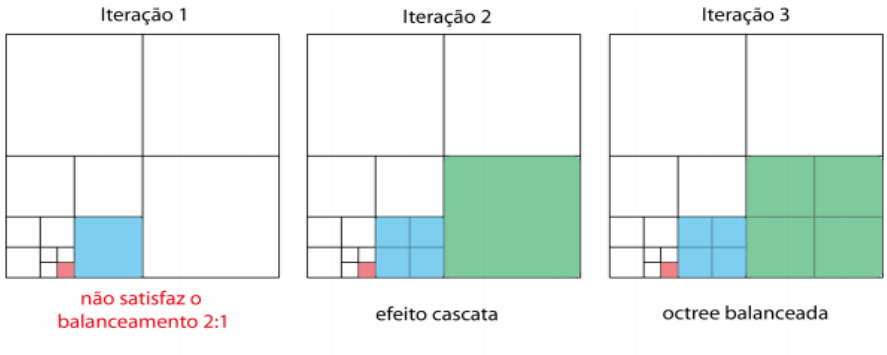
Figura 3 – Exemplo de *octree* subdividindo um domínio 3D



Fonte: <https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter37.html>.

Após realizar a construção, para que haja uma transição mais suave entre as regiões mais refinadas e menos refinadas do modelo, pode ser necessário realizar um refinamento para que a diferença de nível de células vizinhas não seja maior que 1. Essa restrição se chama Balanceamento 2:1 (Figura 4).

Figura 4 – Balanceamento 2:1.

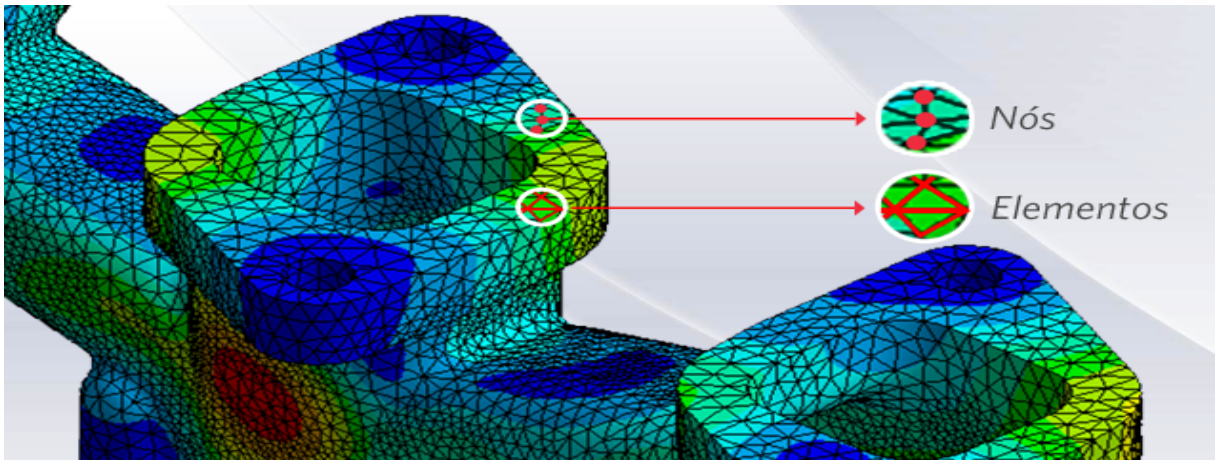


Fonte: Camata (2011).

2.3 Método de Elementos Finitos

Lotti *et al.* (2006) define o método de elementos finitos (MEFs) como um método matemático para discretizar (subdividir) objetos contínuos mantendo as propriedades do original (Figura 5). Esses elementos são representados por equações diferenciais (EDs) e resolvidos por modelos matemáticos. Os MEFs possuem grandes aplicações na ciência e engenharia, permitindo reduzir os custos através de experimentos e simulações computacionais.

Figura 5 – Exemplo de discretização utilizando uma malha triangular baseada no Método de Elementos Finitos



Fonte: <https://www.esss.co/blog/metodo-dos-elementos-finitos-o-que-e>.

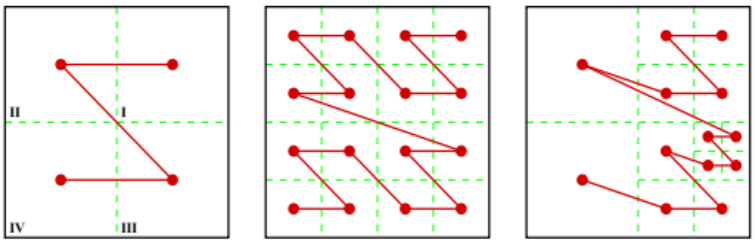
2.4 Curva de Preenchimento de Espaço

Peano (1890) foi o primeiro a definir uma Curva de Preenchimento de Espaço (CPE), que é uma curva que passa por todos os pontos de uma região bidimensional (como, por exemplo, $[0,1]^2$) com conteúdo positivo de área (SAGAN, 2012). Além disso, Valgaerts (2005) define uma CPE como um mapeamento contínuo de um segmento de linha fechado e limitado em P^2 e como cada segmento de linha fechado e limitado é homeomórfico ao intervalo unitário fechado I, pode-se assumir que a curva tenha o domínio I.

Nesse trabalho, as curvas de preenchimento de espaço são consideradas como formas de organizar as *octrees*, utilizadas para linearizar os octantes folhas da *octree* distribuída. As curvas de preenchimento de espaço geram segmentos contíguos percorrendo os octantes sem visitar mais de uma vez cada um. Elas têm uma grande importância no momento do particionamento pois, ao se linearizar a *octree* ou a *quadtree*, é possível criar partições uniformes na lista de octantes ou quadrantes para serem distribuídas entre os processos. As três curvas mais utilizadas são: Ordem de Morton, Ordem de Gray e Ordem de Hilbert (CAMPBELL *et al.*, 2003).

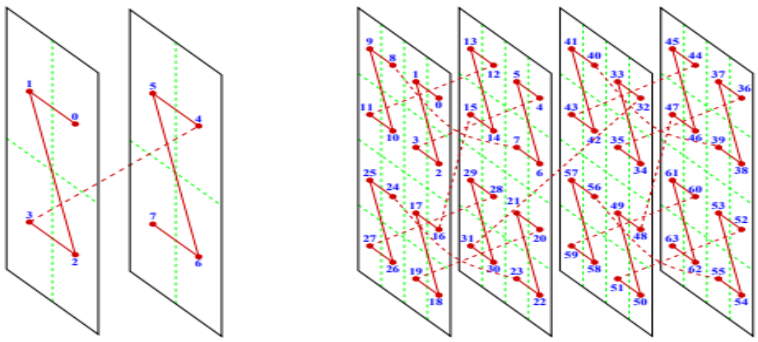
A Ordem de Morton (Ordem-Z ou de Peano) é uma curva de preenchimento de espaço que atravessa todos os filhos de um quadrante com um formato de Z (Figuras 6 e 7). A ordem de Morton é competitiva, simples e serve de base para as outras curvas de preenchimento de espaço.

Figura 6 – Exemplo da Ordem de Morton em uma *quadtree*



Fonte: Campbell *et al.* (2003).

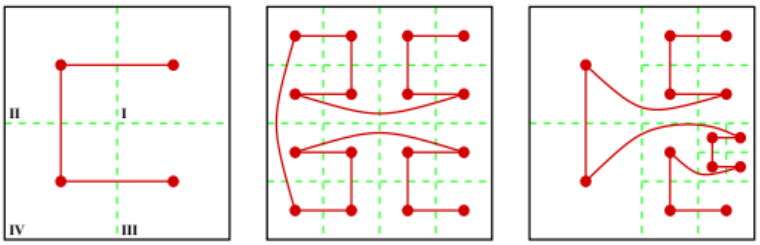
Figura 7 – Exemplo da Ordem de Morton em uma *octree*



Fonte: Campbell *et al.* (2003).

Na Ordem de Gray, os quadrantes são percorridos segundo o "Código de Gray" (Figura 8). A curva percorre os octantes adjacentes que diferem de apenas um bit, 0, 1, 2, 3, o que ficaria 0 (00), 1 (01), 3 (11) e 2 (10). No entanto, a curva resultante é auto-intersectada e não é uma curva de preenchimento de espaço válida.

Figura 8 – Exemplo da Ordem de Gray em uma *quadtree*

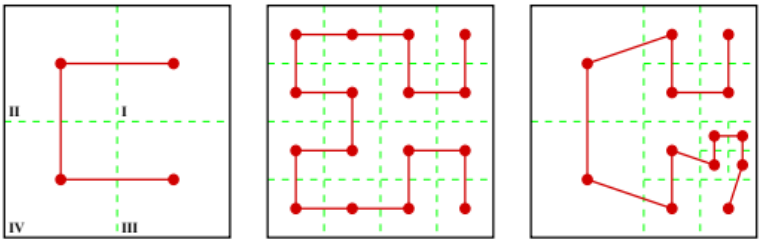


Fonte: Campbell *et al.* (2003).

A Ordem de Hilbert também é utilizada para ordenar quadrantes. É bem parecida com a ordem de Gray, mas apresenta rotações e inversões extras para manter os quadrantes

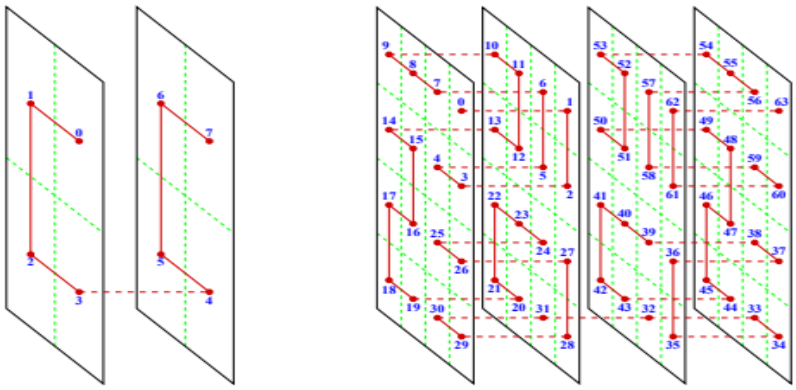
próximos de seus vizinhos (Figuras 9 e 10).

Figura 9 – Exemplo da Ordem de Hilbert em uma *quadtree*



Fonte: Campbell *et al.* (2003).

Figura 10 – Exemplo da Ordem de Hilbert em uma *octree*



Fonte: Campbell *et al.* (2003).

2.5 *Quadtree* Linear

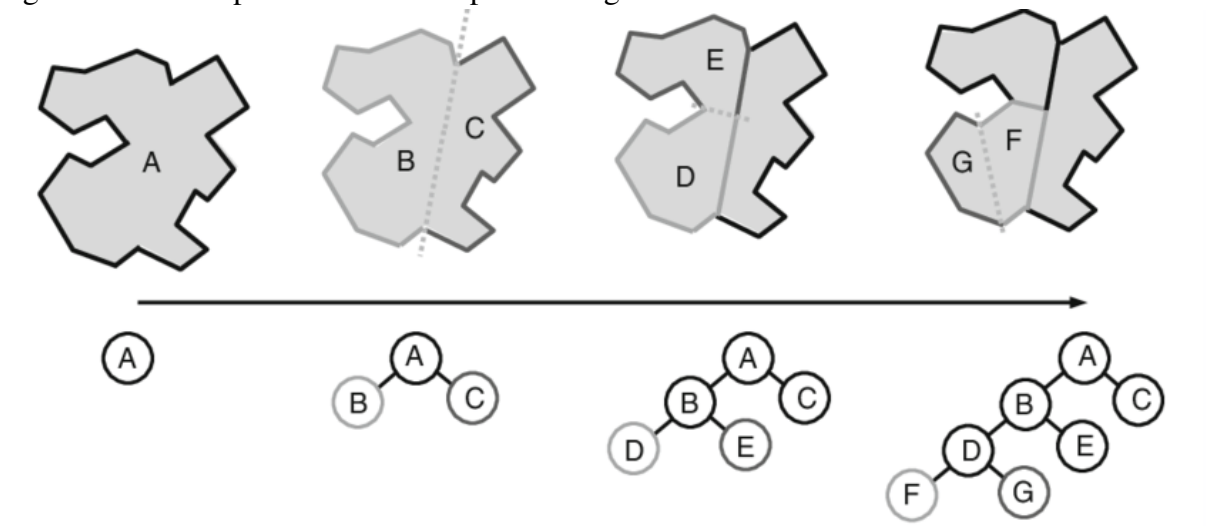
Existem duas maneiras comumente utilizadas de representação de *quadtrees* e *octrees*: em forma de árvore (a forma tradicional) e em forma linear. Na forma linear, os nós folhas da *quadtree* são armazenados em uma lista (um vetor). Essas *quadtrees* recebem o nome de *Quadtrees* Lineares. Nessa representação, os nós internos não são necessários, e são descartados. Assim, utilizar uma curva de preenchimento de espaço nas folhas de uma *quadtree* é uma maneira de linearizá-la.

2.6 *Binary Space Partition* - BSP

Uma BSP (*Binary Spatial Partitioning*) é uma estrutura de dados em forma de árvore binária que subdivide o espaço na metade a cada passo. Em Freitas *et al.* (2016), ela é utilizada

para decompor o espaço de maneira alinhada ao eixo, de forma que a quantidade de carga em cada subdomínio (uma folha da BSP) seja aproximadamente a mesma (Figura 11).

Figura 11 – Exemplo de BSP decompondo a região A



Fonte: (MACHADO *et al.*, 2015).

2.7 Escalabilidade de Tamanho Fixo

A Escalabilidade de Tamanho Fixo verifica o comportamento de uma instância grande de um problema conforme aumenta-se o número de processos. Idealmente, conforme dobra-se o número de processos, o tempo de execução cai pela metade. Caso isso aconteça, afirma-se que a técnica é escalável, e tem *speed-up* linear. Se o tempo de execução cair para menos da metade, diz-se que o *speed-up* é superlinear. Se o tempo de execução não cair muito, ou mesmo se aumentar, diz-se que a técnica não é escalável.

2.8 Escalabilidade Isogranular

A Escalabilidade Isogranular é parecida com a Escalabilidade de Tamanho Fixo. Entretanto, na Isogranular, aumenta-se o tamanho do problema conforme aumenta-se o número de processos. A Escalabilidade Isogranular serve para analisar o comportamento da técnica em uma situação de utilização máxima, em que o problema é tão grande quanto pode ser, para uma determinada quantidade de processos.

3 TRABALHOS RELACIONADOS

3.1 Introdução

Neste capítulo, apresenta-se os trabalhos que, de algum modo, estão relacionados com essa pesquisa. Inicialmente apresenta-se os trabalhos voltados para a construção paralela de *octrees*, *quadtrees* ou ambas. Posteriormente, discute-se os outros trabalhos que complementam o assunto abordado.

3.2 Construção de *quadtrees*/*octrees* em paralelo

Bern *et al.* (1993) é um dos primeiros trabalhos de paralelização de *quadtree*, e que apresenta uma técnica paralela para construção de *quadtree* com o modelo de Gravação Exclusiva e Leitura Exclusiva em Máquina de Acesso Aleatório Paralelo (EREW PRAM), que é uma máquina de memória compartilhada com acesso exclusivo a memória. Foi descrito primeiramente como gerar uma *quadtree* para um conjunto de pontos em um plano. Na quadtree resultante, não há restrições sobre os tamanhos dos quadrantes adjacentes, mas nenhum quadrante-folha contém mais de um ponto.

Primeiramente é realizada a classificação de todos os pontos da entrada pelos valores de suas coordenadas. Posteriormente, é realizada a remoção de todas as folhas dessa árvore que não contenham pontos de entrada, e são contraídos todos os caminhos (bordas dos quadrantes interligados) restantes de cada nó que tenha um filho. Então, é obtida uma árvore em que todos os nós internos que possuem grau dois ou mais. Em seguida, a quadtree completa é construída.

O balanceamento 2:1 é seguido a partir de dois estágios, partindo da *quadtree* desequilibrada. No primeiro estágio, é produzida uma *quadtree* na qual alguns quadrantes não-folha podem ter menos de quatro quadrantes filhos. No entanto, a condição de balanceamento é satisfeita com a inclusão de ponteiros cruzados. No segundo estágio do algoritmo de balanceamento, uma vez que a árvore foi construída, é transformada novamente em uma *quadtree* simplesmente dividindo todos os quadrantes, folhas e nós internos sem os quatro filhos. Isso preserva a condição de balanceamento e restaura o número necessário de filhos por pai. Isso pode levar a *quadtree* balanceada a ter 4 vezes mais filhos que uma *quadtree* balanceada gera por um algoritmo sequencial, por conta da divisão extra.

Esse trabalho apresentou resultados mais teóricos que práticos, tendo realizado um

estudo teórico da geração de malhas bidimensionais paralelas, e tendo demonstrado o uso de técnicas similares aplicadas em algoritmos paralelos. O método de balanceamento desperdiça um fator de quadrante, e outros fatores constantes são perdidos para alcançar a condição de balanceamento.

Yang e Lee (1994), assim como Bern *et al.* (1993), é um trabalho mais teórico que prático. Nesse trabalho, é introduzida a noção de preservação de adjacência multinível (MAP), que caracteriza-se na distribuição dos dados espaciais sobre multiprocessadores interconectados e subjacentes. O trabalho apresenta também um algoritmo paralelo de construção de *quadtree* usando uma árvore binária estendida (*EB-Tree*) com a propriedade MAP. O algoritmo apresentado gera uma *quadtree* para uma imagem bidimensional, e utiliza uma rede de interconexão com a topologia de um hipercubo.

A adjacência espacial desempenha um papel importante no processamento dos dados espaciais, pois muitas operações de aplicações em *quadtrees* possuem esforços extras, como a ordenação, para restaurar as relações de adjacência espacial.

O processo de subdivisão de uma *quadtree* quebra as adjacências espaciais. Espera-se que, sob a adequada atribuição de dados, essas adjacências possam ser pré-atendidas implicitamente na rede de interconexão subjacente para preservar a adjacência espacial. Para cada nível de uma quadtree, se dois nós forem espacialmente adjacentes, seus processadores correspondentes terão um *link* direto no hipercubo. Um esquema de atribuição de dados que satisfaça a propriedade descrita é chamado de preservação de adjacência de multinível (MAP). Nem toda atribuição é de preservação de adjacência multinível. Por exemplo, na Ordem de Morton, embora dois quadrantes sejam adjacentes no mesmo nível, seus processadores correspondentes podem não ser adjacentes, porque seus rótulos podem ter dois bits de diferença.

O primeiro passo para realização da construção é fazer o *download* do domínio entre os processos. O esquema pixel a pixel é muito custoso, então o domínio (imagem) é baixado em um processo, que realiza a sua subdivisão em quatro partes (usando uma *quadtree*), retendo uma parte (quadrante) no processo atual e enviando os outros três quadrantes para três outros processos vizinhos. Conceitualmente, esse esquema de *download* pode ser modelado como uma árvore binária estendida (*EB-Tree*).

Para ter a propriedade MAP, é necessária uma função de alocação, que atribui um código de localização a cada nó da árvore. Esta função de localização será usada posteriormente para controlar a atribuição de dados espaciais em cada subdivisão. Por conveniência, seja o

conjunto de código de localização L = 00, 01, 10, 11 tal que 00, 01, 10 e 11 são códigos de local de noroeste, nordeste, sudoeste e sudeste, respectivamente.

O algoritmo de construção paralela da *quadtree* pode ser descrito da seguinte forma. Inicialmente, cada processador no hipercubo irá definir seu nível para n (nível do nó da folha que o processador correspondente representa), definir sua cor (Preto ou Branco) para a cor do pixel contida no processador e definir seu tipo (*tag* que marca se o processador é um processador de chaves de um nó folha ou não: Y representa um processador de chaves e N, um não processador de chaves) para N. Além disso, define-se o nível de processamento atual cp = n. Para cada processador, as etapas a seguir são executadas em paralelo. Cada processador envia sua cor para o processo pai, em um nível acima na árvore. O processador pai, por sua vez, verifica se as cores recebidas de seus filhos, se são todas pretas ou todas brancas. Caso não sejam todas da mesma cor, o processador pai envia uma "chave" de cor Cinza para os seus filhos. Ao receber a "chave" de cor Cinza, os nós filhos definem o seu tipo para Y. O processo é repetido avançando-se o nível até chegar ao nível máximo da *EB-Tree*.

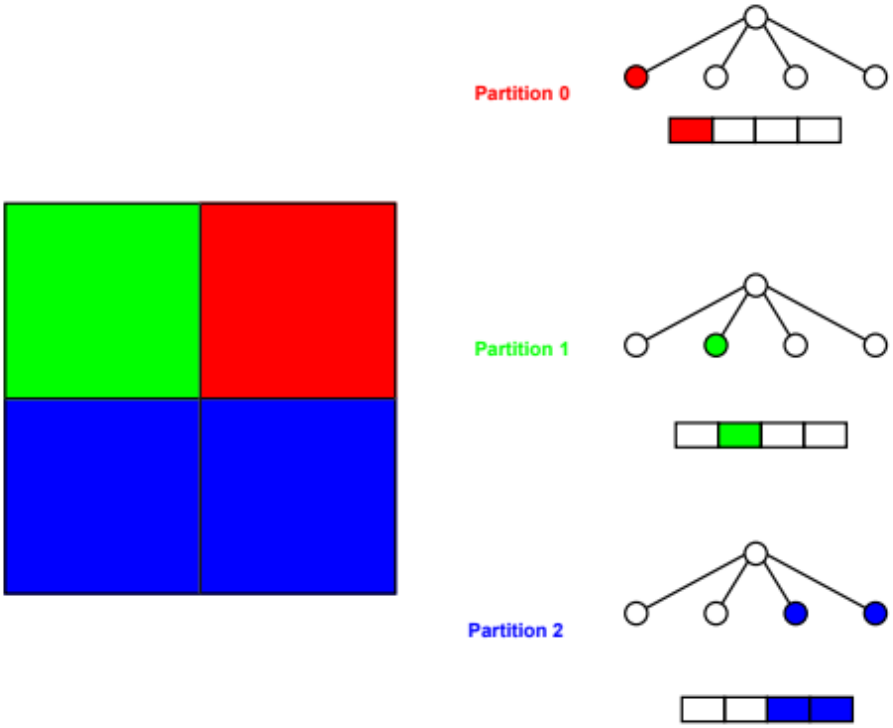
Algumas análises de desempenho e testes empíricos são apresentados para mostrar que o algoritmo de construção da *quadtree* é melhor que os resultados anteriores a este trabalho. Diante dessas análises e testes, foram apresentadas três vantagens em relação a trabalhos anteriores. A primeira vantagem é que nesse algoritmo a adjacência espacial é preservada, não há ordenação extra paralela ou cálculo de soma de prefixo. A segunda vantagem apresentada é que os ponteiros da *quadtree* são incorporados implicitamente nos *links* do hipercubo, não são necessários registros explícitos para pais e filhos. Além disso, nenhum nó interno é armazenado explicitamente. A terceira vantagem é que quaisquer dois processadores que tenham que se comunicar durante a execução do algoritmo de construção terão uma distância máxima de comunicação de dois saltos.

Em Campbell *et al.* (2003), é apresentado um algoritmo para construção de uma octree em paralelo, comparando três curvas de preenchimento de espaço: a Ordem de Morton, a Ordem de Hilbert e a Ordem de Gray.

O trabalho define que, para a partição da *octree*, cada subárvore tem um peso, que pode ser o número de elementos contidos em cada subárvore. É realizada uma primeira travessia de profundidade para determinar todos os "custos" de uma subárvore. Assumindo que os custos são determinados pela quantidade de elementos, é, então, determinado um tamanho de uma partição ideal, definido por OP = TC/NP, em que OP é a partição ótima, TC é o custo total e NP o

número de processos (TC e NP são conhecidos). Na segunda travessia, são adicionados octantes à partição atual, desde que não se exceda o valor de OP. Caso OP seja excedido, é necessário descer na árvore e continuar. Como os octantes folhas são indivisíveis, caso não seja possível a adição de novos octantes, a partição é deixada sub-cheia, e começa-se a trabalhar na próxima partição (Figura 12).

Figura 12 – Particionamento de três vias de nós de folha da árvore global. Nós extras são designados para processos de números mais altos. Cada processo tem uma entrada de matriz de mapa para cada quadrante global.



Fonte: Campbell *et al.* (2003).

O trabalho afirma que, para realizar a construção de uma octree em paralelo, deve-se atentar para alguns fatores: o octante deve saber informações do seu espaço, seus pais, seus filhos e, em alguns casos, o processo que o contém e os processos que contêm seu pai e seus filhos. Os processos podem ficar sobrecarregados com tanta troca de informações.

O algoritmo é descrito da seguinte maneira. Primeiramente, é calculado o centróide de cada elemento, e o domínio definido através de uma caixa delimitadora, o nó raiz, que é criado em todos os processos. Todos os processos refinam esse nó até um certo nível, definido *a priori*. Ao final da criação inicial, cada processo possui uma instância idêntica da *octree*. Enquanto a octree se encontra no seu estado global inicial, é fácil inserir novos octantes nela. Uma matriz de mapa representa um *octree* global por inteiro em uma forma linear. Depois de ser realizado o

processo de linearização, os níveis superiores não são mais necessários.

Em seguida, é realizado o cálculo dos octantes folhas em cada processo. Se a divisão não for uniforme, é necessário alocar os octantes restantes em uma nova partição que tenha, no máximo, um octante a mais que o limite permitido. Quando inserido um objeto na *octree* e o limite por bloco é excedido, tem-se a necessidade de refinar o octante. O objeto pode residir em um processo diferente do qual se encontra o seu octante de destino. Esses objetos são chamados de órfãos, e devem ser migrados para o processador ao qual eles pertencem.

Após a inserção de novos objetos em um processo, realiza-se o cálculo novamente da carga, pois a partição pode ficar inválida. Depois, é feito o cálculo da carga global e do prefixo, o que determina a posição do processo na cadeia de processos. Um processo passa os objetos em excesso para os processos posteriores sem necessitar de comunicação, apenas utiliza um contador que determina a carga do processo para o qual estão sendo enviados os objetos. Quando o número de objetos recebidos pelo processo excede o número máximo permitido OP, os objetos que ultrapassam são passados a frente, até chegar ao último processo, que, mesmo que ultrapasse o seu OP, deve ficar com esses objetos.

Para a obtenção dos resultados, o trabalho realiza uma avaliação de desempenho de carga de octree usando malhas tetraédricas. Foi avaliada a qualidade da malha e foi medido o tempo de execução para uma Escalabilidade de Tamanho Fixo. Foram utilizadas quatro malhas para a realização do estudo: Cone, com uma malha de 42.786 elementos; Focinho, com uma malha de 169.733 elementos; Asa, com malha de 85.567 elementos; e Artéria com malha de 1.103.018 elementos.

Para medir a qualidade da partição, o trabalho apresenta métricas de índice de superfície e de adjacência interprocesso, utilizando as três curvas de preenchimento de espaço: a Ordem de Morton, a de Gray e a de Hilbert. Foi utilizado um computador IBM com suporte a 56 processos. A Ordem de Hilbert geralmente atinge os melhores índices de superfície média e máxima para três malhas (Cone, Asa e Artéria), seguida pela Ordem de Morton e a Ordem de Gray. Pode-se ver que, após o refinamento e repartição, a Ordem de Hilbert produz índices de superfície superiores para a malha do focinho também.

Para demonstrar o desempenho dinâmico do balanceamento de carga, foi feita a resolução de um problema de elementos finitos. Foram mostradas as métricas de qualidade da partição para cada reequilíbrio do problema do tubo de choque perfurado em 28 e 56 processos, usando as Ordens de Morton e de Hilbert (os testes realizados com a Ordem de Gray produziram

decomposições mais pobres e não foram incluídos). Os casos dos índices de superfície para os casos de 28 e 56 foram, respectivamente, marginalmente e aproximadamente 10% melhores com a Ordem de Hilbert do que com a Ordem de Morton. Em ambos os casos a Ordem de Hilbert atinge uma melhor conectividade entre os processos e, para o caso com 56, obteve uma diferença de 25%.

Comparativamente, a curva de Hilbert tem um custo semelhante às curvas de Gray e de Morton, mas apresenta um melhor desempenho, pois leva a partições de melhor qualidade. Geralmente, as métricas de qualidade para as travessias são classificadas pelas descontinuidades evidentes nas ordens, sendo a Ordem de Hilbert a melhor, seguida pela de Morton e pela de Gray. Entretanto, a extensão dessas diferenças é altamente dependente da estrutura da malha, necessitando um estudo mais aprofundado em relação às curvas de preenchimento de espaço. Apesar deste trabalho demonstrar que a Ordem de Hilbert se sobressaiu sobre as demais, os trabalhos apresentados a seguir utilizam a Ordem de Morton.

Para contemplar os requisitos da geração de malha em ambientes paralelos, o trabalho descrito em Tu *et al.* (2005) desenvolveu os seguintes algoritmos: propagação de onda prioritária (que equilibra de forma eficiente uma *octree* em paralelo), um rebuscado sistema de gerenciamento de memória (a fim de prover ganhos na troca de dados entre os processos), e um extrator de malhas. A aplicação final foi chamada de *Octor*, que apresentou uma boa Escalabilidade Isogranular e de Tamanho Fixo.

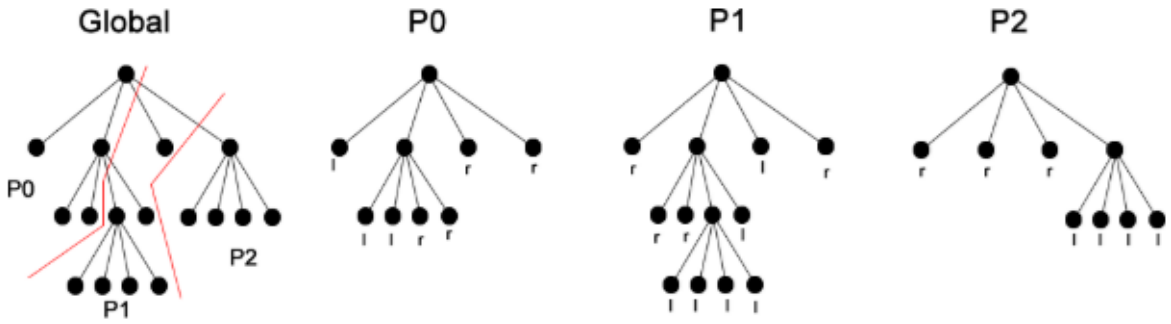
A *Octor* consiste nas seguintes etapas. Primeiramente, o *NEWTREE* cria uma pequena octree em cada processo. Em seguida, essa árvore pode ser ajustada com *REFINETREE* e *COARSENTREE*, de maneira estática ou dinâmica. O *BALANCETREE* realiza o balanceamento 2:1 em paralelo. Então, o *PARTITIONTREE* distribui os octantes folhas entre os processadores. Por fim, o *EXTRACTMESH* obtém informações de cada elemento e nó de malha e determina as correlações entre eles. Através das etapas apresentadas, é possível gerar dois tipos de malhas: as estáticas, nas quais pode ser dispensável a utilização do *COARSENTREE*; e as dinâmicas, nas quais pode ser desnecessária a utilização de *PARTITIONTREE*, se o número de octantes folhas nos processadores diferir apenas em uma pequena quantidade.

Nesse trabalho, também foi utilizada uma *octree* linear baseada em uma curva de preenchimento de espaço: a Ordem de Morton.

Para se ter o paralelismo de dados, com a divisão do *octree* entre os processadores, cada processo mantém sua instância local da octree global subjacente. Cada instância local é

uma *octree* por si só, e os seus nós folhas são marcados como locais ou remotos, como na Figura 13 mostrada abaixo.

Figura 13 – Organização de quadtree em paralelo em três processos (P0, P1 e P2). Os quadrantes marcados com o rótulo l são denominados locais e os marcados com r são os remotos.



Fonte: ROCHA F. W. C. (2018).

Como pode-se ver, o particionamento é feito de forma muito parecida a um percurso em pré-ordem da árvore, dada pela Ordem de Morton. Para manter a consistência da árvore, um nó marcado como local em um processador não pode ser marcado dessa forma em nenhum outro. Portanto as instâncias são únicas para cada processo.

O trabalho mostrou uma análise de Escalabilidade Isogranular e de Tamanho Fixo. Na Escalabilidade Isogranular, a diferença da folha no maior nível para a folha no menor nível é de 64 vezes. Devido à multi-resolução da malha, existe uma grande quantidade de nós pendentes na malha (11% e 20%). O tempo de execução aumenta logaritmicamente com o tamanho do problema, e o *PARTITIONTREE* é o principal responsável pelo aumento tempo de execução. A *Octor* não atinge o desempenho isogranular ótimo teórico, mas a tendência do tempo sugere que a aplicação possui uma boa escalabilidade e nenhuma das etapas se torna um gargalo para a aplicação. Fazendo uma análise da porcentagem do tempo aproximado de execução (para 2000 processos) de cada algoritmo, tem-se:

- *REFINETREE* dura cerca 15% do tempo.
- *BALANCETREE* corresponde a cerca de 13% do tempo.
- *PARTITIONTREE* executa em 68% do tempo.
- *EXTRACTMESH* dura cerca de 4% do tempo.

Na Escalabilidade de Tamanho Fixo, foram utilizados 3 problemas, de tamanhos pequeno, médio e grande: o problema pequeno varia de 1 até 16 processos; o problema médio de

8 até 128 processos; e o problema grande de 128 até 2000 processos.

Embora não esteja exibindo a escalabilidade ideal, a Octor consegue apresentar um bom tempo de execução para um grande número de processadores.Fazendo uma análise da porcentagem do tempo aproximado de execução (para 2000 processos) de cada algoritmo, tem-se:

- *REFINETREE* dura cerca 18% do tempo.
- *BALANCETREE* corresponde a cerca de 14% do tempo.
- *PARTITIONTREE* executa em 58% do tempo.
- *EXTRACTMESH* dura cerca de 10% do tempo.

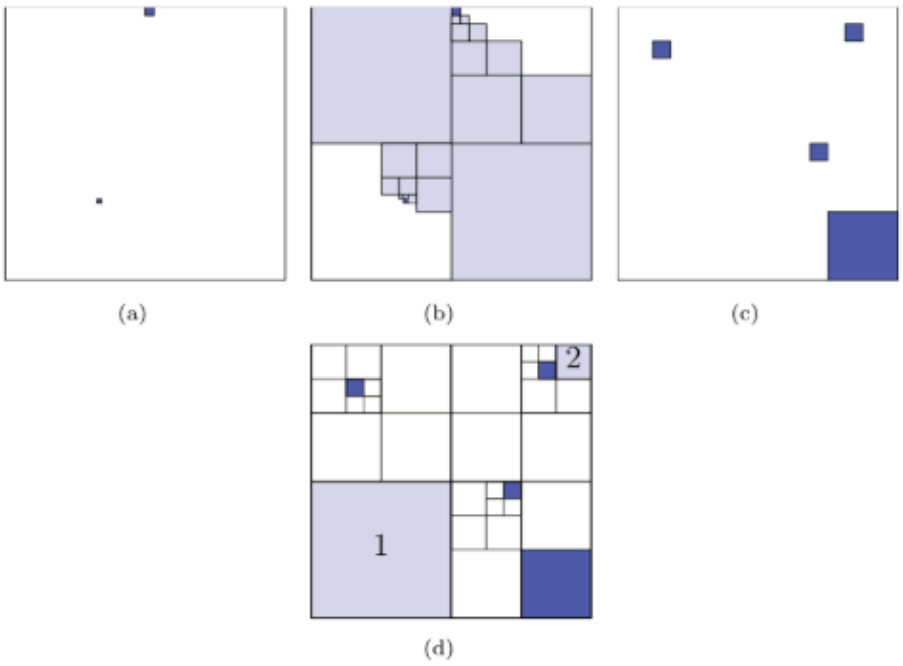
Em Sundar *et al.* (2008), é apresentada uma abordagem de construção de baixo para cima (*bottom-up*), com complexidade de tempo O(N/np log np log (N/np) + np log np) em paralelo e de armazenamento O(N). Essa técnica é diferente de uma abordagem tradicional, de cima para baixo (*top-down*), que consiste em realizar a subdivisão recursiva partindo nó raiz até os nós folhas no nível máximo da árvore. A abordagem *bottom-up* consiste em construir os nós internos da árvore a partir dos nós folhas. Geralmente esta construção é feita a partir de uma *octree* (ou *quadtree*) linear.

O trabalho determina que o principal componente para construção de octrees é o particionamento da entrada, a fim de conseguir um bom balanceamento de carga. O uso de curvas de preenchimento de espaço como a Ordem de Morton (usada em seu trabalho), são bastante comuns para este propósito. O algoritmo descrito tem como finalidade evitar problemas de sincronização, normalmente associados a algoritmos paralelos de construção *top-down* de *octrees*. Ele também apresenta custos de armazenamento menores do que os de outras representações, por utilizar somente os nós folhas para construção e por não usar ponteiros, que adicionam custos indiretos de sincronização e comunicação para implementações paralelas.

O algoritmo pode ser definido em cinco passos para construir uma árvore octree. O primeiro passo consiste em transformar um conjunto de pontos (com suas coordenadas, x, y e z) em octantes, de acordo com uma profundidade máxima definida. Esta profundidade é um limite superior para a profundidade máxima da árvore. No segundo passo, é realizada a classificação dos octantes folhas, ou seja, é feita a linearização usando uma curva de preenchimento de espaço, a Ordem de Morton. O terceiro passo é distribuir os octantes de maneira paralela em um sistema de memória distribuída, realizando o particionamento da lista ordenada de octantes de modo que a carga seja dividida uniformemente entre os processos. O quarto passo consiste em selecionar

o menor e o maior octante em cada processo e construir um árvore entre eles. O quinto passo realiza a construção da *octree* linear completa particionado os octantes entre os processos de maneira balanceada, completando as lacunas, construindo uma *octree* mínima entre cada octante e, posteriormente, gerando o restante da *octree* segundo o critério do número máximo de pontos por octante (Figura 14).

Figura 14 – (a) Menor e maior octante folha contido no processo, de acordo com a Ordem de Morton. (b) O número mínimo de octantes entre as células dadas em (a). (c) Octantes mais grosseiros selecionados em (b). A quadtree linear completa menos refinada possível contento todas as células em (c).



Fonte: Sundar *et al.* (2008).

Para demonstrar a boa escalabilidade deste algoritmo, o trabalho mostra testes de Escalabilidade Isogranular (para distribuições: gaussianas, log-normal e regular) e de Tamanho Fixo. Pôde-se constatar que, ao manter o tamanho do problema por processador (relativamente) constante à medida que o número de processadores é aumentado (Escalabilidade Isogranular), foram identificados problemas de comunicação relacionados ao tamanho e frequência das mensagens, bem como reduções globais e problemas com escalabilidade algorítmica.

Na Escalabilidade Isogranular, a distribuições gaussiana e log-normal mostraram-se desequilibradas pois, em média, o número de octantes desequilibrados é três vezes maior que o número de pontos de entrada e o número de octantes duplica após o balanceamento. Já na

distribuição regular, foi observado que o número de octantes se assemelha ao número de pontos da entrada. As parcelas dos resultados obtidos demonstram uma boa escalabilidade, conseguindo alcançar uma escalabilidade quase que ideal para as três distribuições.

Para a análise da Escalabilidade de Tamanho Fixo, utilizou-se os resultados da distribuição gaussiana para os tamanhos de 1M, 32M e 128M, e foi medida a relação de tempo entre as diferentes quantidades de processadores. Dessa forma, divide-se o tempo com menos processadores pelo tempo com mais processadores. Por exemplo, para analisar a escalabilidade com 2 processadores, divide-se o tempo de 1 processadores pelo tempo de 2 processadores. Caso seja maior que 2, a escalabilidade é considerada super-linear; caso seja igual a 2, é linear; caso seja muito abaixo de 2, é considerada ruim. No geral, na maioria dos casos o algoritmo apresenta uma boa Escalabilidade de Tamanho Fixo.

Em Camata e Coutinho (2010), é apresentada uma técnica bottom-up utilizando a Ordem de Morton para a construção da octree baseada na abordagem proposta por Sundar *et al.* (2008).

O algoritmo de construção da *Octree* consiste na geração de uma *octree* linear onde uma lista de octantes O é igualmente distribuída entre os núcleos de processamento. O ponto inicial do algoritmo consiste na determinação do número corrente de processos participantes da geração (p) e o identificador do processo (*rank*). Com base nessas informações, o algoritmo determina o número mínimo de octantes N que podem ser distribuídos entre os processos, além de determinar o nível inicial da árvore. Tomando como base o número total de octantes, cada processo calcula os códigos de localização do primeiro (IDa) e do último (IDb) octantes que pertencem a ele. Com a posse de seus códigos de localização, é possível obter as coordenadas dos dois octantes OCTa e OCTb.

Com esses octantes, OCTa e OCTb, e sabendo-se que OCTb > OCTa, gera-se o número mínimo de octantes que abrangem a região entre OCTa e OCTb de acordo com a Ordem de Morton. O algoritmo primeiramente obtém o ancestral comum mais próximo dos octantes OCTa e OCTb. Este octante é subdividido em seus oito filhos. Desses, apenas os octantes que são ou maiores que OCTa e menores do que OCTb ou ancestrais de OCTa são mantidos, e os restantes são descartados. Os ancestrais de OCTa ou OCTb são recursivamente subdivididos até que não haja mais necessidade de subdivisões. Esse processo produz uma lista completa de octantes entre os dois octantes OCTa e OCTb. Assim, com a união das listas de cada processo, chega-se a uma *octree* linear completa. Nota-se que o algoritmo é baseado justamente nas

propriedades da Ordem de Morton. A complexidade computacional deste algoritmo é O(n log n), onde n é o tamanho da lista O .

Foi realizada uma análise da Escalabilidade Isogranular da geração da *octree* para dois modelos, variando o número de processadores de 8 a 128. Nos resultados obtidos das medições, pode-se perceber que a técnica apresenta uma escalabilidade razoável. Houve um aumento significativo de tempo no aumento do nível 9 para o 10 e do número de processadores 64 para 128 em ambos os modelos. Porém houve um aumento mais que considerável no número de octantes no aumento do nível.

As porcentagens para o tempo de execução dos componentes principais para o balanceamento são: para a partição local e o balanceamento local, cerca de 20% à 30%; para o efeito cascata do balanceamento 2:1, cerca de 10%; e para a comunicação entre processadores vizinhos, aproximadamente 5%. Após ser feito o balanceamento 2:1, a *octree* encontra-se em um número de octantes desproporcional entre os processos, e é então particionada, levando em média 15% do tempo.

Em Burstedde *et al.* (2011), uma técnica para a construção de florestas de *octrees* é apresentada. Os algoritmos mostrados têm por finalidade contribuir para uma aplicação paralela escalável respeitando a codificação e o armazenamento da floresta de *octree*, assegurando o que se espera, que é a restrição de balanceamento 2:1. Foi também desenvolvida uma soma de verificação da floresta independente da partição. Os principais aspectos para realizar essas eficiências são: um esquema de codificação flexível da conectividade entre octrees, que permite orientações relativas arbitrárias de nós em duas e três dimensões, e a identificação periódica e não periódica mista de limites da *octree*. As conectividades (de faces, arestas e vértices) entre octantes internos e externos ao domínio são utilizadas para passar informações de vizinhança entre os limites dos octantes. Por fim, a Ordem de Morton é usada para determinar com eficiência o processo proprietário dos pares de octante-octante.

Burstedde *et al.* (2011) adota algumas ideias importantes introduzidas nas octrees: o armazenamento das folhas em uma matriz e dois estágios para o balanceamento 2:1. No armazenamento, as octantes folhas são guardadas em uma matriz (vetor), denominada de octree linear. O balanceamento 2:1 é feito em dois estágios: o primeiro no momento do balanceamento local e o segundo depois de se receber octantes compartilhados e de uni-los com os locais. Assim como o trabalho proposto por Sundar *et al.* (2008), não é realizada a partição local de blocos grosseiros.

Esse trabalho define que os octantes dentro de uma *octree* podem ser atribuídos a uma ordenação natural por uma travessia em todas as folhas. Pela equivalência de nós de árvores e octantes, essa sequência unidimensional corresponde a uma curva de preenchimento da Ordem de Morton. Esse conceito pode ser estendido a uma floresta de *octrees*, conectando a curva de preenchimento de espaço entre as *octrees*, gerando uma ordem global de octantes (Figura 15). Logo, uma partição paralela é criada a partir da divisão da curva em segmentos de acordo com a quantidade de processos. Assim, a macroestrutura da floresta é estática e compartilhada por todos os processos, e a quantidade de octantes é limitada pela quantidade de memória local em implementações paralelas em memória distribuída.

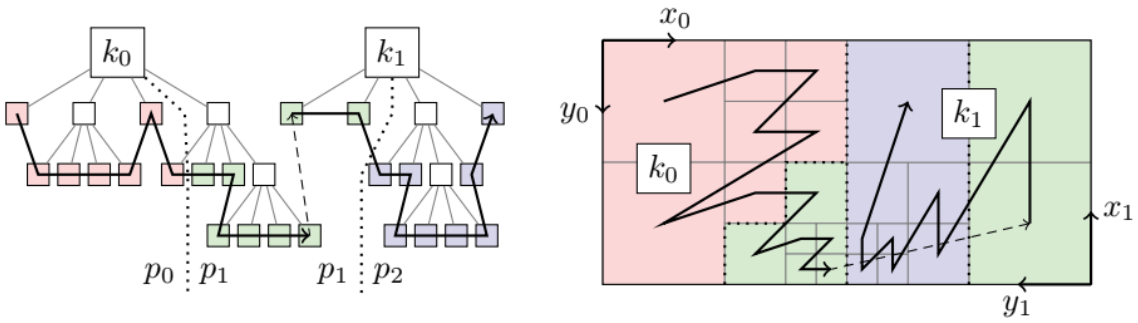
Os algoritmos apresentados de maneira geral são:

1. Criar uma nova floresta igualmente particionada e uniformemente refinada.
2. Subdividir adequadamente octantes com base em um marcador de refinamento, uma vez ou recursivamente.
3. Substituir as famílias de oito octantes filhos pelo seu octante comum dos pais, uma vez ou recursivamente.
4. Redistribuir os octantes em paralelo, de acordo com um determinado número alvo de octantes para cada processo ou pesos prescritos para todos os octantes.
5. Fazer o Balanceamento 2:1, certificando-se que octantes vizinhos tenham relações de tamanho no máximo 2:1.
6. Recolher uma camada de octantes fora do processo, mas tocando os limites da octree do lado de fora.
7. Criar uma enumeração globalmente exclusiva dos vértices nos cantos de octantes, levando em consideração a classificação em nós "não-independentes"(que não respeitam o balanceamento 2:1) e "independentes"(que respeitam o balanceamento 2:1).

O trabalho apresenta estudos para de Escalabilidade Isogranular e de Tamanho Fixo do seu algoritmo, e também mediu a qualidade da partição paralela em termos de faces e processos paralelos.

Para a Escalabilidade Isogranular, foi utilizada uma floresta de 6 octrees. Os cálculos analisados foram realizados em um supercomputador (Jaguar XT5) com 12 núcleos por nó, os resultados da porcentagem do tempo de execução obtidos foram: O tempo dos Algoritmos 1, 2 e 4 é praticamente insignificante (ocupam cerca de 10% do tempo). O Algoritmo 3 leva tanto tempo quanto o Algoritmo 4 . Os Algoritmos 5 e 7 custam cerca de 90% do tempo de

Figura 15 – Correspondência um-para-um entre uma floresta de octrees (à esquerda) com duas octrees k_0 e k_1, e um domínio geométrico particionado em elementos (à direita). Para cada octree, a Ordem de Morton segue a orientação de seus eixos coordenados, gerando uma partição uniforme entre três processos (p_0, p_1 e p_2).



Fonte: Burstedde *et al.* (2011).

execução. Além da porcentagem dos tempos, pode-se constatar um aumento suave do tempo de execução, de aproximadamente 6 segundos (com 12 processos) para 8 a 9 segundos (com 220.320 processos). A eficiência para o Algoritmo 5 é de 65% e a do Algoritmo 7 é de 72%. Portanto, para este caso, uma escala ideal resultaria em tempos constantes, que não é o caso. Porém, a diferença entre os tempos é pequena.

Para a Escalabilidade de Tamanho Fixo, foi utilizada uma floresta com 13 octrees. A cada refinamento, aumenta-se o número de octantes em um fator de 8 e, portanto, foram necessários vários refinamentos para se atingir um octante por processador. Comparando com a utilização do Algoritmo 5, esse procedimento é muito custoso, devido às múltiplas rodadas dos ciclos com o número de processos vazios. A escalabilidade foi considerada boa para uma grande quantidade de processos para problemas com três frequências diferentes. Para a análise da qualidade da partição, foi tomada como base a medição do número de faces vizinhas entre processos diferentes. Uma pequena relação entre as contagens máximas e médias indica uma partição paralela bem equilibrada. O trabalho obteve uma contagem baixa.

Com todos esses resultados, pode-se constatar que a Floresta de *Octrees* é uma abordagem eficaz e confiável para particionamento em grande escala.

3.2.1 Comparativos

Foram realizados comparativos entre as abordagens apresentadas na Seção 3.2 (entre elas e entre a solução proposta). De maneira geral, montamos duas tabelas para expressar as características encontradas em cada uma. Na Tabela 1, expressa as características em termos de

estruturas de dados encontradas em cada uma. A Tabela 2 apresenta as característica de testes de escalabilidade com o número de processos utilizados para realizar tais testes.

Tabela 1 – Comparativos entre as abordagens para construção de *quadtrees*/*octrees* em paralelo.

Autores	**Quadtree**	**Octree**	**Ordem de Morton**	**Outras Curvas**	**Usa BSP**
Bern *et al.* (1993)	Sim	Não	Não	Sim	Não
Yang e Lee (1994)	Sim	Não	Não	Não	Não
Campbell *et al.* (2003)	Sim	Sim	Sim	Sim	Não
Tu *et al.* (2005)	Não	Sim	Sim	Não	Não
Sundar *et al.* (2008)	Sim	Sim	Sim	Não	Não
Camata e Coutinho (2010)	Sim	Sim	Sim	Não	Não
Burstedde *et al.* (2011)	Sim	Sim	Sim	Não	Não
Proposta	Sim	Sim	Não	Não	Sim

Fonte: Produzida pelo autor.

Tabela 2 – Comparação de escalabilidade apresentada pelas abordagens para construção de *quadtrees*/*octrees* em paralelo.

Autores	**Nº de processos**	**Escalabilidade de Tamanho Fixo**	**Escalabilidade Isogranular**
Bern *et al.* (1993)	Não avaliado	Não avaliada	Não avaliada
Yang e Lee (1994)	Não avaliado	Não avaliada	Não avaliada
Campbell *et al.* (2003)	56	Boa	Não avaliada
Tu *et al.* (2005)	2000	Boa	Boa
Sundar *et al.* (2008)	1024	Boa	Boa
Camata e Coutinho (2010)	128	Não avaliada	Razoável
Burstedde *et al.* (2011)	220.320	Boa	Boa
Proposta	Não avaliado	Não avaliada	Não avaliada

Fonte: Produzida pelo autor.

3.3 Algoritmos paralelos usando BSP

Em Freitas et al. (2016), é apresentada uma técnica paralela para geração de malhas tetraédricas tridimensionais pelo método de avanço de fronteira, utilizando de um modelo paralelo para subdividir o domínio em duas partes da maneira mais uniforme possível. A subdivisão do domínio é realizada ao meio com o uso uma estrutura de dados de particionamento espacial binário (BSP), com o auxílio de uma octree para estimativa de carga.

Na construção com uma BSP é executada uma busca binária recursiva, sobre a octree, para a melhor localização do plano de decomposição para subdividir uma célula BSP, ou seja, a

localização onde os dois novos subdomínios têm aproximadamente a mesma estimativa de carga. Isso é feito para os três eixos globais (X, Y e Z) e o melhor é escolhido. Para quebrar os laços, para cada eixo, a carga da interface que conecta os dois subdomínios também é estimada, e o eixo com a interface mais leve é escolhido.

Decomposição em um eixo:

- Inicialmente, um cubo, igual à raiz da octree, é construído como a raiz do BSP. Colocando o plano de decomposição no centro dessa célula, no eixo X.
- Se as cargas dessas células BSP tiverem o mesmo valor, essa posição é a melhor para o eixo X. Caso contrário, a criança mais pesada é detectada e o plano de decomposição é movido para o seu centro, no mesmo eixo. Transferindo células da célula mais pesada para outra.

Esse procedimento é executado recursivamente e possui dois critérios de parada: as cargas dos dois subdomínios são iguais; ou é atingida uma folha da octree, isto é, não é mais possível subdividir.

Esse trabalho apresenta uma grande dependência com a estimativa de carga. Nos modelos testados, a carga foi bem estimada, apresentando erros abaixo de 10%, levando a um bom balanço de carga. No entanto, o tempo gasto em gerar a estimativa de carga de forma seqüencial pelo tempo gasto para gerar a octree em uma situação paralela ideal, pode vir a ser um problema para modelos muito grandes.

3.4 Considerações finais

Diante do que foi exposto, pode-se constatar que todas as abordagens apresentam uma boa escalabilidade e o uso de curvas de preenchimento de espaço é bastante comum para o particionamento dos dados da entrada. Também pode-se destacar: o número de processos que o trabalho de Burstedde *et al.* (2011) utiliza; a análise das curvas de preenchimento de espaço feita no trabalho de Campbell *et al.* (2003), que indicou que a Ordem de Hilbert se saiu melhor nos modelos testados; e o uso de uma outra estrutura de dados em Yang e Lee (1994) para o particionamento da *quadtree*, a *EB-Tree*.

Em comparativo com a solução aqui proposta, as técnicas apresentadas utilizam curvas de preenchimento de espaço, porém os dados linearizados são divergentes. No caso estudado neste trabalho, a curva de preenchimento de espaço seria aplicada sobre os pontos, os quais representam a carga a ser distribuída e, nos outros trabalhos, a curva de preenchimento de

espaço é aplicada sobre os octantes. Outra diferença é o uso de uma outra estrutura de dados, a BSP, para o particionamento dos dados da entrada (octantes) baseado nos pontos do modelo.

A técnica proposta nesse trabalho parece mais adequada ao método de Freitas *et al.* (2016) por que também utiliza uma BSP para decompor o domínio, essa decomposição permanece alinhada aos eixos cartesianos, e faz uso da carga da interface entre dois subdomínios vizinhos. A principal diferença está na maneira como a estimativa carga é calculada pois, nesse trabalho, essa estimativa baseia-se no número de vértices contidos em cada octante, e não no número de octantes, como em Freitas *et al.* (2016).

4 METODOLOGIA

4.1 Introdução

O foco central desse trabalho, é desenvolver uma técnica de construção em paralelo de *quadtrees* (que é de fácil conversão para a abordagem tridimensional) baseado no uso de uma *Binary Space Partitioning* - BSP, para atender a técnica de geração de malhas em paralelo proposta em Freitas *et al.* (2016). Essa abordagem é destinada principalmente à sistemas de memória distribuída. Porém, é possível a sua utilização em memória compartilhada. Essa nova abordagem busca oferecer uma distribuição de carga uniforme, um bom *speed up* (embora não seja o foco deste trabalho), uma maximização da troca de mensagens no procedimento de geração da malha. A solução aqui proposta é baseada na técnica de particionamento usando uma BSP para a geração de malha paralela de Freitas *et al.* (2016).

Para construir uma *quadtree* a partir de um dado modelo, primeiro deve-se criar uma caixa delimitadora (o nó raiz) sobre todo o domínio. Posteriormente, o nó raiz é subdividido recursivamente em 4 nós filhos de mesmo tamanho, até que não seja necessária a realização de mais divisões. Alguns trabalhos, como Sundar *et al.* (2008), usam o número máximo de pontos permitidos em cada octante como critério de parada. Já Freitas *et al.* (2016) determina que será dividida até que o tamanho de um lado (do quadrante que contém o ponto médio da aresta testada) seja menor que o tamanho desse lado multiplicada por um fator que determina o quão refinada deve ser a *quadtree* gerada.

4.2 Metodologia da pesquisa bibliográfica

O trabalho seguiu os preceitos de um estudo exploratório, com uma abordagem qualitativa tomada sobre a análise de documentos (artigos).

Nessa perspectiva, a proposta foi utilizada em 4 macro atividades: **Busca por Fontes**, **Coleta de Dados**, **Análise dos Resultados** e **Interpretação dos Resultados**.

4.2.1 Busca por Fontes

Nesta atividade, foi realizada a busca pelas fontes que fornecem respostas adequadas para a resolução do problema proposto. Até o momento, foram selecionados 7 artigos que apresentam soluções plausíveis. Para a seleção destas fontes, foram considerados trabalhos

que abordassem a construção de *quadtrees*/*octrees* em paralelo, preferencialmente que fossem destinados ao paralelismo em memória distribuída.

4.2.2 Coleta de Dados

A coleta de dados consistiu em uma leitura exploratória de todo o material selecionado, em processo rápido e objetivo, visando selecionar os artigos relevantes para a pesquisa. Posteriormente, foi realizada a leitura dos artigos, e foram extraídos dados em forma de resumo (autores, ano, contexto, método, resultados, conclusões). Esses resumos, por sua vez, eram revisados e, caso houvesse a necessidade, eram corrigidos.

4.2.3 Análise dos Resultados

Nessa atividade, foram realizadas leituras analíticas com a finalidade de ordenar e sumarizar as informações coletadas das fontes, de modo que estas possibilitassem a obtenção de respostas que auxiliem na resolução do problema aqui abordado.

4.2.4 Interpretação dos Resultados

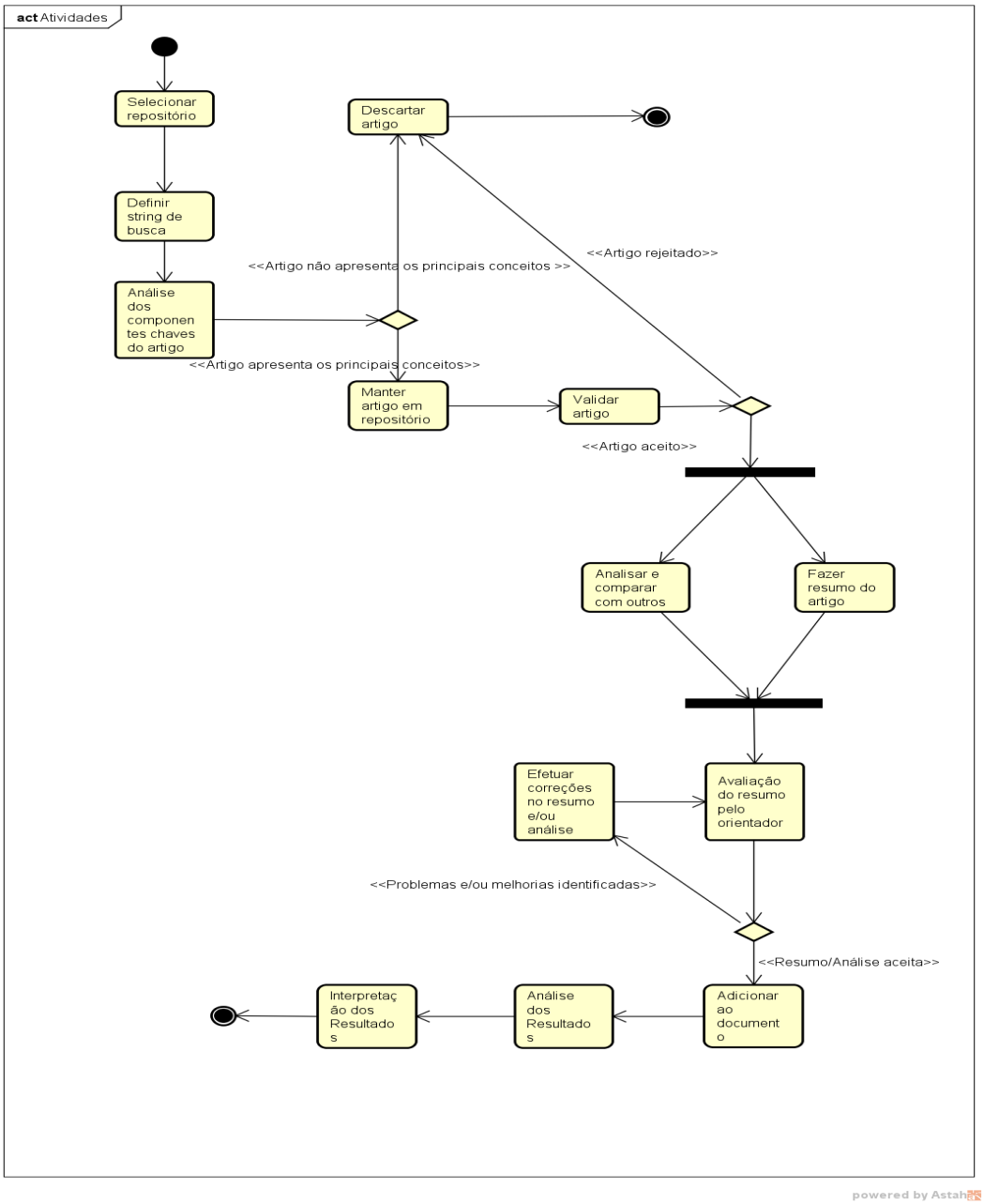
Nessa atividade, foram relacionados os resultados obtidos a partir das leituras analíticas, com problema proposto a ser resolvido. Na Figura 16 abaixo, é mostrado um diagrama de atividades representando o processo metodológico aplicado nesse trabalho:

4.3 Técnica proposta

A técnica aqui proposta é baseada em uma estrutura de dados de árvore binária, a *Binary Partition Space* - BSP. Na BSP, a célula raiz engloba todo o domínio (a *octree*). Cada célula pode ser subdividida em duas novas células filhas. Essa estrutura teoricamente aceita um particionamento qualquer, mas, neste trabalho, os cortes serão realizados na direção dos eixos globais X, Y e Z, como em Freitas *et al.* (2016). A BSP dá a liberdade de particionar em qualquer ponto do domínio da octree e em qualquer um dos eixos. Ela também permite obter uma distribuição de carga uniforme respeitando os limites dos octantes.

Como mencionado anteriormente, cada célula da BSP é dividida em duas novas células, recursivamente. Neste trabalho, o critério de parada é o que número de processos deve ser igual ao número de folhas da árvore (Figura 17). Entretanto, isso nem sempre é possível,

Figura 16 – Diagrama de Atividade representando o fluxo da execução das atividades para a construção desse trabalho.

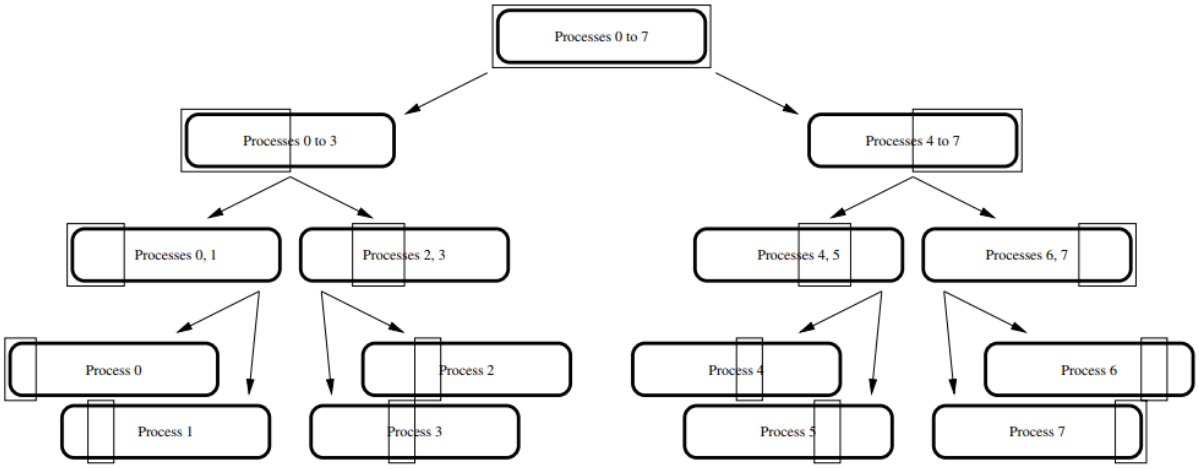


Fonte: Produzida pelo autor (2018).

por exemplo, quando o número de processos não é uma potência de 2. Logo, o número de folhas pode ser maior que a quantidade de processos. Para realizar a estimativa de carga são apresentados cinco algoritmos na Seção 4.3.2.

A técnica para construção de *quadtrees* em paralelo é descrita em três algoritmos: **Criar árvore**, **Particionar**, **Refinar**.

Figura 17 – Exemplo de BSP global decompondo um domínio qualquer entre oito processos



Fonte: Freitas *et al.* (2016).

4.3.1 Criar árvore

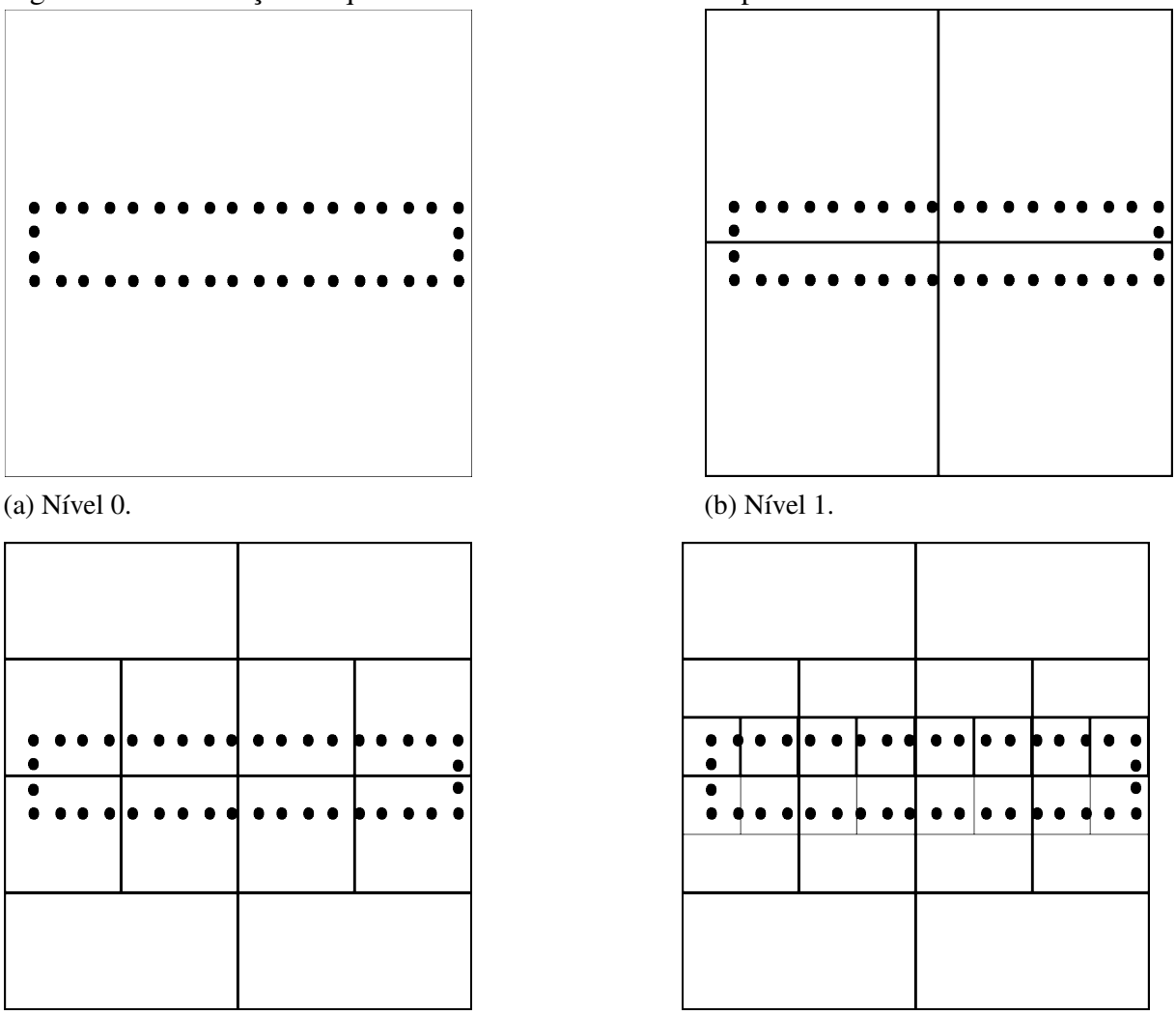
O Algoritmo **Criar árvore** consiste em gerar uma *quadtree* incompleta, criando somente os quadrantes que possuem pontos em seu domínio, gerando a árvore até o nível **L**. A *quadtree* é refinada até que o tamanho da borda da célula folha seja menor que o tamanho da aresta. Para criar um nível que contemple as células mais grosseiras, evitando refinamento excessivo, é usado a maior aresta do modelo. O tamanho da borda de uma célula é definido por $Tq/2^{\mathbf{L}}$, onde Tq é o tamanho da borda da célula raiz. Assim, encontra-se a Fórmula 4.1 para o nível **L**.

$$\mathbf{L} = \lceil \log 2(Tq/Ma) \rceil \tag{4.1}$$

onde Ma corresponte ao tamanho da maior aresta do modelo.

Existem duas abordagens para o procedimento de criação, de cima para baixo (*top-down*) e de baixo para cima (*bottom-up*). A abordagem *top-down* consiste em realizar a construção inicial da maneira tradicional, gerando os octantes a partir do nó raiz até o nível **L** pré-definido (Figura 18). Já na abordagem *bottom-up*, são gerados somente os nós folhas no nível **L** e, após a construção do restante da octree em cada processo, é realizada a construção dos nós internos do nível **L** até a raiz. Nessa abordagem, o processo responsável pelo particionamento da região também é responsável pela construção dos nós internos.

Figura 18 – Construção de quadtree até o nível 3 definido a priori.



(a) Nível 0.

(b) Nível 1.

(c) Nível 2.

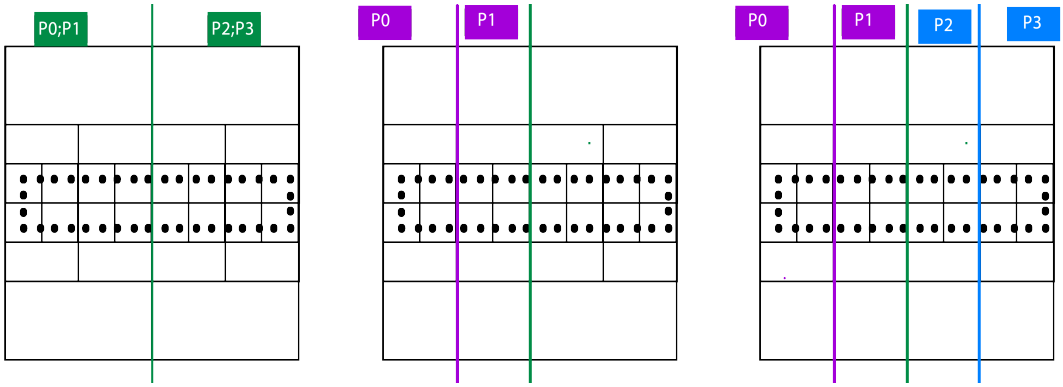
(d) Nível 3.

Fonte: Produzida pelo autor (2018).

4.3.2 Particionar

O Algoritmo Particionar consiste em realizar o particionamento da *quadtree* utilizando uma BSP. O corte que divide a região entre duas células da BSP é determinado a partir da seleção de um dos eixos cartesianos (X e Y) e da carga do domínio. O eixo é escolhido de forma que a superfície da borda interprocesso seja a menor possível, (Figura 19). Se houver empate entre os eixos, é realizado o corte do eixo Y (detalhe de implementação). Após a seleção do eixo, é determinado o local de corte. Para isso, são apresentados cinco algoritmos: **Histograma**, **Mediana**, **Média** e **Média ponderada**.

Figura 19 – Particionamento de *quadtree* entre 4 processos usando BSP. (a) A *quadtree* é subdivida em dois grupos de processos {P0, P1} e {P2, P3}. (b) A região do grupo {P0, P1} é subdivida entre os dois processos do grupo P0 e P1. (c) A região do grupo {P2, P3} é subdivida entre os dois processos do grupo P2 e P3.

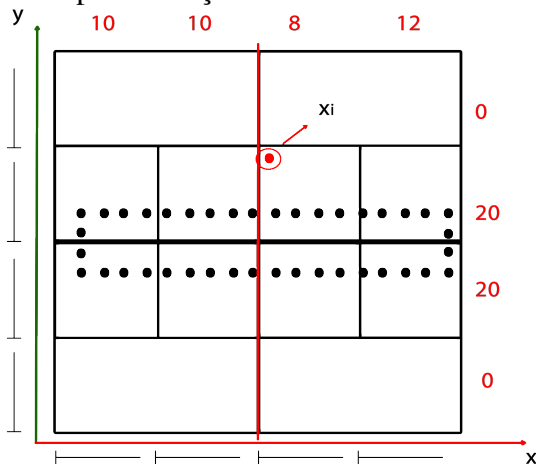


Fonte: Produzida pelo autor (2018).

4.3.2.1 Histograma

Consiste em calcular a média de um histograma, onde um conjunto C de células de mesma âncora corresponde a um intervalo de classes no histograma e a frequência é calculada de acordo com o número de vértices contidos em cada conjunto C. O valor final do corte é dado pelo lado da célula pertencente ao intervalo da média do histograma, que deixe os dois subdomínios o menos desbalanceados possível. No caso de empate, o algoritmo realiza o mesmo processo de seleção do lado da célula, pois selecionar um lado errado pode deixar a partição desbalanceada mesmo que o modelo seja uniformemente distribuído, como na Figura 20.

Figura 20 – Algoritmo Histograma aplicado a uma *quadtree*. Nos eixos X e Y, estão representados os intervalos de classe que correspondem às colunas ou barras de células em cada eixo. O ponto vermelho representa o valor da média e a linha vermelha representa o corte após a seleção de um lado de uma célula no intervalo.



Fonte: Produzida pelo autor (2018).

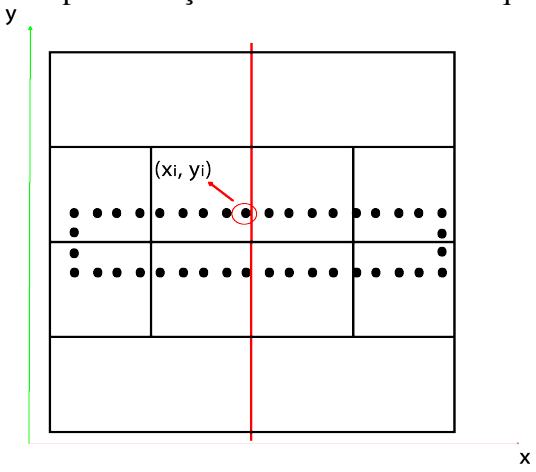
4.3.2.2 Mediana

Consiste na realização da ordenação espacial da lista de vértices de acordo com o eixo selecionado e da seleção do vértice mediano dessa lista. O vértice selecionado está aproximadamente na metade da distribuição espacial dos pontos. Logo, se um lado é mais refinado que outro (contém mais vértices) o corte tende a esse lado. O valor de corte inicial corresponde ao valor da coordenada do ponto no eixo selecionado, como na Figura 21.

Após a seleção de um corte inicial por um dos métodos citados anteriormente, é verificado se o ponto de corte está interno a uma célula folha da árvore. Se estiver, é feito um cálculo para definir qual lado da célula tomar, ou seja, é assumido um novo valor de corte final. O lado da célula selecionado é aquele que torna a partição o menos desbalanceada possível. Se o ponto de corte não estiver interno a uma célula folha da árvore, então ele passa por uma das extremidades das células folhas da árvore, e o corte inicial é mantido. Essa mesma verificação é aplicada também nos próximos algoritmos de escolha de ponto de corte.

Após a seleção do corte final, é verificado se há um grande desbalanceamento entre as partições. Se houver, é refeito o cálculo para o outro eixo e comparado o balanceamento, então é selecionado o que apresente o menor desbalanceamento.

Figura 21 – Algoritmo Mediana aplicado a uma *quadtree*. O vértice selecionado representa o valor do corte do eixo x, que é a coordenada x do vértice mediano e a linha vermelha representa o corte após a seleção de um lado da célula que contém o vértice mediano.

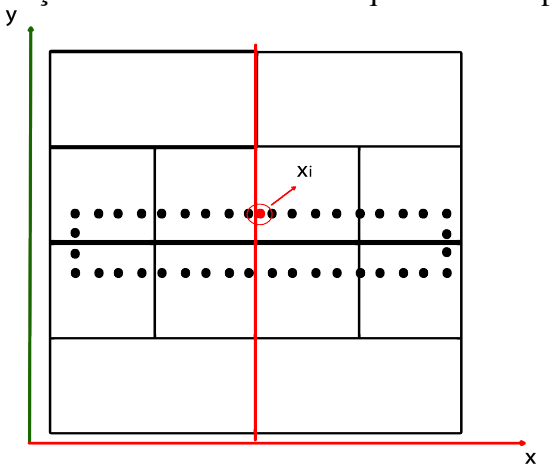


Fonte: Produzida pelo autor (2018).

4.3.2.3 Média

Realiza o cálculo da média aritmética da coordenada de cada ponto, que é selecionada de acordo com o eixo de corte. O valor do corte inicial é determinado pelo valor da média calculada, Figura 22.

Figura 22 – Algoritmo Média aplicado a uma *quadtree*. O ponto selecionado representa o valor da média das coordenadas x dos vértices da fronteira e a linha vermelha representa o corte após a seleção de um lado da célula que contém o ponto da média.

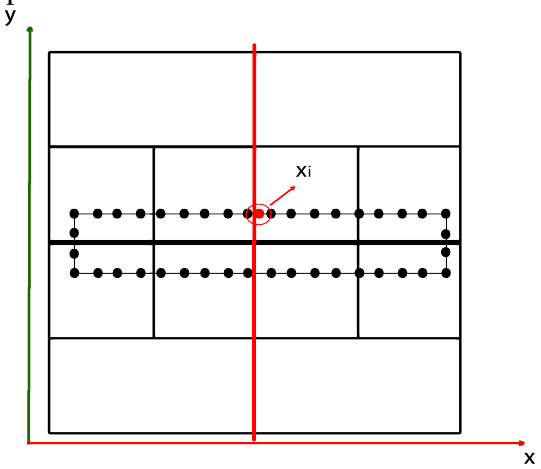


Fonte: Produzida pelo autor (2018).

4.3.2.4 Média Ponderada

Como o próprio nome remete, é calculada uma média ponderada de todos os vértices da fronteira. O peso de um vértice é inversamente proporcional ao tamanho das arestas ligados a ele. Assim, os vértices em regiões mais refinadas têm um peso maior. Esse peso é definido pela média de todas as arestas do modelo dividida pela média das arestas adjacentes ao vértice. O valor inicial de corte é definido pelo valor da média ponderada das coordenadas dos vértices do modelo segundo o eixo de partição, como na Figura 23.

Figura 23 – Algoritmo Média Ponderada aplicado a uma *quadtree*. O ponto vermelho selecionado representa o valor da média ponderada das coordenadas x dos vértices da fronteira e a linha vermelha representa o corte após a seleção de um lado da célula que contém o ponto calculado.



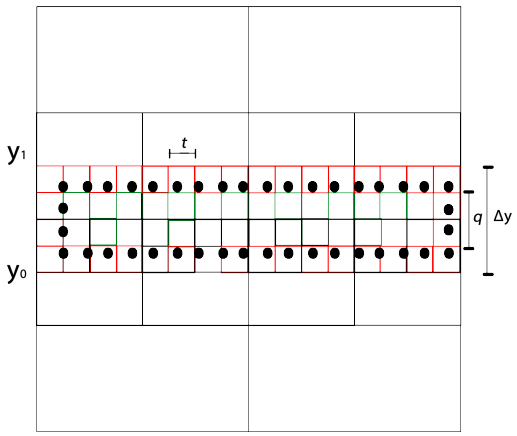
Fonte: Produzida pelo autor (2018).

4.3.2.5 Híbrido

O algoritmo *Híbrido* aborda características dos algoritmos *Histograma* e *Média Ponderada*, e trabalha da seguinte maneira: após a seleção do eixo, é calculada a quantidade de intervalos de classes do histograma. Porém, em vez de pegar todo o domínio que engloba o modelo (raiz da árvore), são selecionadas as células extremas que fazem parte da borda do modelo (seguindo a orientação do eixo), é calculada a distância entre as células e dividida pelo tamanho da célula no nível **L**, o que resulta na quantidade de classes do histograma. Depois é estimada a carga de cada célula da quadtree que compõe o modelo.

Para as células na fronteira, a carga consiste na média do peso de todos os vértices dividida pela média dos pesos dos vértices contidos no interior da célula. Para as células no interior, que não têm vértices, o peso é estimado como a média dos pesos de todas as células da fronteira para aquele intervalo de classes, como na Figura 24.

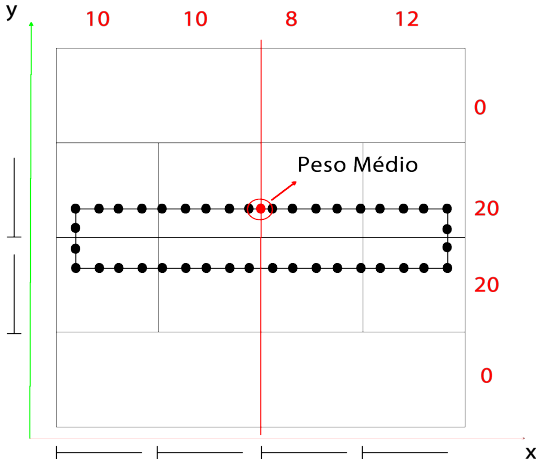
Figura 24 – Cálculo do número q de células do intervalo i internas ao modelo com cf célunas na fronteira, para o corte do eixo x ($Eixo = 1$).



Fonte: Produzida pelo autor (2018).

Posteriormente, é calculado o valor médio das cargas de cada intervalo de classe. Em seguida, é selecionada uma célula que pertença ao intervalo em que o valor médio está inserido. O valor do corte final será o valor da coordenada do lado direito (corte do eixo X) ou superior (corte do eixo Y) da célula da *quadtree*, que é determinada de acordo com o eixo de partição selecionado, como na Figura 25.

Figura 25 – Algoritmo Híbrido aplicado a uma *quadtree*. O ponto vermelho selecionado representa o valor do peso médio do eixo x. Também é mostrada a seleção do lado da célula para o corte.

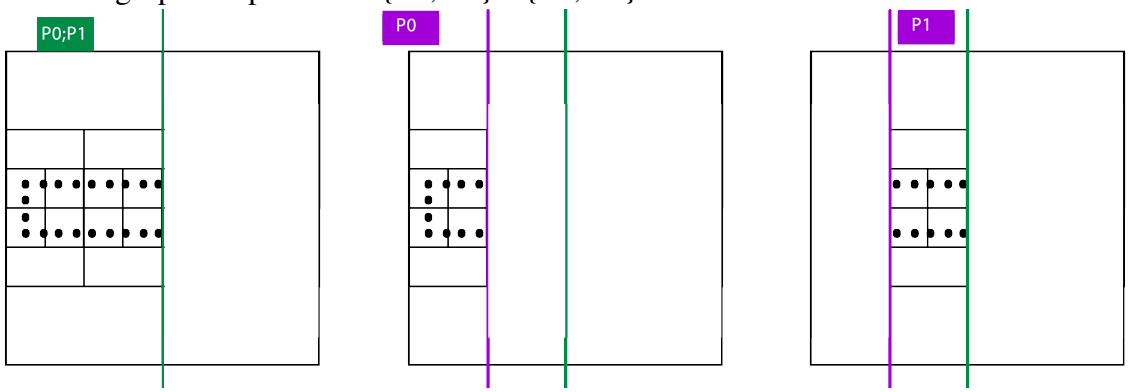


Fonte: Produzida pelo autor (2018).

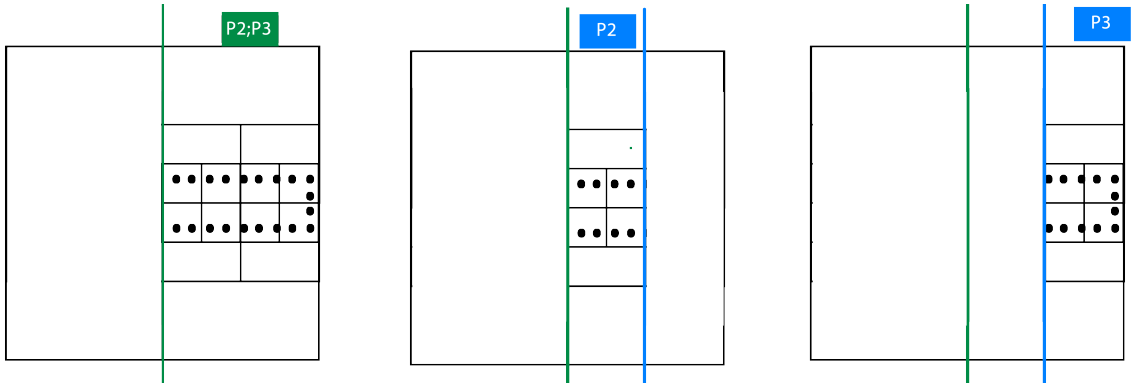
4.3.2.6 Poda

Após a seleção do eixo e local do corte da partição, é realizada a poda da árvore. Com essa poda, um processo fica somente com os nós folhas que pertencem ao seu lado da partição. Os nós internos cortados pelo corte encontrado anteriormente são replicados nas duas partições (para o caso do algoritmo *top-down*).

Figura 26 – Poda de *quadtree* entre 4 processos usando BSP. A *quadtree* é podada em dois grupos de processos {P0, P1} e {P2, P3}.



(a) Poda da *quadtree* para o grupo {P0, P1}.



(b) Poda da *quadtree* para o grupo {P2, P3}.

Fonte: Produzida pelo autor (2018).

4.3.3 Refinar

O Algoritmo **Refinar** consiste em subdividir um quadrante, substituindo-o pelos seus quatro filhos. Esse algoritmo pode ser utilizado: dentro do Algoritmo **Criar árvore**, para criar a árvore inicial na abordagem *top-down* até o nível L; dentro do Algoritmo **Particionar**, para uma melhor distribuição de carga; na construção do restante da *quadtree* em cada processo, na abordagem *top-down*, a partir do nível L.

4.4 Considerações finais

Essa seção mostrou cinco algoritmos para o particionamento da *quadtree*, identificar os algoritmos que apresentem os melhores balanceamentos e estimativas de carga. A próxima seção faz uma análise comparativa entre os cinco algoritmos, levando em conta a quantidade de células da quadtree refinada geradas. Como mencionado anteriormente, não foram realizadas análises de tempo e escalabilidade, pois o algoritmo paralelo é de difícil implementação para o pouquíssimo tempo dado para a sua conclusão.

5 EXEMPLOS E RESULTADOS

5.1 Introdução

Esta seção apresenta os resultados que foram obtidos com a implementação da técnica de construção de *quadtrees* em paralelo apresentada neste trabalho. Esses resultados englobam somente o particionamento da árvore, não levando em consideração o speed-up e o tempo de execução.

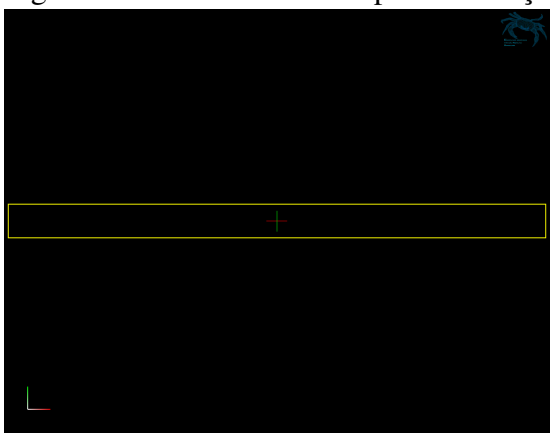
O programa foi desenvolvido na linguagem de programação C++ (COMMITTEE, 2018). Na criação da interface, foram utilizadas as bibliotecas de renderização OpenGL (glu e glut) (BOARD, 2017b).

O computador usado para realização dos testes e coletada dos dados possui um processador Intel(R) Core(TM) i5-4460 CPU @3.20GHz, 16 GB de RAM com um sistema operacional Linux versão 4.4.0.

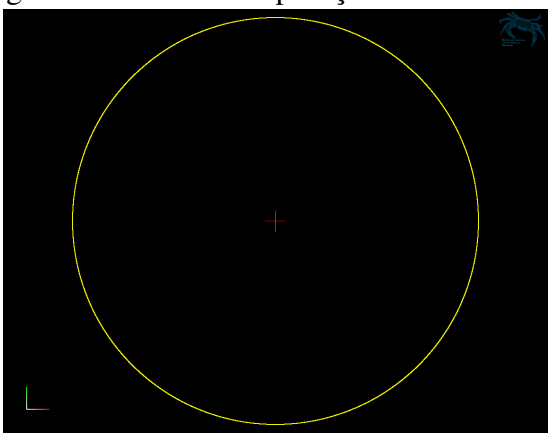
5.2 Exemplos

Os exemplos utilizados neste trabalho são: uma Viga Uniforme (Figura 27a), uma Viga não Uniforme (Figura 27b) e um Círculo (Figura 27c).

Figura 27 – Modelos usados para realização dos algoritmos de corte de partição.



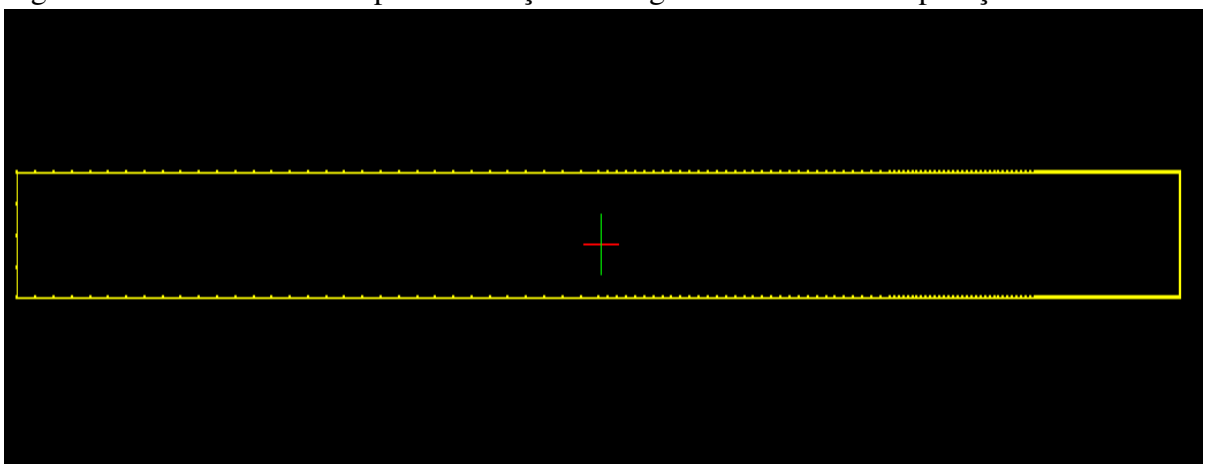
(a) Viga Uniforme.



(b) Círculo.

Fonte: Produzida pelo autor (2018).

Figura 27 – Modelos usados para realização dos algoritmos de corte de partição.



(c) Viga Não-Uniforme.

Fonte: Produzida pelo autor (2018).

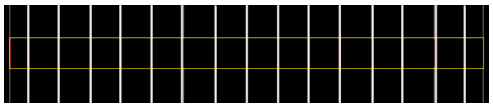
5.3 Viga Uniforme

De início, será analisado o modelo mais simples, a Viga Uniforme. Este modelo apresenta uma forma retangular alinhada aos eixos cartesianos X e Y, com maior lado seguindo o eixo X.

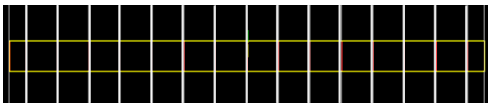
- **Particionamento do modelo.**

Na Figura 28, é mostrado o particionamento da Viga Uniforme para 16 processos após a aplicação dos algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*.

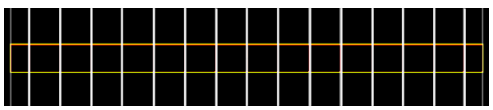
Figura 28 – Particionamento do modelo da Viga Uniforme usando os algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*.



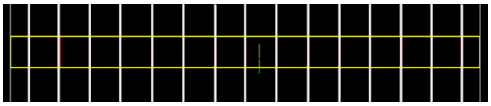
(a) Particionamento do Algoritmo Histograma.



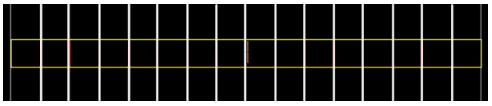
(b) Particionamento do Algoritmo Mediana.



(c) Particionamento do Algoritmo Média.



(d) Particionamento do Algoritmo Média Ponderada.



(e) Particionamento do Algoritmo Híbrido.

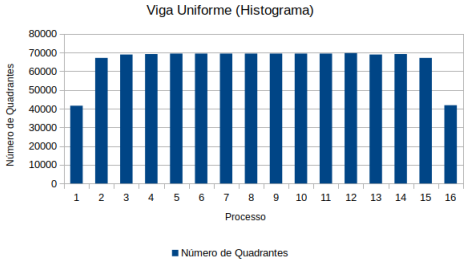
Fonte: Produzida pelo autor (2018).

Nos algoritmos *Histograma*, *Mediana*, *Média* e *Média Ponderada*, pode-se constatar a diminuição do tamanho das partições da extremidade do modelo. Essa diminuição ocorre porque as extremidades do modelo apresentam um maior acúmulo de vértices. No entanto, esses vértices não apresentam tanta influência na geração das células, o que acarreta no mal balanceamento dos subdomínios 1 e 16 para esses algoritmos, como mostra a Figura 29.

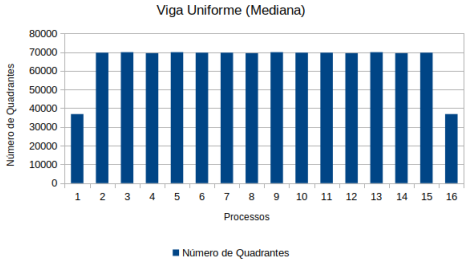
- **Balanceamento da Carga.**

Na Figura 29, é mostrado o balanceamento da carga para 16 processos após a aplicação dos algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido* nos modelos deste trabalho. No balanceamento de carga é importante que a distribuição de carga entre os processos seja aproximadamente uniforme. Caso contrário, alguns processos podem trabalhar mais do que outros. Nos gráficos da Figura 29, as barras verticais representam o número de quadrantes gerados em cada subdomínio.

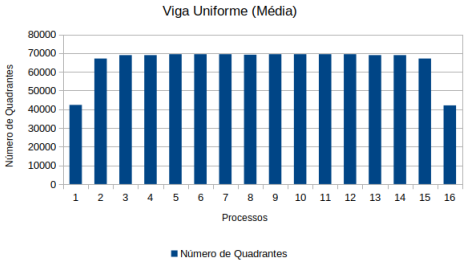
Figura 29 – Balanceamento de carga para o modelo da Viga Uniforme usando os algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*. As barras representam o número de quadrantes gerados em cada subdomínio.



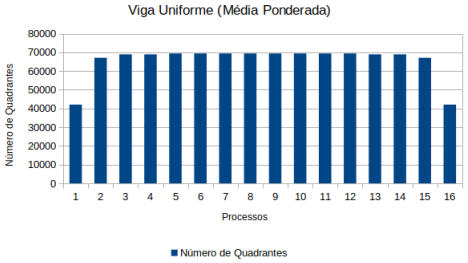
(a) Balanceamento de carga do Algoritmo Histograma.



(b) Balanceamento de carga do Algoritmo Mediana.



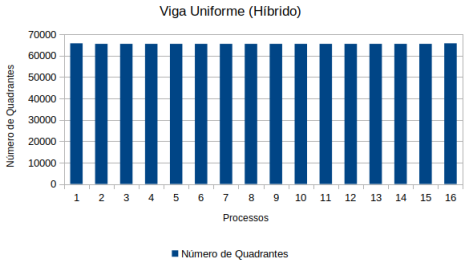
(c) Balanceamento de carga do Algoritmo Média.



(d) Balanceamento de carga do Algoritmo Média Ponderada.

Fonte: Produzida pelo autor (2018).

Figura 29 – Balanceamento de carga para o modelo da Viga Uniforme usando os algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*. As barras representam o número de quadrantes gerados em cada subdomínio.



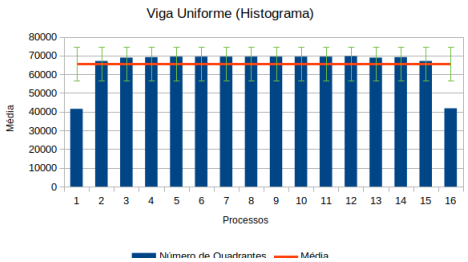
(e) Balanceamento de carga do Algoritmo Híbrido.

Fonte: Produzida pelo autor (2018).

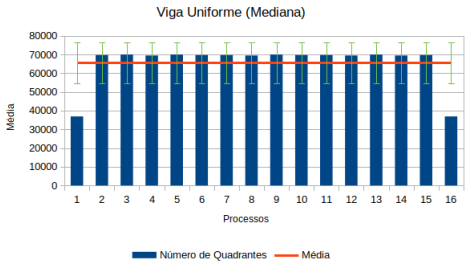
Nos gráficos da Figura 29, pode-se constatar que os algoritmos do Histograma (Figura 29a), Mediana (Figura 29b), Média (Figura 29c) e Média Ponderada (Figura 29d) apresentam um desbalanceamento nos processos 1 e 16 em relação aos outros processos. Isso ocorre devido ao grande número de vértices nas duas extremidades do modelo. Diante desse problema, os algoritmos baseados no uso de uma média tiveram uma leve melhora no balanceamento em relação ao Algoritmo Mediana (Figura 29b). O Híbrido (Figura 29e) apresentou um ótimo balanceamento, onde a divisão das cargas foi aproximadamente uniforme, sobressaindo-se sobre os demais algoritmos.

Para analisar melhor o balanceamento de carga, observa-se os gráficos da Figura 30. Esses gráficos estão representando o número de células geradas nas colunas azuis, a linha laranja representa a média e o desvio padrão. Então, para se avaliar o quão bom foi o balanceamento de carga, analisamos os algoritmos que apresentam o menor desvio padrão. Um desvio padrão grande significa que a carga dos processos variou bastante da média, enquanto que um desvio padrão pequeno significa que a carga dos processo ficou próxima à média.

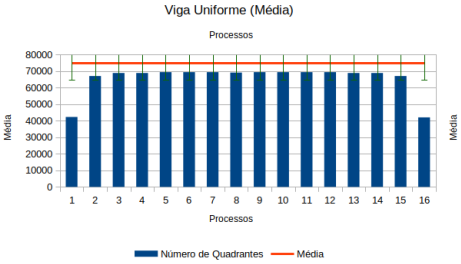
Figura 30 – Resultados para média do balanceamento de cédulas após a aplicação de cada algoritmo no modelo da Viga Uniforme.



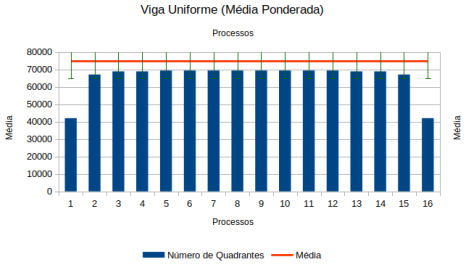
(a) Média para aplicação do Algoritmo Histograma na Viga Uniforme.



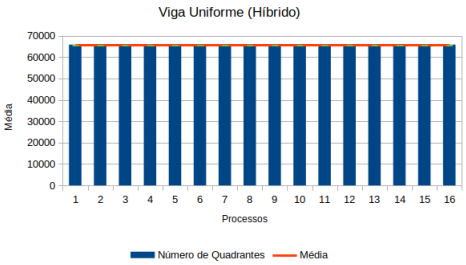
(b) Média para aplicação do Algoritmo Mediana na Viga Uniforme.



(c) Média para aplicação do Algoritmo Média na Viga Uniforme.



(d) Média para aplicação do Algoritmo Média Ponderada na Viga Uniforme.



(e) Média para aplicação do Algoritmo Híbrido na Viga Uniforme.

Fonte: Produzida pelo autor (2018).

Na análise dos gráficos da Figura 30, percebe-se que o Algoritmo Híbrido possui um melhor balanceamento dentre os algoritmos, com o desvio padrão de 84 células. Os demais apresentam desvios padrões próximos uns dos outros, porém não muito bons. Os desvios padrões para os demais algoritmos são: *Histograma* 9064,63; *Mediana* 10839,19; *Média* 8868,54; *Média Ponderada* 8917,67.

- **Estimativa de Carga.**

Como descrito em Freitas *et al.* (2016), a estimativa de carga é uma variável que sozinha não tem tanto significado, mas que, quando colocada em comparação com as demais estimativas, passa a ter valor. Por exemplo, se a estimativa de uma dada região for 100, esse

valor não quer dizer que essa região gerará aproximadamente 100 elementos. Porém, quando se compara tal valor com a estimativa de outra região, por exemplo, de 200, isso quer dizer que a primeira região gerará aproximadamente metade do número de elementos da segunda região.

Como em Freitas *et al.* (2016), a obtenção da carga estimada para cada sub-região foi calculada de acordo com a Equação 5.1, onde N é o número de subdomínios, e_i é o número de elementos gerados em cada subdomínio i, c_i é a carga estimada em cada subdomínio i, $\bar{e}$ e $\bar{c}$ são as médias do número de elementos gerados e da carga estimada em cada subdomínio, respectivamente. A estimativa de carga Est_i para o subdomínio i é dada por $Est_i = c_i \frac{\bar{e}}{\bar{c}}$.

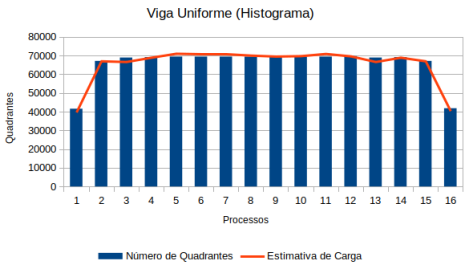
$$\bar{c} = \frac{\sum_{i=1}^{N} c_i}{N}, \tag{5.1}$$

$$\bar{e} = \frac{\sum_{i=1}^{N} e_i}{N}, \tag{5.2}$$

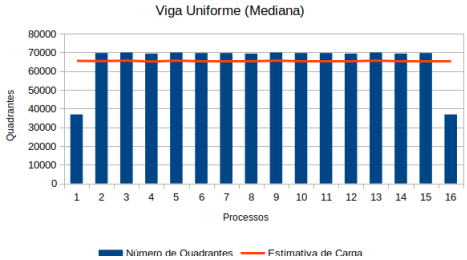
$$Est_i = c_i \frac{\bar{e}}{\bar{c}} = c_i \frac{\left(\frac{\sum_{i=1}^{N} e_i}{N}\right)}{\left(\frac{\sum_{i=1}^{N} c_i}{N}\right)} = c_i \frac{\sum_{i=1}^{N} e_i}{\sum_{i=1}^{N} c_i}. \tag{5.3}$$

Na Figura 31, é mostrada a estimativa de carga e o número de elementos gerados para 16 processos no modelo da Viga Uniforme, para cada algoritmo apresentado neste trabalho. As barras verticais representando número de quadrantes gerados em cada subdomínio, e a linha laranja representa a estimativa de carga para cada subdomínio.

Figura 31 – Resultados da estimativa de carga para aplicação dos algoritmos no modelo da Viga Uniforme.



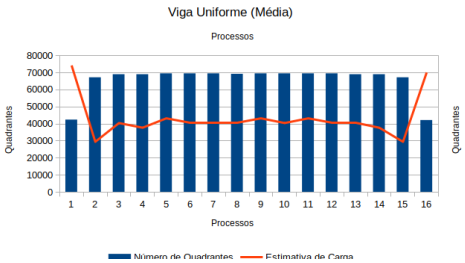
(a) Estimativa de carga realizada pelo Algoritmo Histograma na Viga Uniforme.



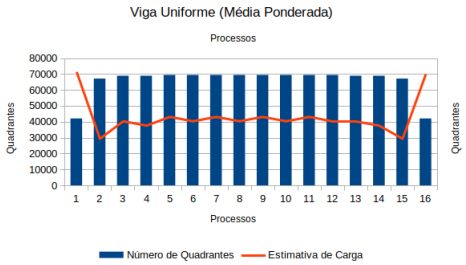
(b) Estimativa de carga realizada pelo Algoritmo Mediana na Viga Uniforme.

Fonte: Produzida pelo autor (2018).

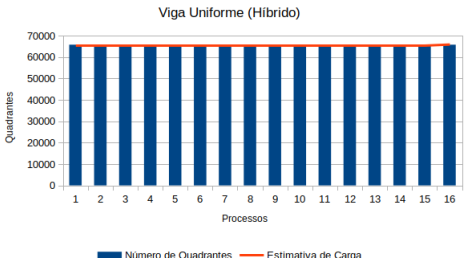
Figura 31 – Resultados da estimativa de carga para aplicação dos algoritmos no modelo da Viga Uniforme.



(c) Estimativa de carga realizada pelo Algoritmo Média na Viga Uniforme.



(d) Estimativa de carga realizada pelo Algoritmo Média Ponderada na Viga Uniforme.



(e) Estimativa de carga realizada pelo Algoritmo Híbrido na Viga Uniforme.

Fonte: Produzida pelo autor (2018).

O Algoritmo Histograma (Figura 31a) apresenta uma estimativa bem próxima do número de quadrantes gerados. Os Algoritmos Mediana (Figura 31b), Média (Figura 31c) e Média Ponderada (Figura 31d) apresentam uma estimativa aproximadamente uniforme, porém os processos 1 e 16, nos Algoritmos de Média e Média Ponderada (Figuras 31c e 31d respectivamente), apresentam uma não uniformidade crescente. O Algoritmo Híbrido apresentou uma boa estimativa de carga, como pode-se ver na Figura 31e.

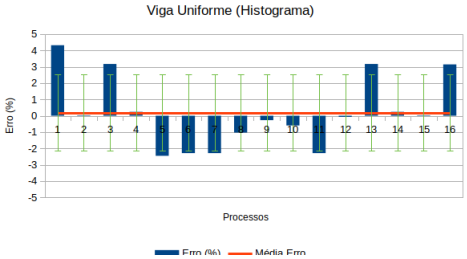
Para uma melhor análise da estimativa de carga, como em Freitas *et al.* (2016) aqui também é realizada uma análise do erro. Esse erro é o erro relativo, expresso em porcentagem, entre o valor estimado e o valor medido. A Equação 5.4 mostra como é realizado o cálculo do erro entre o valor da estimativa de carga e o número de células geradas em um subdomínio i.

$$err_i = 100\frac{Est_i - e_i}{Est_i} \tag{5.4}$$

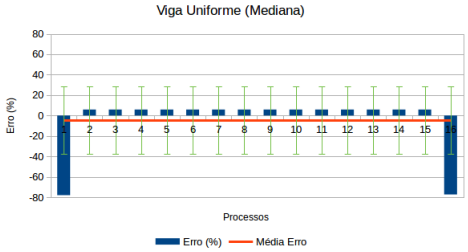
A Figura 32 apresenta os resultados obtidos para os erros das estimativas, onde as barras representam os erros em cada estimativa, e as linhas representam a média do erro e o desvio padrão do erro. Quando se tem um erro positivo, é dito que a estimativa superestimou o

valor real, se é negativo, é dito que a estimativa subestimou o valor real e, se for igual a zero, diz-se que a estimava é equivalente ao valor real. Desse modo, quanto mais próximo de 0 for a média e quanto menor o desvio padrão, melhor é a estimativa da carga.

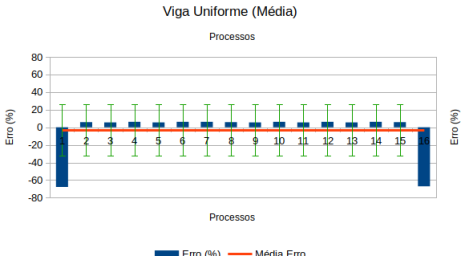
Figura 32 – Resultados para os erros das estimativas de carga nas aplicações dos algoritmos de particionamento da Viga Uniforme.



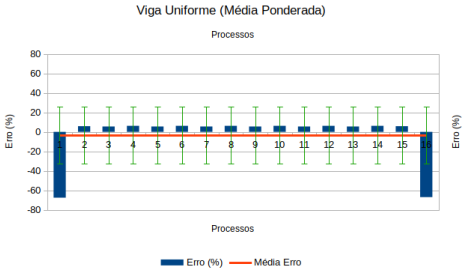
(a) Erro da estimativa de carga realizada pelo Algoritmo Histograma na Viga Uniforme.



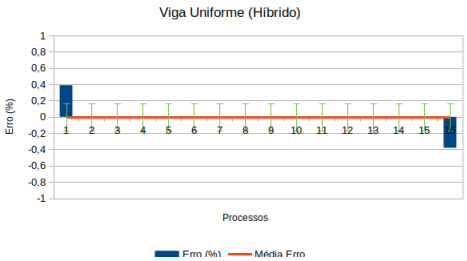
(b) Erro da estimativa de carga realizada pelo Algoritmo Mediana na Viga Uniforme.



(c) Erro da estimativa de carga realizada pelo Algoritmo Média na Viga Uniforme.



(d) Erro da estimativa de carga realizada pelo Algoritmo Média Ponderada na Viga Uniforme.



(e) Erro da estimativa de carga realizada pelo Algoritmo Híbrido na Viga Uniforme.

Fonte: Produzida pelo autor (2018).

Diante dos resultados obtidos para o erro da estimativa na Figura 32, pode-se constatar que os algoritmos *Histograma* e *Híbrido* (Figura 32a e Figura 32e, respectivamente) apresentaram uma boa estimativa com o desvio padrão do erro muito baixo: 2,34% e 0,16%, respectivamente. O *Híbrido* apresenta a melhor estimativa, como analisado anteriormente. Os demais algoritmos apresentaram erros muito altos.

5.4 Modelos do Círculo e da Viga Não-Uniforme

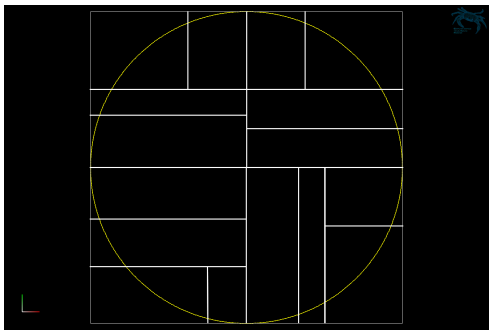
Nesta seção, são apresentados os resultados para a aplicação dos algoritmos sobre os demais modelos: Círculo e Viga Não-Uniforme. Esse modelos foram ilustrados na Figura 27.

Os modelos foram rodados para um número de 16 processos, assim como o modelo da Viga Uniforme mostrado anteriormente.

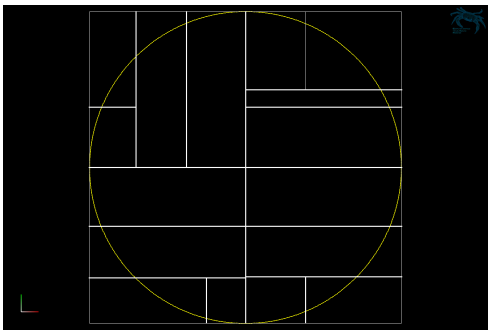
- **Particionamento do modelo.**

Nas Figuras 33 e 34, são apresentados resultados para o particionamento dos modelos do Círculo e da Viga Não-Uniforme, respectivamente.

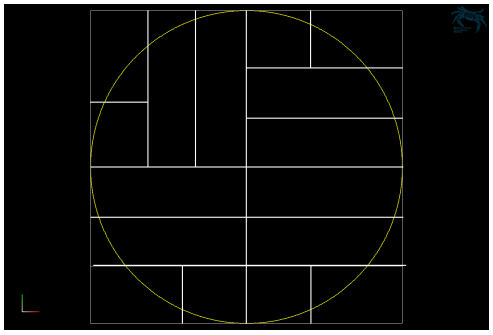
Figura 33 – Particionamento do modelo do Círculo usando os algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*.



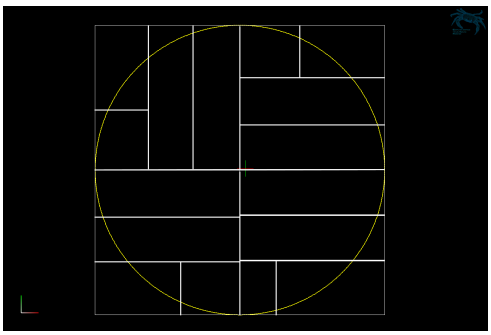
(a) Particionamento do Algoritmo Histograma.



(b) Particionamento do Algoritmo Mediana.



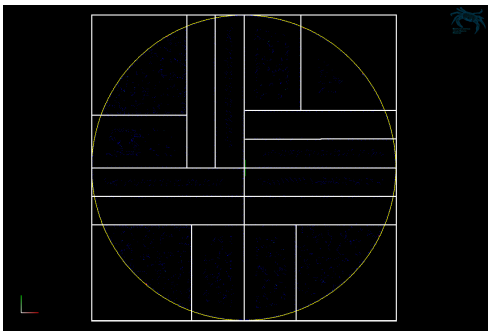
(c) Particionamento do Algoritmo Média.



(d) Particionamento do Algoritmo Média Ponderada.

Fonte: Produzida pelo autor (2018).

Figura 33 – Particionamento do modelo do Círculo usando os algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*.

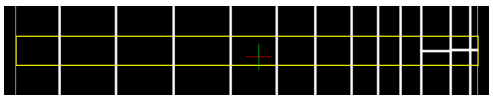


(e) Particionamento do Algoritmo Híbrido.

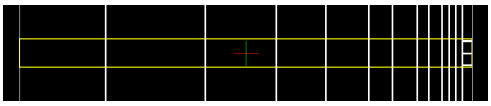
Fonte: Produzida pelo autor (2018).

Na Figura 33, os algoritmos não apresentaram um padrão no momento do particionamento do modelo do Círculo. No entanto, pode-se destacar o Algoritmo Híbrido, que considerando o interior demonstrou um bom balanceamento de carga, como mostra a Figura 38i.

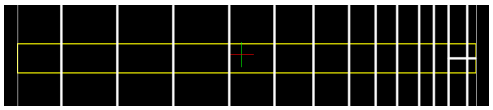
Figura 34 – Particionamento do modelo da Viga Não-Uniforme usando os algoritmos *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*.



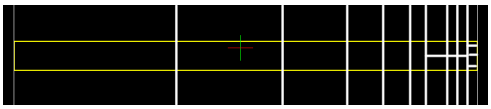
(a) Particionamento do Algoritmo Histograma.



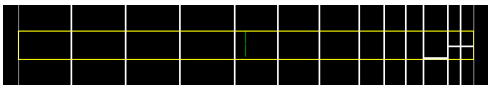
(b) Particionamento do Algoritmo Mediana.



(c) Particionamento do Algoritmo Média.



(d) Particionamento do Algoritmo Média Ponderada.



(e) Particionamento do Algoritmo Híbrido.

Fonte: Produzida pelo autor (2018).

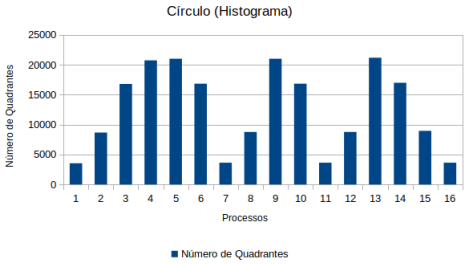
Na Figura 34, os algoritmos *Histograma*, *Mediana*, *Média* e *Média Poderada*, apresentaram uma diminuição gradativa (da esquerda para a direita) no tamanho dos subdomínios. Essa dimuição ocorre por causa do aumento do número de vértices à medida que a região do modelo é mais refinada, como pode-se notar dos gráficos de balanceamento da Figura 37. No particionamento do Algoritmo Híbrido, pode-se destacar o mal balaceamento da região mais refinada, com a região correspondente ao processo 11 (Figura 39j) com a menor quantidade de

quadrantes.

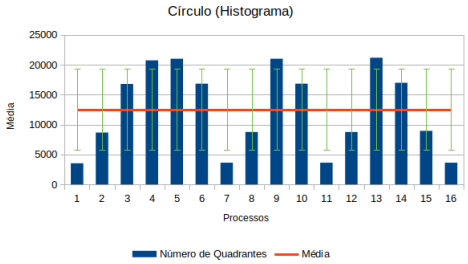
- **Balanceamento de Carga.**

Nas Figuras 35 e 37, são apresentados resultados para o balanceamento de carga dos modelos do Círculo e da Viga Não-Uniforme, respectivamente. Na coluna da esquerda, estão representados os balanceamentos de carga e, na da direita, a média e o desvio padrão do balanceamento.

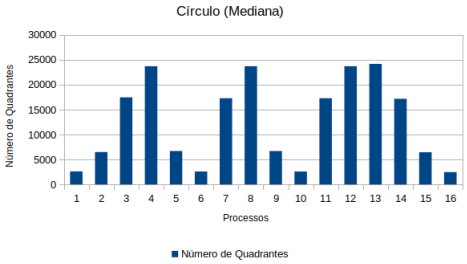
Figura 35 – Balanceamento de carga (esquerda) e média e desvio padrão (direita) para os 5 algoritmos aplicados no modelo do Círculo para 16 processos.



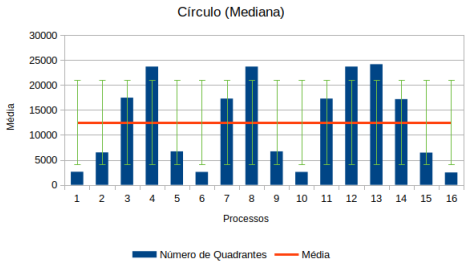
(a) Balanceamento de carga do Algoritmo Histograma.



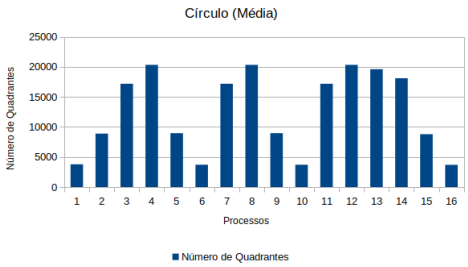
(b) Média para aplicação do Algoritmo Histograma no Círculo.



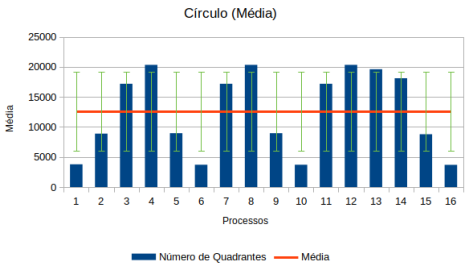
(c) Balanceamento de carga do Algoritmo Mediana.



(d) Média para aplicação do Algoritmo Mediana no Círculo.



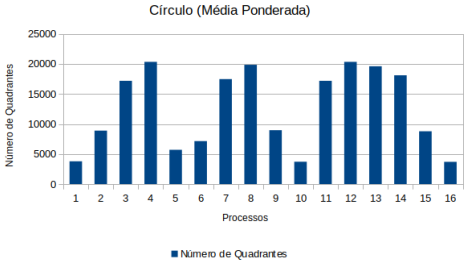
(e) Balanceamento de carga do Algoritmo Média.



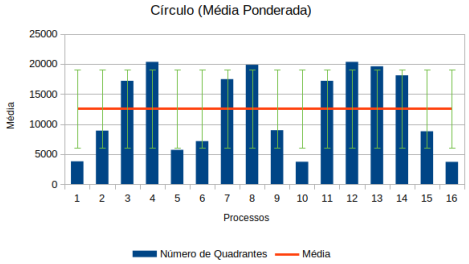
(f) Média para aplicação do Algoritmo Média no Círculo.

Fonte: Produzida pelo autor (2018).

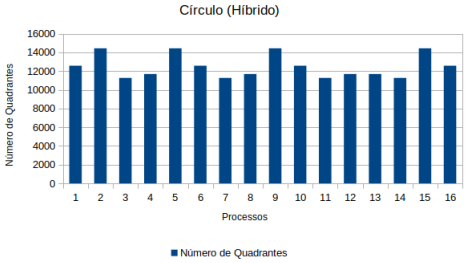
Figura 35 – Balanceamento de carga (esquerda) e média e desvio padrão (direita) para os 5 algoritmos aplicados no modelo do Círculo para 16 processos.



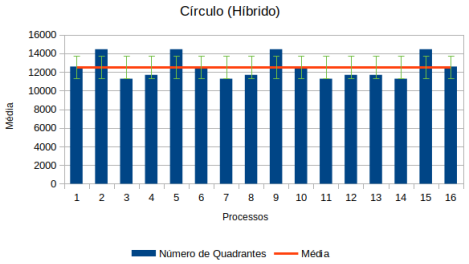
(g) Balanceamento de carga do Algoritmo Média Ponderada.



(h) Média para aplicação do Algoritmo Média Ponderada no Círculo.



(i) Balanceamento de carga do Algoritmo Híbrido.

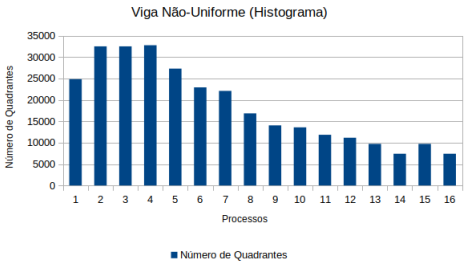


(j) Média para aplicação do Algoritmo Híbrido no Círculo.

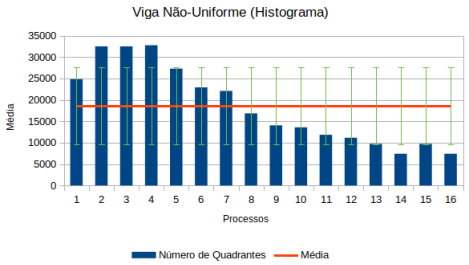
Fonte: Produzida pelo autor (2018).

Nos resultados obtidos para o balanceamento do modelo do Círculo na Figura 35, o *Híbrido* apresentou o desvio padrão de 1215,64, o menor dentre todos os aplicados no modelo, ou seja, apresentou o melhor balanceamento de carga. Os demais apresentaram os seguintes desvios padrões: *Histograma* 6774,54; *Mediana* 8433,05 e Média *Ponderada* 12547,37. O Algoritmo da Média Ponderada apresentou o pior índice de balanceamento, consequentemente, o pior balanceamento.

Figura 36 – Balanceamento de carga (esquerda) e média e desvio padrão (direita) para os 5 algoritmos aplicados no modelo da Viga Não-Uniforme para 16 processos.



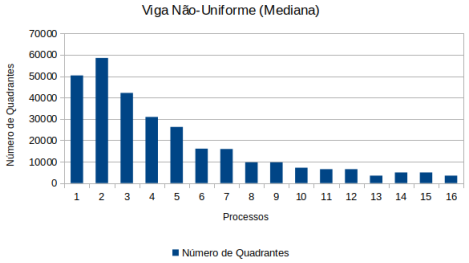
(a) Balanceamento de carga do Algoritmo Histograma.



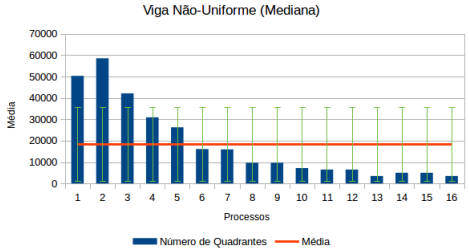
(b) Média para aplicação do Algoritmo Histograma na Viga Não-Uniforme.

Fonte: Produzida pelo autor (2018).

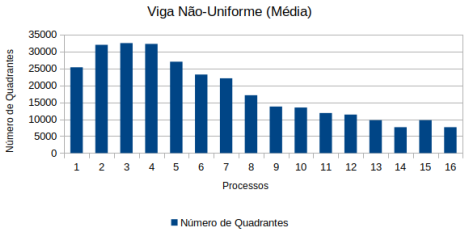
Figura 37 – Balanceamento de carga (esquerda) e média e desvio padrão (direita) para os 5 algoritmos aplicados no modelo da Viga Não-Uniforme para 16 processos.



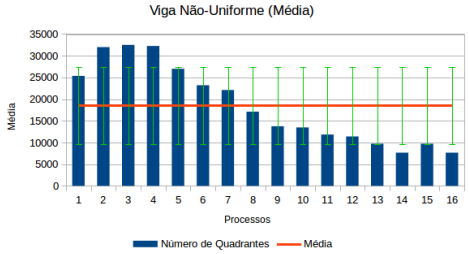
(a) Balanceamento de carga do Algoritmo Mediana.



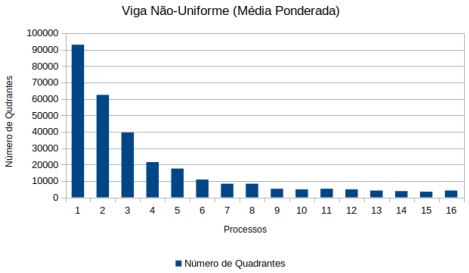
(b) Média para aplicação do Algoritmo Mediana na Viga Não-Uniforme.



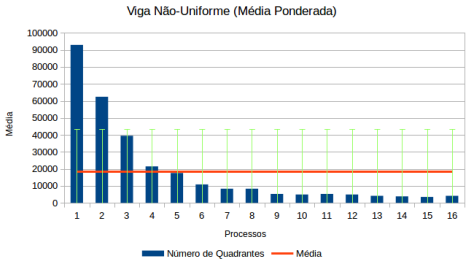
(c) Balanceamento de carga do Algoritmo Média.



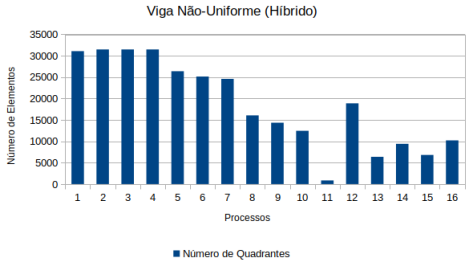
(d) Média para aplicação do Algoritmo Média na Viga Não-Uniforme.



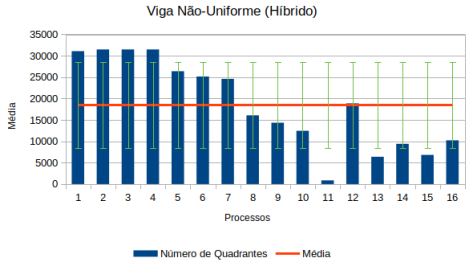
(e) Balanceamento de carga do Algoritmo Média Ponderada.



(f) Média para aplicação do Algoritmo Média Ponderada na Viga Não-Uniforme.



(g) Balanceamento de carga do Algoritmo Híbrido.



(h) Média para aplicação do Algoritmo Híbrido na Viga Não-Uniforme.

Fonte: Produzida pelo autor (2018).

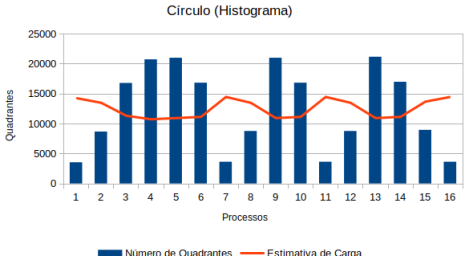
Diferente dos resultados obtidos para a Viga Uniforme, os resultados da Viga não Uniforme não foram tão animadores. Os algoritmos apresentaram um balanceamento de carga não uniforme, com destaque para os algoritmos de Mediana (Figura 39c) e de Média Ponderada (Figura 39g), onde se teve um grande número de células geradas em uma extremidade do modelo

e um pequeno número na outra extremidade. Em uma diminuição gradativa de acordo com o refinamento do modelo, seus desvios padrões foram respectivamente 17291,92 e 24699,79. Para os demais algoritmos, *Histograma* (Figura 36b), *Média* (Figura 37d) e *Híbrido* (Figura 39j), embora não tenham atingido o balanceamento de carga uniforme, apresentaram uma melhor distribuição com os respectivos desvios padrões, 8985,82, 8861,49 e 10039,17. O Algoritmo Média apresentou o menor desvio padrão, no entanto, os algoritmos *Histograma* e *Híbrido* apresentaram valores próximos.

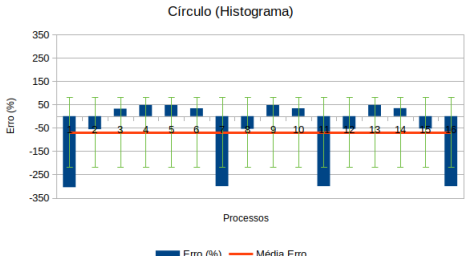
- **Estimativa de Carga.**

Nas Figuras 38 e 39 são apresentados resultados para a estimativa de carga dos modelos do Círculo e da Viga não Uniforme, respectivamente. Na coluna da esquerda, estão representadas as estimativas de carga e, na da direita, o erro relativo para a estimativa de carga.

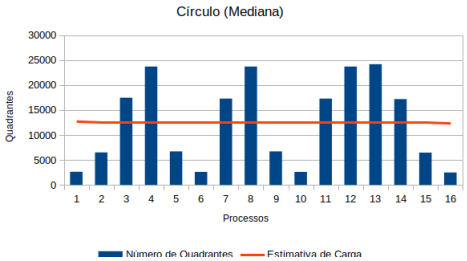
Figura 38 – Estimativa de carga (esquerda) e erro médio e desvio padrão (esquerda) para os 5 algoritmos aplicados no modelo da Círculo para 16 processos.



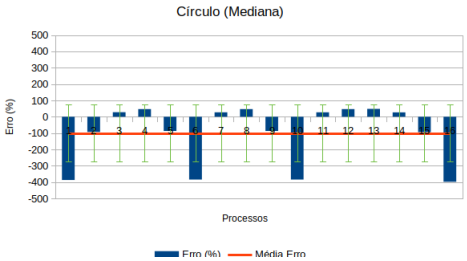
(a) Estimativa de carga do Algoritmo Histograma.



(b) Erro relativo do Algoritmo Histograma no modelo do Círculo.



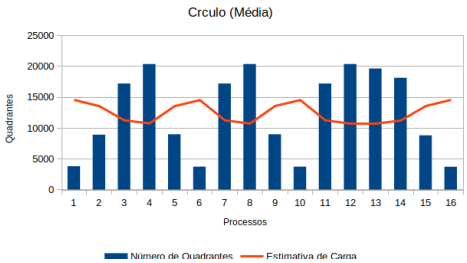
(c) Estimativa de carga do Algoritmo Mediana.



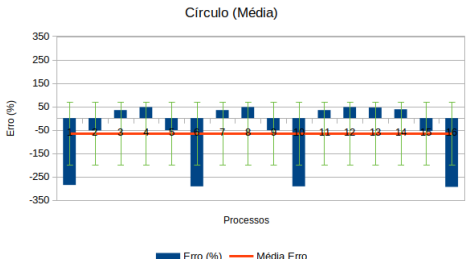
(d) Erro relativo do Algoritmo Mediana no modelo do Círculo.

Fonte: Produzida pelo autor (2018).

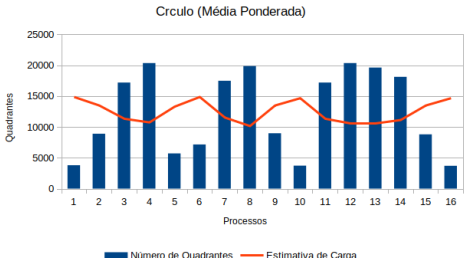
Figura 38 – Estimativa de carga (esquerda) e erro médio e desvio padrão (esquerda) para os 5 algoritmos aplicados no modelo da Círculo para 16 processos.



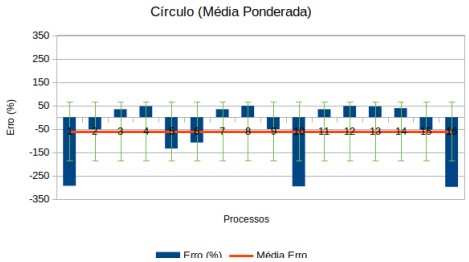
(e) Estimativa de carga do Algoritmo Média.



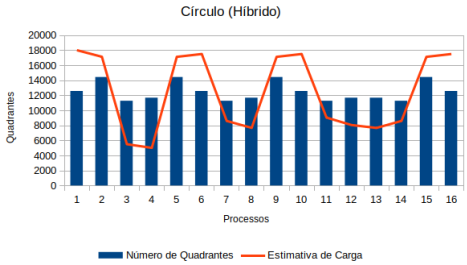
(f) Erro relativo do Algoritmo Média no modelo do Círculo.



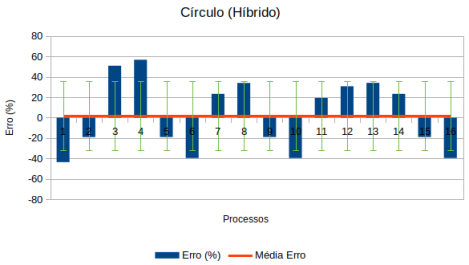
(g) Estimativa de carga do Algoritmo Média Ponderada.



(h) Erro relativo do Algoritmo Média Ponderada no modelo do Círculo.



(i) Estimativa de carga do Algoritmo Híbrido.

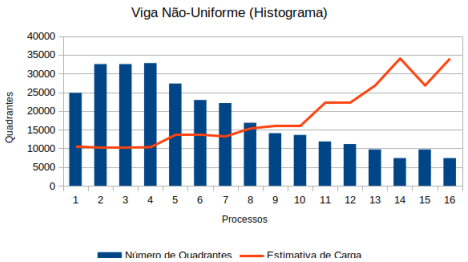


(j) Erro relativo do Algoritmo Híbrido no modelo do Círculo.

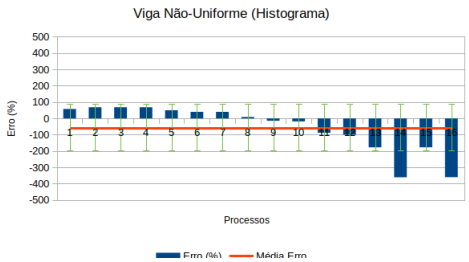
Fonte: Produzida pelo autor (2018).

Na Figura 38, para o caso do Círculo, os algoritmos estimaram mal, embora o modelo do Círculo testado seja uniforme. Pode-se destacar o Algoritmo da Mediana com o maior desvio padrão do erro, com 174, 36% e pior desempenho, e o Algoritmo Híbrido com menor desvio padrão do erro, com 33,99%.

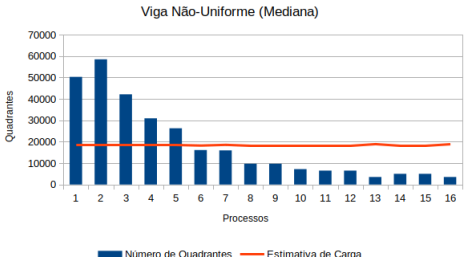
Figura 39 – Estimativa de carga (esquerda) e erro médio e desvio padrão (esquerda) para os 5 algoritmos aplicados no modelo da Viga Não-Uniforme para 16 processos.



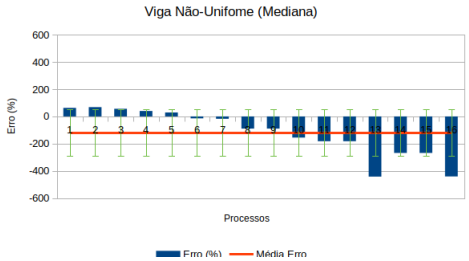
(a) Estimativa de carga do Algoritmo Histograma.



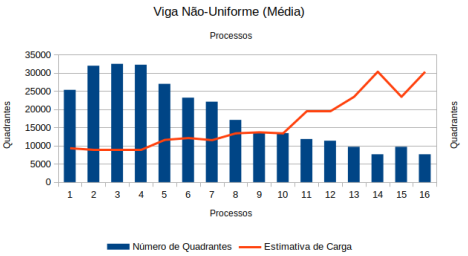
(b) Erro relativo do Algoritmo Histograma no modelo da Viga Não-Uniforme.



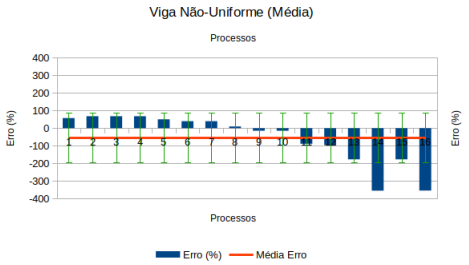
(c) Estimativa de carga do Algoritmo Mediana.



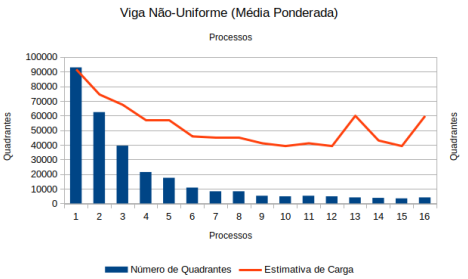
(d) Erro relativo do Algoritmo Mediana no modelo da Viga Não-Uniforme.



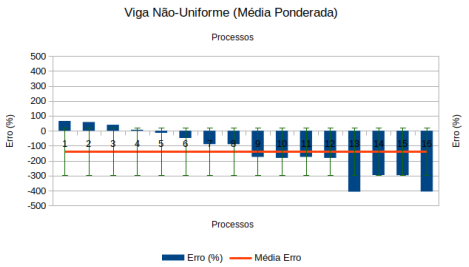
(e) Estimativa de carga do Algoritmo Média.



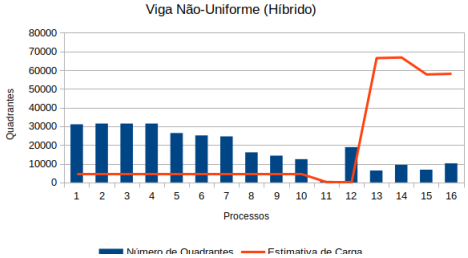
(f) Erro relativo do Algoritmo Média no modelo da Viga Não-Uniforme.



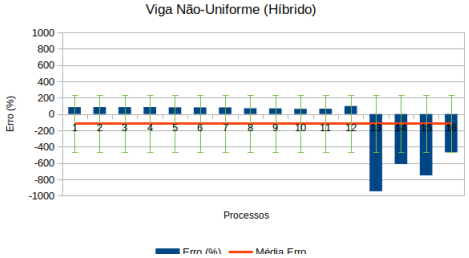
(g) Estimativa de carga do Algoritmo Média Ponderada.



(h) Erro relativo do Algoritmo Média Ponderada no modelo da Viga Não-Uniforme.



(i) Estimativa de carga do Algoritmo Híbrido.



(j) Erro relativo do Algoritmo Híbrido no modelo da Viga Não-Uniforme.

Fonte: Produzida pelo autor (2018).

Na Figura 39, como no Círculo, na Viga Não-Uniforme a estimativa de carga não foi boa, com apenas o Algoritmo da Mediana obtendo uma estimativa uniforme. Entretanto, os processos não tiveram um balanceamento uniforme. Os algoritmos obtiveram os seguintes desvios padrões para os erros: Histograma - 142,55 %; Mediana - 170,07%; Média - 140,96%; Média Ponderada - 158,29%; Híbrido - 347,38%. A melhor estimativa foi realizada pelo Algoritmo da Média e, a pior, pelo Algoritmo Híbrido.

5.5 Considerações finais

Este capítulo apresentou os resultados obtidos na aplicação dos cinco algoritmos de particionamento de carga apresentados nesse trabalho para os modelos da Viga Uniforme, Viga não Uniforme e Círculo. O modelo de estudo Viga Uniforme foi usado como modelo mais simples para verificar a acurácia dos algoritmos.

O próximo capítulo apresenta as conclusões acerca deste trabalho, destacando as suas principais contribuições, e identificando pontos para melhoria deste trabalho.

6 CONCLUSÕES

6.1 Principais contribuições

Esse trabalho apresentou um comparativo entre algumas técnicas de construção paralela de *quadtrees* em sistemas de memória distribuída, no entanto, as técnicas também podem ser usadas para *octrees*. Além disso, também foi apresentada uma técnica em desenvolvimento baseada no uso de uma *Binary Space Partitioning* - BSP para a construção dessas estruturas.

A análise das técnicas dos trabalhos relacionados apresentados mostra que as curvas de preenchimento de espaço são bastante utilizadas para o particionamento de dados da entrada (*octrees*). No entanto, nenhuma utiliza de uma BSP para o particionamento. O uso de uma BSP para a decomposição da *octree* pode se equiparar mais à partição do domínio em Freitas *et al.* (2016) em comparação às outras técnicas.

Esse trabalho também apresentou cinco algoritmos para o particionamento de uma *quadtree*/*octree* usando uma BSP: *Histograma*, *Mediana*, *Média*, *Média Ponderada* e *Híbrido*. Foi realizada uma análise comparativa sobre o balanceamento e a estimativa de carga dos algoritmos de particionamento.

Os resultados demonstraram que o Algoritmo Híbrido demonstrou o melhor balanceamento e uma estimativa de carga relativamente boa para os os modelos uniformes testados, tendo em vista que o algoritmo tenta predizer a quantidade de células geradas em cada sub-região. Este número de células geradas, por sua vez, é proporcional ao número de elementos que serão gerados na sub-região.

Os algoritmos que tomam como base somente os vértices da fronteira não são eficientes no particionamento de alguns modelos. Para uma quantidade maior que dois processos (caso da Viga Uniforme) ou quatro (caso do Círculo), o acúmulo de pontos nas extremidades, que não influenciam tanto na geração da malha, acabam interferindo na estimativa. Além disso, estimar somente sobre o número de vértices desconsidera a possível carga contida no interior do modelo. Como o Algoritmo Híbrido tem influência do interior do modelo, ele se mostrou o melhor de todos.

Apesar de os resultados mostrados, nenhum dos cinco algoritmos se mostrou eficiente no modelo não-uniforme testado. Isso acontece, em parte, por conta da não-uniformidade no tamanho das células da *quadtree* geradas no interior do modelo. Entretanto, isso é difícil de se prever, uma vez que as únicas informações disponíveis de como será o interior do modelo

encontram-se na borda, e essas informações são pobres de detalhes.

Portanto, a utilização de uma estrutura que depende bastante da geometria do modelo, como é a BSP, pode não ser a mais indicada para a geração da *quadtree*.

6.2 Trabalhos futuros

Alguns pontos podem ser continuados e melhorados neste trabalho. A primeira melhoria diz respeito à estimativa de carga para modelos côncavos, com fraturas ou com buracos, tendo em vista, que o Algoritmo Híbrido não apresenta essa distinção. Diante dos resultados obtidos, além desses tipos de modelos, é necessário melhorar a estimativa e a distribuição para modelos não-uniformemente refinados, como a Viga Não-Uniforme e para modelos com a área interna muito grande, como o Círculo.

A segunda melhoria consiste na conclusão da técnica de balanceamento 2:1 em paralelo usando a BSP. Nessa perspectiva, pode-se aplicar um balanceamento 2:1 local intra processo e em seguida balancear a árvore globalmente. Para realizar o balanceamento global, é necessária a comunicação inter-processos. Então, têm-se a necessidade de criar uma abordagem para reger a comunicação inter-processos, usando a BSP. Espera-se que o uso da BSP possibilite uma rápida execução do balanceamento 2:1.

A técnica abordou a construção de *quadtrees* em paralelo. Logo, faz-se necessário uma abordagem para o domínio tridimensional, ou seja, para a construção de *octrees* em paralelo. Essa construção deverá tratar o novo eixo (Z) para a diminuição da interface que conecta os domínios particionados. Ao acrescentar a terceira dimensão, em vez de se usar uma linha de corte, será necessária a utilização de um plano de corte.

Essa técnica foi implementada para rodar em uma única máquina de maneira sequencial. Então, faz-se necessária a implementação paralela, tanto para memória compartilhada utilizando da biblioteca OpenMP (*Open Multi-Processing*) (BOARD, 2017a), quanto para memória distribuída utilizando a biblioteca MPI (*Message Passing Interface*) (FORUM, 2015).

Como a implementação foi sequencial e executada somente em uma única máquina, foram realizados testes e análises somente para o particionamento. Após a implementação da técnica paralela, é necessário realizar testes para medir a Escalabilidade de Tamanho Fixo e Isogranular o tempo de execução do programa.

Apesar de os resultados com relação à quantidade de células geradas não terem sido bons em todos os casos, é necessário analisar o balanceamento com respeito ao tempo de

execução, que pode ter um resultado bastante diferente. Por exemplo, o tempo de criação das células internas ao modelo, no geral, é menor que o da criação das células na borda do modelo. Portanto, o balanceamento com respeito ao tempo de execução pode ter um resultado diferente do apresentado nesse trabalho.

REFERÊNCIAS

BERN, M.; EPPSTEIN, D.; TENG, S.-H. Parallel construction of quadtrees and quality triangulations. In: SPRINGER. **Workshop on Algorithms and Data Structures**. 1993. p. 188–199. Disponível em: https://link.springer.com/chapter/10.1007/3-540-57155-8_247. Acesso em: 9 fev. 2018.

BOARD, O. A. R. **The OpenMP API Specification for Parallel Programming**. 2017. Disponível em: http://www.openmp.org. Acesso em: 3 nov. 2018.

BOARD, T. O. A. R. **OpenGL - The Industry's Foundation for High Performance Graphics**. 2017. Disponível em: http://www.opengl.org. Acesso em: 3 nov. 2018.

BURSTEDDE, C.; WILCOX, L. C.; GHATTAS, O. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. **SIAM Journal on Scientific Computing**, SIAM, v. 33, n. 3, p. 1103–1133, 2011. Disponível em: https://epubs.siam.org/doi/abs/10.1137/100791634. Acesso em: 30 jan. 2018.

CAMATA, J. J. **Estratégias Computacionais de Alto Desempenho Aplicadas ao Método dos Elementos Finitos**. Tese (Doutorado em Engenharia Civil) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2011. Disponível em: http://objdig.ufrj.br/60/teses/coppe_d/JoseJeronimoCamata.pdf. Acesso em: 29 jan. 2018.

CAMATA, J. J.; COUTINHO, A. L. Parallel linear octree meshing with immersed surfaces. In: IEEE. **Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on**. 2010. p. 151–158. Disponível em: https://ieeexplore.ieee.org/abstract/document/5644954. Acesso em: 10 mar. 2018.

CAMPBELL, P. M.; DEVINE, K. D.; FLAHERTY, J. E.; GERVASIO, L. G.; TERESCO, J. D. Dynamic octree load balancing using space-filling curves. **Williams College Department of Computer Science, Tech. Rep. CS-03-01**, 2003. Disponível em: https://pdfs.semanticscholar.org/949f/0d4ea6d730f29aa11d42c061f3ddbd68888d.pdf. Acesso em: 10 mar. 2018.

COMMITTEE, T. C. S. **JTC1/SC22/WG21 - The C++ Standards Committee - ISOCPP**. 2018. Disponível em: http://www.open-std.org/jtc1/sc22/wg21. Acesso em: 3 nov. 2018.

FORUM, M. **The Message Passing Interface (MPI) Standard**. 2015. Disponível em: http://www.mcs.anl.gov/research/projects/mpi. Acesso em: 3 nov. 2018.

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, v. 32, n. 4, p. 655–674, 2016. Disponível em: https://link.springer.com/article/10.1007/s00366-016-0444-3. Acesso em: 23 mai. 2018.

LOTTI, R. S.; MACHADO, A. W.; MAZZIEIRO, Ê. T.; JÚNIOR, J. L. Aplicabilidade científica do método dos elementos finitos. **R Dental Press Ortodon Ortop Facial**, SciELO Brasil, v. 11, n. 2, p. 35–43, 2006. Disponível em: http://www.scielo.br/pdf/dpress/v11n2/a06v11n2. Acesso em: 26 fev. 2018.

MACHADO, M.; FLORES, P.; AMBRÓSIO, J. Techniques for geometrical detection of contact within multibody systems. In: **New Trends in Mechanism and Machine Science**. Springer, 2015. p. 471–478. Disponível em: https://link.springer.com/chapter/10.1007\%2F978-3-319-09411-3_50. Acesso em: 1 nov. 2018.

PEANO, G. Sur une courbe, qui remplit toute une aire plane. **Mathematische Annalen**, Springer, v. 36, n. 1, p. 157–160, 1890. Disponível em: https://link.springer.com/chapter/10.1007/978-3-7091-9537-6_6. Acesso em: 19 fev. 2018.

SAGAN, H. **Space-filling curves**. Springer Science & Business Media, 2012. Disponível em: https://books.google.com.br/books?id=cP_ZBwAAQBAJ&lpg=PA1&ots=sMToNziAvU&dq=Space-filling\%20curves&lr&hl=pt-BR&pg=PA1#v=onepage&q=Space-filling\%20curves&f=false. Acesso em: 19 fev. 2018.

SUNDAR, H.; SAMPATH, R. S.; BIROS, G. Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel. **SIAM Journal on Scientific Computing**, SIAM, v. 30, n. 5, p. 2675–2708, 2008. Disponível em: https://epubs.siam.org/doi/abs/10.1137/070681727.

TU, T.; O'HALLARON, D. R.; GHATTAS, O. Scalable parallel octree meshing for terascale applications. In: IEEE. **Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference**. 2005. p. 4–4. Disponível em: https://ieeexplore.ieee.org/document/1559956. Acesso em: 30 jan. 2018.

VALGAERTS, L. Space-filling curves an introduction. **Technical University Munich**, 2005. Disponível em: http://lxmayr1.informatik.tu-muenchen.de/konferenzen/Jass05/courses/2/Valgaerts/Valgaerts_paper.pdf. Acesso em: 19 fev. 2018.

YANG, S.-N.; LEE, R.-R. Parallel quadtree construction and manipulation algorithmson hypercube. In: **Proceedings of the 6th International Conference on Computingand Information**. [s.n.], 1994. p. Peterborough–Ontario. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.2671. Acesso em: 26 fev. 2018.