Programa de Pós-Graduação em Engenharia
Elétrica Associação Ampla entre CEFET-MG e UFSJ

Biharck Muniz Araújo

TÉCNICAS DE COMPUTAÇÃO PARALELA APLICADAS EM MÉTODOS SEM MALHA

Belo Horizonte

2014

Programa de Pós-Graduação em Engenharia
Elétrica Associação Ampla entre CEFET-MG e UFSJ

Biharck Muniz Araújo

TÉCNICAS DE COMPUTAÇÃO PARALELA APLICADAS EM MÉTODOS SEM MALHA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica - Associação Ampla entre a UFSJ e o CEFET-MG, como parte dos requisitos necessários para a obtenção do título de mestre em Engenharia Elétrica.
Área de Concentração: Sistemas Elétricos
Linha de Pesquisa: Eletromagnetismo Aplicado

Orientador: Dr. Eduardo Henrique da Rocha Copolli
Coorientadora: Dra. Úrsula do Carmo Resende

Belo Horizonte
2014

Folha de Aprovação a ser anexada

A minha esposa Aline Lopes Coelho e meus queridos pais Maria das Graças Muniz Araújo e Gomercino Ferreira de Araújo.

Agradecimentos

Agradeço primeiramente a minha esposa Aline Lopes Coelho por seu carinho, compreensão e amor, aos meus pais Maria das Graças Muniz Araújo e Gomercino Ferreira de Araújo por toda bravura e sabedoria a mim ensinados. Agradeço com muito carinho e respeito o meu orientador Eduardo Henrique da Rocha Coppoli por sua eterna paciência, sabedoria e amizade conquistada. A minha coorientadora Úrsula do Carmo Resende pela oportunidade de realizar este trabalho e a Deus pela missão a mim concedida.

Agradeço também pelo apoio concedido a este trabalho pelo Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG, pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais – FAPEMIG e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES.

Resumo

Dentro da vasta variedade de métodos numéricos aplicados a solução de equações diferenciais parciais estão os Métodos sem Malha ou *Métodos sem Malha*. O que caracteriza esta classe de métodos e o fato dos mesmos não necessitarem de malhas como as utilizadas em métodos como os de Elementos Finitos, por exemplo. Quando se trata de estruturas que constantemente realizam alterações de sua forma, este tipo de abordagem pode ser muito eficiente, reduzindo o custo de reconstrução da malha a cada iteração. Todavia, este tipo de solução apresenta um custo computacional relativamente elevado para sua resolução numérica.

Este trabalho apresenta técnicas de paralelização aplicadas à problemas de eletromagnetismo avaliados por meio de Métodos sem Malha. Ao longo deste trabalho são apresentadas técnicas de paralelismo a nível de memória compartilhada e através de troca de mensagens, além do uso de *Solvers* paralelizados utilizados na solução de sistemas lineares.

O problema utilizado como estudo de caso é o modelo sem malha de uma máquina elétrica de indução trifásica em movimento, em que podem ser avaliadas o fluxo magnético em seu interior, as correntes e tensões em seus condutores, além de possível a visualização de seu movimento.

Palavras-chave: Métodos sem Malha, Programação Paralela, Método de Galerkin.

Abstract

Within the wide range of numerical methods applied to the solution of Partial Differential Equations are the Meshless Methods. What characterizes this class of methods is the fact that they do not have meshes used as in Finite Element Methods. When it comes to structures that constantly changes their shape, this type of approach can be very efficient, reducing the cost of rebuilding the mesh at each iteration. However, such solution has a relatively high computational cost for numerical resolution.

This paper presents parallelization techniques applied to problems in electromagnetics by Meshless Methods. Throughout this paper, parallelism techniques through shared memory and message-passing are presented, and the use of parallel solvers used in the solution of linear systems.

The problem used as a case study is the model without mesh of a three-phase induction electric machine in motion, which can be evaluated in the magnetic flux in its interior, the currents and voltages in their drivers, as well as possible to see their movement.

Key-words: Meshless Methods, Parallel Programming, Element Free Galerking Method.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

- α_I – Valor adimensional relacionado ao domínio de influência.
- A – Potencial vetor magnético dado em $weber/metro$.
- B – Indução magnética dada em $weber/metro^2$.
- d_I – Dimensão do domínio de influência.
- d_c – Distância média próxima ao nó de interesse do domínio de influência.
- H – Campo magnético dados em $ampère/metro$.
- H^0 – Espaço de funções com quadrado integrável
- H^1 – Espaço de funções cuja derivada primeira possua quadrado integrável
- I – Corrente elétrica em $amprère$.
- I_t – Corrente da barra do rotor.
- J – Densidade de corrente dada em $ampère/metro^2$.
- J_T – Densidade total de corrente dada em $ampère/metro^2$.
- J_s – Densidade de corrente externa aplicada ao circuito dada em $ampère/metro^2$.
- K – Matriz de rigidez.
- N – Matriz de massa.
- n – Vetor unitário normal apontado para fora da fronteira.
- U – Tensão dada em $volt$.
- U_t – Tensão da barra do rotor dada em $volt$.
- V – Unidade de tensão elétrica dada em $volt$.
- $\widehat{W}$ – Função Peso.
- w_i – Função peso discreta.
- x – Coordenadas (x, y).
- T_1 – Tempo total de uma execução serial.
- T_P – Tempo total de uma execução paralela composta por P processadores.

- Γ – Fronteira
- Γ_u – Fronteira de Dirichlet.
- Γ_t – Fronteira de Neumann.
- Δ – Operador Lapaciano.
- δu – Função de teste para a função de campo.
- $\delta\lambda$ – Função de teste para o multiplicador de Lagrange.
- ϕ_i – Função de forma discreta.
- λ – Multiplicador de Lagrange.
- μ – Permeabilidade do material dada em $henry/metro$.
- υ – Relutividade do material dada em $(ampère\ x\ metro)/weber$.
- σ – Condutividade do material dada em $siemens/metro$.
- ∇ - Operador gradiente.
- Ω – Domínio.

Lista de Abreviações

- API – *Application Programming Interface*
- CPU – *Central Processing Unit (Unidade Central de Processamento)*
- MED – *Método dos Elementos Difusos*
- EFGM – *Element-Free Galerking Method*
- MDF – *Método das Diferenças Finitas*
- MEF – *Método dos Elementos Finitos*
- MVF – *Método dos Volumes Finitos*
- IMLS – *Interpolating Moving Least Squares*
- LPIM – *Local Interpolation Method*
- MIMD – *Multiple Instruction, Multiple Data*
- MLPG – *Meshless Local Petrov-Galerking*
- MLS – *Moving Least Squares*
- MPI – *Message-Passing Interface*
- MUMPS – *Multifrontal Massively Paralllel Solver*
- OpenMP – *Open Multi Processing*
- PCAM – *Partitioning, Communication, Aglomeration and Mapping*
- EDP – *Equações Diferenciais Parciais*
- PIM – *Point Interpolation Method*
- RKPM – *Reproduciong Kernel Particle Method*
- SIMD – *Single Instruction, Multiple Data*
- SMPD – *Single Program, Multiple Data*
- SPH – *Smoothed-particle hydrodynamics*

Capítulo 1

Introdução

A programação paralela vem sendo aplicada em diversas áreas do conhecimento, pois contribui para a redução do tempo de processamento requerido por determinados algoritmos e, consequentemente, para a diminuição dos valores gastos com a aquisição de equipamentos (*hardware*) de alto desempenho computacional. O avanço das práticas de desenvolvimento utilizando programação paralela foram favorecidas pela necessidade constante de aprimoramento do desempenho computacional; na qual a implementação de cenários para simulações numéricas era necessária, mas era fundamental que essa ocorresse dentro de um escopo de simulação mais preciso [1]. A necessidade de alto desempenho computacional está atrelada diretamente a complexidade da modelagem e execução de problemas computacionais atuais. Este tipo de necessidade favoreceu ao avanço de práticas utilizando a programação paralela [1].

Atualmente, processadores com vários núcleos vem se tornando cada vez mais comuns em computadores, celulares dentre outros dispositivos eletrônicos. Consequentemente, a elevação do número de núcleos aumenta o número de *threads* (fluxos de processamentos) disponíveis para se trabalhar, o que acarreta em uma gama maior de processamento em um mesmo equipamento [2]. A partir desta gama de processamento, foi possível começar a trabalhar com problemas cada vez mais complexos, como por exemplo cálculos numéricos que exigem operações matemáticas complexas com alto nível de precisão. Este tipo de problema, torna-se um grande candidato ao paralelismo computacional [3].

A computação paralela tem ajudado na resolução de diversos problemas distribuídos em áreas diversificadas, tais quais: a) Em engenharia, através de problemas gerais de integração numérica, cálculo de matrizes esparsas, decomposição em valores singulares; b) Em biomedicina, a qual utiliza estes recursos computacionais para determinar a estrutura tridimensional de vírus e calcular a área de superfície acessível à solventes de proteínas [4]; c) Em áreas diversificadas através de modelos experimentais,

tais como a utilização de técnicas não destrutivas em simulações computacionais para modelos que, hoje, são utilizados em cobaias vivas, reduzindo o custo de simulações em laboratórios, além de reduzir também, em casos específicos, riscos biológicos e/ou químicos.

Conforme apresentado pela Figura 1.1, a maior concentração da utilização de computação paralela através de supercomputadores está no setor industrial, responsável por mais da metade do consumo mundial, ou seja, 56,4%. A seguir, com 20,6% da concentração, estão setores de pesquisas, como buscadores de conteúdo. Posteriormente, responsável por 16,6% estão setores de previsão de clima. O restante, 6,4%, estão distribuídos entre conteúdos governamentais, comércio e demais segmentos não classificados [5].

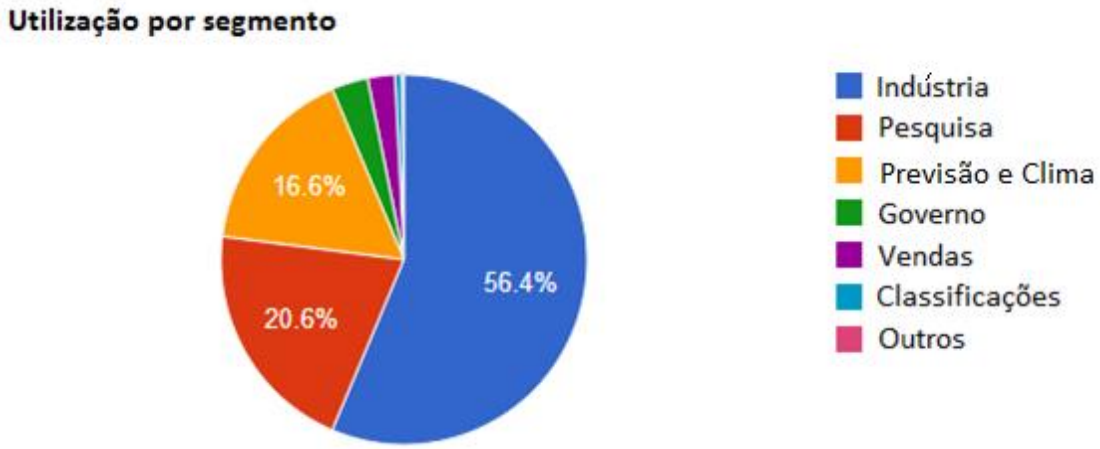

Figura 1.1: Distribuição por seguimento de processamento. Adaptado de [5].

Como forma de suprir a necessidade da crescente demanda de processamento, foram criadas algumas arquiteturas e tecnologias com o intuito de viabilizar a utilização de máquinas paralelas. Dentre esses modelos de programação, pode-se destacar o *Message-Passing Interface* (MPI) e o *Open Multi Processing* (OpenMP).

MPI é um modelo de programação muito utilizado entre máquinas paralelas que trabalham com memória distribuída, o qual consiste em distribuição de processos que se comunicam entre si, com possibilidade de distribuição entre operações singulares, os quais executam de forma independente ou coletivas suas tarefas. Além de prover recursos suficientes para agrupamento de tarefas, realização de contextualizações de comunicação e aplicabilidade a nível de topologia para a aplicação [6]. Uma das características da

utilização do MPI é o fato de que o mesmo código pode ser executado em diversos pontos com arquiteturas diferentes, desde que tenha-se o MPI preparado para tal.

O OpenMP foi construído em 1997 com o objetivo de otimizar o processamento de algoritmos em máquinas que possuem vários núcleos, por ser uma biblioteca baseada em *threads*. Como benefício da utilização de recursos da computação paralela, o OpenMP concentra esforço para realizar o ganho de tempo de processamento através da utilização de memória compartilhada e multi-plataformas [7]. O modelo proposto pelo OpenMP provê funções de programação para as linguagens C/C++ e principalmente Fortran [8]. A aplicação do OpenMP traz como benefício a facilidade da utilização através de uma interface para comunicação entre os processos via memória compartilhada de forma fácil, simples e direta.

Neste trabalho são apresentadas técnicas de utilização de processos em computação paralela para a resolução de problemas relacionados ao eletromagnetismo, especificamente na linha de pesquisa da utilização de Métodos sem Malha. Esses métodos fazem parte de um grupo de métodos numéricos aplicados comumente na resolução de Equações Diferenciais Parciais (EDP) [9]. Eles têm como principal característica o fato de que torna-se desnecessária a utilização de malhas para a discretização de domínios [10] como aquelas usadas em Métodos de Elementos Finitos (MEF) [11] por exemplo. Os nós distribuídos através do *Meshless* não requerem conectividade direta entre os outros nós, tornando este tipo de abordagem bastante atrativa para resolução de problemas com modelagens tridimensionais complexas, ou com grande variação da geométrica. Todavia, esta abordagem possui um alto custo computacional, uma vez que para se determinar a conectividade entre os nós se faz necessário determinar quais são os vizinhos do nó em interesse através de regiões circulares, retangulares, entre outras. Além de que para realizar a construção da matriz da forma fraca, faz-se necessário a inversão de matrizes e outros cálculos matriciais complexos. Esses processos são muito custosos computacionalmente, tornando assim interessante a utilização da programação paralela focada na redução do tempo de execução.

A utilização da programação paralela também é frequentemente utilizada na solução de sistemas lineares que geralmente estão associadas em algum tipo de *Solver* (Solucionadores prontos de sistemas lineares ou não lineares). O Método sem Malha requer a solução de diversos sistemas lineares pequenos e de um sistema linear esparso

grande, assim o emprego de solucionadores pode conduzir a um percentual significativo de redução do tempo de execução do problema sob análise.

Dessa maneira, este trabalho também apresenta a utilização da programação paralela aplicada diretamente sobre os solucionadores através da biblioteca PARDISO [12], uma biblioteca *thread-safe* (manipula estrutura de dados compartilhada garantindo a execução segura através de várias *threads* ao mesmo tempo) com alto desempenho, robusta e com grande eficiência do uso de memória. A biblioteca é usada, principalmente, para a solução de equações lineares esparsos simétricos e não simétricos através do processo de memória compartilhada e distribuída entre processadores.

Além do *PARDISO*, este trabalho apresenta também investiga a aplicação do *Solver Multifrontal Massively Parallel Solver* (MUMPS) [13] [14], um *Solver* também paralelo, mas com a arquitetura desenvolvida através da utilização do MPI e focado em resoluções de problemas em larga escala, possibilitando, assim, a utilização da computação paralela híbrida entre o MPI e o OpenMP.

1.1. Objetivos

O objetivo deste trabalho é o estudo e aplicação de modelos de paralelização para resolução de algoritmos em matemática computacional direcionados à problemas de eletromagnetismo por meio do Método sem Malha objetivando melhorias no tempo de processamento.

Com o intuito de realização do objetivo proposto, tem-se como meta:

- Estudo de técnicas de programação paralela para métodos numéricos direcionados a problemas em eletromagnetismo.
- Estudo e implementação de algoritmos através da linguagem Fortran F90 combinada entre o OpenMP, MPI, e os *Solvers* PARDISO e MUMPS.
- Aplicação das técnicas de programação paralela em um problema matemático de uma máquina de indução trifásica.
- Comparação de resultados obtidos entre modelo serial e o paralelo para a identificação das vantagens e desvantagens da abordagem proposta.

- Apresentação de resultados diretos através de gráficos, planilhas, cálculos de *speedup* (medida de ganho de desempenho) dentre outros mecanismos de mensuração de resultados.

1.2. Justificativa

Cada vez mais se torna evidente que a demanda por supercomputadores é uma necessidade para resolução de problemas mais complexos e extensos. Sabe-se que uma das formas para melhoria de tempo de processamento é a utilização da computação paralela. Em decorrência disso, este trabalho pretende avaliar esse tipo de arquitetura na solução numérica em eletromagnetismo, uma vez que, na maioria de problemas dessa natureza o custo de processamento é extremamente elevado e é fundamental a obtenção de precisões numéricas. Além disso, este estudo pretende contribuir com a difusão do conceito de computação paralela em Método sem Malha.

1.3. Estrutura do Trabalho

O Capítulo 2 apresenta um histórico sobre a computação paralela, o conceito geral do paralelismo, histórico de aplicação, evolução, e tendências da computação paralela. O capítulo apresenta também conceitos fundamentais de *software* e *hardware* paralelo, e as principais métricas. Por fim, são apresentados as tecnologias OpenMP, MPI, *Solvers* PARDISO e MUMPS e o conjunto denominado comutação híbrida.

No Capítulo 3 são apresentados os conceitos físicos do problema proposto neste trabalho, além de uma contextualização dos métodos sem malha, em que são explanados conceitos de domínios de influência, domínios de suporte, funções de forma, e Métodos sem Malha como o *Element-Free Galerking Method* (EFGM). Neste capítulo é realizada uma apresentação do problema proposto neste trabalho com mais detalhes, e as estratégias de paralelização adotadas, além do detalhamento físico dos solucionadores MUMPS e PARDISO.

No Capítulo 4 são apresentados os diagnósticos e resultados dos experimentos realizado neste trabalho, divididos em estratégias de paralelização via OpenMP, seguido do MPI e por fim um modelo híbrido.

O Capítulo 5 apresenta as conclusões deste trabalho, com foco nas contribuições alcançadas além de propostas para os próximos trabalhos dando continuidade ao modelo apresentado.

Capítulo 2

A Utilização do Paralelismo Computacional

Durante o discorrer deste capítulo são apresentados conceitos da programação paralela, o porquê de sua utilização, seu histórico além de requisitos mínimos necessários para sua aplicação.

2.1. Conceituando o Processamento Paralelo

Entende-se por programação paralela a estratégia através da qual é possível a obtenção do resultado de um algoritmo através da divisão e distribuição de sua execução em determinadas partes. Essa abordagem pode ser realizada por meio de pequenas parcelas respectivas ao algoritmo executadas através de *threads* específicas, agrupando o resultado individual no final de sua execução. Este modelo é denominado *fork-join* [15] e é apresentado na Figura 2.1. Já a Figura 2.2 apresenta o modelo tradicional ou serial em que a execução do algoritmo é realizada por um único processo [15] [16].

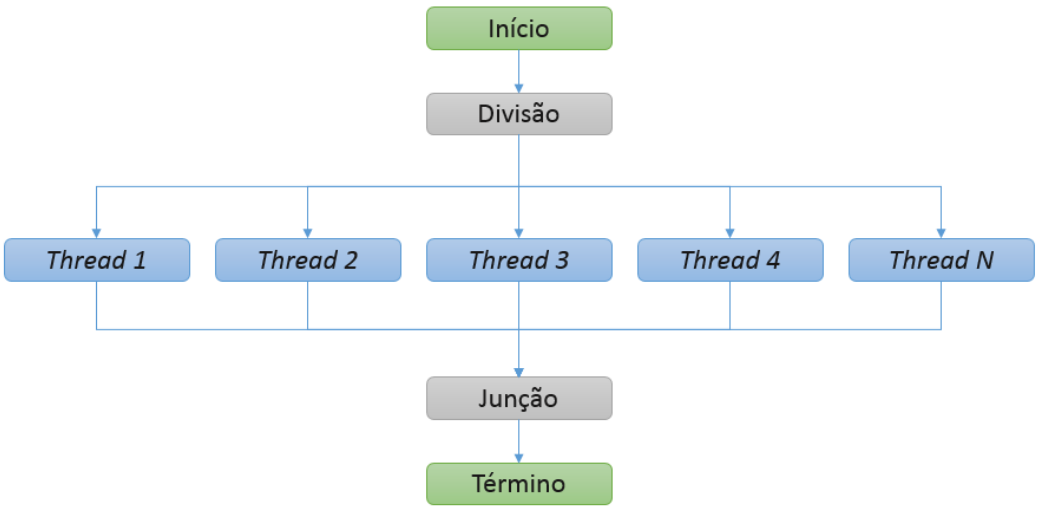

Figura 2.1: Representação de algoritmos paralelos através do modelo *fork-join*. Adaptado de [16][19].

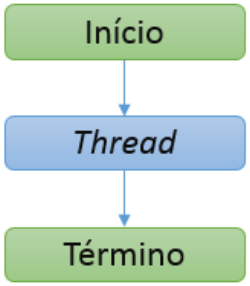

Figura 2.2: Representação da execução de um algoritmo via programação serial. Adaptado de [16].

Outro modelo muito utilizado na programação paralela é a divisão de tarefas em processos, tal que cada processo é responsável pela execução de um trecho do código previamente dividido [16]. Conforme definido pela taxonomia de Flynn [17], um dos modelos propostos para o processamento paralelo é chamado de *Multiple Instruction, Multiple Data* (MIMD). Este modelo propõe que vários processadores autônomos executem simultaneamente diferente instruções com dados compartilhados comuns ou dados diferentes [18] *apud* [17].

Existem dois principais tipos de sistemas MIMD:

- Memória compartilhada (*Shared-memory*) apresentado na Figura 2.3.
  - Uma coleção de processadores autônomos conectados a um sistema de memória através de uma rede, tal que cada processador pode acessar cada localização específica da memória.

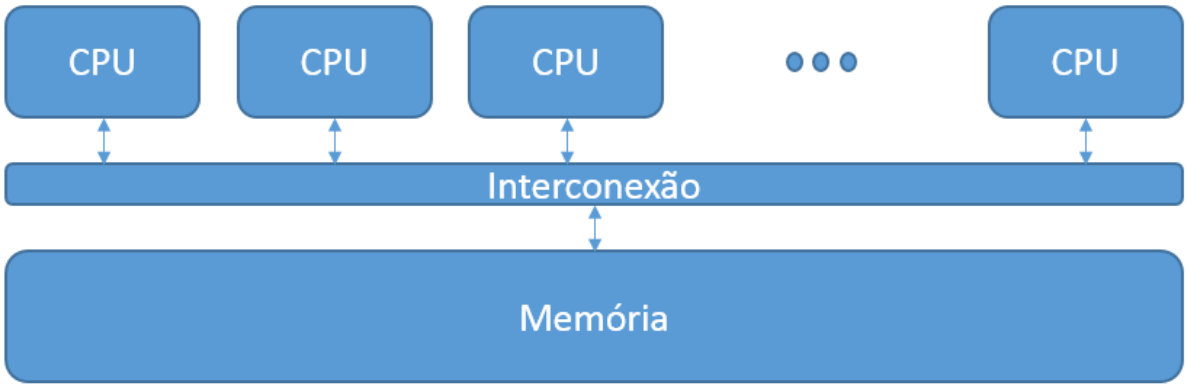

Figura 2.3: Modelo do sistema de Memória Compartilhada. Adaptado de [16].

- Memória distribuída (*Distributed-memory*) conforme ilustra a Figura 2.4.
  - Cada processador é emparelhado com sua própria memória. Um par é composto por (processador + memória).

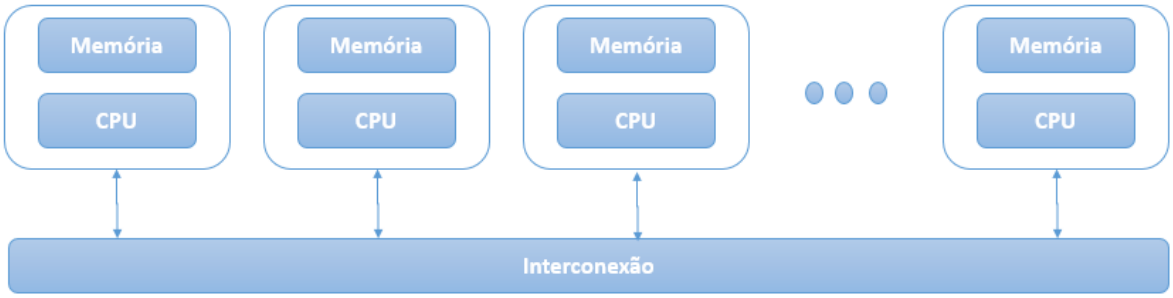

Figura 2.4: Modelo do sistema de Memória Distribuída. Adaptado de [16].

Além do modelo MIMD, existem outros três modelos propostos também por Flynn, *Single Instruction, single data* (SISD), *Single instruction, multiple data* (SIMD) e *Multiple instruction, single data* (MISD).

Sistemas do tipo SISD representam arquiteturas em que um único fluxo de instruções opera sobre um único fluxo de dados. Este tipo de processamento é característico de processamento serial ou sequencial. Este tipo de arquitetura é caracterizado por possuir uma única unidade de controle [17]. No SIMD um processamento de vários dados pode ser executado sob um único comando de apenas uma única instrução. Este modelo, apenas de realizar operações com vários dados, ainda possui características de programas sequenciais. O acesso a múltiplos dados requer uma organização em diversos módulos. Esses sistemas oferecem o compartilhamento de resultados, ou seja, o primeiro resultado pode ser compartilhado com os demais que são executados [17]. Por fim, modelos do tipo MISD são caracterizados por possuírem múltiplas etapas ou processos de controle executando operações distintas sobre um mesmo ponto. Este modelo é considerado impraticável com a tecnologia atual [17].

2.2. O Surgimento da Computação Paralela

A resolução de diversos problemas computacionais deixaram de se tornar viáveis uma vez que, a elevação da carga de dados em conjunto com o aumento da complexidade do algoritmo tornaram o tempo de execução muito extenso quando executado de forma serial. Baseado neste cenário, um grupo de pesquisadores e instituições localizadas principalmente nos Estados Unidos e Europa iniciaram um estudo que posteriormente se tornaria o MPI. Uma grande maioria de fabricantes de processadores também participou

desta pesquisa padronizando assim a forma de troca de mensagens em ambiente de memória distribuída pelo *Center for Reserach on Parallel Computing*, em 1992 [20].

Entre 1986 e 2002 o desempenho de microprocessadores aumentou em uma média de 50% ano [21]. Todavia, a partir de 2002 processadores *single-processor* obtiveram uma desaceleração cerca de 20% ao ano. A diferença de aumento de desempenho foi associada com uma dramática alteração de arquitetura de construção dos novos processadores. Devido a tentativa de elevação do número de registradores em um único processador, a indústria começou a enfrentar dificuldades relacionadas ao aquecimento e ao tamanho dos registradores [16].

Desde 2005, a grande maioria das empresas responsáveis pela construção de microprocessadores decidiram que, para obter um resultado que fosse significativo referente ao processamento, a utilização do paralelismo na execução de processos computacionais deveria ser aplicada [16] ao invés de manter tentativas de melhorias em processadores monolíticos. Essa decisão foi extremamente importante para o rumo da computação moderna, uma vez que, com a adição de mais processadores pode-se obter ganhos muito significativos em programas anteriormente executados de forma serial, ou seja, programas construídos para serem executados através de uma única *thread*.

Graças a estratégia adotada pela indústria de microprocessadores, operações muito complexas puderam ser realizadas; como por exemplo, a decodificação do genoma humano, máquinas de busca, jogos computacionais com inteligência artificial, já que, a ausência dessa abordagem, impossibilitaria este tipo de avanço [16].

2.3. Tendências e Aplicações Presentes

Considerando uma linha do tempo traçada a partir de 2008, a computação paralela quebrou uma importante barreira, a quebra dos *Petaflops* [5]. Em Junho de 2008, o supercomputador *RoadRunner* ultrapassou a velocidade de 1 *Petaflop* por segundo, ou seja, 1 milhão de bilhões de cálculos por segundo. Este desempenho fez do *RoadRunner* mais de duas vezes mais rápido que seu antecessor.

Recentemente, em junho de 2011, um supercomputador japonês quebrou mais uma vez outra barreira, agora a de 8,16 quatrilhões de cálculos por segundo. O sistema chamado computador K, está no Instituto Avançado *Riken* para Ciência Computacional

(AICS) em Kobe. Este nome foi dado homenageando a marca de 10 quatrilhões ou *"kei"*. [5].

O cenário atual, atua com a combinação de diversos recursos objetivando o ganho relacionado ao desempenho, usando cada vez mais a capacidade máxima de núcleos computacionais, memória compartilhada, processamento via hardwares de vídeo, além da computação híbrida, mesclando todos os recursos com o intuito de obter o maior ganho possível no desempenho.

2.4. Software Paralelo

Em programas que usam o modelo de memória compartilhada, *threads* podem atuar com dois modelos distintos de compartilhamento de valores entre variáveis: a) variáveis privadas b) variáveis compartilhadas. É muito comum utilizar o modelo de variáveis compartilhadas tornando a troca de informações entre processos mais simples. Todavia, variáveis compartilhadas podem gerar alguns problemas, tais como problemas condições de corrida ou concorrência de valores [22].

Considere, por exemplo, que em uma execução paralela, duas *threads* tentam acessar o mesmo valor ao mesmo tempo; isso, provavelmente, irá resultar em erro no caso de variáveis compartilhadas. Entretanto, esse tipo de problema pode ser solucionado através da determinação de regiões críticas, ou seja, um bloco que poderá ser executado somente por uma única *thread* por vez [16]. Para a obtenção de bons resultados, recomenda-se que apenas pequenos trechos do código sejam candidatos a regiões críticas.

Outro ponto importante a ser observado é o fato de que um bloco de código pode ser acessado por múltiplas *threads*. Esse bloco de código deve manter o seu estado válido, mesmo sendo compartilhado por múltiplas *threads* ao mesmo tempo. Esse conceito é chamado de *thread safe.* Um bom exemplo para descrever esta situação é o de uma conta corrente em que uma pessoa possui $100,00 disponível para saque. Todavia, uma *thread* é executada com a solicitação de saque de $70,00 e de forma paralela uma outra *thread* dispara uma solicitação de um novo saque no valor de $60,00. Caso esse trecho de código não seja *thread safe* o cliente será beneficiado com um saque de $130,00 com apenas $100 em conta.

As etapas para a realização de um projeto paralelo devem ser bem definidas com objetivo de se obter os melhores resultados possíveis [23]. Visando este tipo de necessidade, a metodologia *Partitioning* (partição), *Communication* (comunicação), *Agglomeration* (Aglomeração), *and Mapping* (Mapeamento) (PCAM) proposta por [23] vem sendo aplicada uma etapa de projeto paralelo. Este modelo é apresentado na Figura 2.5, e consiste em quatro fases:

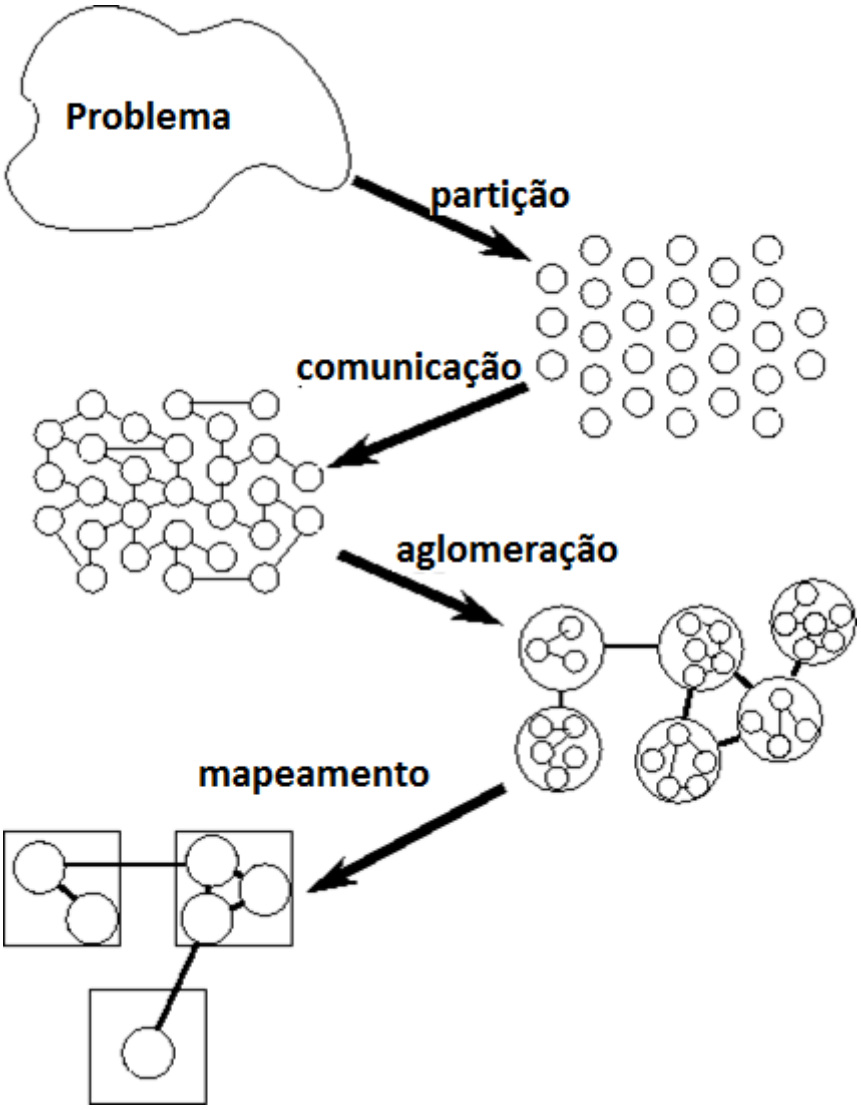

Figura 2.5: Metodologia PCAM de projetos para problemas paralelos. Adaptado de [23].

- Divisão ou partição:

- Todos os dados e cálculos a serem utilizados são decompostos em pequenas tarefas com foco em oportunidades de execução paralela. A divisão pode ser realizada de duas maneiras: Divisão por domínios ou Divisão funcional.
    - Divisões por domínios consistem em subdividir os dados associados a um determinado problema, caso possível, dividir em pequenos pedaços simétricos. Em seguida, divide-se o cálculo a ser executado, geralmente associado aos dados com a operação a ser executada. Este tipo de divisão produz series de tarefas. Pode ser que, em algumas operações, dados podem ser compartilhados, fazendo-se necessário a comunicação para obtenção destes dados. Um bom exemplo desta aplicabilidade é a Figura 2.6, a qual apresenta um modelo de divisão por domínios de uma, duas e três dimensões.

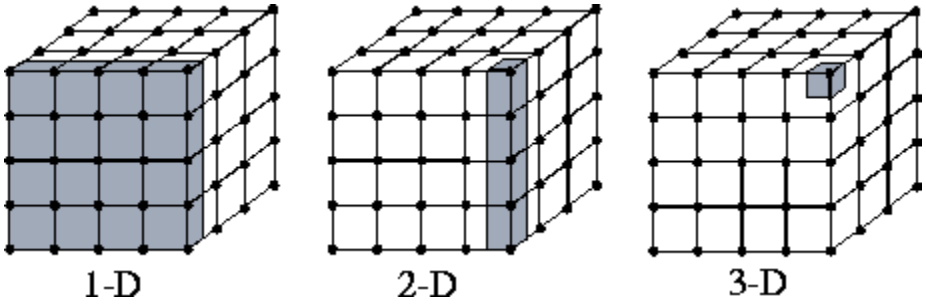

Figura 2.6: Exemplo de uma divisão por domínios [23].

- Divisões Funcionais são formas de divisões focadas exclusivamente em cálculos, ou seja, dispersando os dados em questão. Caso este tipo de divisão seja bem feita, passa-se a realizar análises diretamente sobre as tarefas, conforme apresentado na Figura 2.7, que representa um modelo de clima tal que pode-se pensar em cada etapa como uma tarefa separada. As conexões representam a comunicação entre os dados e os componentes durante a execução dos cálculos, ou seja, cada componentes depende de dados do outro componente.

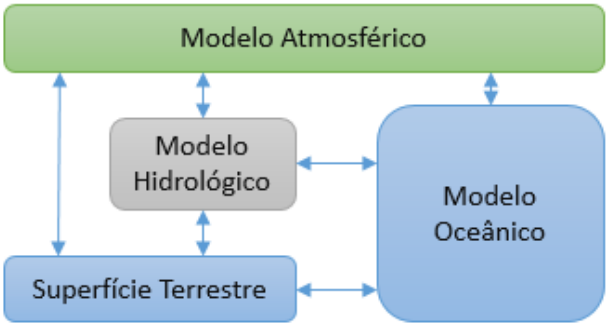

Figura 2.7: Modelo de divisão funcional. Adaptado de [23].

- Comunicação:
  - Definição da estrutura dos algoritmos e o meio e comunicação necessário para coordenar as tarefas. A comunicação pode ser classificada como: Comunicação Local e Comunicação Global.
    - A comunicação local acontece quando uma operação requer dados de um pequeno número de tarefas, facilitando assim a comunicação entre os fornecedores e consumidores dos dados. A Figura 2.8 apresenta um exemplo prático da comunicação local em resolução de problemas de Elementos Finitos em que cada tarefa encapsula um único elemento de grade bidimensional objetivando comunicar apenas com seus quatros vizinhos definidos.

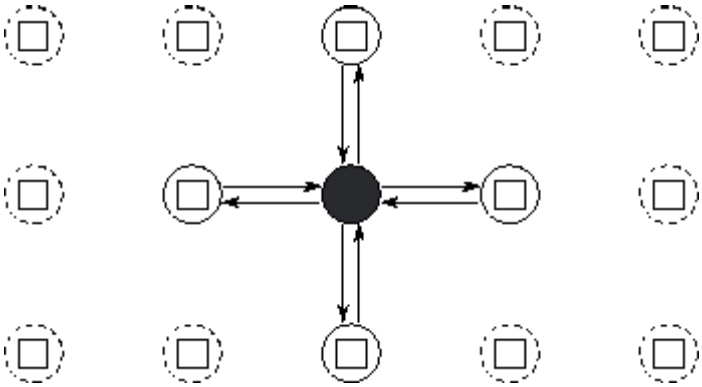

Figura 2.8: Exemplo de Comunicação Local em Elementos Finitos [23].

    - A comunicação global representa o cenário em que várias tarefas compartilham dos mesmos dados, fazendo com que apenas determinar grupos de operadores e dados não sejam suficientes. A

Figura 2.9 apresenta um algoritmo de soma centralizada, em que a tarefa central *S* gerencia cada um dos outros oito canais de cálculo.

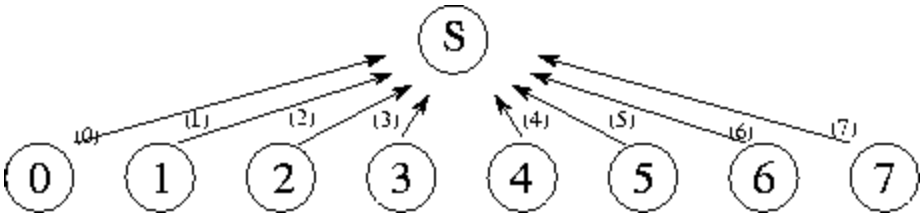

Figura 2.9: Exemplo de operação centralizada utilizando comunicação global [23].

- Aglomeração:
  - Análise das duas primeiras etapas, objetivando desempenho e custo de implementação. Caso necessário, as pequenas tarefas são reagrupadas em tarefas maiores.
  - Esta etapa é extremamente importante, pois neste momento é que a execução das tarefas deixa de ser abstrata, uma vez que ao criar-se muito mais tarefas do que processadores pode gerar ineficiência do lado dos processadores por não ser possível gerenciar estas tarefas. Esta fase determina a forma como as operações devem ser realizadas, após a execução das etapas de divisão e comunicação. Esta etapa também é muito útil para agrupar tarefas semelhantes que foram previamente subdividas em processos de tamanho maior, conforme representado pela Figura 2.10.

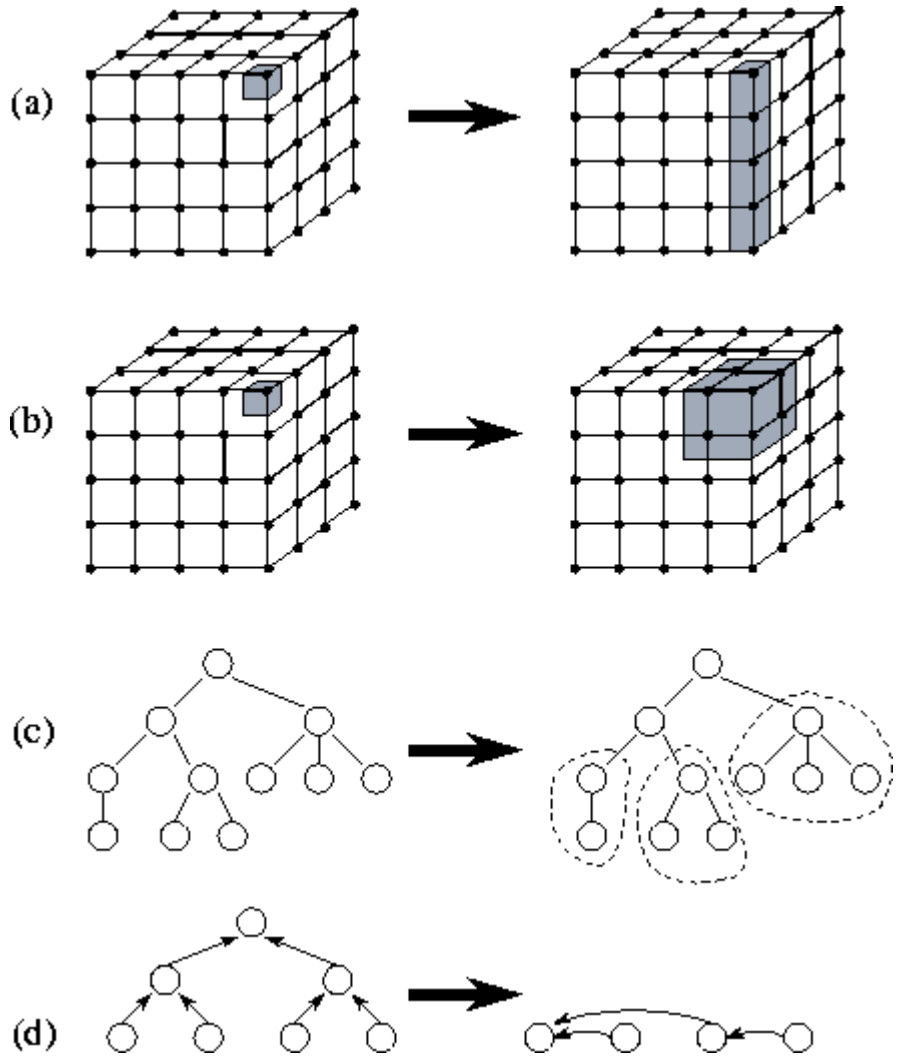

Figura 2.10: Exemplo da etapa de aglomeração [23], a) o tamanho das tarefas é elevado através da redução da dimensão da decomposição de três para dois, b) as tarefas adjacentes são combinadas com o objetivo de produzir uma decomposição tridimensional com maior granularidade, c) fusão de sub árvores de um modelo de divisão e conquista, d) nós de um algoritmo de árvore são combinados.

- Mapeamento:
  - Cada tarefa é associada a um processador com o intuído de garantir um melhor desempenho computacional e otimizar os custos de comunicação.
  - Nesta etapa são definidos onde cada tarefa é executada. A Figura 2.11 apresenta um modelo de divisões por grades, tal que, cada tarefa realiza o mesmo número de operações e que cada tarefa se comunica com no máximo seus quatro vizinhos. Já a Figura 2.12 apresenta um mapeamento balanceado.

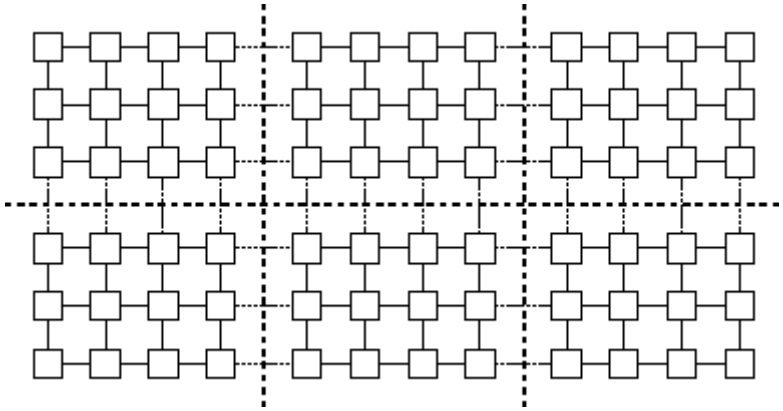

Figura 2.11: Exemplo de mapeamento por grade [18].

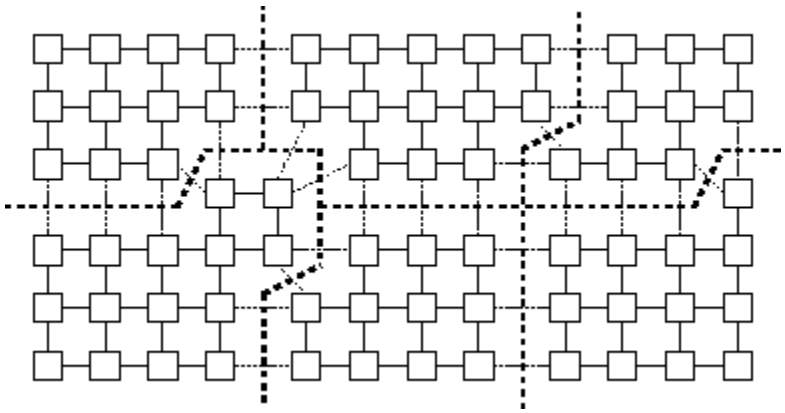

Figura 2.12: Exemplo de mapeamento balanceado [23].

A utilização desta metodologia tende a fornecer a um programa a autonomia de criar e destruir tarefas dinamicamente. Esta arquitetura se comporta muito bem também com modelos *Single Program Multiple Data* (SPMD), em que todos os processadores executam as mesmas funções, ou seja, uma tarefa específica para o processador.

2.5. Desempenho em Processamento Paralelo

Esta seção tem o objetivo de descrever modelos e métricas utilizadas para mensurar o desempenho durante a execução de processos paralelos. A análise do desempenho da aplicação, através de modelos de desempenho, possibilita a comparação de algoritmos, análise de escalabilidade e a identificação de estrangulamentos nas aplicações antes de investir um esforço substancial na implementação proposta.

2.5.1. Speedup

Quando utiliza-se a programação paralela, tem-se como objetivo o aumento na velocidade de execução dos programas. Para calcular o desempenho da execução paralela, utiliza-se o cálculo do *speedup* dado pela razão do tempo de execução serial em uma máquina e o tempo de execução em paralelo, denotado por:

$$Speedup = \frac{T_1}{T_p} \tag{2.1}$$

em que T_1 é o tempo requerido pela execução serial e T_p é o tempo de execução do código paralelo composto por P processadores [24].

2.5.2. Lei de Amdhal

Habitualmente, são comuns em computação paralela o fato de que em meio da execução paralela, existem partes sendo executadas de forma serial. Logo, o *speedup* é limitado por esta parte serial. Este estudo formulado por Gene Amdahl [25] o qual formulou a lei de Amdahl, preconiza a verificação do quanto de um algoritmo pode ser executado de forma paralela. Esta lei descreve problemas enfrentados pela indústria durante o desenvolvimento de máquinas com vários núcleos e com um número cada vez maior de processadores: o software que roda nessas máquinas deve ser adaptado para um ambiente de execução altamente paralelo para explorar o poder do processamento paralelo. [25]

2.6. Passagem de Mensagem (Message Passing Interface - MPI)

MPI é um modelo de programação amplamente utilizando máquinas paralelas, geralmente máquinas com memória distribuída. O conceito geral pode ser caracterizado como processos que se comunicam através de mensagens [6].

Frequentemente acontecem associações entre o modelo MIMD e o MPI devido ao formato nativo do MPI que consiste em um ou vários processos que se comunicam através de envio e recebimento de mensagens entre os processos. Geralmente, um conjunto de

processos é criado estaticamente, todavia, estes processos podem executar diferentes programas em tempos diferentes.

A Figura 2.13 descreve os três modelos possíveis de *hardware* em que pode-se aplicar o paralelismo com o MPI: 1) ambientes com memória compartilhada; 2) ambientes com memória distribuída; 3) ambientes híbridos, com memória compartilhada e distribuída.

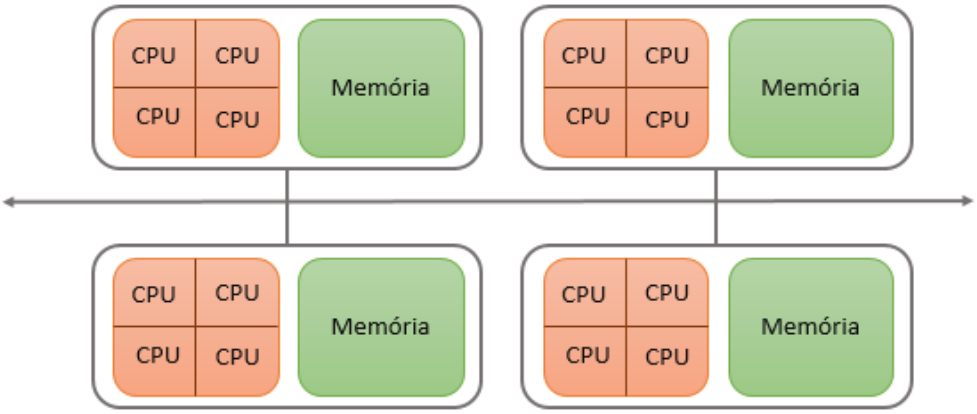

Figura 2.13: Funcionamento do MPI em memória compartilhada, memória distribuída ou de forma híbrida. Adaptado de [26].

O MPI pode usar processos de comunicação ponto-a-ponto, enviando informações de um determinado processo a outro processo. Outro grande fator considerado importante para o MPI é o fato de suportar programação assíncrona e modular através de comunicadores ou *communicator* encapsulando módulos do programa. Em geral, pode-se definir o MPI com algumas funções básicas, as quais já suportam o processo do paralelismo. São elas:

- MPI_INIT
  - Inicialização de uma execução
- MPI_FINALIZE
  - Finalização de uma execução
- MPI_COMM_SIZE
  - Responsável em determinar o número de processos a ser utilizado
- MPI_COMM_RANK
  - Responsável por determina a identificação dos processos
- MPI_SEND
  - Envio de mensagens entre processos
- MPI_RECV

- Recebimento de mensagens entre processos

A troca de mensagens geralmente está associada a um conjunto de computadores com memória distribuída, entretanto o MPI também pode ser utilizado através de uma única máquina multiprocessada. Para tal, existem implementações diferenciadas para cada tipo de solução [6].

Uma outra grande vantagem do MPI é a execução em diferentes ambientes, todos com MPI e com arquiteturas diferentes, ou seja, não é necessário se preocupar com as diferenças existes uma vez que estas são abstraídas. O MPI trabalha com conceitos de intercomunicadores, responsáveis por realizar as comunicações ponto-a-ponto entre os processos de diferentes grupos. O grupo que inicia o processo é chamado de grupo local [27].

Através do MPI é possível a criação de grupos de tarefas utilizadas para identificar tarefas associadas a uma operação coletiva. Este tipo de identificação pode fornecer o paralelismo diferenciado a cada grupo, associando assim o MIMD [6]. A Figura 2.14 descreve um modelo de grupos chamado *three-group pipeline* em que o grupo 0 comunica-se com o grupo 1, o grupo 1 comunica-se com os grupos 0 e 2 e o grupo 2 comunicação com o grupo 1. Este tipo de solução prevê intercomunicadores para cada grupo.

Figura 2.14: Exemplo do modelo *three-group pipeline*. Adaptado de [6].

Já a Figura 2.15 apresenta um modelo mais robusto, chamado *three-group ring*, tal que o grupo 0 comunica-se com o grupo 1 e o grupo 2, o grupo 1 comunica-se com o grupo 0 e 2, e finalmente o grupo 2 comunica-se com os grupos 1 e 0. Este tipo de implementação requer dois intercomunicadores para cada grupo.

Figura 2.15: Exemplo do modelo *three-group ring* adaptado de [6].

2.7. Open Multi-Processing - OpenMP

Considerando um sistema o qual trabalha com memória compartilhada, cada processador tem acesso direto à memória dos demais processadores. Este acesso permite escrita ou leitura de informações em qualquer localização (endereço) da memória compartilhada. Além das áreas comuns, pode-se definir também regiões privadas ou exclusivas a um determinado processador, fazendo com que seja possível gerenciar de forma extremamente eficiente o paralelismo computacional.

Com a modernização de computadores, fabricantes começaram a fornecer mecanismos de programação via memória compartilhada com mecanismos proprietários feitos em linguagem C e/ou Fortran. Além do MPI, criou-se o modelo *Pthreads* [8], em que se utiliza o modelo de memória compartilhada mas em baixo nível de desenvolvimento tanto em Fortran quanto em C, além de não ser escalável.

O OpenMP é uma *Application Programming Interface* (API) para programação paralela utilizada através do uso de memória compartilhada implementada de forma muito simples e direta, além de ser bem preciso focando diretamente no ponto específico a ser paralelizado. O sufixo "MP" em OpenMP apresenta a ideia de multi-processamento, que remete a computação paralela em memória compartilhada [7]. Em suma, o OpenMP consiste em um conjunto de instruções de compilação desenvolvidos em C/C++ e Frotran objetivando o paralelismo de trechos de código via memória compartilhada. Uma outra grande vantagem do OpenMP é o fato de não especificar o compilador a ser utilizado, desde que o mesmo dê suporte.

Após o processo de compilação, o trecho de código demarcado pelas instruções do OpenMP, transforma a seção serial em tarefas paralelas [28]. O processo utilizado pelo OpenMP para etapas de paralelismo é baseado no modelo *fork-join*, apresentado na Figura

2.1, em que todos os programas possuem uma *thread* principal a qual é executada de forma sequencial até que os trechos paralelos sejam inicializados através do conjunto de *threads* disponíveis. Por fim, a junção dos resultados é realizada, ocasionando o fim da solução paralela [7] [16].

A programação paralela utilizada via OpenMP prevê dois tipos de dados:

- Privados
  - Dados que são compartilhados entre todas as threads durante a execução do trecho de código paralelo.
- Compartilhados.
  - Dados que não exclusivos para a *thread* responsável pela sua execução durante o processo paralelo.

A Figura 2.16 apresenta um modelo de código paralelizado através da programação paralela via memória compartilhada com o OpenMP.

```
1   PROGRAM  Hello
2      IMPLICIT NONE
3      use omp_lib
4
5      INTEGER :: numThreads, identificador
6      INTEGER, EXTERNAL :: OMP_GET_THREAD_NUM, OMP_GET_NUM_THREADS
7
8
9      !$OMP PARALLEL private(numThreads, identificador)
10     identificador = OMP_GET_THREAD_NUM()
11
12     print *, "Ola! Eu sou a Thread numero: ", identificador
13
14     if (identificador == 0) then
15        numThreads = OMP_GET_NUM_THREADS()
16        print *, "Numero de Threads = ", numThreads
17     end if
18
19     !$OMP END PARALLEL
20     END
```

Figura 2.16: Código exemplo utilizando diretivas simples do *OpenMP*.

2.8. Modelos Híbridos de Programação Paralela

Devido aos avanços tecnológicos e arquiteturais recentes, tornou-se possível a junção de dois paradigmas da programação paralela. A troca de mensagens MPI e a programação via memória compartilhada OpenMP [23] [16]. Este fator contribui para obter o máximo de cada tipo de arquitetura, como a troca de mensagens realizada pelo

MPI e o paralelismo via memória compartilhada do OpenMP conforme apresentado na Figura 2.17 [29].

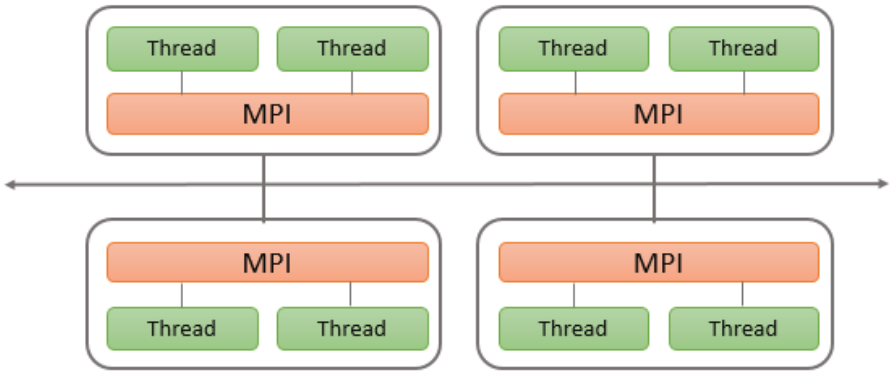

Figura 2.17: Modelo genérico híbrido. Adaptado de [31].

A programação híbrida entre o MPI e o OpenMP podem oferecer diversas vantagens[6] [8], tais como:

- Adequação às arquiteturas atuais baseadas em clusters multiprocessados reduzindo significativamente o número de comunicações entre diferentes nós.
- Aplicações que necessitam limitar o número de processos, podem utilizar o OpenMP para explorar o poder computacional dos processos residuais.
- Aplicações muito complexas em que se utiliza o balanceamento de carga, podem trabalhar com a distribuição granular com MPI e explorar o processamento de memória compartilhada pelo OpenMP.

2.9. Considerações Finais

Este capítulo apresentou os conceitos fundamentais da programação paralela bem como a metodologia PCAM proposta por [23] descrevendo o conceito de *design* paralelo. A utilização desta metodologia faz com que seja mais fácil identificar potenciais pontos paralelos em tempo de *design.* Além do PCAM, apresentou-se métrica como o *speedup*. Também foram apresentados conceitos do MPI e do OpenMP, duas ferramentas muito poderosas utilizadas para aplicar na prática o paralelismo, além da junção de ambas, formando o conceito de computação híbrida.

Capítulo 3

Paralelização do Método Sem Malha

3.1. Introdução

A modelagem computacional de um fenômeno físico frequentemente utiliza equações matemáticas que descrevem seu comportamento, sendo a maioria destes modelos baseados em EDPs. A próxima etapa constitui na solução destas EDPs através de um método numérico. A escolha correta do método e seus parâmetros intrínsecos passam a ser fatores decisivos em questões relacionadas à precisão, estabilidade e ao custo computacional envolvidos.

Dentre os diversos métodos comumente utilizados na solução de modelos matemáticos destacam-se aqueles que fazem uso de malhas como o MEF, os Métodos de Diferenças Finitas (MDF) [30] e o Método dos Volumes Finitos (MVF) [32]. Contudo, algumas características verificadas em determinados tipos de problemas motivaram a busca por novas técnicas numéricas na tentativa de resolver tais questões. Considerando, por exemplo, problemas em que as fronteiras são alteradas com o tempo, como uma máquina elétrica em movimento, ou quando trincas em uma superfície aumentam continuamente. Outro exemplo é uma estrutura que se deforma à medida que o tempo evolui. Nestes casos, quando é aplicado um método como o MEF, o que acontece normalmente é uma acentuada deformação nos elementos gerando perda de precisão. Para contornar este tipo de situação, uma nova malha é construída à medida que estas fronteiras se movem. Este processo além de não ser trivial, pode ser computacionalmente caro, principalmente em se tratando de geometrias tridimensionais. Para problemas desta natureza uma nova classe de método tem sido desenvolvida, os denominados Métodos sem Malha.

A origem dos métodos sem malha se deu por volta de 1977, quando Gingold e Lucy iniciaram os estudos focados na resolução de problemas em astrofísica. Esta classe de métodos teve origem com o *Smoothed Particle Hydrodynamics* (SPH) [33] ou Método de

Hidrodinâmica de Partículas Suavizado. O objeto deste método era a modelagem de um fluido através de um conjunto de partículas.

Contudo, somente a partir de 1990 tais métodos experimentaram um forte desenvolvimento, propiciado principalmente pela evolução da tecnologia de computadores digitais. A partir dessa data, surgiram diversos métodos sem malha tais como o Método dos Elementos Difusos (MED) [34], o EFGM, o *H-p Cloud Method* [35], o *Reproducing Kernel Particle Method* (RKPM) [36], o Método *Meshless Local Petrov-Galerkin* (MLPG) [37], o *Point Interpolation Method* (PIM) [38], o *Local Point Interpolation Method* (LPIM) [39] e o *Partition of Unit Method* [40], dentre outros.

Inicialmente os Métodos sem Malha tiveram como principais aplicações problemas ligados à mecânica computacional, área ainda explorada intensamente. Quanto à sua aplicação em eletromagnetismo, apesar de haver registros de trabalhos em 1992 [41], seu uso efetivo é bem mais recente, datando de meados da década de 1990 [42]. Atualmente estas aplicações alcançaram números expressivos com publicações em diversos periódicos e anais de congressos especializados. A principal característica que difere estes métodos de outros, como por exemplo, o MEF, consiste basicamente que nos Métodos sem Malha é feita a distribuição de uma quantidade de nós no domínio em estudo, sendo que nenhuma conexão ou relação é pré-estabelecida entre os mesmos. Estes nós constituirão o local em que as incógnitas deverão ser determinadas. Como é mostrado neste trabalho, esta característica facilita a modelagem de estruturas móveis.

Os Métodos sem Malha são métodos voltados a resolução de equações diferenciais com condições de contorno ou iniciais trabalham com a subdivisão de domínios principais em subdomínios. Estes subdomínios por sua vez contém pontos chamados nós dos quais obtêm-se os seus valores. Ao contrário do (MEF), que caracteriza a subdivisão dos domínios em subdomínios denominados elementos e compartilham seus nós e lados com seus vizinhos. Em Métodos sem Malha os subdomínios são regiões que cercam cada uma das partículas e apresentam pontos comuns sem conexões fortes entre fronteiras. Cada subdomínio está vinculado a um determinado nó, denominado domínio de influência [43].

Ainda assim, pode-se considerar recente os estudos de métodos sem malha voltados ao eletromagnetismo. Pode-se considerar que [41] foi um dos pioneiros na introdução dos métodos sem malha em eletromagnetismo através da aplicação do Método dos Elementos Difusos (MED) para simular um problema eletrostático bidimensional.

Esse trabalho apresenta as vantagens dos métodos sem malha tais como a ausência da construção de malhas e um conjunto de refinamentos facilitando a solução de regiões específicas.

3.2. Conceitos dos Métodos sem Malha

Métodos sem Malha são embasados por alguns conceitos fundamentais, são eles: a) Domínios de influência; b) Domínios de suporte; c) Funções de forma.

3.2.1. Domínios de Influência

O domínio de influência de um nó é definido como a região em que o nó exerce influência no domínio do problema [10] ou seja, para cada nó existe um subdomínio, no qual o ponto nodal respectivo contribui para a solução. Este subdomínio consiste simplesmente da região em torno do ponto em que uma função *w* (função peso) é diferente de zero. Ao contrário do FEM, em que os elementos devem ser disjuntos dois a dois, domínios de influência em algumas técnicas podem se superpor. Os domínios de influência podem ser circulares, retangulares ou outra geometria qualquer. A Figura 3.1 apresenta o modelo do domínio de influência retangular, já a Figura 3.2, o domínio de influência circular.

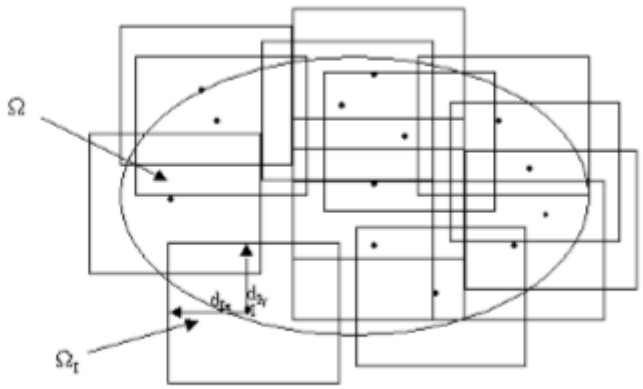

Figura 3.1: Domínios de influência retangulares [42].

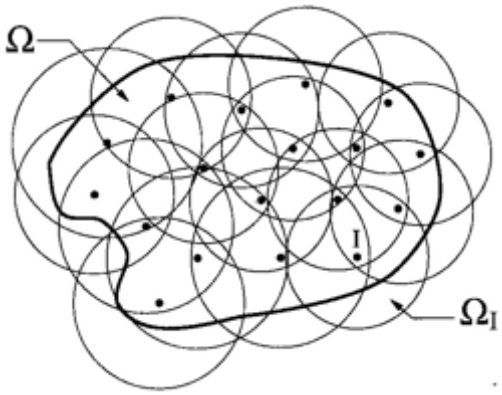

Figura 3.2: Domínios de influência circulares [42] *apud* [44].

Observando um ponto de interesse "*I*", pode-se concluir que a dimensão do domínio de influência é dada por:

$$d_I = \alpha_I d_c \tag{3.1}$$

tal que α_I é um valor adimensional relacionado ao domínio de influência e d_c é a distância média próxima ao nó de interesse. Entende-se que α_I influencia diretamente o tamanho do domínio de influência avaliado. Conforme apresentado em [42] e [43], valores de α_I entre 2,0 e 4,0 tendem a apresentar bons resultados.

3.2.2. Domínios de Suporte

Domínio de suporte para um ponto $\hat{x}$ qualquer dentro da região do problema pode ser descrito como sendo a região formada pela interseção de todos os domínios de influência relacionados àquele ponto, conforme apresentado na Figura 3.3 pela região sombreada representando o domínio de suporte [44].

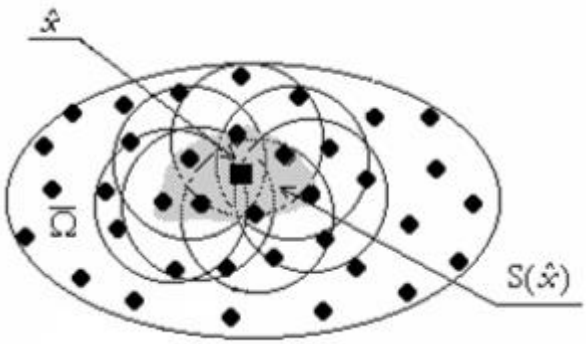

Figura 3.3: Domínio de suporte para um ponto $\hat{x}$ genérico [44].

3.2.3. Função de Forma

A decisão da escolha da função de forma tem um papel fundamental para a formulação de métodos sem malha segundo Liu em [43]. Liu também relaciona uma lista de passos a serem seguidos durante a fase de escolha e construção das funções, que caso sejam atendidas, garantiram uma facilidade maior para a implementação dos algoritmos e uma boa precisão nos resultados finais. Contudo, o desafio nos Métodos sem Malha está no desenvolvimento de funções de forma estáveis sem dependências de distribuições nodais pré-definidas. [42] *apud* [43].

3.3. Element-Free Galerkin Method - EFGM

O EFGM é um método sem malha cujas principais características são: 1) *Moving least squares* (MLS) [45] são geralmente empregados para a construção da função de forma [46] [47]; 2) O Método de *Galerkin* é utilizado para se chegar ao sistema final de equações; 3) Uma grade de células de integração é colocada em todo o domínio para se efetuar as integrais presentes na formulação [48] [49].

O MLS utiliza uma função de aproximação local dada por:

$$\boldsymbol{u}^{\boldsymbol{h}}(\boldsymbol{x}) = \sum_{j=1}^{m} \boldsymbol{p}_j(\boldsymbol{x})\boldsymbol{a}_j(\boldsymbol{x}) = \underbrace{\{1\ x\ y\ \dots\ p_m(\boldsymbol{x})\}}_{\boldsymbol{p}^T(\boldsymbol{x})} \underbrace{\begin{Bmatrix} \boldsymbol{a}_1(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{a}_m(\boldsymbol{x}) \end{Bmatrix}}_{\boldsymbol{a}} \tag{3.2}$$

tal que $\mathbf{x^T}$=[x,y] e $\boldsymbol{p}_j(\boldsymbol{x})$ é um vetor formado por monômios. Os parâmetros desconhecido $\mathbf{a}(\mathbf{x})$ são determinados minimizando a norma discreta em L_2 dada por [48]:

$$\boldsymbol{J} = \sum_{I=1}^{n} w(\boldsymbol{x} - \boldsymbol{x}_I)[\boldsymbol{p}^T(\boldsymbol{x}_I)\boldsymbol{a}(\boldsymbol{x}) - \boldsymbol{U}_I]^2 \tag{3.3}$$

tal que *w* é a função peso. A minimização da equação (3.3) leva a:

$$\boldsymbol{a}(\boldsymbol{x}) = \boldsymbol{A}^{-1}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})\boldsymbol{u} \tag{3.4}$$

em que

$$\boldsymbol{a}(\boldsymbol{x}) = \sum_{I=1}^{n} w(\boldsymbol{x} - \boldsymbol{x}_I)\boldsymbol{p}(\boldsymbol{x}_1)\boldsymbol{p}^T(\boldsymbol{x}_I) \tag{3.5}$$

e

$$\boldsymbol{B}(\boldsymbol{x}) = [w(\boldsymbol{x} - \boldsymbol{x_I})\boldsymbol{p}(\boldsymbol{x_I}) \cdots w(\boldsymbol{x} - \boldsymbol{x_n})\boldsymbol{p}(\boldsymbol{x_n})]. \tag{3.6}$$

Substituindo (3.4) em (3.3)

$$\boldsymbol{u}^h(\boldsymbol{x}) = \boldsymbol{p}^T(\boldsymbol{x})\boldsymbol{A}^{-1}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})\boldsymbol{u} \tag{3.7}$$

ou

$$\boldsymbol{u}^h(\boldsymbol{x}) = \sum_{I=1}^{n} \boldsymbol{\phi_I}(\boldsymbol{x})\boldsymbol{u_I} \tag{3.8}$$

em que

$$\boldsymbol{\phi_I}(\boldsymbol{x}) = \boldsymbol{p}^T\boldsymbol{A}^{-1}\boldsymbol{B}_I \tag{3.9}$$

é a função de forma utilizando o MLS.

Esta função de forma é utilizada para aproximar a incógnita da forma fraca do problema de modo similar ao FEM. Considere o seguinte problema bidimensional no domínio Ω delimitado por $\boldsymbol{\Gamma} = \boldsymbol{\Gamma}_t \mathbf{U} \boldsymbol{\Gamma}_u$

$$-\nabla.(k\nabla u) = b \; em \; \Omega \tag{3.10}$$

$$-k\frac{\partial u}{\partial n} = \bar{t} \; em \; \Gamma_t \tag{3.11}$$

$$u = \bar{u} \; em \; \Gamma_u \tag{3.12}$$

Pelo fato de que a função desenvolvida através do MLS não atende ao delta de Kronecker, faz-se necessário a imposição das condições de contorno essenciais através dos Multiplicadores de Lagrange ou usando o Método das Penalidades por exemplo [42].

Assim, considerando as funções de teste $\mathbf{u(x)} \in \mathbf{H}^1$ e os multiplicadores de Lagrange $\lambda \in \mathbf{H}^0$ a forma variacional da Equação 3.10 é colocada da seguinte forma: para todas as funções de teste $\boldsymbol{\delta}\mathbf{u}(\mathbf{x}) \in \mathbf{H}^1$ e $\delta\lambda(\mathbf{x}) \in \mathbf{H}^0$,

$$\begin{aligned} &\int_{\Omega} (\nabla\delta u)^T (k\nabla u) d\Omega - \int_{\Omega} \delta u b d\Omega + \int_{\Gamma_t} \delta u \bar{t} d\Gamma_t \\ &- \left[\int_{\Gamma_u} \delta\lambda(u - \bar{u}) d\Gamma_u + \int_{\Gamma_u} \delta u \lambda d\Gamma_u\right] = 0. \end{aligned} \tag{3.13}$$

Note que H^1e H^0 denotam o espaço de Hilbert de grau um e zero, respectivamente. Com o intuito de impor as condições de contorno de Dirichlet, faz-se necessário a substituição do MLS pelo *Interpolating Moving Least Squares* (IMLS) [42] *apud* [45] nos termos em (3.13), descartando assim os multiplicadores de Lagrange, uma vez que as funções de forma do IMLS possuem propriedades da função de delta de Kronecker. Modelo semelhante ao adotado em MEF [21].

O Sistema final de equações, após descartar os termos relacionados aos multiplicadores de Lagrange, fazendo-se uso de elementos do IMLS, pode ser obtido substituindo a forma discreta da função de teste em (3.6):

$$\boldsymbol{KU} = \boldsymbol{F} \tag{3.14}$$

$$\boldsymbol{K}_U = \int_{\Omega} \boldsymbol{T}_I^T k \boldsymbol{T}_J d\Omega \tag{3.15}$$

$$\boldsymbol{T}_I = \begin{bmatrix} \phi_{I,x} \\ \phi_{I,y} \end{bmatrix} \tag{3.16}$$

$$\boldsymbol{F}_I = \int_{\Omega} \boldsymbol{\phi}_I b d\boldsymbol{\Omega} - \int_{\Gamma_t} \boldsymbol{\phi}_I \bar{t} d\boldsymbol{\Gamma} \tag{3.17}$$

3.4. Modelagem da Máquina de Indução

O caso para estudo adotado neste trabalho consiste de uma máquina de indução trifásica de 2HP, quatro polos, com alimentação em corrente e rotor do tipo gaiola de esquilo. Como aproximação o ferro da máquina foi considerado um material com resposta linear. A Figura 3.4 apresenta a configuração do motor utilizado. O modelo proposto está apresentado em duas dimensões e com o domínio reduzido para um quarto da máquina por motivo de simetria.

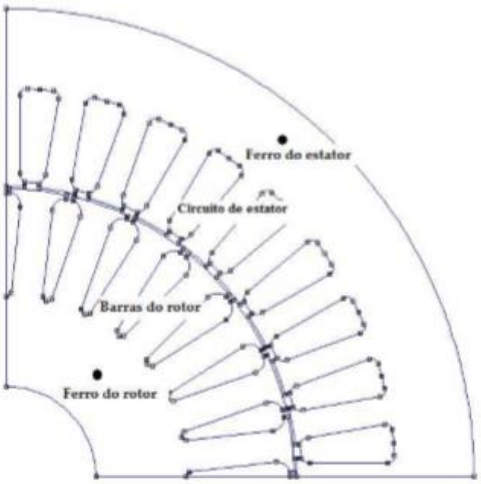

Figura 3.4: Geometria da máquina de indução [42]

As seguintes equações representam o modelo de acoplamento campo circuito elétrico da máquina [42], [43]:

$$\nabla . v \nabla \boldsymbol{A} - \boldsymbol{J}_S = 0 \tag{3.18}$$

no estator:

$$\nabla . v \nabla \boldsymbol{A} - \sigma \frac{\partial \boldsymbol{A}}{\partial t} + \sigma \frac{U_t}{l} = 0 \tag{3.19}$$

no rotor, com:

$$\boldsymbol{U}_t = \boldsymbol{R}_t \boldsymbol{I}_t + \boldsymbol{R}_t \int_{S_t} \sigma \frac{\partial \boldsymbol{A}}{\partial t} ds \tag{3.20}$$

em que $\boldsymbol{A}$ é o potencial vetor magnético, v é a relutância magnética, $\boldsymbol{\sigma}$ é a condutividade elétrica e $\boldsymbol{J_S}$ é a densidade de corrente imposta ao estator. $\boldsymbol{U_t}$ é a tensão nos terminais das barras do rotor e $\boldsymbol{I_t}$ é a corrente que atravessa o mesmo. $\boldsymbol{S}_t$ é a área da seção de cada condutor do rotor, l seu comprimento e $\boldsymbol{R}_t = (\boldsymbol{\sigma S}_T)^{-1}$ refere-se a resistência d.c. do rotor.

As equações de circuito do rotor podem ser expressas por (3.21), em que foi utilizada a Lei de Kirchhoff [51].

$$\boldsymbol{C_1^T C_1 U_t} + \boldsymbol{C_2 I_t} = 0 \tag{3.21}$$

em que

$$\boldsymbol{I} = [I_1 I_2 I_3 \cdots I_n]^T \tag{3.22}$$

$$\boldsymbol{C_1} = \begin{bmatrix} -1 & 0 & 0 & \dots & f \\ 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{bmatrix} \tag{3.23}$$

$$\boldsymbol{C_2} = \begin{bmatrix} 2r & 0 & 0 & \dots & 0 \\ 1 & 2r & 0 & \cdots & 0 \\ 0 & 1 & 2r & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2r \end{bmatrix} \tag{3.24}$$

$$\boldsymbol{U}_t = [\boldsymbol{U}_{t1} \boldsymbol{U}_{t2} \boldsymbol{U}_{t3} \cdots \boldsymbol{U}_{tn}]^T \tag{3.25}$$

Considerando que a relação $\boldsymbol{I}$ e $\boldsymbol{I}_t$ pode ser expressa por

$$\boldsymbol{I}_t = \boldsymbol{C}_1^T \boldsymbol{I} \tag{3.26}$$

segundo a lei de Kirchhoff.

O EFGM é então aplicado nas equações de (3.18) a (3.20), resultando em:

$$\boldsymbol{KA} + \boldsymbol{N} \frac{d}{dt} \boldsymbol{A} - \boldsymbol{PU}_t - \boldsymbol{J} = 0 \tag{3.27}$$

$$\boldsymbol{Q} \frac{d}{dt} \boldsymbol{A} + \boldsymbol{C}_3 \boldsymbol{U}_t + \boldsymbol{RI}_t = 0 \tag{3.28}$$

em que:

$$\boldsymbol{K}(k,j) = \int_{\Omega} \nabla \boldsymbol{\phi}_k^T v \nabla \boldsymbol{\phi}_j d\boldsymbol{\Omega} \tag{3.29}$$

$$\boldsymbol{N}(k,j) = \int_{\Omega} \sigma \boldsymbol{\phi}_k^T \phi_j d\boldsymbol{\Omega} \tag{3.30}$$

$$\boldsymbol{P}(k,j) = \int_{\Omega} \frac{\sigma_j}{l} \boldsymbol{\phi}_k d\boldsymbol{\Omega} \tag{3.31}$$

$$\boldsymbol{Q}(k,j) = \int_{\Omega} \boldsymbol{R}_{tk} \boldsymbol{\sigma}_k \boldsymbol{\phi}_j d\boldsymbol{\Omega} \tag{3.32}$$

Os elementos do vetor $\boldsymbol{J}$ são expressos por:

$$\boldsymbol{J}(k) = \int_{\Omega} \boldsymbol{J}_S\,(t) \boldsymbol{\phi}_k d\boldsymbol{\Omega} \tag{3.33}$$

Após aplicar o esquema de diferenças finitas de Euler em (3.27) e (3.28), utilizando (3.21) e fazendo uso das matrizes auxiliares $\boldsymbol{C_1}$ (3.23) $\boldsymbol{C_2}$ (3.24) necessárias nas equações de circuito, e $\boldsymbol{C_3}$ dada por (3.34),

$$\boldsymbol{C_3} = \begin{bmatrix} -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{bmatrix} \tag{3.33}$$

o sistema final de equações é obtido:

$$\begin{bmatrix} \boldsymbol{K} + \dfrac{\boldsymbol{N}}{\Delta t} & -\boldsymbol{P} & \boldsymbol{0} \\ \dfrac{\boldsymbol{Q}}{\Delta t} & \boldsymbol{C_3} & \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{C_1^T C_1} & \boldsymbol{C_2} \end{bmatrix} \begin{bmatrix} \boldsymbol{A}(t+\Delta t) \\ \boldsymbol{U_t} \\ \boldsymbol{I_t} \end{bmatrix} = \begin{bmatrix} \dfrac{\boldsymbol{NA}(t)}{\Delta t} + \boldsymbol{J}(t+\Delta t) + \boldsymbol{T} \\ \dfrac{\boldsymbol{QA}(t)}{\Delta t} \\ \boldsymbol{0} \end{bmatrix} \tag{3.34}$$

Um dos resultados obtido por [42] empregando a metodologia apresentada e ilustrado pela Figura 3.5, é a distribuição do fluxo magnético ao longo da máquina.

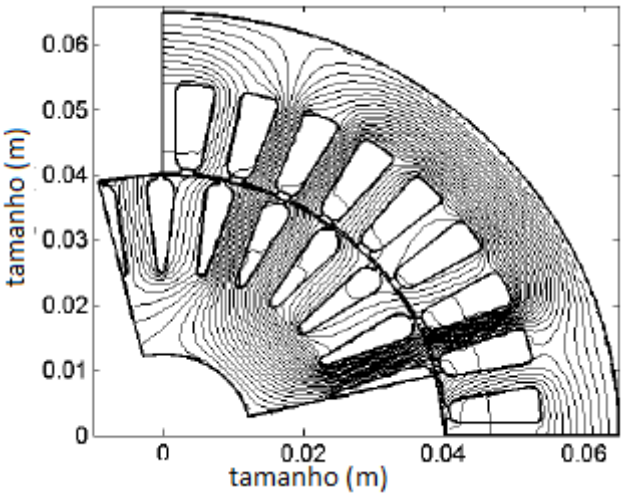

Figura 3.5: Distribuição do fluxo magnético (Δt = 17)

3.5. Algoritmo Computacional para Solução do Problema da Máquina de Indução

Nesta Seção, será apresentado o algoritmo desenvolvido para a solução do problema proposto da máquina de indução trifásica. A Figura 3.6 apresenta o fluxograma do *Meshless* utilizado para resolução do referido problema.

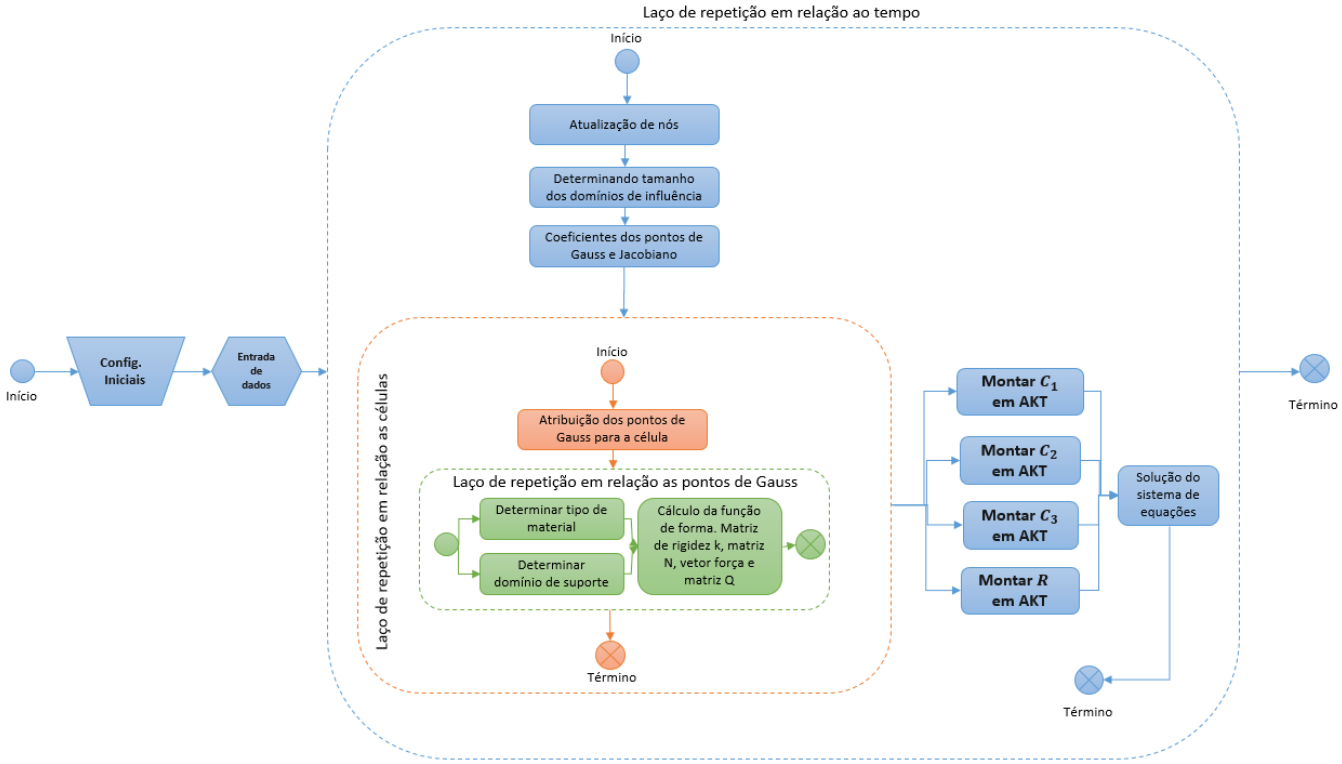

Figura 3.6: Fluxograma para uma máquina de indução baseada em um modelo *Meshless.* Adaptado de [31].

Conforme descrito por [42] a escolha do método sem malha se deu pelas características do problema e levou-se em conta alguns aspectos, a saber:

- A utilização de fronteiras móveis com a combinação de geometria complexa.
- Utilização de materiais diversos na estrutura.
- O problema possui natureza dinâmica.
- Existência de fronteiras periódicas no modelo.
- Considerando o grau de complexidade do problema, são necessários métodos reconhecidamente estáveis.

Baseado nas questões levantadas, o método escolhido para a representação do problema da máquina de indução é o EFGM [42].

O algoritmo apresentado por [42] consiste em preparar os dados iniciais para execução do modelo, ou seja, fontes de dados de entrada, posteriormente o problema entra em um laço de repetição por tempo ($\Delta T\ da\ máquina$). A cada laço de tempo, o algoritmo realiza as seguintes operações:

- Atualização dos nós
- Determina o tamanho do domínio de influência
- Efetua o cálculo dos coeficientes dos pontos de *Gauss e o Jacobiano*

Ainda dentro do laço de tempo da máquina, um outro laço importante é iniciado sobre as células de integração, com o objetivo de realizar as integrais da formulação e sequencialmente montar as matrizes responsáveis por obter a solução do problema. Além disto o laço sobre as células apontam os pontos de Gauss da célula em questão. Esta ação é necessária para a determinação dos domínios de suporte que consequentemente agregarão conteúdo para a função de forma. Outro aspecto importante dentro deste laço, consiste da atualização das coordenadas dos nós, que corresponderá ao movimento da máquina.

3.6. Estratégia de Paralelização

O fato do problema proposto ser complexo e trabalhar com a manipulação de uma grande quantidade de dados compostos de valores com precisões decimais, faz com que

as decisões de paralelismo do problema sejam tomadas com cuidado, com o intuito de que não ocorra falha de arredondamento comprometendo o resultado.

Considerando o fluxograma apresentado pela Figura 3.6, após as simulações em um equipamento equipado com uma CPU de processador Intel core I7-990X *Extreme Edition* [53] (3.46 GHz, 12 M Cache, 8 GB de seis cores com sistema operacional Ubuntu 12.04.3 (Kernel Linux 3.8) o percentual de consumo de tempo requerido pelas partes mais importantes do algoritmo executado de forma serial é apresentado na Tabela 3.1.

É notado que o laço de repetição referente às células de integração apresenta o maior consumo do tempo do algoritmo.

Região	**Percentual de consumo em relação ao tempo de execução**
Configurações iniciais e entrada de dados	3,17%
Laço de Repetição referente às células de integração	55,65%
Determinação das condições de fronteira	7,23%
Determinação do domínio de suporte	6,78%
Cálculo da função de forma	9,03%
Solver	18,14%

Tabela 3.1: Percentual de consumo de tempo dos principais módulos do problema considerando $\Delta t = 1$

Baseado no custo computacional do laço de repetição sobre as células de integração apresentando pela Tabela 3.1, foi modelado um sub fluxo apresentado na Figura 3.7.

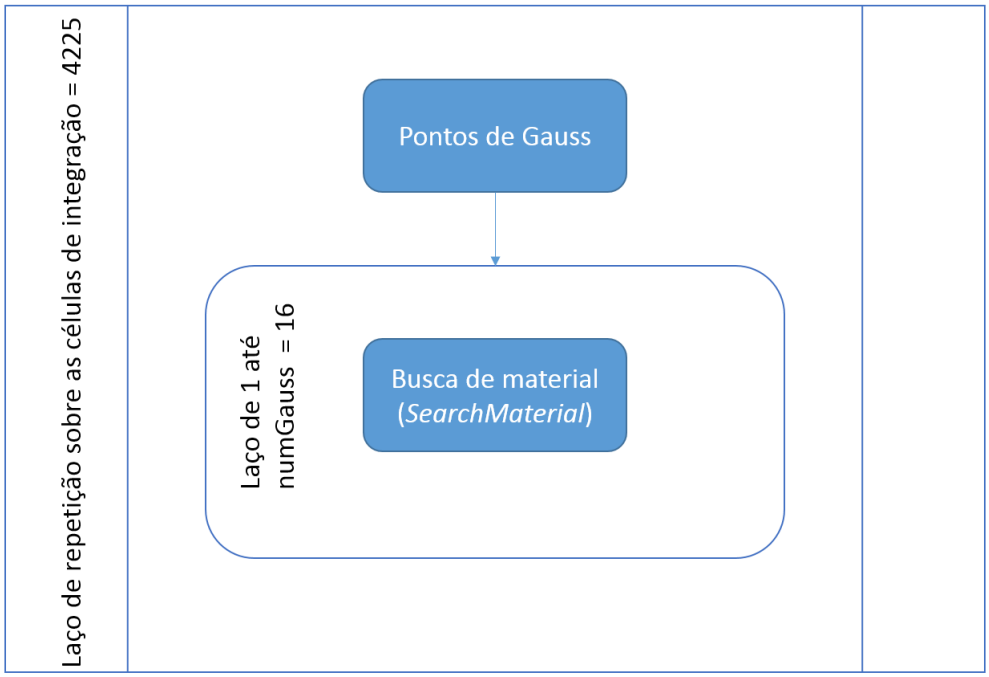

Figura 3.7: Sub fluxo referente a um dos trechos candidatos à paralelização

Este trecho do código tem o objetivo de realizar um laço de repetição para cada célula da malha de integração do *Meshless Method*, e para cada célula calcular os pontos de Gauss e invocar a operação para busca do tipo de material (*searchMaterial*).

O *serachMaterial* é uma rotina desenvolvida em Fortran F90 com o objetivo de localizar o tipo de material referente ao ponto de integração corrente. Como entrada de dados, esta rotina precisa de receber os pontos de Gauss e devolve o tipo de material. Internamente, o processo para localizar o material é feito primeiramente identificando se o ponto está dentro de uma micro região com material definido. Caso encontrado, o método de busca em largura é disparado para identificar o tipo de material através de uma estrutura de dados do tipo *kd-tree* utilizada para armazenamento das informações durante a execução do programa.

Para esta estimativa, foi desconsiderado o tempo gasto pelo laço principal de tempo, uma vez que ele é variável de acordo com o tempo. Além do laço sobre as células de integração, nota-se que a etapa do *Solver* também possui uma grande representatividade do percentual de tempo de execução do algoritmo, tornando-se um candidato ao processo de paralelismo.

O foco na resolução desta etapa é através do uso de dois solucionadores paralelos:

- PARDISO
  - Solver paralelo com foco em memória compartilhada

- MUMPS
    - Solver paralelo com foco em troca de mensagens

3.7. Solver PARDISO

PARDISO é um conjunto de bibliotecas *thread-safe*, de alto desempenho, robusto, de manipulação eficiente de memória focado em resolução de sistemas de equações lineares tanto simétricos quanto não simétricos esparsos. A resolução destes sistemas acontece via memória compartilhada e memória distribuída entre multiprocessadores.

Conforme descrito pelo website o qual mantém o projeto, o *Solver* possui sua utilização em diversas universidades e laboratórios científicos internacionais desde sua origem em 2004 [54].

O PARDISO pode ser usado com: matrizes não simétricas, simétricas, com precisões complexas ou reais, positiva definida ou indefinida [12]. Esta biblioteca utiliza rotinas BLAS nível 3, aumentando o rendimento. PARDISO calcula a solução de um sistema de equações lineares com múltiplos lados direitos de problemas do tipo,

$$Ax = b \tag{3.35}$$

e utiliza a fatoração paralela do tipo LU, LDU e LL^T. A Figura 3.8 apresenta os possíveis tipos de matrizes que podem ser solucionadas com o *Solver* PARDISO, sendo elas: a) simétrica real indefinida; b) simétrica real positiva definida; c) simétrica hermitiana indefinida; d) simétrica hermitiana positiva indefinida; e) simétrica complexa; f) não simétrica real; g) não simétrica complexa.

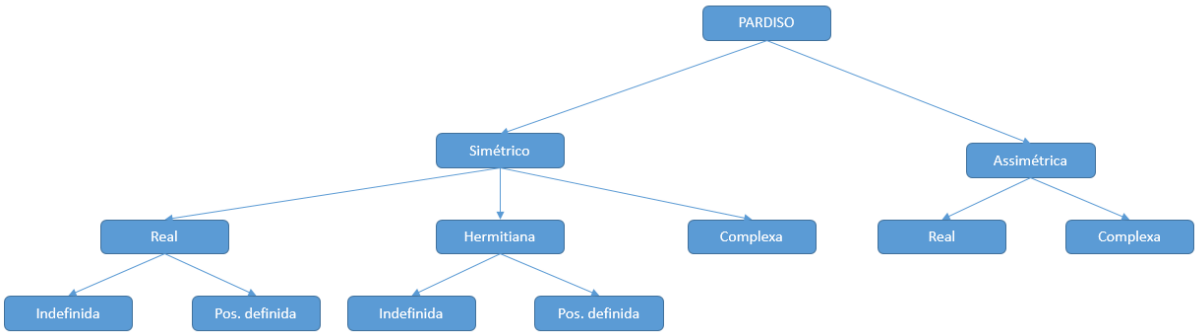

Figura 3.8: Modelos de matrizes esparsas que pode ser solucionadas com o *solver PARDISO* [54].

3.8. Solver MUMPS

MUMPS é um pacote de bibliotecas com objetivo de resolução de sistemas de equações lineares do tipo,

$$Ax = b \tag{3.36}$$

tal que A representa uma matriz quadrada esparsa podendo ser, não simétrica, simétrica positiva definida, ou somente simétrica. MUMPS compõe uma implementação de métodos diretos baseados em abordagens multifrontal [55] seguindo o conceito de fatoração do tipo:

$$A = LU \tag{3.37}$$

tal que L representa uma matriz triangular inferior e U uma matriz triangular superior. Caso a matriz seja simétrica, a fatoração será do tipo

$$A = LDL^T \tag{3.38}$$

em que D é um bloco da matriz diagonal com blocos de fim 1 ou 2.

O pacote MUMPS oferece recursos como entrada de matrizes montadas (distribuída ou centralizada), ou no formato elementar, análise de erros, refinamento iterativo, e o principal, análise paralela [14]. Originalmente, o MUMPS foi escrito em Fortran F90 e possui uma interface em C. Seu funcionamento requer o MPI [6] [27] com o objetivo da passagem de mensagens, além das bibliotecas *BLAS* [3], BLACS e SCALAPACK [56]. Todavia, caso seja necessário apenas a execução serial, basta a biblioteca BLAS [14].

A execução do pacote MUMPS consiste em distribuir as tarefas entre os processadores, todavia, um destes deve ser identificado como *host* responsável por executar a maior parte da fase analítica, objetivando a distribuição da matriz de entrada para os demais processadores. O sistema $Ax = b$ é resolvido em 3 fases:

- Análise
  - A fase de análise está disponível tanto para o formato serial quanto para o formato paralelo. Esta etapa inclui uma ordenação baseada em padrões de simetria entre $A + A^T$ e uma fatoração simbólica.
- Fatoração
  - Durante esta fase, uma fatoração direta é realizada no formato

    $$A_{pre} = LU \tag{3.39}$$

    $$A_{pre} = LDL^T \tag{3.40}$$

    Em seguida, a matriz original é distribuída para os processadores de acordo com o mapeamento gráfico da dependência da fatoração, chamada árvore de eliminação [57]. A árvore de eliminação expressa

a independência entre as tarefas permitindo várias frentes serem processadas simultaneamente. Essa abordagem é chamada abordagem multifrontal. Durante a fase de fatoração, as matrizes são mantidas em memória ou em disco, e é utilizada na fase de solução.

- Solução
  - A solução de

    $$LUX_{pre} = b_{pre} \tag{3.41}$$

    ou

    $$LDL^T X_{pre} = b_{pre} \tag{3.42}$$

    tal que X_{pre} e b_{pre} são respectivamente a transformada da solução X e o *right-hand side* ou lado direito b associado ao pré-processamento da matriz A_{pre} obtida através da pós eliminação:

    $$Ly = b_{pre} \tag{3.43}$$

    ou

    $$LDy = b_{pre} \tag{3.44}$$

    Seguido pela passo da pré-eliminação:

    $$Ux_{pre} = y \tag{3.45}$$

    ou

    $$L^T x_{pre} = y \tag{3.46}$$
  - O lado b é inicialmente pré-processado, e em seguida transmitida para os demais processadores. Lados direitos esparsos podem ser utilizados como limitadores de volume de dados durante este passo.
  - A pós-eliminação (3.43) e (3.44) e pré-eliminação (3.45) e (3.46) são utilizados para se obter a solução de x_{pre}. Finalmente a solução de x_{pre} é processada obtendo assim a solução X do sistema original (3.37).

Cada uma das três fases pode ser utilizada separadamente por várias instâncias do MUMPS simultaneamente [14].

3.9. Considerações Finais

Este capítulo apresentou o histórico e a formulação básica de *Métodos sem Malha*. Foram definidos conceitos fundamentais para a execução dos métodos como as funções de forma, domínios de influência e de suporte. Foi apresentado também suas equações. Foi apresentado o método EFGM utilizado neste trabalho para resolução do problema proposto da máquina de indução trifásica. Por fim, foi apresentado o fluxo de execução do algoritmo computacional para resolução do problema e os principais pontos de custo computacional, evidenciando principais candidatos a paralelização. Esta que realizada através das ferramentas OpenMP, MPI, MUMPS e PARDISO.

Capítulo 4
Análise de resultados

Neste capítulo é apresentada uma sequência de resultados obtidos através da simulação da uma máquina de indução trifásica relacionada na Seção 3.6 por meio de Métodos sem Malha e implementada por algoritmos paralelos. Os resultados são distribuídos em três etapas: A primeira envolve o paralelismo do laço de repetição sobre as células de integração do EFGM via memória compartilhada OpenMP e em seguida por troca de mensagens utilizando o MPI. A segunda, apresenta simulações do paralelismo do *Solver*, através das bibliotecas PARDISO e MUMPS. Por fim, a terceira etapa consiste em uma combinação entre as duas primeiras abordagens, resultando no paralelismo híbrido.

4.1. Paralelização do Laço de Repetição

Após a identificação do custo computacional requerido pelo laço de repetição sobre as células de integração do EFGM apresentado pela Tabela 3.1, iniciou-se o trabalho de paralelismo através do OpenMP. A abordagem adotada foi a distribuição do laço entre processos conforme apresentado pelas Figuras 4.1 e 4.2.

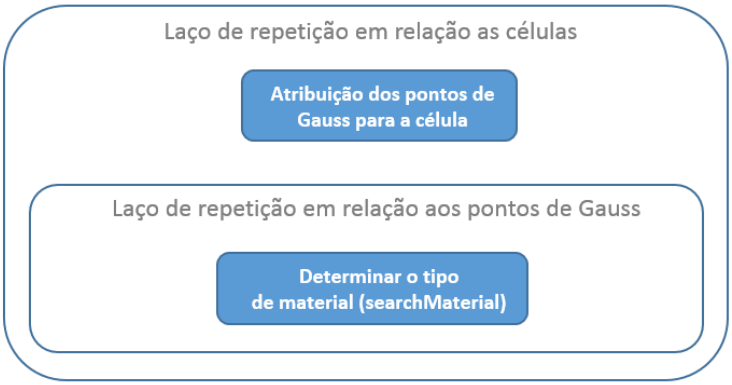

Figura 4.1: Paralelismo realizado sobre as células de integração.

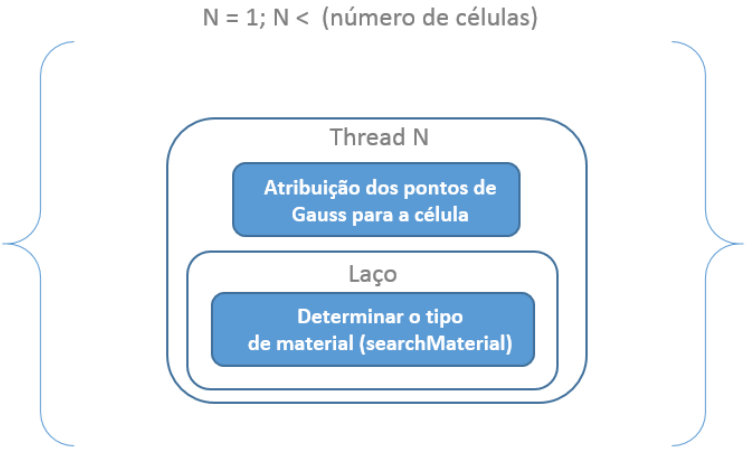

Figura 4.2: Representação da distribuição no laço de repetição par para o OpenMP.

Para cada iteração sobre as células de integração, um processo é utilizado para realizar as operações de atribuição dos pontos de Gauss e determinação do tipo de material utilizando a função *searchMaterial* (função apresentada na Seção 3.8). Por padrão, é informado em cada simulação a quantidade de *threads* utilizadas.

A estratégia de paralelização do laço de repetição sobre as células de integração obteve uma redução do percentual de distribuição de consumo de 55,65% para 4,37% conforme apresentado pelas Tabelas 3.1 (programa executado de maneira serial), e Tabela 4.1 (programa executado com a paralelização).

Região	**Percentual de consumo em relação ao tempo de execução**
Configurações iniciais e entrada de dados	2,14%
Laço de Repetição referente às células de integração	4,37%
Determinação das condições de fronteira	2,23%
Determinação do domínio de suporte	4,22%

Cálculo da função de forma	10,03%
Solver	77,01%

Tabela 4.1: Percentual de representatividade de consumo de tempo no algoritmo paralelo apenas no laço de repetição sobre as células de integração

Todavia, esta abordagem elevou consideravelmente o percentual de consumo em relação ao tempo de execução da etapa de *Solver* de 18,14% para 77,01% conforme descrito também pela Tabela 4.1. Considerando que a etapa referente ao *Solver* está dentro de um laço de repetição em relação ao tempo, esta elevação de distribuição do tempo é muito custosa computacionalmente, uma vez que o aumento é relativo em relação ao número de repetições.

4.2. Paralelização do Solver

Com o objetivo de diminuir a distribuição do percentual de participação entre as etapas mais importantes do problema, atuou-se na paralelização do *Solver.*

O processo de paralelização do *Solver* foi distribuído em duas etapas:

- Paralelização com PARDISO via memória compartilhada (OpenMP).
- Paralelização com MUMPS via troca de mensagens (MPI).

A primeira solução abordada, foi a junção da paralelização do laço sobre as células de integração feita com o OpenMP e a paralelização do *Solver* através da biblioteca PARDISO. Com esta configuração, foi obtido uma redução de 77,01% para 23,40%, resultados que podem ser analisados através da Tabela 4.2.

Região	Percentual de consumo em relação ao tempo de execução
Configurações iniciais e entrada de dados	15,13%
Laço de Repetição referente às células de integração	17,45%
Determinação das condições de fronteira	12,77%
Determinação do domínio de suporte	14,13%
Cálculo da função de forma	17,12%
Solver	23,40%

Tabela 4.2: Representatividade do percentual de tempo após paralelização do laço de repetição sobre as células de integração via OpenMP e o *Solver* PARDISO

É notado também na Tabela 4.2 uma equalização de participação em cada etapa durante a execução do algoritmo. Esta equalização reflete em uma elevação das outras etapas que antes não eram tão representativas. Este resultado era esperado uma vez que a equalização tende a manter o percentual de consumo próximo entre todas as regiões.

Após a análise da redução do percentual de tempo, foram realizados novos testes de execução do algoritmo, desta vez variando o número de *threads* até o máximo de cores do processador [53]. A Figura 4.3 apresenta o abordagem de paralelização do laço de repetição e do *Solver* PARDISO variando de 1 *thread* (formato serial) até 6 *threads* (formato paralelo), ou seja, uma *thread* para cada *core*.

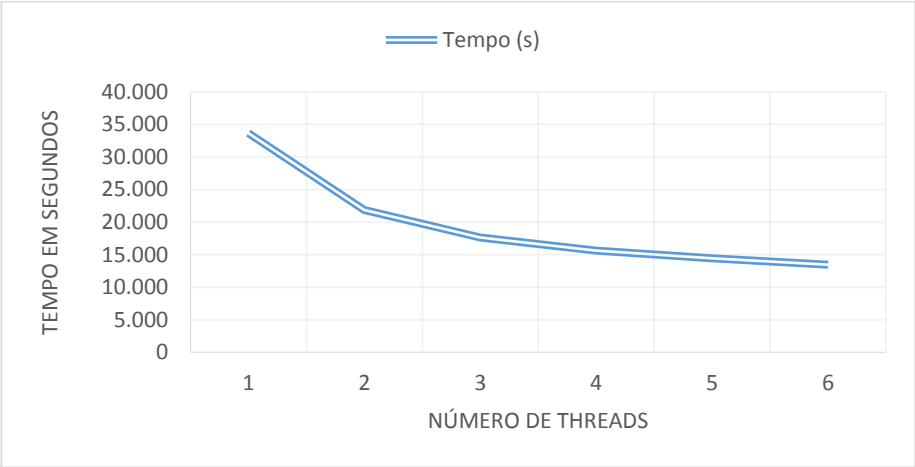

Figura 4.3:Processo de paralelização do laço de repetição sobre as células de integração e do Solver via PARDISO com variação de 1 à 6 *threads*

Nota-se o ganho significativo de tempo durante a execução do PARDISO a partir da utilização de duas *threads,* ou seja, o *Solver* deixa de ser executado de forma serial e passa a ser executado paralelamente. Em seguida, nota-se que a linha de inclinação do tempo torna-se mais suave, mas ainda assim com redução do tempo de execução. A próxima etapa destes experimento conta com a tentativa de execução com a variação do número de *threads* entre 7 e 12 alcançando a capacidade máxima disponível segundo fabricante [53]. O resultado deste experimento é apresentado pela Figura 4.4. Nota-se que ainda existe ganho no tempo de execução do algoritmo com a utilização de até 11 *threads* atingindo o melhor resultado de tempo de execução com 12,440 segundos. A partir de 12 *threads* o tempo de execução tende a saturar por dificuldades de gerenciamento de memória do OpenMP durante a realização de cálculos em regiões paralelas. Com o objetivo de demonstrar nitidamente esta saturação, foi simulado uma variação de 15 à 50 *threads* apresentada na Figura 4.5.

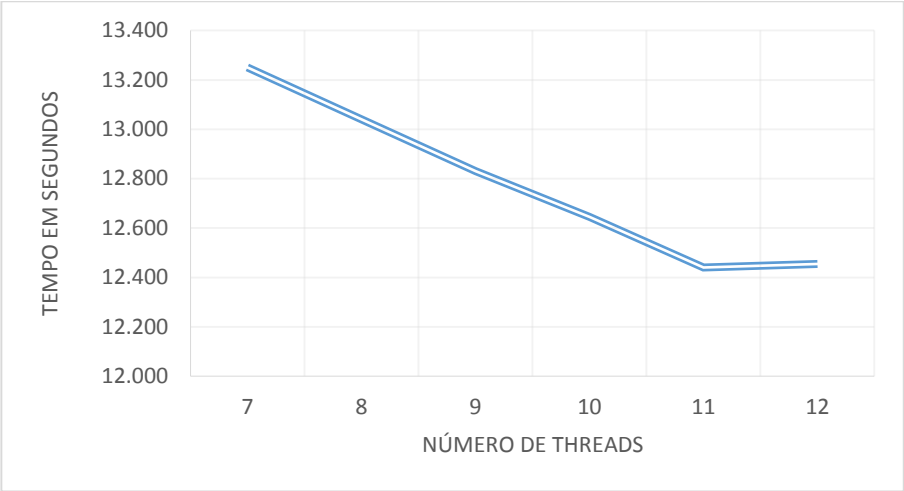

Figura 4.4: Processo de paralelização do laço de repetição sobre as células de integração e do *Solver* via PARDISO com variação de 7 à 12 *threads*

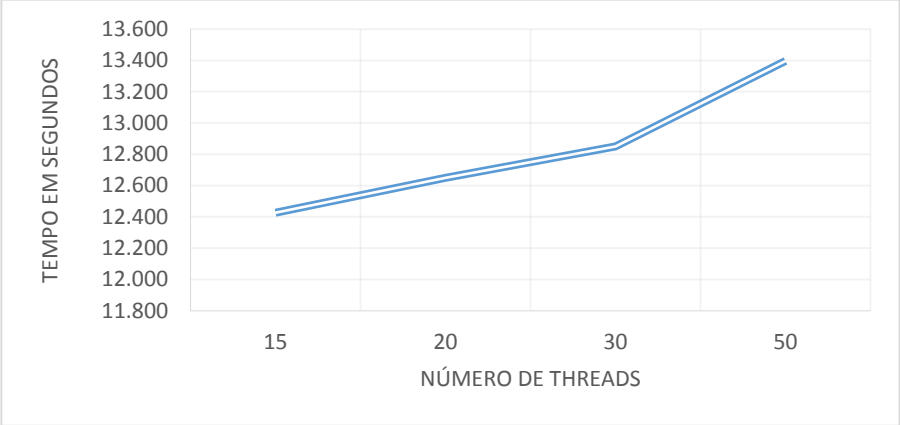

Figura 4.5: Processo de paralelização do laço de repetição sobre as células de integração e do *Solver* via PARDISO com variação de 15 à 50 *threads*

A próxima análise realizada foi a paralelização do laço de repetição sobre as células de integração em conjunto com o *Solver* MUMPS. Esta abordagem possibilita a integração entre o OpenMP (paralelização no laço de repetição) e o MPI (paralelização no MUMPS). O processo de paralelização do MUMPS faz com que as etapas de fatoração e de solução do sistema linear sejam subdivididas e executadas em outros processos MPI. Esta abordagem favoreceu a distribuição do percentual de execução dos tempos, tornando-os mais uniformes conforme apresentado pela Tabela 4.3. Foi adotado o número de processos MPI igual ao número de *cores* do processador utilizado para testes, ou seja, 6 processos [53]. Considerando as configurações para o MUMPS, foram realizados testes com variação de 1 à 6 *threads* OpenMP e 6 processos MPI conforme apresentado pela Figura 4.6.

Região	Percentual de consumo em relação ao tempo de execução
Configurações iniciais e entrada de dados	16,02%
Laço de Repetição referente às células de integração	18,01%
Determinação das condições de fronteira	14,02%
Determinação do domínio de suporte	14,76%
Cálculo da função de forma	16,99%
Solver	20,20%

Tabela 4.3: Representatividade do percentual de tempo após paralelização do laço de repetição sobre as células de integração via OpenMP e o *Solver* MUMPS

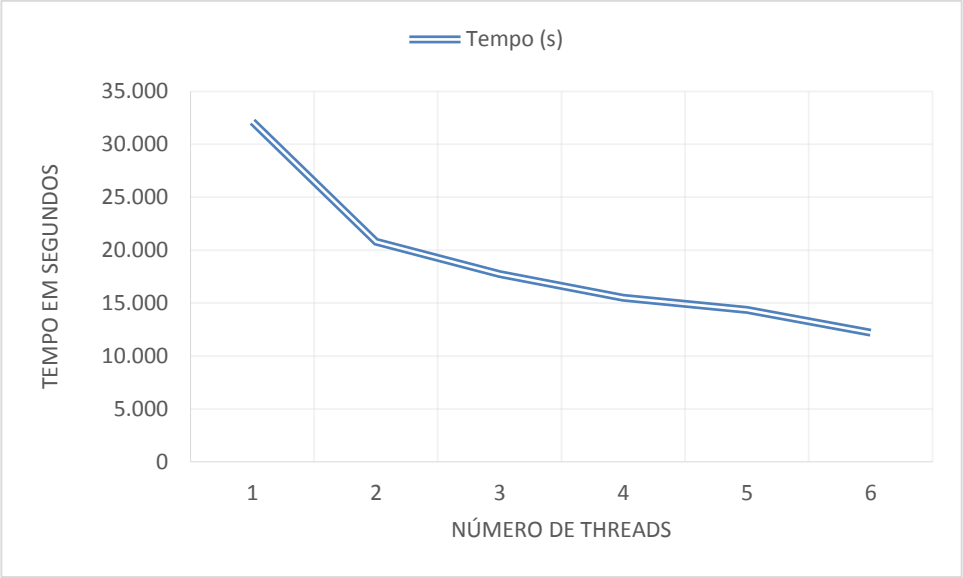

Figura 4.6: Processo de paralelização do laço de repetição sobre as células de integração e do *Solver* via MUMPS com 6 processos e com variação de 1 à 6 *threads* OpenMP

Pode-se perceber que o ganho de tempo de execução entre o modelo serial e o paralelo híbrido (OpenMP e MPI) é bastante significativo. Esta abordagem mostrou-se mais eficiente também do que o modelo paralelo através do *Solver* PARDISO. Para esta configuração, foi testada também a tentativa de saturação através da elevação de *threads* OpenMP. Os resultados são apresentados pelas Figuras 4.7 com variação entre 7 à 12 *threads* e Figura 4.8 com variação entre 15 e 50 *threads.* Nota-se que o experimento apresentado pela Figura 4.8 tende a partir de aproximadamente 30 *threads* se tornar mais custoso computacionalmente do que sua versão original, ou seja, sua versão serial.

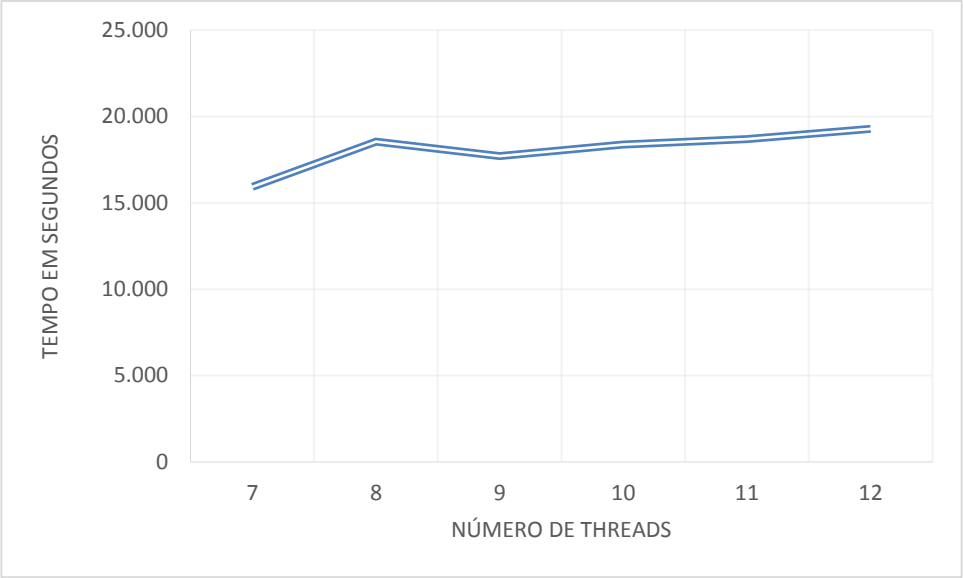

Figura 4.7: Processo de paralelização do laço de repetição sobre as células de integração e do *Solver* via MUMPS com variação de 7 à 12 *threads*

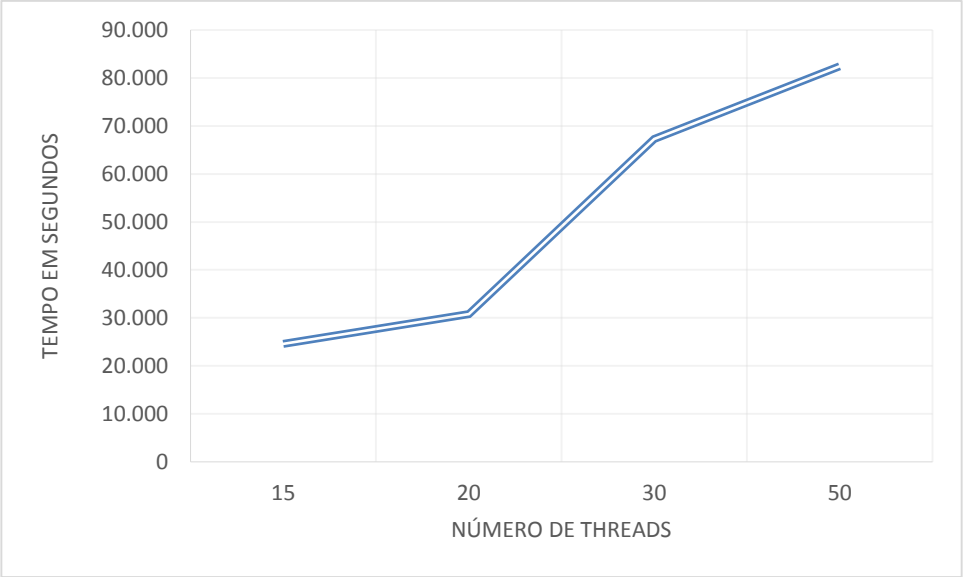

Figura 4.8: Processo de paralelização do laço de repetição sobre as células de integração e do *Solver* via PARDISO com variação de 15 à 50 *threads*

O melhor resultado obtido foi de 12.221 segundos com a utilização de 6 *threads* e 6 processos MPI, tornando-se até o momento melhor resultado em relação ao experimento serial.

O último experimento, foi aplicado somente com troca de mensagens, tanto no laço de repetição, quanto no *Solver,* neste caso o MUMPS. O resultado foi o pior entre as distribuições de percentuais paralelas conforme apresentado pela Tabela 4.4.

Região	Percentual de consumo em relação ao tempo de execução
Configurações iniciais e entrada de dados	14,34%
Laço de Repetição referente às células de integração	25,65%
Determinação das condições de fronteira	11,17%
Determinação do domínio de suporte	13,23%
Cálculo da função de forma	16,59%
Solver	20,54%

Tabela 4.4: Representatividade do percentual de tempo após paralelização do laço de repetição sobre as células de integração via MPI e o *Solver* MUMPS

Para este tipo de solução foi observado que o processo de distribuição das informações pelo laço de repetição através do MPI fez com que o tempo fosse maior do que o tempo requerido pelo *Solver*. Esta abordagem fez com que a etapa do laço de repetição fique maior do que a etapa do *Solver*. Além da distribuição do percentual de participação em relação ao tempo de execução não ser a esperada neste experimento, o melhor tempo para esta abordagem é de 14.231 segundos, atuando com 6 processos MPI. Com o modelo puramente MPI não atual com threads OpenMP, foi realizada uma tentativa de saturação do MPI, elevando o número de processos até 50, alcançando o resultado de 85.345 segundos.

Por fim, a tabela 4.5 e a Figura 4.9 apresentam os melhores resultados de acordo com cada abordagem apresentada neste capítulo.

Abordagem	Melhor tempo de execução em segundos	Quantidade de *Threads* utilizada pera o melhor tempo	Speedup
Laço Serial e Solver com *PARDISO*	68.013	1	0
Laço com *OpenMP* e *Solver* com *PARDISO*	12.440	11	5,467
Laço com *OpenMP* e *Solver* com *MUMPS*	12.221	6	5,565
Laço com *MPI* e Solver com *MUMPS*	14.231	*6	4,779

Tabela 4.5: Comparativo ente os melhores tempos por abordagem. *processos MPI.

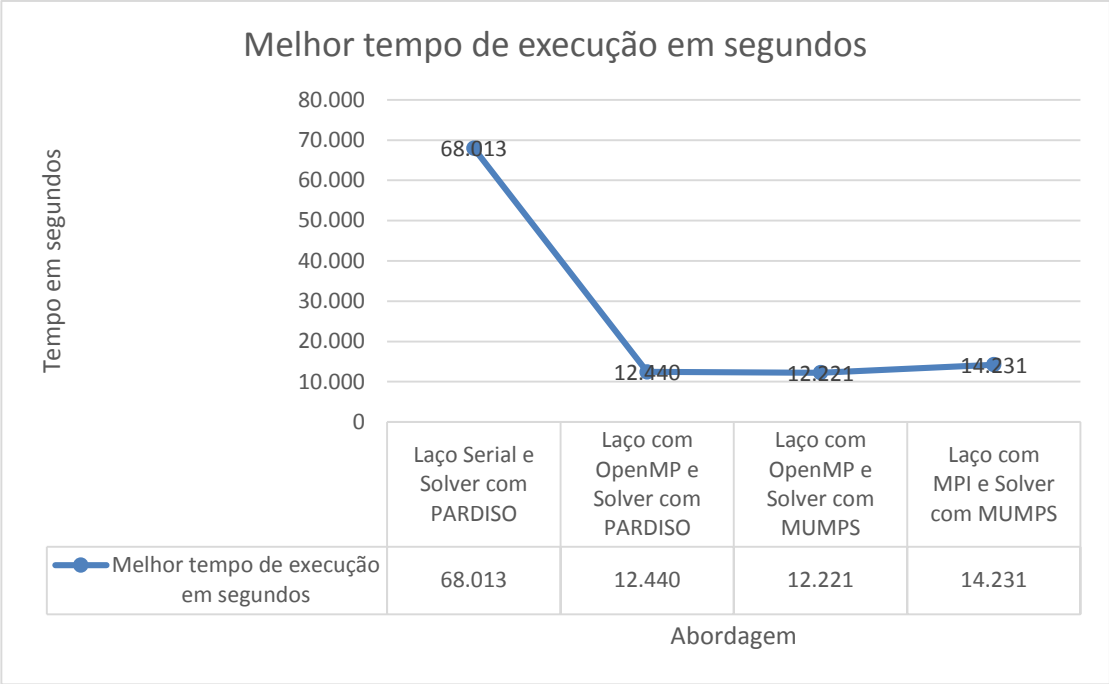

Figura 4.9: Gráfico comparativo entre os melhores tempos de acordo com a abordagem

4.3. Considerações Finais

Neste capítulo foram apresentados os experimentos segundo as abordagens propostas: a) paralelização somente do laço de repetição com OpenMP; b) paralelização do laço com OpenMP e do *Solver* através do PARDISO; c) paralelização do laço com OpenMP e do *Solver* através do MUMPS; d) paralelização do laço de repetição com MPI e do *Solver* com MUMPS.

De posse dos resultados obtidos através destas abordagens propostas, pode-se notar que o melhor resultado para o problema da máquina de indução trifásica modelada com *Métodos sem Malha* é o modelo híbrido. Por fim, o capítulo apresenta gráficos e tabelas para facilitar o do resultado obtido.

Capítulo 5

Conclusão

Neste trabalho procurou-se identificar os principais trechos com maior consumo de processamento existentes em um algoritmo que representa um modelo de um motor de indução utilizando o método sem malha EFGM. Durante o processo de identificação, foram utilizadas ferramentas para mensuração destas informações como GProf [58], com o intuito de realizar um diagnóstico de consumo do tempo de código serial e posteriormente implementar técnicas de paralelismo em regiões que consumiam mais tempo. O levantamento destas informações, apresentou duas áreas candidatas ao processo de paralelização sendo o laço de repetição referente às células de integração para o *Meshless* e a solução do sistema linear.

Pôde-se notar também que, o paralelismo de um segmento pode afetar diretamente outra etapa, ou seja, após o paralelismo do laço de repetição em função das células de integração, o processo do *Solver* tornou-se o mais crítico. Esta etapa foi também paralelizado em seguida através do *Solver* PARDISO via OpenMP e em seguida através do MUMPS via MPI.

Observou-se também que, a limitação de *threads* deve ser observada de acordo com cada hardware uma vez que, a elevação de *threads* bem acima do disponível por cada core do processador na máquina pode, por sua vez, aumentar o tempo de execução, quebrando o objetivo principal do paralelismo. Todavia, a utilização correta da quantidade de *threads* disponíveis tende a gerar bons resultados.

Após toda a execução dos processos em paralelo, notou-se um speedup de 5,56 de *speedup* através da formação híbrida entre o *OpenMP* atuando diretamente no laço de repetição, e o *Solver MUMPS* resolvendo o sistema linear final.

Este trabalho auxiliou também a observar que a elevação do número de *threads* não caracteriza diretamente uma diminuição do tempo de execução, uma vez que a execução de tarefas é monitorada tanto pelo *OpenMP* quanto pelo *MPI*, e uma elevação

descontrolada pode acarretar a um número maio de *threads* avaliando tarefas do que executando propriamente dita.

Por fim, notou-se que o ganho de uma paralelização está intrinsecamente relacionado ao design e técnicas adotadas, todavia, a aplicação de forma errônea pode acarretar em um tempo de execução muito maior que sua forma serial.

5.1. Trabalhos Futuros

Como este trabalho, foi possível analisar o ganho computacional referente a arquiteturas paralelas comparadas a arquiteturas seriais. Todavia, o modelo híbrido foi aplicado diretamente a etapas mais custosas computacionalmente, sendo a integração sobre as células de integração do *Meshless* e sobre o *Solver*.

Pretende-se em trabalhos futuros a paralelização híbrida também para a montagem do sistema a ser solucionado conforme apresentado em [31]. Outro ponto a ser abordado em novos trabalhos é o estudo de outros algoritmos de busca, resolvendo a busca de material.

Além deste modelo, pretende-se também trabalhar com a programação a nível de aceleração GPU com a linguagem *CUDA*.

Referências Bibliográficas

[1] BYRDE, O., SAWLEY, M.L. Parallel computation and analysis of the flow in a static mixer. Computers & Fluids 28, 1999.

[2] ZHENG, Z., CHEN, X., WANG, Z., SHEN, L. & LI, J. Performance Model for OpenMP Parallelized Loops. 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE), 16-18 de dezembro de 2011.

[3] DONGARRA, J. J., DU CROZ, J., DUFF, I. S., & HAMMARLING, S. Algorithm 679. A set of Level 3 Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software, 16:1–17, 1990.

[4] MARTINO, R. L., JOHNSON, C. A., SUH, E. B., TRUS, B. L. & YAP, T. K. Parallel Computing in Biomedical Research. Science, ;265(5174):902-8, 1994

[5] TOP500 Supercomputers Site. Disponível em <http://top500.org>. Acesso em 03 de abril de 2013.

[6] MESSAGE-PASSING Interface Forum. Disponível em: <http://www.mpi-forum.org/docs/mpi-1.1/mpi1-report.pdf>. Acesso em: 10 de junho de 2013.

[7] CHAPMAN, B., JOST, G., & PAS, R. Using OpenMP - Portable Shared Memory Parallel Programming. Massachussetts: Massachussetts Institute of Technology, 2008.

[8] THE OPENMP Group. Disponível em: <http://openmp.org>. Acesso em 7 de abril de 2013.

[9] PETROVSKII, J., G. Equações Diferenciais Parciaiss, W. B. Saunders Co. Philadelphia 1967.

[10] GU, G. R. An Introduction to Meshfree Methods and their. Em G. R. Gu, An Introduction to Meshfree Methods and their. Netherlands: Springer, 2005.

[11] BELYTSCHKO, T., LU, Y., & GU, L. Element-free Galerkin Methods. International Journal for Numerical Methods in Engineering, v. 37, p. 229–256. Janeiro 1994.

[12] SCHENK, O., GÄRTNER, K., FICHTNER, W. & STRICKER, A. PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation. Future Generation Comp. Syst. 18(1): 69-78, 2001.

[13] AMESTROY, P., L'EXCELLENT, J.-Y., ROUET, F.-H., SID-LAKHDAR, W., Modeling 1D distributed-memory dense kernels for an asynchronous multifrontal sparse solver, High-Performance Computing for Computational Science, VECPAR 30 de Junho de 2014, Eugene, Oregon, USA.

[14] MUMPS, A Multifrontal Massively Parallel Sparse Direct Solver. Disponível em: <http://mumps.enseeiht.fr/ > . Acesso em 23 de setembro de 2013.

[15] BRESHERS, C. The Art of Concurrency: A Thread Monkeys Guide to Writing Parallel Applications, O'Reilly, Sebastopol, CA, 2009.

[16] PACHECO, P. S. An Introduction to Parallel programming. San Francisco: Elsevier, 2011.

[17] FLYNN, M. J. Some Computer Organizations and Their Effectiveness. Computers, IEEE Transactions on, 948 – 960, setembro, 1972

[18] EL-REWINI, H., ABD-EL-BARR, M. Advanced Computer Architecture And Parallel Processing. New Jersey: John Wiley & Sons, 2005.

[19] MAROWKA, A. Parallel Computing on Any Desktop. Communications of the ACM. 75-78. Volume 50 I. 9, Setembro 2007. New York, NY, USA.

[20] THE OPEN MPI Development Team. Disponível em <http://icl.cs.utk.edu/open-mpi/about/members>. Acesso em 15 de outubro de 2013.

[21] HERLIHY, M., SHAVIT, N. The Art of Multiprocessor Programming. Boston: Morgan Kaufmann, 2012.

[22] FUHRER, R., LIN, B., & NOWICK, S. Algorithms for the optimal state assignment of asynchronous state machines. Advanced Research in VLSI, março 1995.

[23] FOSTER, J. Designing And Building Parallel Programs: Concepts And Tools For Parallel Software Engineering. Addison-Wesley; 1 edition, fevereiro, 1995.

[24] TUNCER, I.H., GÜLCAT, Ü., EMERSON, D.R. & MATSUNO, K. Parallel Computational Fluid Dynamics 2007 Implementations and Experiences on Large Scale and Grid Computing. Berlin, Springer, 2009.

[25] SEVERANCE, C.; DOWD, K. High Performance Computing, Second Edition, O'Reilly, julho de 1998.

[26] LIVERMORE Computing Center. Disponível em: <https://computing.llnl.gov/tutorials/mpi>. Acesso em 25 de outubro de 2013.

[27] SNIR, M., OTTO, S. W., HUSS-LEDERMAN, S., WALKER, D. & DONGARRA, J. MPI: The Complete Reference. The MIT Press, Cambridge, Massachusetts, 1996.

[28] DAGUM, L., MENON, R. OpenMP: Industrystandard API for shared-memory programming. In IEEE Computacional Science & Engineering, 1998.

[29] JONES, M. D., YAO, R. Parallel programming for OSEM reconstruction with MPI, OpenMP, and hybrid MPI-OpenMP. IEEE Nuclear Science symposium and Medical Imaging Conference ,Roma, Italia, 2004

[30] RÜBENKÖNIG, O. The Método das Diferenças Finitas (FDM) – An introduction. Albert Ludwigs University of Freiburg, 2006.

[31] COPPOLI, E. H. R., RESENDE, U. C., Afonso, M. M., MESQUITA, R. C., GERVASIO, J. P., ARAUJO, B. M. Hybrid Parallel Meshless Algorithm for Electromagnetic Applications. In: COMPUMAG Conference on the Computation of Electromagnetic Fields, 2013, Budapeste.

[32] LEVEQUE, R. J. Método dos Volumes Finitoss for hyperbolic problems. Cambridge New York: Cambridge University Press, 2002. Print.

[33] GINGOLD, R. A., MONAGHAN, J., J. Smoothed particle hydrodynamics: theory and application to non-spherical stars. Mon. Not. R. Astron. Soc., Vol 181, pp. 375–89, 1977.

[34] NAYROLES, B., TOUZOT, G., VILLON, P. Generalizing the Método dos Elementos Finitos: Diffuse approximation and diffuse elements. Computational Mechanics. 1992, Volume 10, Issue 5, pp 307-318.

[35] LISZKA, T., J. DUARTE, C. A. M., TWORZYDLO, W. W. hp-Meshless cloud method. Computer Methods in Applied Mechanics and Engineering. Volume 139, Issues 1–4, 15 December 1996, Pages 263–288.

[36] LIU, W. K., JUN, S., ZHANG, Y. F. Reproducing kernel particle methods. International Journal for Numerical Methods in Fluids. Volume 20, Issue 8-9, pages 1081–1106, 30 April - 15 May 1995.

[37] ATLURI, S., SHEN, S. The meshless local Petrov-Galerkin method: A simple & less-costly alternative to the finite-element and boundary element methods. CMES, vol. 3, no. 1, pp. 11–51, 2002.

[38] LIU, G. R., GU, Y. T. A point interpolation method for two-dimensional solid. 2001, Int. J. Num. Methods. Eng., 50, 937-951.

[39] GU, Y. T., LIU, G. R. A local point interpolation method for static and dynamic analysis of thin beams. 2001, Comput Methods Appl Mech Eng 190:5515–5528.

[40] BABUŠKA, I., MELENK, J. M. The partition of unity Método dos Elementos Finitos: Basic theory and applications. Comp. Meth. in Appl. Mech. and Eng. Vol. 4, 1996, p. 289-314.

[41] MARÈCHAL, Y., COULOMB, J. L., MEUNIER, G. & TOUZOT, G. Use of difuse element. In Digest of Fifth Biennial IEEE Conference, 1992

[42] COPPOLI, E. H. Modelagem de Dispositivos Eletromagnéticos Através de Métodos sem Malha. Modelagem de Dispositivos Eletromagnéticos Através de Métodos sem Malha. Petrópolis, RJ: Tese de doutorado submetida ao LNCC, 2010.

[43] LIU, G. R. Mesh Free Methods - Moving Beyond the Finite Element. Em G. R. Liu, Mesh Free Methods - Moving Beyond the Finite Element. Washington, D. C.: CRC Press, 2003.

[44] VIANA, S. A. .Métodos sem Malha Applied to Computational Electromagnetics. University of Bath, United Kingdom. Tese de doutorado submetido ao Department of Electronic & Electrical Engineering, 2006.

[45] LANCASTER, P. SALKAUSKAS, P. Surfaces generated by moving least square methods. Mathematics of Computation - STOR, 37(155):141–158, 1981

[46] XUAN, L., SHANKER, B., ZENG, Z. & UDPA, L. Element-free galerkin method in pulsededdy currents. International Journal of Applied Electromagnetics and Mechanics, 19(1-4):463–466, 2004.

[47] XUAN, L., ZENG Z. & UDPA, L. Element free galerkin method for static and quasi-static electromagnetic field computation. IEEE Transactions on Magnetics, 40 (1):12–20, 2004.

[48] CINGOSKI, V., MIYAMOTO, N. & YAMASHITA, Y.. Element-free galerkin method for electromagnetic field computations. IEEE Transactions on Magnetics, 34 (5):3236–3239, 1998.

[49] BOTTAUSCIO, O., CHIAMPI, M., & MANZIN, A. Eddy current problems in nonlinear media by the element-free galerkin method. Journal of Magnetism and Magnetic Materials, 34(2):e823–e825, 2006.

[50] RESENDE, U. C., COPPOLI, E. H. R., AFONSO, M. M. Analysis of EFG Interloping Moving Least Square Method in a Electrostatic Problem. Compumag 2013 - Conference on the Computation of Electromagnetic Fields – Budapeste, Hungary – Julho de 2013

[51] COPPOLI, E. H. R., MESQUITA, R. C., SILVA, R. S. Induction Machine Modeling with Métodos sem Malha. IEEE Transactions on Magnetics, vol. 48, pp. 847-850, Fevereiro, 2012.

[52] PARREIRA, G. F., SILVIA, E. J., FONSECA, A. R. & MESQUITA, R. C. The element-free galerkin method in three-dimensional electromagnetic problems. IEEE Transactions on Magnetics, 42(2):711–714, 2006.

[53] INTEL Specifications. Disponível em: <http://ark.intel.com/products/52585/Intel-Core-i7-990X-Processor-Extreme-Edition-12M-Cache-3_46-GHz-6_40-GTs-Intel-QPI> . Acesso em 08 de janeiro de 2014.

[54] PARDISO PROJECT. Disponível em <http://www.pardiso-project.org> . Acesso em 15 de fevereiro de 2014.

[55] AMESTOY, P. R. Recent progress in parallel multifrontal solvers for unsymmetric sparse matrices. In Proceedings of the 15th World Congress on Scientific Computation, Modelling and Applied. Mathematics, IMACS 97, Berlin, 1997

[56] BLACKFORD, L. S., CHOI, J., CLEARY, A., D'AZEVEDO, E., DEMMEL, J., DHILLON, I., DONGARRA, J., HAMMARLING, S., HENRY, G., PETITET, A., STANLEY, K., WALKER, D., & WHALEY, R. C. ScaLAPACK Users' Guide. SIAM Press, 1997.

[57] LIU, J. W. H. The role of elimination trees in sparse factorization. SIAM Journal on Matrix Analysis and Applications, 11:134–172, 1990.

[58] THE GNU GPROF. Diposnível em: <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>. Acesso em 05 de junho de 2013.