UFRRJ

INSTITUTO DE CIÊNCIAS EXATAS

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

DISSERTAÇÃO

Estudo do Desempenho de Aplicações da Mecânica dos Sólidos em Computação Paralela

Ronilson Rodrigues Pinho

2014

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

Estudo do Desempenho de Aplicações da Mecânica dos Sólidos em Computação Paralela

RONILSON RODRIGUES PINHO

Sob a Orientação do Professor
Carlos A. Reyna Vera-Tudela

Co-orientação do Professor
Sérgio Manuel Serra da Cruz

Dissertação submetida como requisito parcial para obtenção do grau de **Mestre em Modelagem Matemática e Computacional**, no Curso de Pós-Graduação em Modelagem Matemática e Computacional, Área de Concentração em Inteligência Computacional e Otimização.

Seropédica, RJ
Outubro de 2014

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA COMPUTACIONAL

RONILSON RODRIGUES PINHO

Dissertação submetida como requisito parcial para obtenção do grau de **Mestre em Ciências**, no Curso de Pós-Graduação em Modelagem Matemática e Computacional, Área de concentração Inteligência Computacional e Otimização.

DISSERTAÇÃO APROVADA EM ___/___/____

_____
Carlos A. Reyna Vera-Tudela. Dr. UFRRJ

_____
Robson Mariano da Silva. Dr. UFRRJ

_____
Alexandre Silva de Lima. Dr. CEFET-RJ

AGRADECIMENTOS

Ao Deus eterno que residem todos os tesouros da sabedoria;
Aos meus Pais, meus primeiros mestres na ciência do viver, que compartilharam os meus ideais e os alimentaram, cujo sacrifício e renúncia permitiu chegar até aqui;
A toda minha a família e em especial, a minha esposa Teresa Faria Campos, sempre do meu lado apoiando, nos momentos difíceis, e aos dois filhos, meus tesouros, Leone e Levi, amor incondicional;
Aos meus orientadores, Professor Carlos Andres e o Professor Sergio Serra, que num esforço comum, consciente e honesto, repartiram comigo os seus conhecimentos;
Aos meus colegas do Mestrado que contribuíram de uma forma ou de outra para o meu êxito;
O mérito a essa conquista.

RESUMO

PINHO, Ronilson Rodrigues. Estudo do Desempenho de Aplicações da Mecânica dos Sólidos Aplicado em Computação Paralela.: Seropédica, RJ. 2014, 50p.
Dissertação (Mestrado em Modelagem Matemática e Computacional). Instituto de Ciências Exatas, Departamento de Matemática, Universidade Federal Rural do Rio de Janeiro.

O Método de Elementos de Contorno (MEC) é um método computacional para a solução de sistemas de equações diferenciais, formuladas em forma de integrais. Aplicado na Mecânica dos fluidos, Acústica, Eletromagnéticos, Estudo de fraturas etc. O MEC requer discretização apenas no contorno da geometria do problema, mas não do seu interior como um todo, diminuindo o esforço computacional. Com o intuito em diminuir o esforço computacional, a Computação paralela é uma forma eficiente de processamento de informação com ênfase na exploração de eventos simultâneos na execução de um software. Ele surge principalmente devido às elevadas exigências de desempenho computacional e à dificuldade em aumentar a velocidade de um único núcleo de processamento. Apesar das CPUs multiprocessadas, ou processadores multicore, serem facilmente encontrados atualmente, diversos algoritmos ainda não são adequados para executar em arquiteturas paralelas. O presente estudo objetivou-se com o intuito de prosseguir na pesquisa sobre paralelismo, atuando num programa sequencial, desenvolvido na linguagem Fortran 77 (VERA-TUDELA, 2003), que efetua análises numéricas de problemas específicos tensão e deformação em 2D) da Mecânica dos Sólidos via MEC com representação física da barra engastada e tracionada. A implementação da aplicação, visa explorar o máximo o paralelismo.

Palavras-chave: *Paralelismo, Elemento-Contorno, Thread, multicore*

ABSTRACT

PINHO, Ronilson Rodrigues. *Solid Mechanics Applied to Parallel Computing Performance Study*. Seropédica, RJ. 2014, 50p.
Dissertation (MSc in Mathematical Modeling and Computational). Instituto de Ciências Exatas, Departamento de Matemática, Universidade Federal Rural do Rio de Janeiro (UFRRJ).

The Boundary Element Method (BEM) is a computational method for differential equations solutions, formulated in the form of integral domains. Thus, it is applied in Fluid Mechanics, Acoustics, Electromagnetics and Fractures study. The BEM requires discretization only regarding boundary geometry of the problem, but not inside as a whole, reducing the computational effort. In order to reduce computational effort, parallel computing is an efficient form of information processing emphasizing concurrent events exploitation during software execution. This processing status arises primarily due to high computational performance requirements and difficulty in increasing single processor core speed. Despite central processing units (CPUs), whether multiprocessors or multicore processors, are easily found today, several algorithms are not suitable to run on parallel architectures yet. The present study aimed to develop parallelism research, acting in a sequential program, using Fortran 77 language (VERA-TUDELLA, 2003), making numerical analysis of stress and strain 2D specific problems) of Solids Mechanics with BEM, as well as, its clamped and tensioned bar physical representation. This application implementation is intended to exploit the maximum parallelism.

Key words: *Parallelism, Boundary Element, Thread, multicore*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

LISTA DE ABREVIATURAS

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho analisa o desempenho paralelo de uma implementação de um programa em linguagem Fortran 77 que efetua análises numéricas de problemas específicos da Mecânica dos Sólidos via Método dos Elementos de Contorno.

O processamento paralelo é uma forma eficiente de processamento de informação com ênfase na exploração de eventos simultâneos na execução de um software. Ele surge principalmente devido às elevadas exigências de desempenho computacional e à dificuldade em aumentar a velocidade de um único núcleo de processamento. Apesar das CPUs multiprocessadas, ou processadores *multicore*, serem facilmente encontrados atualmente, diversos algoritmos ainda não são adequados para executar em arquiteturas paralelas. No desenvolvimento de aplicações paralelas, é necessário, também, utilizar paralelismo dentro da tarefa. O estudo da implementação, da aplicação, usando Método dos Elementos de Contorno (MEC), para problema elástico 2D numa barra, testando tensão e deformação, visa explorar o máximo do paralelismo. O algoritmo explora os diversos blocos (refinamentos sucessivos, matrizes, cálculos vetoriais e funções), esses blocos são otimizados em threads que são executados em diversos processadores. Avaliação, será em cima do tempo de execução, O speedup[1] com aumento do número de processadores, aumento da instância do problema para verificar a eficiência de processamento na otimização da aplicação.

1.1 Revisões Bibliográficas

Neste item são abordados três assuntos: Primeiro parágrafo está relacionado ao Método de Elemento do Contorno, o segundo parágrafo está relacionado a computação paralela e o terceiro parágrafo , faz abordagem a trabalhos escritos, unido a computação paralela e o Método de Elemento de Contorno.

A busca de soluções mais precisas aliada à redução de custos favoreceu o desenvolvimento dos métodos numéricos baseados nas equações integrais. Dentre vários, pode-se citar o Método de Elementos de Contorno, que apresenta como vantagens a melhor adaptação para certos tipos de problemas, como os que envolvem regiões com áreas de concentração de tensões e regiões infinitas, a possibilidade de selecionar os pontos internos onde se deseja a solução e o reduzido volume de dados necessários para definir o problema, tendo em vista que o método reduz sua dimensão em uma unidade. Com este método são definidos elementos apenas no contorno do problema em análise, e são utilizadas soluções aproximadas de forma semelhante aos elementos finitos, interpolando-se os valores ao longo dos elementos em função dos valores nodais. Este procedimento resulta em matrizes menores uma vez que trabalha apenas com incógnitas no contorno, mas em contrapartida estas matrizes ficam completamente cheias, enquanto os elementos finitos fornecem uma matriz em banda. O Método dos Elementos de Contorno (MEC), cuja formulação é baseada em equações integrais, surgiu há apenas 30 anos.

[1] Speedup: aumento de velocidade observado quando se executa um determinado processo em p processadores em relação à execução deste processo em 1 processador.

Porém, desde o início do século 20, a partir do trabalho de (FREDHOLM, 1903), as equações integrais são utilizadas para a solução de alguns problemas físicos particulares. Nos anos 60, surge a primeira formulação indireta do Método dos Elementos de Contorno, embora ainda não tendo essa denominação, de autoria de (KUPRADZE, 1965), aplicado a problemas potenciais e elásticos.

Somente a partir de 1967, com a publicação do primeiro artigo sobre a formulação direta do método das equações integrais de contorno, para problemas elásticos bidimensionais, de autoria de (FRANK J. RIZZO 1967), é que os métodos integrais começam a despertar interesse na comunidade científica.

O método passa a ser conhecido como "Método dos Elementos de Contorno", com a publicação do primeiro livro sobre o método pelo professor (CARLOS A. BREBBIA, 1978), onde o autor formula o método a partir do método dos resíduos ponderados, usando uma função ponderada conveniente. Em (BREBBIA, DOMINGUES, 1992), escreveu – "BOUNDARY ELEMENTS IN INTRODUCTORY COURSE", no qual, faz varias abordagens de aplicações mecânicas usando o MEC.

No mundo da Computação, a demanda por alto desempenho e baixo custo energético motivam novas pesquisas no campo da computação. O processamento paralelo surge, então, como um dos processos chaves de tecnologia que possibilita a computação moderna (TASOULIS et al., 2004; YANG et al., 2006). Dentre essas característica surge a *"Era Multicore"*, cuja ideia engloba o princípio de duplicação da quantidade de núcleos de processamento em um único chip a cada geração, é comentada por (BORKAR, 2007). Processadores *Multicore* possuem o potencial para resolver grandes problemas, permitindo o uso de diferentes núcleos para o processamento de diferentes porções de dados e, consequentemente, reduzir o tempo de resposta.

Diversos trabalhos,nas áreas de Modelagem Matemática e Computação têm sido apresentados como: (GEORGESYMM , 1984) escreveu programa de MEC usando a linguagem Fortran num processador de matriz distribuída; (MIERS, 2003), escreveu trabalho de sobre a implementação paralela de códigos do MEC utilizando métodos iterativos na solução do sistema de equações;

(Cunha, 2004) implementou desenvolvimento das características de paralelização computacional do MEC em diferentes plataformas paralelas; (VERA-TUDELA, BARBOSA, FONTES JÚNIOR, 2009), Desenvolvimento de Software para a Resolução de Problemas da Mecânica dos Sólidos; (FONTES JÚNIOR, 2010),uma abordagem multithreading para o acoplamento entre os Métodos dos elementos de contorno e finitos.

Muito destes trabalhos citados, foram desenvolvidos para arquiteturas paralelas de memória compartilhada quanto para arquiteturas paralelas de memória distribuída.

1.2 Objetivos

O presente estudo tem como objetivo dar continuidade às pesquisa sobre paralelismo, atuando num programa sequencial, desenvolvido na linguagem Fortran 77 (VERA-TUDELA, 2003), que efetua análises numéricas de problemas específicos (Tensão e Deformação em 2D) na área de mecânica dos sólidos através do MEC com representação física de uma barra engastada e tracionada. Com isto espera atingir alguns objetivos específicos:

- ✓ Diminuir o tempo de execução da aplicação.
- ✓ Implementar programa paralelo de forma eficiente.
- ✓ Aperfeiçoar o código da aplicação sequencial para uma versão atualizada da linguagem.
- ✓ Executar aplicação na plataforma multicore em 1, 2 e 4 núcleos e comparar resultados.
- ✓ Comparar o desempenho computacional da aplicação na versão sequencial e na versão paralela.

1.3 Justificativa e Motivação

Muitas técnicas numéricas foram desenvolvidas nas últimas décadas para examinar o comportamento dos sólidos submetidos a uma carga dinâmica. Os métodos mais adequados parecem ser o Método dos Elementos Finitos (MEF) e, mais recentemente, o Método de Elementos de Contorno (MEC). Estes métodos têm sido aplicados com sucesso para uma ampla classe de problemas tais como a propagação de ondas, a vibração, e estrutura solo Chyou-Chi Chien , Tong-Yue Wu, 2001).

O Método de Elementos de Contorno (MEC) requer discretização apenas no contorno da geometria do problema, mas não do seu interior como um todo, diminuindo o esforço computacional.

1 CONCEITOS MATEMÁTICOS

2.1 Introdução

Este capítulo trata da teoria básica do Método dos Elementos de Contorno (MEC). As equações da teoria de elasticidade, Delta de Kronecker, Delta Dirac, Coeficiente de Poisson, Quadratura Gaussiana, Lei de Hooke e Módulo Young as quais são necessárias para a formulação do MEC são brevemente apresentadas.

2.2 Delta de Kronecker

Considere os eixos de coordenadas ortogonais por 1,2 e 3 e os vetores unitários por $\hat{e}_i$, onde i corresponde ao eixo (1,2 e 3). O produto escalar entre $\hat{e}_i$ e $\hat{e}_j$ pode ser escrito como:

$$\hat{e}_i \cdot \hat{e}_{j=} \delta_{ij} \tag{2.1}$$

Onde δ_{ij} é o Delta de Kronecker. A quantidade de δ_{ij} é definida assim:

$$\delta_{ij} = \begin{cases} 1, \textit{se } i = j \\ \\ 0, \textit{se } i \neq j \end{cases} \tag{2.2}$$

2.3 Delta Dirac

As excitações concentradas, como forças pontuais em mecânica dos sólidos e dos fluidos, cargas elétricas pontuais em eletrostática ou forças de impulso em acústica, pode ser alcançada com a adoção da função Delta de Dirac.

Considere-se o ponto ξ como o de aplicação da força ($\xi \in \Omega$).

$$\begin{cases} \Delta(\xi, x) = 0 & \text{se } x \neq \xi \\ \int_\Omega \Delta(\xi, x)\ d\Omega(x) = 1 & \textit{se } x \in \Omega \end{cases} \tag{2.3}$$

2.4 Coeficiente de Poisson

Representa a relação entre as deformações lateral e longitudinal na faixa de elasticidade. A razão entre essas deformações é uma constante denominada coeficiente de Poisson.

$$v = -\frac{\varepsilon_{lat}}{\varepsilon_{lon}} \tag{2.4}$$

O sinal negativo é utilizado pois o alongamento longitudinal (deformação positiva) provoca contração lateral (deformação negativa) e vice-versa.
O coeficiente de Poisson é adimensional e seu valor se encontra entre zero e meio.
$0 \leq \nu \leq 0{,}5$.

2.5 Lei de Hooke

Os materiais de forma geral, apresentam relação linear entre tensão e deformação na região de elasticidade. Consequentemente, um aumento na tensão provoca um aumento proporcional na deformação. Essa característica é conhecida como Lei de Hooke.

$$\delta = \varepsilon_{lon}.E \tag{2.5}$$

Onde: E = módulo de elasticidade ou constante de proporcionalidade.

ε_{lon}= Deformação elástica longitudinal

2.6 Módulo Young

Módulo de elasticidade longitudinal ou módulo de Young É uma grandeza proporcional à rigidez de um material quando este é submetido a uma tensão externa de tração ou compressão. Basicamente, é a razão entre a tensão aplicada e a deformação sofrida pelo corpo, quando o comportamento é linear, como mostra a equação E= δ/ε_{lon}, em que:
E= Módulo de elasticidade ou módulo de Young (Pascal)
δ= Tensão aplicada (Pascal)

ε_{lon}= Deformação elástica longitudinal do corpo de prova (adimensional).

2.7 Quadratura Gaussiana

Muitas das integrais que são necessárias calcular no âmbito da aplicação do MEF e MEC não são triviais, ou a primitiva da função integrada não existe explicitamente, ou é demasiado complicada para viabilizar a sua utilização prática. Por este motivo é essencial recorrer a técnicas de integração numérica, que também recebem a designação de quadratura. Neste método os pontos escolhidos seguem um critério bem definido, com o objetivo de fornecer resultados exatos para polinômios até a ordem *(2n-1)*.
O procedimento de integração numérica genericamente designado quadratura de Gauss tem como principal vantagem o fato de poder ser facilmente incluído num programa de computador destinado à análise de estruturas pelo MEF e MEC. A principal dificuldade associada à sua utilização reside na necessidade de escolher um número de pontos de Gauss adequado à precisão pretendida.

A solução numérica da integral $A = \int_a^b f(x)dx$ foi resolvida a partir da integração de um polinômio interpolador, gerado pela forma de Gregory-Newton[2], que aproxima a função *f(x)*.

2.8 Elasticidade Bidimensional

Essa seção apresenta as principais relações matemáticas inerentes a teoria da elasticidade linear, que serão necessárias para o desenvolvimento do Método de Elementos de Contorno e suas aplicações a problemas de Estado Plano de Tensões e Estado Plano de Deformações. O seu estudo pode ser feito de um ponto de vista totalmente matemático tanto como físico (TIMOSHENKO, GOODIER, 1970).

2.8.1 Estado Plano de Tensão

Tensão corresponde a uma força ou carga, por unidade de área, aplicada sobre um material. Um corpo solido pode está sujeito a um conjunto de forças, (Figura 1), podem ser cargas localizadas como F1 e F2, pode ser distribuído ao longo de um comprimento como Q1.
No Estado Plano de Tensão, não há restrições a deformação ao longo da espessura, *e*, o que implica que a tensão na direção X3 seja nula ao longo da espessura.

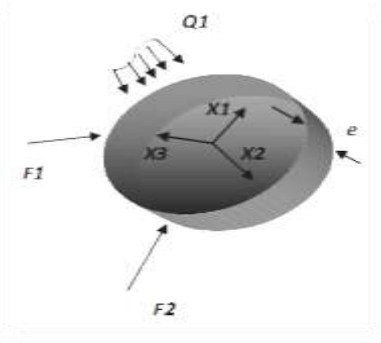

Figura 1: Estado Plano Tensão

O tensor de elasticidade D que relaciona as tensões com as deformações é representado na forma matricial por:

$$D = \frac{E}{1-v^2}\begin{pmatrix} 1 & v & 0 \\ v & 1 & 0 \\ 0 & 0 & \frac{1-v}{2} \end{pmatrix} \quad (2.6)$$

[2] James Gregory Matemático, físico teórico e astrônomo escocês nascido em Drumoak, nas proximidades de Aberdeen, predecessor de Isaac **Newton**, que desenvolveu na Inglaterra o *teorema binomial* para potências inteiras fracionárias.

Onde E é o módulo de Young e ν o coeficiente de Poisson, os quais representam as características físicas do material.

2.8.2 Estado Plano de Deformação

Deformação é a mudança nas dimensões, por unidade da dimensão original.
No Estado Plano de Deformação (Figura. 2), a deformação na direção x3 é nula, o que implica na existência da tensão na direção x3.

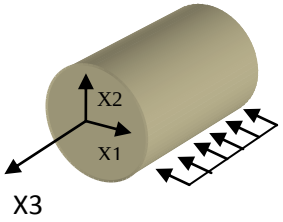

Figura 2: Plano de Deformação.
Fonte: Fontes Júnior, et al 2010.

O tensor de elasticidade D pode ser representado de forma matricial por:

$$D = \frac{E}{(1+v)(1-2v)} \begin{pmatrix} 1-v & v & 0 \\ v & 1-v & 0 \\ 0 & 0 & \frac{1-2v}{2} \end{pmatrix} \tag{2.7}$$

Onde E é o módulo de Young e v o coeficiente de Poisson, os quais representam as características físicas do material.

2.8.3 Equação de Governo

Os problemas pertinentes à mecânica dos sólidos são, na sua maior parte, problemas de campo vetorial, pois a cada ponto estão associadas grandezas cuja definição requer a identificação de módulo, direção e sentido, como no caso dos deslocamentos e forças de superfície (VERA-TUDELA, 2003). Aqui são apresentadas em notação matriciais corpo de domínio Ω.

$$E = \frac{1}{2}[\nabla U + \nabla U^T] \tag{2.8}$$

da relação entre tensão e deformação para um material elástico isotrópico (lei de Hooke)

$$S=DE, \tag{2.9}$$

e a equação de equilíbrio elástico

$$\text{Div } S + b = 0 \tag{2.10}$$

onde E é o tensor de deformação infinitesimal, u é o campo de deslocamentos, S é o tensor de tensões e b é o vetor das forças de volume.

Sejam Γu e Γp subconjuntos do contorno do domínio Ω, tal que $\partial\Omega = \Gamma_u \cup \Gamma_p$: Então o problema de elasticidade linear pode ser enunciado como: Dados $\Omega, \Gamma_u, \Gamma_p$ D e b em Ω , deslocamentos prescritos ū em Γ_u e forças de superfície prescritas $\bar{p}$ em Γp (Figura 3).

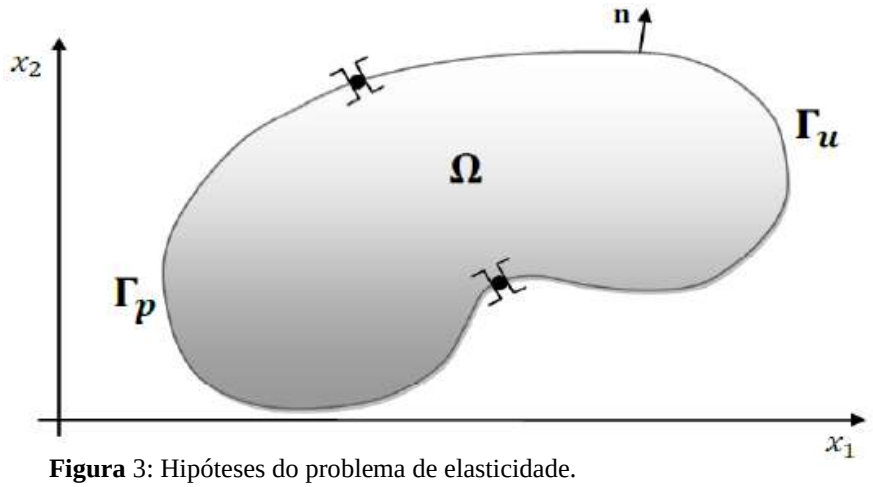

Figura 3: Hipóteses do problema de elasticidade.
Fonte: Fontes Júnior, et al 2010

Determinar **u** satisfazendo as condições de contorno

$$u= \bar{u} \text{ em } \Gamma_u, \tag{2.11}$$

$$Sn= \bar{p} \text{ em } \Gamma p, \tag{2.12}$$

sendo **n** o vetor dos cossenos diretores da normal ao contorno, apontando para fora.
Para um material elástico isotrópico, a lei de Hooke (Equação 2.9) pode ser substituída por:

$$S=2\mu E + \lambda(\text{tr}\mathbf{E})\mathbf{I}, \tag{2.13}$$

onde μ e λ são as constantes de Lamê, definidas por:

$$\mu = \frac{E}{2(1+v)} \tag{2.14}$$

e

$$\lambda = \frac{Ev}{(1+v)(1-2v)}. \tag{2.14}$$

Sendo o corpo representado por homogêneo e isotrópico, a (Equação 2.8) e a (Equação 2.10) podem ser combinadas com a (Equação 2.13) e por meio de operações algébricas formar a equação de equilíbrio de Navier:

$$\mu\Delta\mathbf{u} + (\lambda+\mu)\,\Delta(\Delta.\mathbf{u})+\mathbf{b}=0 \text{ em } \Omega \tag{2.15}$$

A (Equação 2.15) é um sistema de equações diferenciais, que sujeito às condições de contorno (Equação 2.11) e (Equação 2.12), definem o problema de elasticidade estática linear relacionando todos os campos de deformações, tensões e deslocamentos.
As equações anteriores foram definidas para o estado plano de deformação, para problemas de estado plano de tensão, basta substituir *v* por $\tilde{v}$, tal que

$$\tilde{v} = \frac{v}{(1+v)} \tag{2.16}$$

O teorema de Reciprocidade de BETTI é uma das formas para a formulação do Métodos dos Elementos de Contorno que será desenvolvida na próxima etapa.
A demonstração para o Teorema BETTI pode ser encontrada nas referências (BREBBIA, DOMINGUEZ, 1992).

Teorema 1 (Teorema de Reciprocidade de BETTI) Seja [u;E; S] e [ū; Ē; ŝ] os estados elásticos correspondentes as forças de volume b e ˜b respectivamente e ainda assuma que o tensor D seja simétrico. Então:

$$\int_{\partial\Omega} Sn.\bar{u}\, dA + \int_{\Omega} b.\bar{u}\, dV = \int_{\partial\Omega} \hat{s}n.u\, dA + \int_{\Omega} \sim b.u\, dV = \int_{\Omega} S.\bar{E}\, dV = \int_{\Omega} \hat{s}.E\, dV \tag{2.17}$$

O teorema de reciprocidade de BETTI estabelece que se dois estados elásticos [u;E; S] e

[ū; Ē; ŝ] existem e satisfazem às equações de equilíbrio, então o trabalho realizado pelas forças externas do sistema [u;E; S] sobre os deslocamentos do sistema [ū; Ē; ŝ] é igual ao trabalho realizado pelas forças de [ū; Ē; ŝ] sobre os deslocamentos de [u;E; S].

2.9 Método de Elemento de Contorno

Este método foi a principio chamado de Método das Equações Integrais. Mas para distingui-lo dos outros métodos que envolviam também equações integrais, ele foi finalmente chamado de Método dos Elementos de Contorno.

O Método dos Elementos de Contorno (MEC) tem sido estabelecido como um método numérico alternativo ao Método dos Elementos Finitos (MEF). Isto se deve a sua simplicidade e redução na dimensionalidade do problema. Por exemplo, um problema bidimensional se reduz somente a linha unidimensional de contorno do domínio necessário a ser discretizado dentro dos elementos e, um problema tridimensional se reduz a uma superfície do domínio que necessita ser discretizado.

A solução de problemas em ciência e engenharia passa por diversas etapas de simplificação. Entre elas está a proposição do modelo matemático aproximado, utilizando-se equações diferenciais. A escolha do método de solução destas equações diferenciais e a simplificação numérica através da discretização do problema.

O Método dos Elementos de Contorno é um dos métodos aproximados utilizados em Ciência e Engenharia. Ele é aplicado na solução de equações diferenciais, onde estas são transformadas em equações integrais aplicadas ao contorno do problema. Este por sua vez é discretizado em elementos que podem ser constantes lineares, quadráticos ou cúbicos.

2.9.1 Formulação da Equação de contorno para Elasticidade 2D

O método de Elemento de Contorno, representa uma das mais sofisticadas ferramentas existentes para a resolução de problemas da Ciência e da Engenharia.

Estuda-se o desenvolvimento matemático para a obtenção das equações integrais de contorno que governam o problema da elasticidade 2D. Os problemas pertinentes à mecânica dos sólidos são, na sua maior parte, problemas de campo vetorial, pois a cada ponto estão associadas grandezas cuja definição requer a identificação de módulo, direção e sentido, como no caso dos deslocamentos e forças de superfície.

Inicia-se o estudo a partir da equação de Navier, mas levando em conta algumas simplificações, como carga de domínio nula ($b_i = 0$) e problema estático (u_i= 0). Assim, a equação de Navier pode ser escrita da seguinte forma:

$$\mu u_{j,ii} + (\lambda + \mu) u_{j,ii} = 0 \qquad (2.18)$$

A formulação do MEC consiste em ponderar a (Equação 2.18) por uma função u^*, com características especiais e depois integrá-la no domínio. Por meio de um tratamento matemático adequado, essa nova equação integral ponderada de domínio é transformada em uma equação integral de contorno.

Então, após os procedimentos descritos, temos a seguinte expressão:

$$\mu \int_\Omega u_{j,ii}\ u_j^* d\Omega + (\lambda + \mu) \int_\Omega u_{i,ij}\ u_j^* d\Omega = 0 \qquad (2.19)$$

Ω corresponde ao domínio da região.
Para realizar o tratamento matemático adequado, são necessárias duas propriedades matemáticas importantes: a propriedade da integração por partes e, do teorema da divergência, respectivamente:

$$\int_\Omega u\, v_{,i}\, d\Omega = \int_\Omega (u\, v)_{,i}\ d\Omega - \int_\Omega v\, u_{,i}\, d\Omega \qquad (2.20)$$

$$\int_\Omega (u\, v)_{,i}\, d\Omega = \int_\Gamma u n_i \ \mathrm{d}\Gamma \qquad (2.21)$$

Só para lembrar, sempre que depararmos com a seguinte nomenclatura: $v_{,i}$, o índice precedido de uma vírgula, indicará diferenciação (derivação), ou seja, $v_{,i}$ corresponde a derivada de v.

A partir da propriedade de integração por partes, podemos desmembrar nossa equação em várias equações integrais menores e mais fáceis de manipular, essa é a importância da utilização desta propriedade. Uma vez, desmembrada a equação a partir de integração por parte, utilizamos o teorema da divergência com o intuito de converter um dos termos da equação que está no domínio para o contorno.

Aplica o teorema da divergência no primeiro termo originado na (Equação 2.20) que está no domínio e reescreve como uma integral no contorno.
Após os procedimentos rescritos e utilização das propriedades descritas, obtivemos o seguinte resultado:

$$\int_\Omega [\mu(u_j\ u^*_{j,ii}) + (\lambda+\mu)(u_j\, u^*_{i,ij})]d\Omega + \int_\Gamma [\mu(u_{j,i}\, u^*_j n_i - u_j u^*_{j,i} n_i)]d\Gamma +$$
$$+ \int_\Gamma [(\lambda+\mu)(u_{i,i}\, n^*_j n_j - u_i u^*_{j,j} n_i)]d\Gamma = 0 \qquad (2.23)$$

2.10 Elasticidade Aplicada ao Estudo de Chapas

Uma força atuando sobre um material pode provocar deformação, quando terminar a força sobre o material, e o mesmo voltar ao seu estado original, essa propriedade é chamado de elasticidade. Se o retorno for apenas parcial, denomina-se material parcialmente elástico, porém, quando a deformação for permanente denomina-se plástico.

3. COMPUTAÇÃO PARALELA

O grande interesse por problemas cada vez mais complexos tem levado a necessidade de computadores cada vez mais potentes para resolvê-los.
No entanto, limitações físicas e econômicas têm restringido o aumento da velocidade. A primeira seria aumento no consumo de energia. "O consumo de Energia aumenta proporcionalmente com a frequência de operação" (KOCH, 2005). O número de transistores por chip aumentou nos últimos anos e os novos desafios da engenharia é trabalhar na criação de dispositivos elétricos que consumam menos energia e produzam menos calor. O segundo é que a velocidade das memórias não aumentam tão rapidamente quanto a dos processadores. O terceiro problema trata do tamanho dos transistores. Como eles se Tornam cada vez menores, os chips atuais são mais densos e precisam de maiores comprimentos de fios de interconexão.

Entretanto, a grande capacidade de integração, combinada com a redução dos custos de fabricação, torna o uso de múltiplos processadores independentes atraente, dando início á era dos chips multicore (CHYOU-CHI, TONG-YUE, TANENBAUM, 2003). Tais processadores apresentam desde uns poucos até várias dezenas de cores (núcleos de processamento) homogêneos ou heterogêneos. Embora sua disponibilidade beneficie imediatamente ambientes multiprogramados, processadores multicore exigem que desenvolvedores de aplicações explorem paralelismo no nível de threads (CHYOU-CHI, TONG-YUE, BREBBIA, 2010) para habilitar melhorias no tempo de execução (OLUKOTUN, 2000).

Neste capitulo, vamos abordar sobre conceitos básicos de arquitetura de computadores envolvendo assuntos ligados a computação paralela: tecnologia Multicore, uso de memória compartilhada, o conceito de Thread (CHYOU-CHI, TONG-YUE, 1999), Taxonomia de computadores paralelos, avaliação de desempenho de aplicações paralela e modelo de programação paralela (OPENMP).

3.1 Taxonomia de computadores paralelos

Muitos tipos diferentes de computadores paralelos têm sido proposto e construídos ao longo dos anos. Permaneceu a taxonomia de Flynn (FlYNN, 1972) que baseia-se em dois conceitos de instruções e sequência de dados. Como resultado, temos as 4 (quatro) grandes classes, relacionadas a seguir:Veja (figura 4).

3.1.1 SISD (*Single Instruction Single Data*)

Modelo de sistemas que suportam uma única sequência de instruções e apenas uma sequência de dados. Computadores com apenas um processador e com apenas um núcleo. Mesmo em sistemas com um único processador e um núcleo é possível encontrar algum nível de paralelismo, utilizando a técnica de Pipeline .

3.1.2 SIMD (*Single Instruction , Multiple Data*)

Modelo de execução em paralelo que permitem a manipulação de vetores inteiros simultaneamente, possibilitando a execução de uma mesma instrução sobre diferentes elementos de um ou mais vetores.

3.1.3 MIMD (*Multiple Instruction stream, Multiple Data stream*)

Modelo de execução em paralelo que trata múltiplas sequencias de instruções e múltiplas sequências de dados. Este modelo é mais flexível que o SIMD, porém as interações entre processadores tendem a ser mais complicadas e menos eficientes.

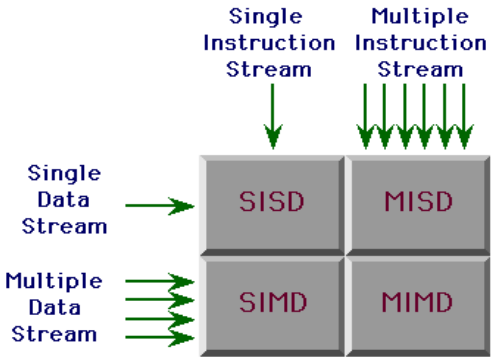

Figura 4- Modelos de arquitetura

3.2 Memória Compartilhada

A memória pode ser acessada por diferentes CPU em um sistema de multiprocessamento. Um sistema de memória compartilhada é simples de ser programado já que todos os processadores compartilham a mesma visão dos dados, e a comunicação entre processadores pode ser tão rápida quanto o acesso à memória na mesma posição.

3.3 Multicore

O processamento paralelo é uma forma eficiente de processamento de informação com ênfase na exploração de eventos simultâneos na execução de um *software*.
Meados de 2005, fabricante de processadores como Intel, começou a desenvolver uma tecnologia chamada de Multicore. Essa representa uma grande revolução na ares de tecnologia computacional. São capazes de prover maior capacidade de processamento com um custo/benefício melhor.
A tecnologia Multicore (múltiplos núcleos) consiste em colocar duas ou mais unidades de execução (cores) no interior de um único 'pacote de processador' (um único chip), (Figura 5). O sistema operacional trata esses núcleos como se cada um fosse um processador diferente, com seus próprios recursos de execução. Na maioria dos casos, cada unidade possui seu próprio cache e pode processar várias instruções simultaneamente. Adicionar novos núcleos de processamento a um processador possibilita que as instruções das aplicações sejam executadas em paralelo em vez de serialmente, como ocorre em um núcleo único.

Os processadores de múltiplos núcleos permitem trabalhar em um ambiente multitarefa. Em sistemas de um só núcleo, as funções de multitarefa podem ultrapassar a capacidade da CPU, o que resulta em queda no desempenho enquanto as operações aguardam serem processadas. Em sistemas de múltiplos núcleos, como cada núcleo tem seu próprio cache, o sistema operacional dispõe de recursos suficientes para lidar com o processamento intensivo de tarefas executadas em paralelo. Portanto, melhora-se a eficiência do sistema e o desempenho dos aplicativos em computadores que executam vários aplicativos simultaneamente.

Arquitetura Multicore é geralmente um multiprocessamento simétrico (SMP) implementado em um único circuito VLSI (*Very Large Scale Integration*). O objetivo é melhorar o paralelismo no nível de threads, ajudando especialmente as aplicações que não conseguem se beneficiar dos processadores superescalares atuais por não possuírem um bom paralelismo no nível de instruções.

Esta arquitetura propicia o chamado paralelismo ao nível de chip. Onde dois ou mais processadores idênticos são conectados a uma única memória principal. Isso permite que qualquer processador trabalhe em qualquer tarefa, não importando onde que ela esteja localizada. Assim, é possível ficar movendo as tarefas entre processadores de modo a tornar a carga de trabalho o mais eficiente possível. Porém, há um custo a se pagar: como a memória é muito mais devagar do que o processador, se em arquiteturas single-core é gasto uma grande parcela do tempo esperando pelos dados da memória.

Suas vantagens são: melhor localidade de dados se comparado com outras arquiteturas de multiprocessamento; melhor comunicação entre as unidades; economia de espaço e energia. É importante observar que o aumento de throughput não ocorre no caso da execução de uma única aplicação que não possa ser paralelizada, mas, nos outros casos, e sempre que se considera o sistema como um todo (rodando várias aplicações simultaneamente), tal aumento é bem notável. (CARDOSO, 2005).

É importante notar que, para uma total utilização do poder de processamento oferecido pelo Multicore, as aplicações devem ser escritas de modo a usar intensivamente o conceito de threads. Assim, melhora-se o desempenho de cada aplicação. Esse trabalho foi baseado na tecnologia multicore.

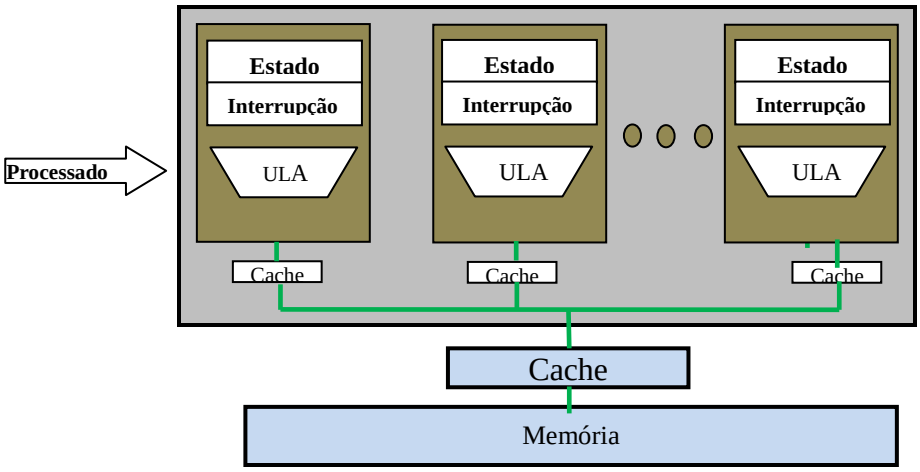

Figura 5: CPU com Core

3.4 Threads

Um programa é composto por uma sequência de instruções, laços, desvios e procedimentos e funções. Quando é executado precisa de recursos de hardware, software e espaço de endereçamento. Esse ambiente que o sistema disponibiliza é chamado de processo. Um processo só suporta um único programa no seu espaço de endereçamento e apenas uma instrução é executada por vez (MACHADO, MAIA, 2002).

Numa aplicação paralela são criados múltiplos processos e subprocessos independentes, fazendo com aplicação possa ser dividida.

A comunicação entre processo e subprocesso é lenta, pois processo possui seu espaço de endereçamento próprio, e a troca de mensagens feitas pelas técnicas como, pipe, semáforos e sinais não é simples consumindo muito tempo do sistema.

O conceito de thread veio na tentativa de reduzir o tempo gasto na criação, eliminação e troca de contexto de processos nas aplicações concorrentes, bem como economizar recursos do sistema como um todo. Em ambiente multithread, um único processo pode suportar múltiplos threads, cada qual associado a uma parte do código da aplicação (Figura 6). Neste caso não é necessário haver diversos processos para implementação da concorrência. Threads compartilham o processador da mesma maneira que um processo, ou seja, enquanto um thread espera por uma operação E/S, outro thread pode ser executado.

Cada thread possui o seu próprio contexto de hardware, porém compartilha o mesmo contexto de software e espaço de endereçamento com os demais threads do processo. O compartilhamento do espaço de endereçamento permite que a comunicação de threads dentro do mesmo processo seja realizada de forma simples e rápida diminuindo overhead.

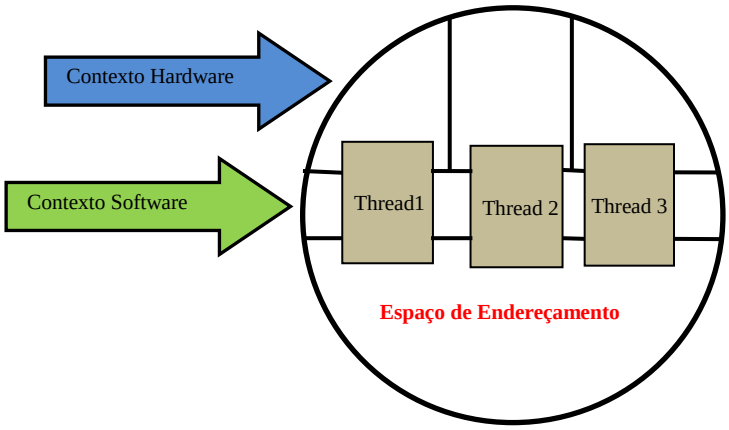

Figura 6: Processo Multithread

3.5 Performance

O principal objetivo do projeto de máquina paralela é de fazê-la rodar mais rápido que uma máquina dotada de um processador single-core. Para que isto aconteça devemos examinar algumas questões relativas à performance das arquiteturas de máquinas paralelas tendo como

perspectiva hardware e eficiência de algoritmos.

Medidas de Hardware

Da perspectiva de hardware, as medidas de interesse são a velocidade do processador e dos dispositivos de E/S e a performance da rede de interconexão. As velocidades do processador e dos dispositivos de E/S são as mesmas que a obtidas em um sistema com um único processador. Portanto, os parâmetros chave para performance em sistemas paralelos são aqueles associados à rede de interconexão. Existem dois parâmetros principais para medida de performance das redes de interconexão: a latência e a banda passante.

Define-se como latência total (ida e volta) como tempo gasto por um processador para enviar um pacote e receber uma resposta. Se o pacote for enviado à memória, a latência mede o tempo de leitura ou escrita de uma palavra ou de um bloco de palavras. Se o pacote for enviado a outro processador, a latência mede o tempo de comunicação entre os processadores para aquele pacotes daquele tamanho.

Outra medida da performance do hardware é a banda passante. A maioria dos programas paralelos, desloca um conjunto de dados de um lado para outro, de maneira que a quantidade de bits por segundo que um sistema pode deslocar é um dado critico para a performance. Um exemplo, é a banda passante média de cada processador. Se cada processador for capaz de colocar na saída 1 MB/s, a comunicação será limitada pela quantidade de dados que cada processador puder colocar na saída.

Medidas de Software

As medidas de efetuadas por hardware, como a latência e a banda passante, mostram aquilo que o hardware é capaz de realizar.

No entanto, os usuários têm uma perspectiva diferente. Eles querem saber quanto seus programas vão rodar mais rápido em computadores paralelos. Para eles a medida fundamental é o ganho: quanto mais rápido os programas rodam em uma máquina com *n* processadores em comparação com o tempo gasto para rodar em uma máquina com um único processador.

Quase nenhum programa de computador consegue o ganho perfeito, mas alguns chegam perto. A virtual impossibilidade de se obter o ganho perfeito, deve-se , em parte, ao fato de todos os programas têm componentes sequencias, na maioria das vezes em sua fase de inicialização, leitura de dados ou obtenção de resultados. A existência de diversos processadores não ajudam neste caso, (Tanenbaum,2001).

Para simplificar os aspectos de análise de escalabilidade dos algoritmos, a análise será realizada sobre os aspectos da **Lei de AMDAHL** e da **Lei de GUSTAFSON-BARSIS**. As duas análises serão comentadas a seguir.

3.5.1 Lei de AMDAHL

O *speedup* *S*, expressa em quantas vezes o algoritmo paralelo é mais rápido que o sequencial. Para isto dividimos o tempo de processamento do algoritmo sequencial T_s pelo tempo do algoritmo paralelo T_p. Já a eficiência ou ganho *E*, aponta a fração de tempo em que as unidades de processamento são usadas. É calculada pela relação entre o speedup, *S*, e o número de processadores, *P*, como mostra as (Equação 3.1, Equação 3.2).

$$S = \frac{T_s}{T_p} \tag{3.1}$$

$$E = \frac{S}{P} \tag{3.2}$$

A primeira notação de escalabilidade de AMDAHL, é uma das formulações que tenta explicar o ganho, que é obtido em uma aplicação aumentando o numero de processadores. Dessa forma, quanto maior for o número de unidades de processamento (CPU's), para um problema de tamanho fixo, maior será o *speedup*, já que tempo de processamento paralelo será menor. Todavia, o *speedup* possui um valor máximo devido à fração sequencial do código.

A segunda notação de AMDAHL: a velocidade de processamento paralelo é limitada. Ela afirma que uma pequena porção do programa que não pode ser paralelizada limitará o aumento de velocidade disponível com o paralelismo. A consequência é que haverá um limite onde o processamento paralelo pode ser aplicado e o fator limitador é o trecho do código sequencial. Idealmente, a eficiência seria 1 para qualquer quantidade de p processadores. Supondo uma aplicação com fração sequencial igual a f e fração paralelizável igual a $(1 - f)$. Então o speedup S esperado é (Equação 3.3):

$$S = \frac{1}{f + \frac{(1-f)}{p}} \tag{3.3}$$

AMDAHL, iniciou suas pesquisas seguindo a ideia de que um trecho do programa é puramente sequencial, e o desempenho deste trecho não é aumentado quando aumenta o número de processadores de execução. Apenas o desempenho do trecho que não é sequencial pode ser aumentado.

A (Figura 7) exemplifica tal limitação. Nela existem dois algoritmos, *Algoritmo. A* e *Alg. B*, de tamanhos diferentes. *S* e *P* representam a porção sequencial e paralela do código, respectivamente. *Alg. B* possui maior *speedup*, quando comparado ao *Alg. A*, porque possui menor parcela sequencial e maior paralela. Apesar disso, seu *speedup* será limitado, já que o paralelismo não reduz o tempo de processamento da porção sequencial.

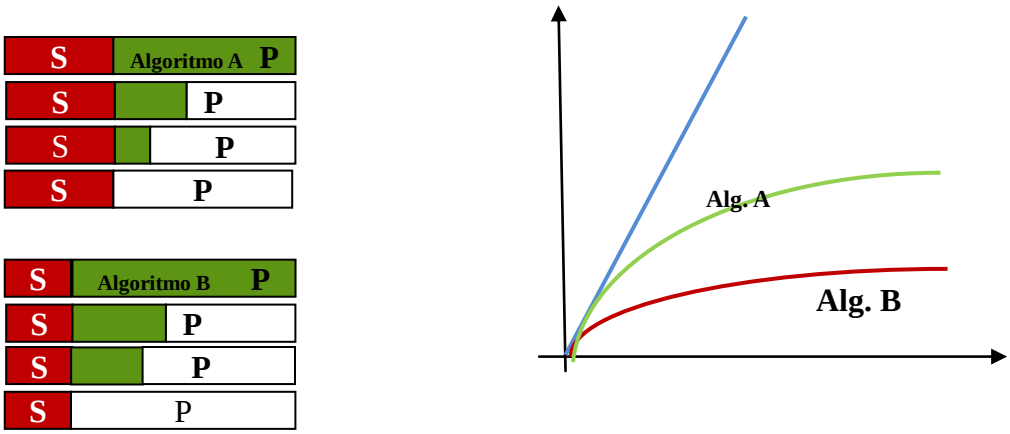

Figura. 7 Fonte: Silva, 2013

3.5.2 Lei de GUSTAFSON-BARSIS

Uma das principais suposições da Lei de Amdahl é o fato de que o tamanho do problema (workload) é fixo e, portanto não é escalável para atingir o poder de computação disponível à medida que o número de máquinas disponíveis aumenta. Isso leva quase sempre ao não aproveitamento dos recursos disponíveis quando uma máquina muito poderosa é empregada na resolução de um pequeno problema.

Devido às limitações da Lei de Amdahl, em 1988, John Gustafson e Ed Barsis propuseram o que veio a ser chamado de Lei de Gustafson-Barsis afirmando que o gargalo seqüencial pode ser diminuído se a suposição de tamanho fixo para o problema for deixada de lado. Eles propuseram a fixação do tempo para a resolução do problema e não o seu tamanho. Em outras palavras, o que se quer é resolver o maior problema possível em uma máquina com elevado poder de processamento com aproximadamente o mesmo tempo de execução gasto para resolver um problema menor em uma máquina menor.

$$S(p) = P - \alpha(P - 1) \qquad (3.4)$$

em que P é o número de processadores, S é o aumento de velocidade e α é a parte não paralelizável do processo. A lei de Amdahl assume um tamanho de problema fixo e que o tamanho da seção sequencial é independente do número de processadores, enquanto a lei de Gustafson não parte dessas premissas.

3.6 OpenMP(Open Multi-Processing)

É uma interface de programação, portável, baseada no modelo de programação paralela de memória compartilhada para arquiteturas de múltiplos processadores. É composto por diretivas de Compilação, biblioteca de execução e Variáveis de Ambiente.

Está disponível para uso com os compiladores C/C++ e Fortran, podendo ser executado em ambientes Unix e Windows (Sistemas Multithreads).

Modelo de Programação Paralela:

- ✓ Baseado em threads: Vários threads compartilham a mesma memória.
- ✓ Explicito: O programador tem o total controle sobre a paralelização do código.
- ✓ Fork – Join: Cria o thread Mestre que executa sequencialmente até encontrar uma região com definição para paralelizar. Em seguida cria um conjunto de threads paralelos e executa-os em paralelo. Quando os threads terminam suas respectivas execuções na região então são sincronizados (JOIN) permanecendo apenas o thread Mestre.
- ✓ Diretivas do compilador: Todo paralelismo do OPENMP é feito em cima da especificação das diretivas do compilador. No Fortran as diretivas geralmente trabalham em pares.
- ✓ Paralelismo recursivo: Dentro de uma região paralela eu posso ter sucessíveis regiões o paralelizáveis que serão executadas.
- ✓ Threads Dinâmicos: Na execução do programa posso aumentar o diminuir o número de Threads.

Figura 8: Exemplo de programação paralela: Modelo de programação que utiliza memória compartilhada, permitindo a todas as tarefas acesso a estrutura de dados na memória global; O trabalho paralelo é efetuado em um conjunto de dados, e os dados devem estar organizados na forma de conjuntos ("loops"), aonde cada tarefa irá trabalhar com partições diferentes dessa estrutura de dados e efetuar a mesma operação em sua partição da estrutura de dados; Implementações: Fortran, HPF - High Performance Fortran, MPI e OpenMP.

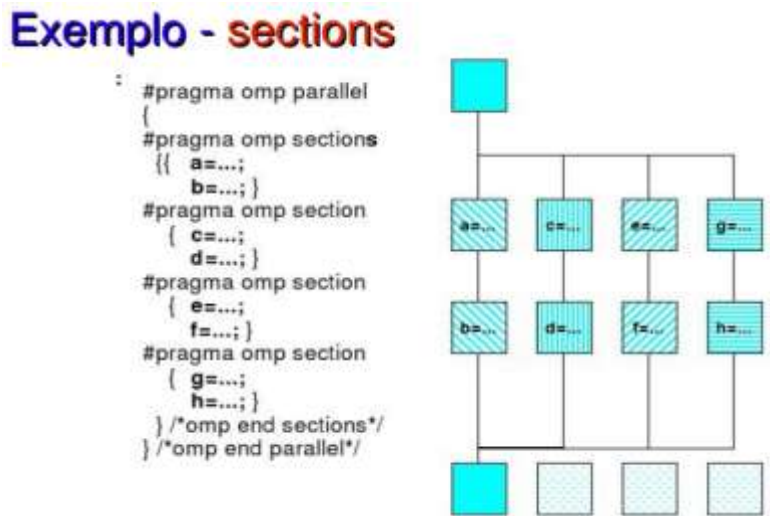

Figura 8: Uso do OPENMP
Fonte Unicamp 2008

4 IMPLEMENTAÇÃO COMPUTACIONAL

4.1 Introdução

Neste capitulo é exibido a problemática, que consiste em paralelizar uma aplicação escrita em linguagem Fortran, que resolve problemas uma placa retangular engastada (Figura.9), alguns resultados de desempenho computacional obtidos com o desenvolvimento de uma estratégia de programação multithreading , para minimizar o gasto de tempo de processamento na análise numérica realizada com MEC, o qual é desenvolvido para uma aplicação paralela para problemas da elasticidade 2D. A eficiência computacional ao se utilizar a programação multithreading na implementação de métodos numéricos em geral,é considerável quando se utiliza um processador com mais de um núcleo de processamento, estes que por sua vez estão ficando cada vez mais populares. Portanto o paralelismo é obtido via programação multithreading. O programa desenvolvido pode ser executado em processadores que possuam somente um núcleo ou vários, (neste caso as threads executarão concorrentemente).

Sistemas operacionais Unix ou Windows podem ser usados., porem para sistema Unix, teremos que fazer alguns ajustes em algumas bibliotecas. Os testes foram feitos usando sistema operacional Windows com processadores Intel e Unix com processador AMD.

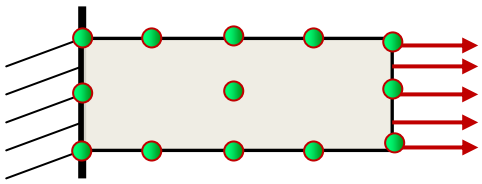

Figura. 9: Representação de uma Barra Engastada

Inicialmente a aplicação consiste em uma série de blocos muito bem definido: Entrada dos parâmetros (uma série de bibliotecas como: NUMERICAL_LIBRARIES, DFPORT e IMSL, e omp_lib.h), Leitura dos dados de arquivo externo, declaração de variáveis, vetores e matrizes. Na leitura do arquivo externo, são processadas as coordenadas do contorno como os nós duplos, conforme (Tabela 1), as coordenadas do ponto interno, conforme (Tabela 2), os elementos de conectividades que fazem referencia ao item número da (Tabela 1), com o apontamento de referencia à (Tabela 3), assim os itens número 1, número 2 e número 3 da referente tabela, está interligando os pontos formando a ideia da barra conforme (Figura 11). Aplicação faz leitura no arquivo texto externo, do itens, conforme (Tabela 4).

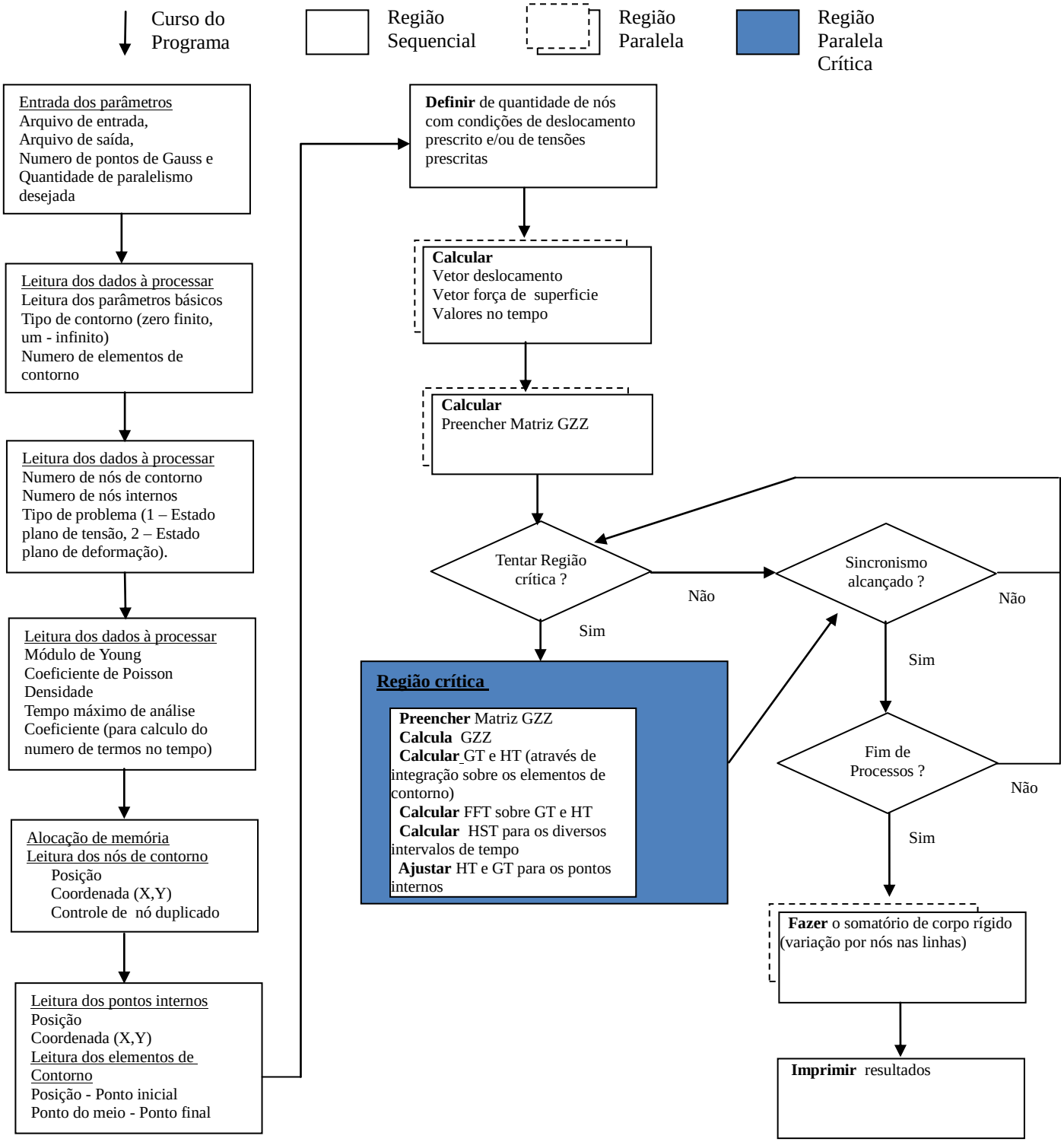

Figura 10 – Implementação da Aplicação. O programa inicia na parte sequencial com, abertura de bibliotecas, declaração de variáveis e leitura de arquivo texto.Em seguida preenche uma serie de vetores e matriz através de cálculos das possíveis soluções nos blocos paralelos. Para atualizar as soluções, ele deverá adentrar a região crítica. Essa mesma região é utilizada para atualizar a solução. A região crítica única garante que todas as operações sejam realizadas atomicamente, evitando assim inconsistência numérica.

Tabela 1: Coordenadas dos Nós de Contorno

Coordenadas dos Nós do Contorno com Nós Duplo			
Numero	X	Y	Duplo
1	0.000	0.000	76
2	0.500	0.00	--
3	1.000	0.00	--
4	1.500	0.00	--
--	--	--	--
--	--	--	25
27	12.000	0.500	--
28	12.000	1.000	--
--	12.000	1.500	--
--	--	--	--
--	--	--	--
76	0.000	0.000	--

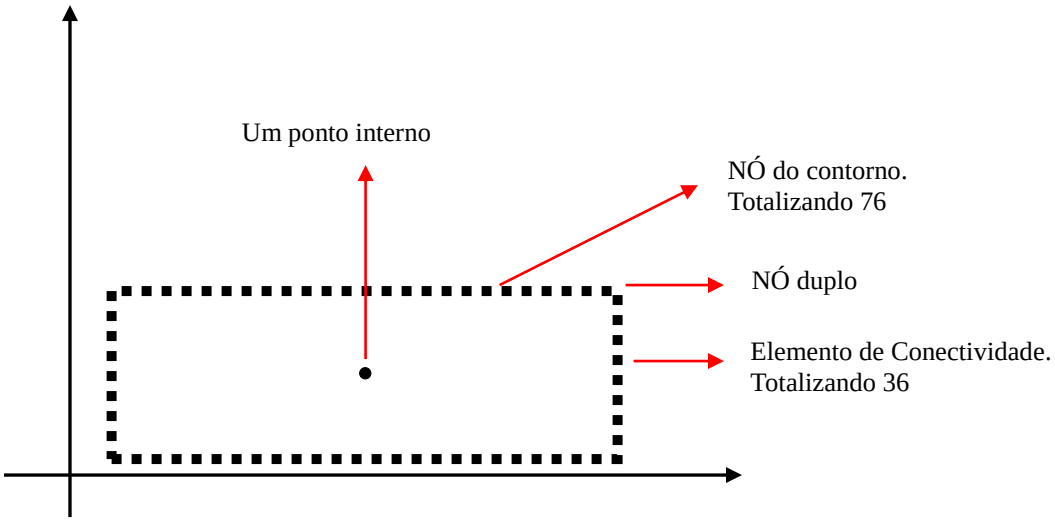

Figura 11: Formação da Barra

Tabela 2: Posição do Ponto Interno

Coordenadas do ponto Interno		
Ponto	X	Y
77	6.0000	3.0000

Tabela 3: Elementos de Conectividade para o Contorno

Elemento de Conectividade				
Elemento	Número 1	Número 2	Número 3	Largura
1	1	2	3	1.0000
2	3	4	5	1.0000
3	5	6	7	1.0000
4	7	8	9	1.0000
5	9	10	11	1.0000
6	11	12	13	1.0000
7	13	14	15	1.0000
--	--	--	--	1.0000
--	--	--	--	1.0000
--	--	--	--	1.0000
35	72	73	74	1.0000
36	74	75	76	1.0000

Tabela 4: Entrada de Dados, lido de um arquivo externo

Entrada de Dados	
Entrada de dados	Valor
Contorno Finito	0
Número de Elementos de Contorno	36
Número de NÓ	76
Número de Pontos Internos	1
Tipo de Problema	2
Modulo de YOUNG	1
Coeficiente de POISSON	0.0
Densidade	1
Tempo Máximo de Análise	100
Pontos e Pesos de Gauss	12
Expoente de 2	6

Seguindo a sequencia sobre aplicação as figuras 10 e 13, da uma visão sobre aplicação.
A (figura 10), mostra uma série de blocos que são paralelizados. Aplicação usou OpenMp para fazer paralelismo. OpenMP é uma interface de programação de aplicativo (API) para a programação multi-processo de memória compartilhada em múltiplas plataformas em C/C++ e Fortran. Ela implementa o modelo de execução multitarefa, que permite que várias threads possam existir dentro do contexto de um processo único. Threads compartilham recursos do processo, mas são capazes de executar de forma independente. Ao serem criadas, as threads são

distribuídas aos processadores por um escalanador de tarefas segundo algum algoritmo de escalonamento. Vale frisar que pode haver mais de uma thread executando em um único processador. Como o algoritmo do escalonador de tarefas visa utilizar o máximo da computação disponível, sempre ocorria a divisão de uma thread para cada processador.
A introdução de uma região crítica foi a principal característica inserida na versão paralela proposta. Em programação concorrente, uma região crítica é uma área de código de um algoritmo que acede um recurso compartilhado que não pode ser executado concorrentemente por mais de uma linha de execução. O objetivo é tornar a operação sobre o recurso compartilhado atômica. Uma técnica utilizada para evitar que dois processos ou threads acessem simultaneamente um recurso compartilhado é a exclusão mútua. Um meio simples de obter exclusão mútua é através de semáforo binário, uma variável compartilhada que só pode assumir dois valores distindos, 0 e 1, por exemplo. O travamento por semáforo deve ser feito antes de utilizar o recurso, e após o uso o recurso, deve ser liberado. Enquanto o recurso estiver em uso, qualquer outro processo que queira utilizar o recurso deve esperar a liberação. A (Figura 12) exemplifica o funcionamento de uma região crítica. Um único semáforo binário foi utiliza na aplicação, delimitando assim uma única região crítica (porção em destaque na Figura 12). Entretanto, em OpenMP existe uma função semelhante a um semáforo binário, mas que não força um thread a esperar a região crítica ser liberada, ou seja, uma função não bloqueante.

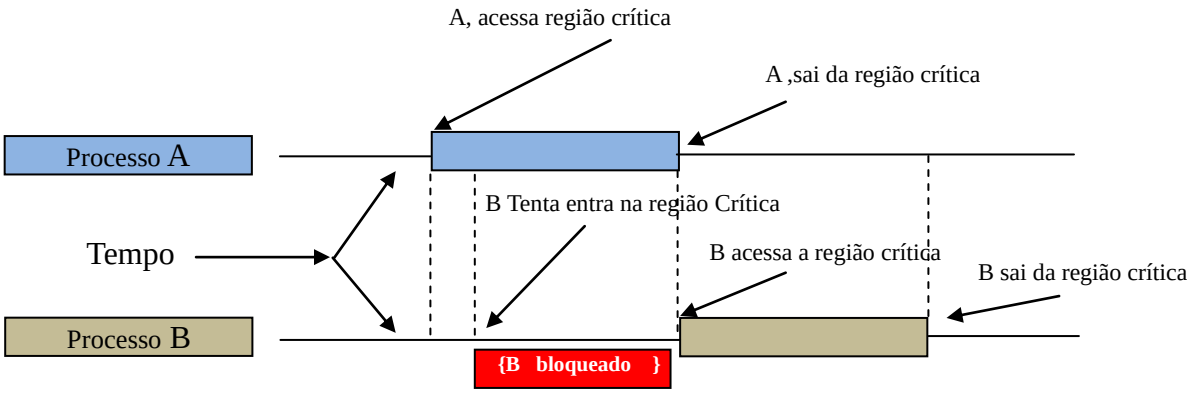

Figura 12: Exemplo do uso da Região Crítica

Foi necessário inserir uma região crítica no algoritmo, pois as operações envolvidas na região crítica devem ser realizadas atomicamente. Segundo expresso nas Equações e funções envolvidas, é necessário utilizar todas as soluções correntes do sistema. Uma simples alteração em alguma das funções correntes, durante o processo de cálculo das Matrizes H,G e HST pode acarretar inconsistência, pois seriam utilizados valores desatualizados. Logo, os recursos compartilhados entre as *threads* são, em sua maioria, variáveis residentes na região crítica. Estas variáveis juntamente com linhas de comando em OpenMP são:

- $OMP PARALLEL DO: Neste caso, começa uma região ou bloco paralelo.
- !$OMP& SHARED(NTT,NT,X,Y,NE,LT,EPS,RLT,INC,NPG,HST): São exemplo de variáveis que são compartilhadas em memória.
- !$OMP& PRIVATE(N,NAUX,RN,I,XFONT,YFONT,J,III,IMI,IFI,ICOD,TAXA,JJ,KK,

- !$OMP& L,LAUX,RL,Z,GZ,K,M,NX,NV,IC): São variáveis que embora estejam numa região compartilhadas, elas são privadas a uma região.
- !$OMP& FIRSTPRIVATE(H,G,HS): Matrizes que embora estejam numa região compartilhada, são privadas.
- !$OMP& SCHEDULE(DYNAMIC,1): Escalonador dinamico
- !$OMP CRITICAL: Inicia uma região critica.

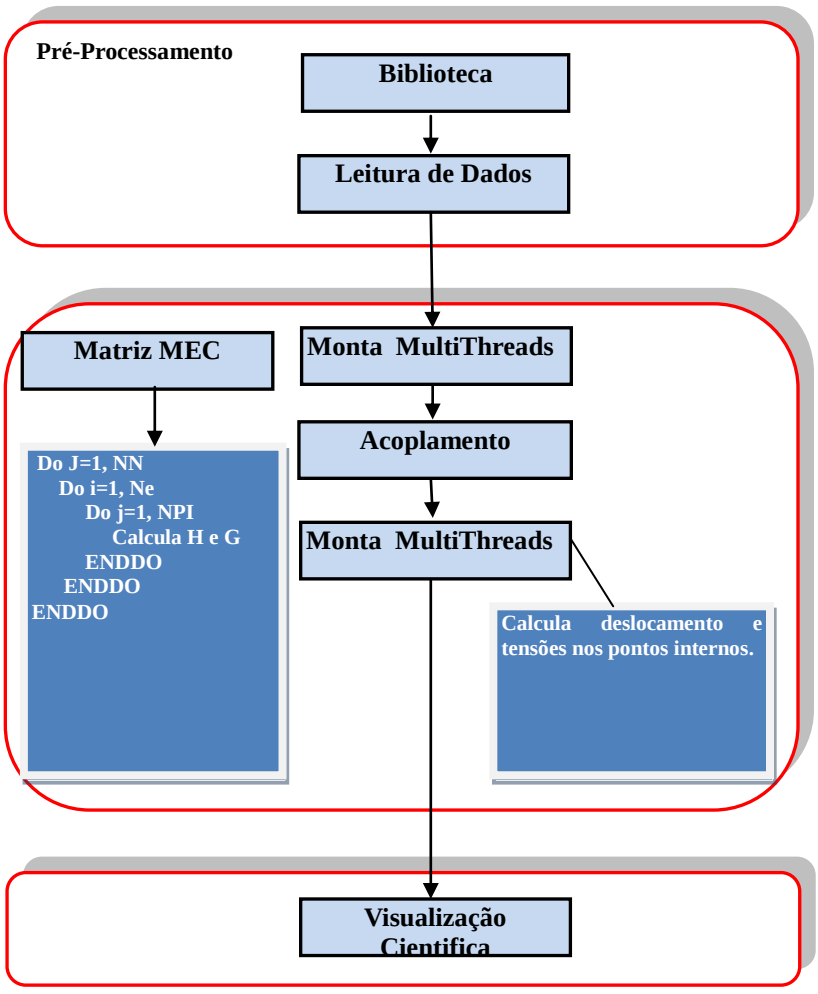

Figura 13**.** Visão Macro da Aplicação.

5 RESULTADOS

Neste estudo não estamos fazendo a comparação da velocidade entre as máquinas, mesmo porque são padrões de velocidades diferentes e sim estamos comparando a diminuição do tempo em função de paralelizar aplicação em núcleos.

Primeira Analise

Foram feitos 4 experimentos , em cada experimento aumentando o número de threads de 1 a 6, isto consigo validar os dados.

O speedup encontrado foi baseado dividindo o tempo da aplicação sequencial (mesmo resultado para encontrado para uma thread) pelo menor tempo, neste caso com duas threads, (Equação 3.1). Isto mostra o quanto mais rápida é aplicação paralela com relação a sequencial.
O ganho se baseou na (Equação 3.2), onde o divide o speedup pelo numero de processadores. Exibindo a fração de tempo de cada CPU.

Tabela 5: Resultados dos testes usando Processador Intel, Primeiro Teste

Equipamento1: Dual–Core Cpu de 2GHz, 3Giga de RAM. SO Windows 7				
Nro Threads	Experimento 1	Experimento 2	Experimento 3	Experimento 4
1	359 s	360 s	359 s	358 s
2	186 s	185 s	186 s	186 s
3	264 s	263 s	263 s	264 s
4	235 s	239 s	235 s	238 s
5	260 s	260 s	259 s	261 s
6	247 s	242 s	246 s	247 s
Speedup	1,93	1,94	1,93	1,92
Ganho	0,96	0,97	0,96	0,96

Com uma máquina com dois núcleos e distribuindo aplicação em dois Threads, podemos perceber (Tabela 5) ,que o seu melhor desempenho é quando temos o número de núcleos igual a número de threads. Quando aumenta o número de threads aumenta o tempo de processamento isto em função do escalonamento gerado pelo número de threads ser maior que o número de núcleos.

Tabela 6: Tempo gasto por cada parte da aplicação

USANDO	1 Threads.	USANDO	2 Threads.
1410658161	0 START	1410657108	0 START
1410658161	0 INICIO LEITURA DE DADOS	1410657108	0 INICIO LEITURA DE DADOS
1410658161	0 FIM LEITURA DE DADOS	1410657108	0 FIM LEITURA DE DADOS
1410658161	0 INICIO DOS CALCULOS	1410657108	0 INICIO DOS CALCULOS
1410658161	0 CONTORNO CONST. NO TEMPO	1410657108	0 CONTORNO CONST. NO TEMPO
1410658161	0 FIM	1410657108	0 FIM
1410658161	0 GZZ	1410657108	0 GZZ
1410658161	0 FIM	1410657108	0 FIM
1410658161	0 MONTA GT E HT	1410657108	0 MONTA GT E HT
1410658168	7 FIM	1410657112	4 FIM
1410658168	7 TRANSFERÊNCIA H e G	1410657112	4 TRANSFERÊNCIA H e G
1410658168	7 FIM	1410657112	4 FIM
1410658168	7 CALCULO DE HST	1410657112	4 CALCULO DE HST
1410658523	355 FIM	1410657293	181 FIM
1410658523	355 SOMATORIO DE CORPO RIGIDO	1410657293	181 SOMATORIO DE CORPO RIGIDO
1410658523	355 FIM	1410657293	181 FIM
1410658523	355 TROCA H E G	1410657293	181 TROCA H E G
1410658523	355 FIM	1410657293	181 FIM
1410658523	355 INVERTE H	1410657293	181 INVERTE H
1410658523	355 FIM	1410657294	182 FIM
1410658523	355 CALCULA VI NO TEMPO	1410657294	182 CALCULA VI NO TEMPO
1410658523	355 FIM	1410657294	182 FIM
1410658523	355 CALCULA VI E RESP	1410657294	182 CALCULA VI E RESP
1410658527	359 FIM	1410657298	186 FIM
1410658527	359 FINAL	1410657298	186 FINAL

A (Tabela 6), demonstra o tempo de cada parte da aplicação. O seu melhor tempo 185 segundos, com duas Threads e pior com tempo 359 segundos, com uma thread, podemos perceber que aplicação é quase toda paralelizável, ou seja analisando por etapas usando o resultado com duas Threads, a parte sequencial durou apenas 4 segundos o restante 182 foi para montagem e cálculos em cima das matrizes totalmente paralelizável. Com conseguimos obter aproximadamente 3,62 da aplicação sendo sequencial e 96,38% sendo paralelizável.

Segunda Analise

Foram feitos 4 experimentos , em cada experimento aumentando o número de threads de 1 a 6, isto consigo validar os dados.

O speedup encontrado foi baseado dividindo o tempo da aplicação sequencial (mesmo resultado para encontrado para uma thread) pelo menor tempo, neste caso com duas threads, (Equação 3.1). Isto mostra o quanto mais rápida é aplicação paralela com relação a sequencial.
O ganho se baseou na (Equação 3.2), onde o divide o speedup pelo numero de processadores. Exibindo a fração de tempo de cada CPU.

Tabela 7. Resultados do Segundo experimento. Processador Intel

Equipamento2: i3 Dual Core 1.47 GHz, 3 Giga de Ram. SO Windows 7				
Nro Threads	Experimento 1	Experimento 2	Experimento 3	Experimento 4
1	420 s	418 s	420 s	418 s
2	246 s	238 s	236 s	228 s
3	199 s	196 s	197 s	196 s
4	231 s	230 s	232 s	2302 s
5	267 s	272 s	274 s	268 s
6	296 s	328 s	278 s	312 s
Speedup	2,11	2,13	2,13	2,13
Ganho	0,70	0,71	0,71	0,71

Com uma máquina com dois núcleos e um virtual e distribuindo aplicação em três Threads podemos perceber que aplicação tem o seu melhor desempenho diminuindo o tempo de processamento da aplicação e que a partir do momento em que aumenta o número de threads aumenta o tempo de processamento isto em função do escalonamento gerado pelo número de threads ser maior que o número de núcleos. Repetindo o feito da aplicação no Equipamento 1. Melhor tempo 196 segundos usando 3 threads. O speedup encontrado (Tabela 7), 2,13 para o experimento 2 ,3,e 4 e o ganho calculado foi 0,71 ou seja, é máximo que aplicação poderá ganhar a nível de performance usando a configuração acima.

Terceira Analise

Foram feitos 4 experimentos , em cada experimento aumentando o número de threads de 1 a 6, isto consigo validar os dados.

O speedup encontrado foi baseado dividindo o tempo da aplicação sequencial (mesmo resultado para encontrado para uma thread) pelo menor tempo, neste caso com duas threads, (Equação 3.1).

O ganho se baseou na (Equação 3.2), onde o divide o speedup pelo numero de processadores. Exibindo a fração de tempo de cada CPU.

Tabela 8: Resultados do Terceiro experimento, Processador Intel

Equipamento3: i5 Quad Core 3.4 GHz com 8 Giga de Ram. SO Windows 7				
Nro Threads	Experimento 1	Experimento 2	Experimento 3	Experimento 4
1	167 s	167 s	165 s	164 s
2	86 s	87 s	84 s	84 s
3	60 s	60 s	60 s	60 s
4	46 s	46 s	45 s	45 s
5	65 s	66 s	65 s	66 s
6	66 s	70 s	73 s	66 s
Speedup	3,60	3,60	3,66	3,64
Ganho	0,9	0,9	0,91	0,91

Com uma máquina com quatro núcleos físicos e distribuindo da aplicação em quatro Threads podemos perceber que aplicação tem o seu melhor desempenho diminuindo o tempo de processamento da aplicação e que a partir do momento em que aumenta o número de threads aumenta o tempo de processamento isto em função do escalonamento gerado pelo número de threads ser maior que o número de núcleos. Repetindo o feito da aplicação no Equipamento 1 e Equipamento 2. Melhor tempo 46 segundos usando 4 threads.

O speedup encontrado (Tabela 8), 3,60 para o tempo 1 e o ganho calculado foi 0,9 ou seja, aplicação foi 3,6 vezes mais rápida que na versão sequencial e cada CPU operou 0,9.

Quarta Analise

Foi feito 1 experimento , neste experimento aumentou-se o número de threads de 1 a 6, para validar os dados.

A exibição da (Tabela 9), serve para verificar em que parte da aplicação foi demandada maior uso de processamento.

O speedup encontrado foi baseado dividindo o tempo da aplicação sequencial (mesmo resultado para encontrado para uma thread) pelo menor tempo, neste caso com duas threads, (Equação 3.1).

O ganho se baseou na (Equação 3.2), onde o divide o speedup pelo numero de processadores. Exibindo a fração de tempo de cada CPU.

Tabela 9: Resultados do quarto experimento, Processador Intel

Equipamento 4: Máquina Vaio i7, 1.8 GHz, 8Giga de RAM, SO Windows 8	
Nro Threads	Experimento 1
1	208 s
2	111 s
3	116 s
4	90 s
5	92 s
6	95 s
Speedup	2,31
Ganho	0,57

Com uma máquina com quatro núcleos físicos e distribuindo da aplicação em quatro Threads podemos perceber que aplicação tem o seu melhor desempenho, diminuindo o tempo de processamento da aplicação e que a partir do momento em que aumenta o número de threads aumenta o tempo de processamento isto em função do escalonamento gerado pelo número de threads ser maior que o número de núcleos. Repetindo o feito da aplicação no Equipamento 1 e Equipamento 2 e Equipamento 3. Melhor tempo 90 segundos usando 4 threads.
O speedup 2,31 mostra que aplicação rodou 2,31 vezes mais rápido que na versão sequencial e que o cada CPU teve 0,57% de operação.

Tabela 10: Tempo gasto por cada parte da aplicação

USANDO	1 Threads.	USANDO	4 Threads.
1410824155	0 START	1410825336	0 START
1410824155	0 INICIO LEITURA DE DADOS	1410825336	0 INICIO LEITURA DE DADOS
1410824155	0 FIM LEITURA DE DADOS	1410825336	0 FIM LEITURA DE DADOS
1410824155	0 INICIO DOS CALCULOS	1410825336	0 INICIO DOS CALCULOS
1410824155	0 CONT. CONSTANTE NO TEMPO	1410825336	0 CONT. CONSTANTE NO TEMPO
1410824155	0 FIM	1410825336	0 FIM
1410824155	0 GZZ	1410825336	0 GZZ
1410824155	0 FIM	1410825336	0 FIM
1410824155	0 MONTA GT E HT	1410825336	0 MONTA GT E HT
1410824159	4 FIM	1410825338	2 FIM
1410824159	4 TRANSF DE FOURIER H E G	1410825338	2 TRANSF DE FOURIER H E G
1410824159	4 FIM	1410825338	2 FIM
1410824159	4 CALCULO DE HST	1410825338	2 CALCULO DE HST
1410824361	206 FIM	1410825423	87 FIM
1410824361	206 SOMATORIO DE CORPO RIGIDO	1410825423	87 SOMATORIO DE CORPO RIGIDO
1410824361	206 FIM	1410825423	87 FIM
1410824361	206 TROCA H E G	1410825423	87 TROCA H E G
1410824361	206 FIM	1410825423	87 FIM
1410824361	206 INVERTE H	1410825423	87 INVERTE H
1410824361	206 FIM	1410825423	87 FIM
1410824361	206 CALCULA VI NO TEMPO	1410825423	87 CALCULA VI NO TEMPO
1410824361	206 FIM	1410825423	87 FIM
1410824361	206 CALCULA VI E RESP	1410825423	87 CALCULA VI E RESP
1410824363	208 FIM	1410825426	90 FIM
1410824363	208 FINAL	1410825426	90 FINAL

A (Tabela 10), demonstra o tempo gasto por cada bloco da aplicação. Verificando o maior tempo 208 segundos, é possível observar que a parte sequencial durou 4 segundos o que equivale a 1,94% do tempo da aplicação, a parte das matrizes paralelizáveis consumiu 202 segundos equivalente a 97,1% do tempo da aplicação e a parte da montagem para resultados também paralelizável durou 2 segundos.
Fazendo uma verificação no menor tempo de 90 segundos de duração, a parte sequencial durou 2 segundos, a parte das matrizes paralelizável durou 85 segundos e a montagem para acoplamento 3 segundos.

6 CONCLUSÃO

A carência de implementação de algoritmos paralelos eficientes devido a *era multicore* e a necessidade de maior robustez de algoritmos quanto aos parâmetros de inicialização. Faz com que os programadores devam se preocupar com outros aspectos que não somente o algoritmo, como a hierarquia de memória, coerência de *cache* e o tipo de interconexão.

Neste trabalho, a eficiência dos processadores multi-core, com as técnicas de programação multithreading foi comprovado em computadores pessoais, para o programa de elasticidade bidimensional proposto por VERA-TUDELLA. As técnicas de programação multithreading são implementadas especialmente para o código do MEC e para resolução do sistema de equações.

Podemos constatar que o tempo diminui a partir do momento em que o número de threads seja igual ao numero de núcleos do processador (melhor tempo), quando temos um número maior de threads que o número de núcleos no processador, o tempo da aplicação tende aumentar depois estabiliza. Com isto poderemos constatar que uso da computação paralela é bastante eficiente e que poderá proporcionar inúmeros benefícios. Benefícios como o aumento de produtividade e maior eficiência na tomada de decisões baseadas na análise de grandes conjuntos de dados.

6.1 Trabalhos Futuros

A implementação atual é voltada para computadores pessoais com processadores de um ou mais núcleos. Como sugestão de trabalho futuro, pode-se estudar um esquema de implementação paralelo que execute em clusters de computadores utilizando a programação *multithreading* para utilizar os diversos núcleos dos processadores que apresentou níveis de *speedups* muito bons em um esquema de memória compartilhada neste trabalho e o protocolo de rede MPI[3] para computação de alto desempenho em ambiente de memória distribuída.
Após implementado, pode-se realizar diversos experimentos funcionais a fim de atestar os melhores e avalia-se seu desempenho comparando-o com outras versões de algoritmos, paralelos ou não, seja quanto ao refinamento da solução, seja quanto à análise de escalabilidade paralela.

[3]MPI Message Passing Interface é um padrão para comunicação de dados em computação paralela

7 REFERÊNCIA BIBLIOGRÁFICA

Akhter, S. ; Roberts, J. Multi-Core Programming: Increasing Performance through Software Multi-threading. Intel Press, 2006.

Amdahl, G. M. (1967), 'Validity of the single processor approach to achieving large scale computing capabilities', *Proc. AFIPS 1967 Spring Joint Computer Conf* pp. 483–485.
United States of America, Atlantic City.

Andrews, G. Concurrent Programming: Principles and Practice. The Benjamin Cummings, 1991.

Cardoso, R. B., S. R. A.S, Fernandes T. M. Multicore, Artigo tecnologia Multicore. 2005

C. A. R. Vera-Tudela, Formulações Alternativas do MEC para Problemas Elastodinâmicos de Mecânica da Fratura com o Uso da Função de Green Numérica, Tese de Doutorado, UFRJ, Engenharia Civil, 2003, Rio de Janeiro.

Casazza, J., "Intel Core I7-800 Processor Series and Intel Core I5-700 Processor Series Based on Intel Microarchitecture (Nehalem)", Intel Corporation, http://download.intel.com/products/processor/corei7/319724.pdf.

C.A. Brebbia, J. C. F. Telles, L. C. Wrobel. Boundary Element Techniques, Theory and Applications in Engineering. Springer-Verlag, Belin Heidelberg New York Tokyo 1984.

C. A. Brebbia, J. C. Dominguez. Boundary Elements an Introductory Course, Second Edition. WIT Press/ Computational Mechanics Publications. Southamption, SO40 7AA, UK.1992

Carissimi, A. S. ; Toscani, S. S. ; Oliveira, R. S. Sistemas Operacionais e Programação Concorrente. Porto Alegre : Sagra Luzzato, 2003.

Chandra, R. ; Dagum, L. ; Kohr, D. ; Maydan, D. ; Mcdonald, J. ; Menon, R. Parallel Programming in OpenMP. San Francisco : Morgan Kaufmann Publishers, 2001.

Chyou-Chi Chien , Tong-Yue Wu, *A particular integral BEM/time-discontinuous FEM methodology for solving 2-D elastodynamic problems*, International Journal of Solids and Structures 38 (2001) 289±306

Cunha, M. Cristina C., *Métodos Numérico*, 2 ª Edição, Editora Unicamp, 2000
E. F. Fontes Júnior, Uma Abordagem Multithreading para o Acoplamento entre os Métodos dos Elementos de Contorno e Finitos, Dissertação de Mestrado UFRJ, Engenharia Civil, 2010, Rio de Janeiro.

F. B. Machado, L.P. Maia, Arquitetura de Sistemas Operacionais, 3 ª Edição, Editora LTC, 2002.

Gustafson, J. L. (1988), 'Reevaluating amdahl's law', *Communications of the ACM* pp. 532–533.

Introdução ao OpenMP, CENAPAD, Unicamp, notas de aula, "www .cenapad .unicamp.br/servicos/treinamentos/openmp"

Kurbel, K., Schneider, B. ; SINGH, K. (1998), 'Solving optimization problems by parallel recombinative simulated annealing on a parallel computer - an application to standard cell placement in vlsi design', *IEEE Transactions on Systems, Man, and Cybernetics* **28**, 454–461.

Miers, L. S., Implementação do MEC para elasticidade tridimensional em ambiente
paralelo de memória distribuída, Master's Thesis, Prog. de Eng. Civil -Coppe - UFRJ, Brazil, 2003.

Microsoft Visual Studio 2005 Professional Guided Tour. OpenMP. Disponível em: <http://msdn.microsoft.com/vstudio/tour/vs2005_guided_tour/VS2005pro/Framework/CPlusOpenMP.htm>. Acessado em 12 de maio de 2006.

OpenMP: Simple, Portable, Scalable SMP Programming. OpenMP. Disponível em: <http:/ /www.openmp.org/>. Acessado em 12 de maio de 2006.

Passos, B. C., Lorena, Avaliação de desempenho de método para a resolução da evolução temporal de sistemas auto-gravitantes em dois paradigmas de programação paralela: troca de mensagens e memória compartilhada, dissertação de mestrado,UNB 2006.

Petzold, C. Programming Windows. 5th ed. Microsoft Press, 1998.

Tanenbaum, A. S., *Organização Estruturada de Computadores*, 4ª Edição, Vrije Universiteit , Amsterdam, Holanda, editora LTC, 2001.

Telles, J. C. F., "A self-adaptive co-ordinate transformation for efficient numerical evaluation of general boundary element integrals", Int. J. *Numer. Meth. Engng.*, v. 24, pp. 959–973, 1987.

Telles, J. C. F., Oliveira, R. F., “Third degree polynomial transformation for boundary element integrals:further improvements”, *Engng. Anal. Bound. Elem.*, v. 13, pp. 135–141, 1994.

Timoshenko, S. P., Goodier, J. N., Theory of Elasticity. Tokyo, McGraw-Hill, 1970

Silva , K,.G., Análise de Escalabilidade de uma Implementação Paralela do Simulated Annealing Acoplado, Dissertação de Mestrado, UFRN 2013.

8 ANEXOS

Descrição Algoritmo

1 – Início
- Entrada dos parâmetros
  - Arquivo de entrada, Arquivo de saída, Numero de pontos de Gauss e
  - Quantidade de paralelismo desejada
- Leitura dos dados a serem processados
  - Leitura dos parâmetros básicos
    - Tipo de contorno (zero finito, um - infinito)
    - Numero de elementos de contorno
    - Numero de nós de contorno
    - Numero de nós internos
    - Tipo de problema (1 – Estado plano de tensão, 2 – Estado plano de deformação).
    - Módulo de Young
    - Coeficiente de Poisson
    - Densidade
    - Tempo máximo de análise
    - Coeficiente (para calculo do numero de termos no tempo)
  - Alocação de memória (baseado nos parâmetros básicos)
  - Leitura dos nós de contorno
    - Posição, Coordenada (X,Y), Controle de nó duplicado
  - Leitura dos pontos internos
    - Posição, Coordenada (X,Y)
  - Leitura dos elementos de Contorno
    - Posição, Ponto inicial, Ponto do meio, Ponto final
  - Definir nós com condições de deslocamento prescrito e/ou de tensões prescritas
  - Leitura das condições de deslocamento prescritas (se houver)
  - Leitura das condições de tensão prescrita (se houver)

2 – Processamento (parte paralizavel)
- Calculo da distancia entre ponto inicial e final (para os elementos de contorno
- Calculo da menor distância
- Calculo dos parâmetros iniciais e das constantes
- Região Critica
- Calcula GZZ
- Calcula GT e HT (através de integração sobre os elementos de contorno)
- Calcula FFT sobre GT e HT
- Calcula HST para os diversos intervalos de tempo
- Ajusta HT e GT para os pontos internos
- Fim-Região Critica
- Faz o somatório de corpo rígido (variação por nós nas linhas)
  - Ajusta HT
- Calcula o vetor independente (VI) para cada ponto no tempo
  - Se IFIP = 1 para o ponto acumula GT * U
  - Senão acumula GT * P
- Para cada ponto no tempo calcula o vetor independente (VI) e a resposta
  - Calcula VI somando (-HT*U + GT*P)
  - Calcula resposta somando HT (no primeiro momento) *VI
  - Atualiza U ou P com o valor da resposta (dependendo do valor de IFIP)

3 – Impressões dos resultados
- Para cada nó imprime os valores de U e P para cada ponto no tempo

Fim-Inicio