Bárbara de Melo Quintela

Implementação Computacional Paralela da Homogeneização por Expansão Assintótica para Análise de Problemas Mecânicos em 3D

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Modelagem Computacional.

Orientadora: Profa. D.Sc. Michèle Cristina Resende Farage

Coorientador: Prof. D.Sc. Marcelo Lobosco

Juiz de Fora

2011

Quintela, Bárbara de Melo.
Implementação computacional paralela da homogeneização por expansão assintótica para análise de problemas mecânicos em 3D / Bárbara de Melo Quintela. – 2011.
95 f. : il.

Dissertação (Mestrado em Modelagem Computacional)—Universidade Federal de Juiz de Fora, Juiz de Fora, 2011.

1. Computadores (computadores). 2. Modelos matemáticos. 3. Método dos elementos finitos. I. Título.

CDU 681.3.06

Bárbara de M. Quintela

Implementação Computacional Paralela da Homogeneização por Expansão Assintótica para Análise de Problemas Mecânicos em 3D

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional, da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Modelagem Computacional.

Aprovada em 31 de Janeiro de 2011.

BANCA EXAMINADORA

Profa. D.Sc. Michèle C. Resende Farage - Orientadora
Universidade Federal de Juiz de Fora

Prof. D.Sc. Marcelo Lobosco - Coorientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Elson Magalhães Toledo
Universidade Federal de Juiz de Fora

Profa. D.Sc. Regina Célia Paula Leal-Toledo
Universidade Federal Fluminense

Profa. D.Sc. Lucia Catabriga
Universidade Federal do Espírito Santo

Aos meus pais e minha irmã.

AGRADECIMENTOS

Primeiramente à Deus.

Aos orientadores, Profa. Michèle Farage e Prof. Marcelo Lobosco, sempre presentes e solidários e acima de tudo pelo constante exemplo de ética e conduta profissional. Aprendi muito mais do que as páginas que seguem poderiam conter.

À Rejane Lobosco pelo apoio e torcida.

Aos professores e colaboradores do Programa de Pós Graduação em Modelagem Computacional, especialmente à Natália, por tornar tantas coisas possíveis.

À todos os professores do Departamento de Mecânica Aplicada Computacional da Faculdade de Engenharia da UFJF. De forma especial à Professora Flávia Bastos pelo constante incentivo e ajuda com as malhas e ao Professor Elson Magalhães Toledo pelo interesse e disposição em ensinar sempre.

À todos os colegas do Mestrado em Modelagem Computacional que contribuíram direta ou indiretamente para o desenvolvimento deste trabalho: Anna Paula, Michelli, Gildo, Franciane, Victor, Ricardo (Cadim), Caroline, Bernardo, Eduardo, Roger, Alessandra e Vivi.

Ao professor da graduação Marco Antônio Araújo por expandir meus horizontes.

Aos meus familiares amados pelo apoio incondicional.

À todos os amigos pela paciência.

Às agências de fomento que financiaram esse trabalho: FAPEMIG, CAPES, CNPq.

À Universidade Federal de Juiz de Fora.

"O homem nasceu para aprender, aprender tanto quanto a vida lhe permita." (Guimarães Rosa)

"Você pode não saber quais resultados virão da sua ação mas, se não fizer nada, não haverá resultado algum." (Mahatma Gandhi)

RESUMO

A *H*omogeneização por *E*xpansão *A*ssintótica (HEA) é uma técnica multiescala empregada ao cálculo de propriedades efetivas de meios contínuos com estrutura periódica. As principais vantagens desta técnica são a redução do tamanho do problema a resolver e a possibilidade de se empregar uma propriedade homogeneizada que guarda informações da microestrutura heterogênea. Quando associada ao *M*étodo dos *E*lementos *F*initos (MEF), a HEA demanda o emprego de malhas que permitam a imposição de condições de contorno periódicas – sendo portanto necessário especificar tal particularidade quando da geração dos modelos em MEF. Tais modelos representam as células periódicas, que são volumes representativos do meio heterogêneo e, em alguns casos, apresentam uma complexidade geométrica e física que torna imprescindível o emprego de malhas com alto grau de refinamento – levando a um custo computacional significativo.

Este trabalho tem por objetivo a obtenção de um programa em Elementos Finitos para a aplicação da HEA à Elasticidade em 3D, empregando técnicas de programação paralela. Foram desenvolvidas versões do programa em 2D: uma sequencial em C e duas paralelas empregando OpenMP e CUDA. Foi implementado com sucesso o programa HEA3D em uma versão sequencial, em linguagem FORTRAN e uma paralela, empregando OpenMP. Para validação dos programas, foram analisadas células periódicas bifásicas e os resultados apresentaram boa concordância com valores experimentais e numéricos disponíveis na literatura. A versão paralela obteve expressivos ganhos de desempenho, com acelerações de desempenho de até 5.3 vezes em relação a versão sequencial.

Palavras-chave: Modelagem Multiescala. Elementos Finitos. Homogeneização. Programação Paralela.

ABSTRACT

The Asymptotic Expansion Homogenization (AEH) is a multiscale technique applied to estimate the effective properties of heterogeneous media with periodical structure. The main advantages of this technique are the reduction of the problem size to be solved and the ability to employ an homogenized property that keeps information from the heterogeneous microstructure. In association with the Finite Element Method (FEM), the AEH requires the application of periodic boundary conditions, which must be taken into account during the generation of FE meshes. Such models represent periodic cells, which are representative volumes for heterogeneous media and, in some cases, present a geometric and physics complexity that demands refined meshes, leading to a significant computational cost.

The aim of this work is to develop a parallel program that applies both FEM and AEH to estimate the elasticity properties of 3D bodies. A sequential version of the 2D program using C, and parallel versions using OpenMP and CUDA were implemented. A sequential version of the program, called HEA3D, was successfully implemented using FORTRAN. Also, a parallel version of the code was implemented using OpenMP. The validation of the codes consisted of comparisons of the numerical results obtained, with numerical and experimental data available in the literature, showing good agreement. Significant speedups were obtained by the parallel version of the code, achieving speedups up to 5.3 times over its sequential version.

Keywords: Multiscale Modelling. Finite Elements. Homogenization. Parallel Programming.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 Introdução

1.1 Motivação

O comportamento macroscópico de materiais heterogêneos pode ser fortemente afetado por características microestruturais, principalmente em relação aos aspectos de durabilidade e condições ambientais que levam a degradação progressiva [10, 11, 12]. Nesses casos, é conveniente considerar as características da microescala na modelagem de sistemas heterogêneos para simular adequadamente os efeitos da macroescala. Entretanto, para simular fenômenos físicos relacionados a meios heterogêneos, a representação detalhada das interações da microestrutura frequentemente resulta em problemas extremamente complexos, que podem tornar a solução impossível devido à enorme demanda de esforço computacional. Esse fato justifica a aplicação não só de computação paralela, como também de técnicas numéricas multiescala para estimar propriedades efetivas homogeneizadas, cujos valores são similares aos obtidos a partir de medidas experimentais [13, 14, 15, 16, 17].

Dentre as técnicas de modelagem multiescala mais utilizadas para a análise de meios heterogêneos, cita-se a Homogeneização por Expansão Assintótica [14, 18, 19], que é aplicada a meios que possuem estrutura periódica ou repetitiva. Basicamente, nestes casos é possível aproximar o meio original e heterogêneo por um meio equivalente, com propriedades efetivas ou homogeneizadas determinadas com base em um volume elementar que seja representativo da estrutura em questão e, por ser periódico, é denominado célula periódica. As análises em âmbito macroscópico da estrutura completa podem, então, ser efetuadas considerando-se o meio equivalente e homogeneizado, o que reduz de modo significativo a dimensão do problema. Destaca-se que as propriedades macroscópicas calculadas através de técnicas multiescala mantêm informações das escalas microscópicas, o que permite a recuperação de aspectos relacionados à microestrutura através de procedimentos conhecidos como técnicas de localização [19].

Um exemplo de materiais heterogêneos que podem ser aproximados por meios com estrutura periódica é o tecido ósseo. A Figura 1.1 é extraída de [1] em que a modelagem multiescala é aplicada para simular a regeneração óssea. A Figura mostra o corpo a

analisar (a), e dois tipos de volumes elementares representativos (d) e (e), que são empregados como células periódicas das quais se extraem as características efetivas do meio heterogêneo. A Figura 1.2.(a) mostra a geometria de um material compósito, que também apresenta periodicidade. Trata-se de uma estrutura composta por cilindros feitos de nanotubo de carbono – resultando em um meio com periodicidade em vários níveis [2].

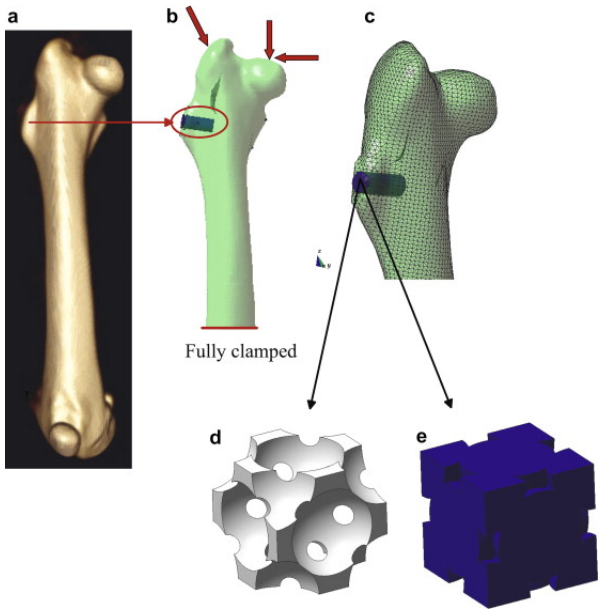


Figura 1.1: Análise multiescala da regeneração óssea. Extraída de [1]

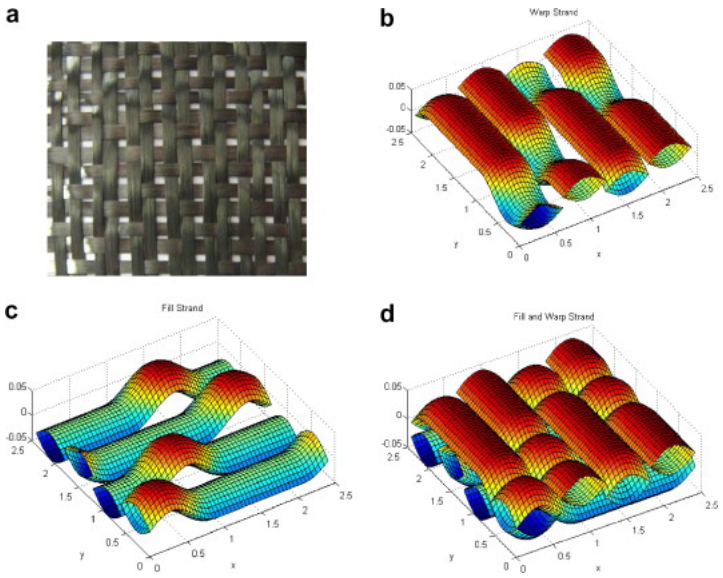


Figura 1.2: Configurações geométricas de um compósito feito de fibras de carbono. Extraída de [2]

As aplicações na Engenharia Civil são muito diversas. A estrutura representada na

Figura 1.3 é uma parede de alvenaria cujo comportamento foi analisado por [3] através de uma técnica multiescala. Existem trabalhos que representam o concreto através de células periódicas, como, por exemplo, o de Lee et al. [4] que utilizou a célula apresentada na Figura 1.4, onde as inclusões esféricas representam 4 tipos de agregados distintos.

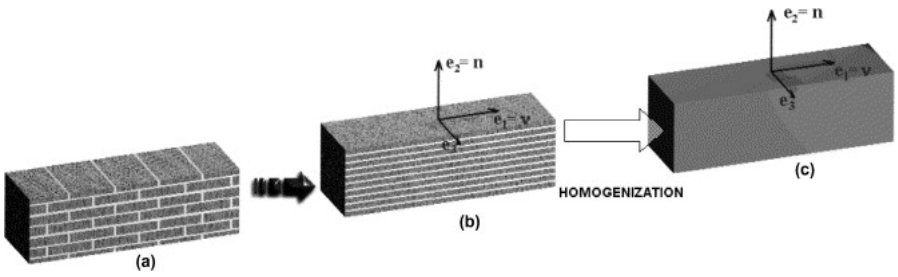


Figura 1.3: Análise multiescala de Parede de alvenaria. Extraída de [3]

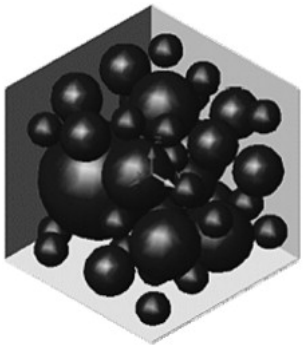


Figura 1.4: Célula periódica representativa de concreto com quatro tipos de agregados. Extraída de [4].

O cálculo de propriedades efetivas através da Homogeneização por Expansão Assintótica pode ser realizado com o auxílio de métodos numéricos, como o Método dos Elementos Finitos (MEF): o modelo a analisar é a célula periódica, que deve então ser discretizada em elementos finitos, gerando como resultado as propriedades efetivas do meio, que podem ser então empregadas em análises de âmbito macroscópico. Dispensa-se, assim, o emprego de malhas de elementos finitos muito refinadas na estrutura original - e os resultados podem ser obtidos de modo mais eficiente, demandando menos recursos computacionais do que os que seriam necessários na aplicação do MEF de modo direto, para analisar o meio não-homogêneo original. Ainda assim, alguns tipos de geometrias demandam um maior refinamento na malha de elementos finitos para garantir resultados válidos. O maior refinamento, por sua vez, implica na necessidade de resolução de um sistema de equações maior. Nestes casos, a melhor alternativa para reduzir o tempo necessário para o processamento é aplicar técnicas de computação

paralela. Mas mesmo nos casos em que o tempo de processamento não seja um fator determinante, o emprego de computação paralela ainda é importante em virtude do melhor aproveitamento dos recursos computacionais disponíveis: grande parte dos processadores hoje fabricados possuem capacidade para realizar processamento paralelo; não utilizar toda a sua capacidade computacional representa um desperdício de recursos.

1.2 Descrição Geral e Objetivos

Este trabalho teve início em um estudo preliminar desenvolvido no NUMEC (Núcleo de Pesquisa em Métodos Computacionais em Engenharia - UFJF) em que a HEA foi implementada utilizando o ambiente de programação MATLAB® para a obtenção de propriedades mecânicas efetivas de estruturas periódicas planas [20, 21]. A partir dessa versão inicial foi desenvolvida uma nova versão utilizando a linguagem C [22], para melhor compreensão do modelo mecânico e como base para a posterior generalização para 3D. Foram empregadas ainda técnicas de programação paralela gerando duas novas versões paralelas – OpenMP e CUDA – visando necessidades futuras de avaliação de domínios mais complexos [23, 24].

O objetivo deste trabalho é a implementação da técnica da HEA aplicada à Elasticidade, para obtenção de propriedades mecânicas efetivas de meios com estrutura periódica, empregando o MEF e técnicas de processamento paralelo. Para este fim, procedeu-se primeiramente a adaptação de um programa-base desenvolvido em linguagem FORTRAN, para a resolução do problema da HEA em domínios com geometria hexaédrica empregando o MEF. A partir dessa versão sequencial, foi desenvolvida uma versão paralela para multiprocessadores empregando OpenMP.

Para a validação dos programas, foram realizadas comparações com resultados disponíveis na literatura, apresentando uma boa concordância. A análise do desempenho entre as versões sequencial e paralela indica que o objetivo de redução no tempo de computação necessário para a obtenção das propriedades mecânicas foi alcançado: ganhos de até 5.3 vezes foram obtidos quando 6 processadores foram utilizados.

1.3 Escopo

Esta dissertação foi organizada da seguinte forma:

O Capítulo 2 apresenta a formulação da HEA aplicada à problemas de Elasticidade e descreve a aplicação do MEF à HEA.

O Capítulo 3 trata de técnicas de programação paralela, apresentando de forma breve conceitos de arquitetura e modelos de programação paralela. Aborda em particular OpenMP e CUDA, que foram empregados no presente trabalho.

As implementações das versões sequenciais e paralelas são descritas nos Capítulo 4 e 5. O Capítulo 4 trata das funcionalidades do programa para análise bidimensional e no Capítulo 5 são apresentadas as principais rotinas do programa 3D. Ambos apresentam aspectos importantes dos programas desenvolvidos, como a imposição de condições de contorno periódicas e as estratégias utilizadas para a programação paralela.

Em seguida, no Capítulo 6, são apresentados os exemplos de aplicação e as análises de desempenho das versões desenvolvidas dos programas HEA2D e HEA3D.

Por fim, o Capítulo 7 traz as considerações finais e perspectivas futuras.

2 Homogeneização por Expansão Assintótica

2.1 Introdução

Técnicas de modelagem multiescala têm sido empregadas há algumas décadas, com bastante êxito, na simulação dos mais variados problemas heterogêneos, tais como: sistemas biológicos, meios porosos e materiais compósitos [13, 17, 25]. Estes problemas caracterizam-se pela heterogeneidade, apresentando composições em escalas com dimensões hierárquicas cujo tamanho característico pode variar de ângstrons a quilômetros. A descrição detalhada de meios heterogêneos pode tornar a análise de tais problemas difícil ou até impossível. Para tanto determina-se, quando possível, um meio macroscópico equivalente conhecido como homogeneizado [26].

Este capítulo tem por objetivo apresentar a técnica da *H*omogeneização por *E*xpansão *A*ssintótica (HEA) aplicada à Elasticidade, bem como a utilização do *M*étodo dos *E*lementos *F*initos (MEF) para a determinação de propriedades efetivas de meios periódicos.

A HEA tem se mostrado uma excelente metodologia para modelar fenômenos físicos em meios que possuam microestrutura periódica [27]. A teoria da homogeneização se baseia em encontrar equações diferenciais parciais homogeneizadas que descrevam os processos físicos que ocorrem nos meios heterogêneos quando a escala das heterogeneidades tende a zero [28]. O que se espera, dessa forma, é adotar propriedades macroscópicas que considerem a influência da microestrutura, através de procedimentos semelhantes à determinação de médias.

Admite-se a existência de pelo menos duas escalas distintas, associadas a fenômenos microscópicos e macroscópicos [18, 29], as quais podem ser desacopladas. Na literatura [18, 19], a notação frequentemente utilizada considera x como o referencial da macroescala, que caracteriza o sistema global, e y como o referencial da microescala associada às heterogeneidades da microestrutura analisada. Em determinadas aplicações, como por exemplo tecidos biológicos, podem existir mais do que duas escalas e, com isso, há a

necessidade de expandir a nomenclatura com índices que possam representar as demais escalas [29].

Considerando um problema com duas escalas, estas são relacionadas por um parâmetro ϵ real positivo, que é a razão entre as dimensões características da microescala y e macroescala x, dada pela relação 2.1 [19]:

$$y = x/\epsilon. \tag{2.1}$$

A HEA é uma técnica de perturbação baseada na expansão em séries assintóticas em torno de ϵ. A incógnita do problema depende das duas variáveis relativas às escalas, global x e local y. A expansão assintótica aproxima a grandeza que está sendo homogeneizada nas escalas adicionando perturbações causadas pela periodicidade das heterogeneidades microestruturais [27, 29].

Considera-se o caso limite em que a dimensão do período da microescala é muito pequena em relação às outras dimensões que aparecem no problema, sendo possível considerar que a solução do problema em questão é aproximadamente igual à solução de um material *homogeneizado* [18]. Como as escalas são desacopladas, o esforço computacional do problema é reduzido [19], de forma que, ao invés de analisar toda hierarquia do material de uma só vez, opta-se por analisá-la por partes *representativas* da estrutura completa. Estas partes são conhecidas como volume elementar representativo (VER), e devem representar de forma estatística a microestrutura do material que está sob análise [29]. O VER deve ainda ser pequeno em relação ao volume macroscópico e, quando se trata de um meio periódico, reduz-se a uma célula unitária periódica [26]. A Figura 2.1 apresenta uma estrutura periódica de modo esquemático.

A base da aproximação é a premissa de que o parâmetro que relaciona as escalas tende a zero, indicando que o número de células periódicas tende a infinito, conforme representado na Figura 2.2. No limite em que $\epsilon \to 0$, a estrutura não-homogênea é aproximada por uma homogênea.

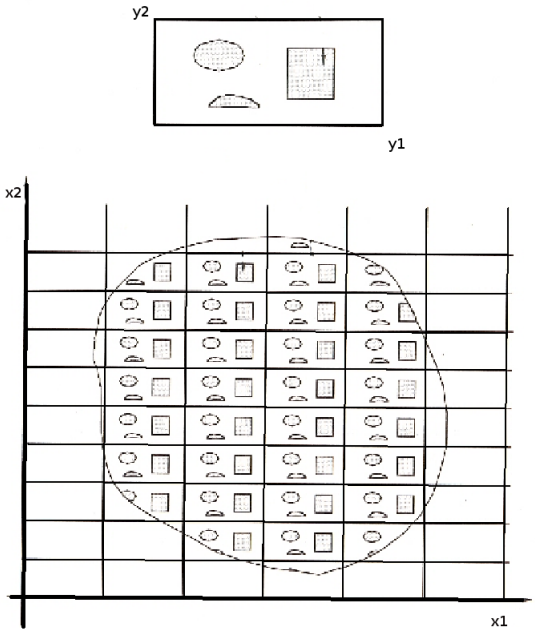


Figura 2.1: Representação de uma estrutura heterogênea. Adaptada de [5]

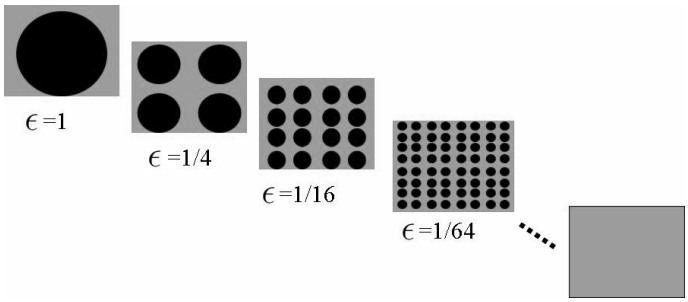


Figura 2.2: Representação da aproximação de uma estrutura periódica não-homogênea por uma homogênea equivalente quando $\epsilon \to 0$

2.2 Homogeneização por Expansão Assintótica Aplicada à Elasticidade

Considera-se um material cuja microestrutura é composta por múltiplas fases, cujas orientações e formas devem ser tais que estejam repetida e periodicamente distribuídas nas três dimensões na microestrutura do material [18, 19]. As propriedades elásticas do material periódico são definidas pela relação 2.2:

$$D^{\epsilon}_{ijkl} = D_{ijkl}\left(\frac{x}{\epsilon}\right), \tag{2.2}$$

em que $(\ )^\epsilon$ denota os valores relacionados à estrutura real não-homogênea e a função D_{ijkl} representa as variações das propriedades do material na microestrutura heterogênea Y. Isso significa dizer que o volume está representado no sistema de coordenadas x_i^ϵ e as variações que ocorrem na microestrutura são modeladas por D_{ijkl}^ϵ. No problema homogeneizado, as mesmas variações ocorrem de forma *aumentada*, somente sobre a célula unitária periódica Y [19]. A célula pode ser repetida no espaço para gerar o mesmo sistema descrito por D_{ijkl}^ϵ.

O problema elástico-linear é descrito pela Equação de equilíbrio (2.3), condições de contorno (Equações 2.4 e 2.5), relação deformação-deslocamento (Equação 2.6) e relação constitutiva (Equação 2.7):

$$\frac{\partial \sigma_{ij}^\epsilon}{\partial x_j^\epsilon} + f_i = 0 \text{ em } \Omega; \tag{2.3}$$

$$u_i^\epsilon = 0 \text{ em } \partial_1\Omega; \tag{2.4}$$

$$\sigma_i^\epsilon n_j = F_i \text{ em } \partial_2\Omega; \tag{2.5}$$

$$\varepsilon_{ij}(u^\epsilon) = \frac{1}{2}\left(\frac{\partial u_i^\epsilon}{\partial x_j^\epsilon} + \frac{\partial u_j^\epsilon}{\partial x_i^\epsilon}\right); \tag{2.6}$$

$$\sigma_{ij}^\epsilon = D_{ijkl}^\epsilon \varepsilon_{kl}(u^\epsilon), \tag{2.7}$$

em que o parâmetro de escala ϵ identifica as quantidades relacionadas ao comportamento do material heterogêneo; σ_{ij}^ϵ é o termo ij do tensor de tensões internas e f_i é a força de volume no domínio Ω; u_i^ϵ é o deslocamento na direção i; n_j é o vetor normal ao contorno $\partial\Omega$; F_i é a força por unidade de área no contorno $\partial_2\Omega$ e ε_{ij} é o termo ij do tensor de deformações.

Assumindo a existência de duas escalas distintas associadas ao comportamento do material, o campo de deslocamento, que é a incógnita do problema, é aproximado por uma série assintótica em ϵ, dada pela Equação (2.8), em que $u_i^{(0)}$ é o deslocamento na escala macro e $u_i^{(1)}, u_i^{(2)}, \ldots$ são os deslocamentos periódicos nas escalas inferiores.

$$u_i^\epsilon(x^\epsilon) = u_i^{(0)}(x,y) + \epsilon u_i^{(1)}(x,y) + \epsilon^2 u_i^{(2)}(x,y) + \ldots \tag{2.8}$$

Como o meio heterogêneo é representado por dois sistemas de coordenadas (x e $y = x/\epsilon$), as derivadas originalmente em x^ϵ são expandidas pela regra da cadeia, dada pela

Equação 2.9:

$$\frac{\partial}{\partial x_i^\epsilon} = \frac{\partial}{\partial x_i} + \frac{1}{\epsilon}\frac{\partial}{\partial y_j}. \tag{2.9}$$

Substituindo a expansão assintótica dos deslocamentos (Equação 2.8) na relação deformação-deslocamento (Equação 2.6) chega-se a Equação (2.10):

$$2\varepsilon_{ij}(u^\epsilon) = \left(\frac{\partial u_i^{(0)}}{\partial x_j^\epsilon} + \epsilon\frac{\partial u_i^{(1)}}{\partial x_j^\epsilon} + \epsilon^2\frac{\partial u_i^{(2)}}{\partial x_j^\epsilon} + \ldots \frac{\partial u_j^{(0)}}{\partial x_i^\epsilon} + \epsilon\frac{\partial u_j^{(1)}}{\partial x_i^\epsilon} + \epsilon^2\frac{\partial u_j^{(2)}}{\partial x_i^\epsilon} + \ldots\right) \tag{2.10}$$

Empregando a regra da cadeia (Equação 2.9) e agrupando o lado direito da Equação (2.10) em potências de ϵ, tem-se:

$$\begin{aligned} 2\varepsilon_{ij}(u^\epsilon) &= \epsilon^{-1}\left(\frac{\partial u_i^{(0)}}{\partial y_j} + \frac{\partial u_j^{(0)}}{\partial y_i}\right) + \epsilon^0\left(\frac{\partial u_i^{(0)}}{\partial x_j} + \frac{\partial u_i^{(1)}}{\partial y_j} + \frac{\partial u_j^{(0)}}{\partial x_i} + \frac{\partial u_j^{(1)}}{\partial y_i}\right) + \\ &+ \epsilon^1\left(\frac{\partial u_i^{(1)}}{\partial x_j} + \frac{\partial u_j^{(1)}}{\partial x_i} + \frac{\partial u_i^{(2)}}{\partial y_j} + \frac{\partial u_j^{(2)}}{\partial y_i}\right) + \epsilon^2\left(\frac{\partial u_i^{(2)}}{\partial x_j} + \frac{\partial u_j^{(2)}}{\partial x_i} + \ldots\right) + \ldots \end{aligned} \tag{2.11}$$

que, substituído em (2.7), leva a:

$$\begin{aligned} \sigma_{ij}^\epsilon &= \epsilon^{-1}D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial y_l} + \frac{\partial u_l^{(0)}}{\partial y_k}\right) + \epsilon^0 D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial x_l} + \frac{\partial u_k^{(1)}}{\partial y_l} + \frac{\partial u_l^{(0)}}{\partial x_k} + \frac{\partial u_l^{(1)}}{\partial y_k}\right) \\ &+ \epsilon^1 D_{ijkl}\left(\frac{\partial u_k^{(1)}}{\partial x_l} + \frac{\partial u_l^{(1)}}{\partial x_k} + \frac{\partial u_k^{(2)}}{\partial y_l} + \frac{\partial u_l^{(2)}}{\partial y_k}\right) + \epsilon^2 D_{ijkl}\left(\frac{\partial u_k^{(2)}}{\partial x_l} + \ldots\right) + \\ &+ \ldots \end{aligned} \tag{2.12}$$

O primeiro termo da Equação de equilíbrio (2.3) é $(\partial\sigma_{ij}^\epsilon/\partial x_j^\epsilon)$, em que se aplica a regra da cadeia:

$$\frac{\partial\sigma_{ij}^\epsilon}{\partial x_j^\epsilon} = \frac{\partial\sigma_{ij}^\epsilon}{\partial x_j} + \frac{1}{\epsilon}\frac{\partial\sigma_{ij}^\epsilon}{\partial y_j}, \tag{2.13}$$

e substituindo σ_{ij}^ϵ na Equação (2.13) pela Equação (2.12), obtém-se a Equação (2.14), em

que o lado direito já se encontra organizado em potências de ϵ:

$$\begin{aligned}
\frac{\partial \sigma_{ij}^{\epsilon}}{\partial x_j^{\epsilon}} &= \epsilon^{-2}\left[\frac{\partial}{\partial y_j}\overbrace{D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial y_l}\right)}^{\sigma^{(0)}}\right] + \\
&+ \epsilon^{-1}\left[\frac{\partial}{\partial x_j}\overbrace{D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial y_l}\right)}^{\sigma^{(0)}} + \frac{\partial}{\partial y_j}\overbrace{D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial x_l} + \frac{\partial u_k^{(1)}}{\partial y_l}\right)}^{\sigma^{(1)}}\right] + \\
&+ \epsilon^{0}\left[\frac{\partial}{\partial x_j}\overbrace{D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial x_l} + \frac{\partial u_k^{(1)}}{\partial y_l}\right)}^{\sigma^{(1)}} + \frac{\partial}{\partial y_j}\overbrace{D_{ijkl}\left(\frac{\partial u_k^{(1)}}{\partial x_l} + \frac{\partial u_k^{(2)}}{\partial y_l}\right)}^{\sigma^{(2)}}\right] + \\
&+ \epsilon^{1}\left[\frac{\partial}{\partial x_j}D_{ijkl}\left(\frac{\partial u_k^{(1)}}{\partial x_l} + \frac{\partial u_k^{(2)}}{\partial y_l}\right) + \frac{\partial}{\partial y_j}D_{ijkl}\left(\frac{\partial u_k^{(2)}}{\partial x_l} + \ldots\right)\right] + \\
&+ \epsilon^{2}\left[\frac{\partial}{\partial x_j}D_{ijkl}\left(\frac{\partial u_k^{(2)}}{\partial x_l} + \ldots\right)\right] + \ldots
\end{aligned} \tag{2.14}$$

A equação de equilíbrio (2.3) passa então a ser escrita como:

$$\epsilon^{-2}\frac{\partial \sigma_{ij}^{(0)}}{\partial y_j} + \epsilon^{-1}\left(\frac{\partial \sigma_{ij}^{(0)}}{\partial x_j} + \frac{\partial \sigma_{ij}^{(1)}}{\partial y_j}\right) + \epsilon^{0}\left(\frac{\partial \sigma_{ij}^{(1)}}{\partial x_j} + \frac{\partial \sigma_{ij}^{(2)}}{\partial y_j} + f_i\right) + \ldots = 0, \tag{2.15}$$

sendo $\sigma_{ij}^{(0)}$, $\sigma_{ij}^{(1)}$ e $\sigma_{ij}^{(2)}$ termos indicados na equação 2.14.

Admitindo válida a hipótese de que o meio heterogêneo tende para o meio homogeneizado quando $\epsilon \to 0$, ocorre que nas expressões aqui apresentadas todos os termos com potência negativa de ϵ devem ter coeficientes nulos. Então, tomando a Equação (2.12) tem-se:

$$D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial y_l}\right) = 0, \tag{2.16}$$

daí:

$$\frac{\partial u_k^{(0)}}{\partial y_l} = 0. \tag{2.17}$$

indicando que o deslocamento macroscópico $u^{(0)}$ não depende de y, sendo constante na

escala microscópica Y. Consequentemente, tem-se:

$$\sigma_{ij}^{(0)} = D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial y_l}\right) = 0. \tag{2.18}$$

Tomando a segunda parcela da equação 2.15, chega-se à equação 2.19:

$$\frac{\partial \sigma_{ij}^{(0)}}{\partial x_j} + \frac{\partial \sigma_{ij}^{(1)}}{\partial y_j} = \frac{\partial}{\partial x_j}\overbrace{D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial y_l}\right)}^{\sigma^{(0)}=0} + \frac{\partial}{\partial y_j} D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial x_l} + \frac{\partial u_k^{(1)}}{\partial y_l}\right) = 0, \tag{2.19}$$

que se reduz à equação 2.20:

$$\frac{\partial}{\partial y_j} D_{ijkl}\left(\frac{\partial u_k^{(0)}}{\partial x_l} + \frac{\partial u_k^{(1)}}{\partial y_l}\right) = 0, \tag{2.20}$$

que relaciona o termo microscópico $u^{(1)}$ com o termo macroscópico $u^{(0)}$ e pode ser reescrita na forma da equação 2.21:

$$-\frac{\partial}{\partial y_j}\left(D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l}\right) = \frac{\partial}{\partial y_j}\left(D_{ijkl}\frac{\partial u_k^{(0)}}{\partial x_l}\right). \tag{2.21}$$

Multiplicando-se ambos os lados da Equação (2.21) por uma função teste v e integrando em Y obtém-se a Equação 2.22:

$$-\int_Y \frac{\partial}{\partial y_j}\left(D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l}\right) v dy = \int_Y \frac{\partial}{\partial y_j}\left(D_{ijkl}\frac{\partial u_k^{(0)}}{\partial x_l}\right) v dy. \tag{2.22}$$

Empregando-se integração por partes dada pela Equação (2.23):

$$\int_\Omega U dV = UV\rceil_\Gamma - \int_\Omega V dU, \tag{2.23}$$

ao lado esquerdo da Equação (2.22), fazendo $V = \frac{D_{ijkl}\partial u_k^{(1)}}{\partial y_l}$ e $U = v$, chega-se à Equação (2.24):

$$\int_Y \frac{\partial}{\partial y_l} D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l} v dy = v D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l}\rceil_\Gamma - \int_Y D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l}\frac{\partial v}{\partial y_l} dy \tag{2.24}$$

em que o termo $v D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l}\rceil_\Gamma = 0$, considerando-se a periodicidade no contorno.

Tomando o lado direito da Equação 2.21, aplica-se a regra da cadeia:

$$\frac{\partial}{\partial y_j}\left(D_{ijkl}\frac{\partial u_k^{(0)}}{\partial x_l}\right) = \frac{\partial D_{ijkl}}{\partial y_j}\frac{\partial u_k^{(0)}}{\partial x_l} + D_{ijkl}\frac{\partial^2 u_k^{(0)}}{\partial x_l \partial y_j}, \tag{2.25}$$

em que o termo $D_{ijkl}\frac{\partial^2 u_k^{(0)}}{\partial x_l \partial y_j} = 0$. Daí:

$$\int_Y \frac{\partial}{\partial y_j}\left(D_{ijkl}\frac{\partial u_k^{(0)}}{\partial x_l}\right) v dy = \int_Y \frac{\partial D_{ijkl}}{\partial y_j}\frac{\partial u_k^{(0)}}{\partial x_l} v dy, \tag{2.26}$$

em que $\frac{\partial u_k^{(0)}}{\partial x_l}$ é constante em Y, e, portanto, o lado direito da Equação (2.26) pode ser reescrita como (2.27):

$$\frac{\partial u_k^{(0)}}{\partial x_l}\int_Y \frac{\partial D_{ijkl}}{\partial y_j} v dy. \tag{2.27}$$

A Equação (2.21) é então reescrita na forma variacional (2.28):

$$\int_Y D_{ijkl}\frac{\partial u_k^{(1)}}{\partial y_l}\frac{\partial v}{\partial y_j} dy = \frac{\partial u_k^{(0)}}{\partial x_l}\int_Y \frac{\partial D_{ijkl}}{\partial y_j} v dy, \tag{2.28}$$

em que a função teste $v = 0$ em $\partial\Omega$.

Como não é conveniente resolver $u_i^{(1)}$ na célula periódica para toda variação de $u_i^{(0)}$, para desacoplar as escalas, emprega-se uma solução indireta [18, 19] que consiste em adotar:

$$u^{(1)} = \chi^{ij}\frac{\partial u^{(0)}}{\partial x} + \tilde{u}^{(1)}(x), \tag{2.29}$$

em que $\tilde{u}^{(1)}(x)$ é uma constante de integração e χ^{kl} é uma função periódica em y, chamada de corretor elástico ou função característica [19, 30].

Substituindo (2.29) na forma variacional (2.28), temos que χ^{ij} é a solução do problema variacional auxiliar dado por:

$$\int_Y D_{ijkl}\frac{\partial \chi_k^{ij}}{\partial y_l}\frac{\partial v_i}{\partial y_j} dy = \int_Y v_j \frac{\partial D_{ijkl}}{\partial y_i} dY. \tag{2.30}$$

Para resolver o problema na macroescala, é necessário determinar o tensor de propriedades elásticas que relaciona as tensões e deformações homogeneizadas em x. Com

este fim, o operador de média de uma função periódica em y é definido em 2.31 [26].

$$\langle . \rangle = \frac{1}{Y} \int_Y (.) dy. \tag{2.31}$$

Este operador de média é então aplicado a (2.15):

$$\langle \frac{\partial \sigma_{ij}^{(1)}}{\partial x_j} \rangle_Y + \langle \frac{\partial \sigma_{ij}^{(2)}}{\partial y_j} \rangle_Y + \langle f_i \rangle_Y = 0. \tag{2.32}$$

Empregando o teorema da divergência e levando em consideração a periodicidade em y, o segundo termo da Equação (2.32):

$$\langle \frac{\partial \sigma_{ij}^{(2)}}{\partial y_j} \rangle_Y = \frac{1}{|Y|} \int_Y \frac{\partial \sigma_{ij}^{(2)}}{\partial y_j} dY = \frac{1}{|Y|} \int_\Gamma \sigma_{ij}^{(2)} n_j d\Gamma = 0. \tag{2.33}$$

Considerando ainda que o operador de média tem a propriedade de comutação com os operadores $\partial(.)/\partial x_j$ e que f_i é constante em Y [27], a Equação 2.32 se reduz a:

$$\frac{\partial \langle \sigma_{ij}^{(1)} \rangle_Y}{\partial x_j} + f_i = 0. \tag{2.34}$$

A Equação (2.34) representa a Equação diferencial de equilíbrio homogeneizada da macroescala. Substituindo a Equação (2.29) em $\sigma_{ij}^{(1)}$, tem-se:

$$\sigma_{ij}^{(1)} = D_{ijkl} \left(\frac{\partial u_k^{(0)}}{\partial x_l} + \frac{\partial u_k^{(1)}}{\partial y_l} \right) = D_{ijkl} \left(I_{ij}^{kl} - \frac{\partial \chi^{kl}}{\partial y_l} \right) \frac{\partial u_{(0)}}{\partial x_k}. \tag{2.35}$$

Aplicando o operador de média e lembrando que $u^{(0)}$ é constante em y, tem-se:

$$\langle \sigma_{ij}^{(1)} \rangle_Y = D_{ijkl}^h \frac{\partial u^{(0)}}{\partial x_k}. \tag{2.36}$$

Daí, observando as Equações (2.35) e (2.36), pode-se definir o tensor de propriedades elásticas homogeneizadas D_{ijkl}^h, que relaciona $\langle \sigma_{ij}^{(1)} \rangle_Y$ com $\frac{\partial u^{(0)}}{\partial x_k}$, como:

$$D_{ijkl}^h = \frac{1}{Y} \int_Y D_{ijkl}(y) \left[I_{ij}^{kl} - \frac{\partial \chi^{kl}}{\partial y_l} \right] dY. \tag{2.37}$$

Em suma, a determinação do tensor de propriedades homogeneizadas D^h é denominada resolução do problema da microescala. Dadas as características geométricas e a

variação de propriedades mecânicas em uma célula periódica, pode-se obter D^h que, então, é empregado na resolução de problemas macroscópicos, considerando-se o meio homogeneizado equivalente ao meio original, não-homogêneo.

O procedimento consiste basicamente em: resolver a Equação (2.30), obtendo o corretor elástico χ, que é então aplicado na Equação (2.37) para o cálculo de D^h. Na Equação (2.30), o lado direito, aqui denominado F_{ij}:

$$F_{ij} = \int_Y v_j \frac{\partial D_{ijkl}}{\partial y_i} dY, \tag{2.38}$$

é um termo forçado, análogo a uma função de carregamento do problema clássico de elasticidade. Para problemas em $2D$, χ e F são matrizes com 3 colunas e em $3D$ são matrizes com 6 colunas, uma vez que na Equação (2.38) aplica-se o operador diferencial ao tensor elástico D.

2.3 Método dos Elementos Finitos Aplicado à HEA

2.3.1 Cálculo do Corretor Elástico

O corretor elástico χ, obtido da resolução da Equação (2.30), é composto por n vetores sendo n o número de colunas da matriz elástica a determinar. Estes vetores são análogos a vetores de deslocamento e geram as autodeformações da célula periódica. Empregando o Método dos Elementos Finitos para a resolução do problema, a Equação (2.30) é aqui reescrita como:

$$\sum_{i=0}^{nelm} \int_{Y^e} B^T D B dY \chi = \overbrace{\sum_{i=0}^{nelm} \int_{Y^e} B^T D dY}^{F^D}, \tag{2.39}$$

em que B é o operador diferencial, D é o tensor elástico, Y_e é o volume e $nelm$ é o número total de elementos finitos. F^D é então descrito na forma de cargas nodais equivalentes e a função χ é obtida a partir destas aproximações.

Sabendo que o corretor χ é uma matriz e não um vetor e o segundo termo da Equação (2.39) consiste na matriz F^D, os resultados são n (3 em $2D$ ou 6 em $3D$) soluções diferentes que compõem as n colunas de χ, sendo que cada uma pode ser representada na célula periódica como um modo de autodeformação. A definição da matriz F^D (Equações 2.30 e 2.39) mostra que os vetores de "força" são montados a partir da integração do gradiente

das propriedades elásticas dos materiais que formam o meio compósito [27].

2.3.2 Condições de Contorno Periódicas

A solução da Equação (2.39) (χ_k^{ij}) é obtida empregando-se condições de contorno periódicas de $u_k^{(1)}(x,y), k = 1,2,3$. Wang [6] explica as condições de contorno periódicas através do seguinte exemplo: se a célula periódica for um paralelepípedo, como o mostrado na figura 2.3, as 6 faces podem ser descritas pela Equação (2.40):

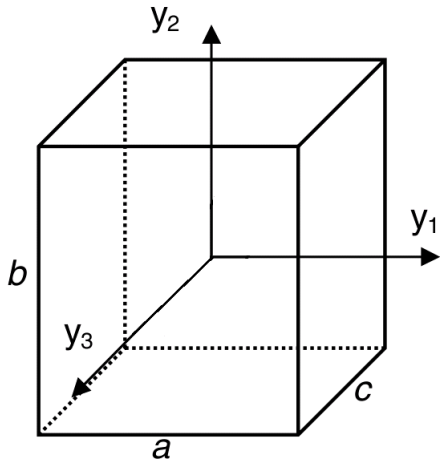


Figura 2.3: Célula unitária representativa hexaédrica. Adaptada de [6]

$$
\begin{aligned}
(y_1,y_2,y_3)|_{face3}^{face1} &= \left(\pm\frac{a}{2},y_2,y_3\right), \\
(y_1,y_2,y_3)|_{face4}^{face2} &= \left(y_1,\pm\frac{b}{2},y_3\right), \\
(y_1,y_2,y_3)|_{face6}^{face5} &= \left(y_1,y_2,\pm\frac{c}{2}\right).
\end{aligned}
\tag{2.40}
$$

sendo a, b e c os comprimentos das arestas da célula nas direções y_1, y_2 e y_3, respectivamente. As condições de contorno periódicas de $u_k^{(1)}$ (Equação 2.29) requerem, portanto, as seguintes igualdades:

$$
\begin{aligned}
u_k^{(1)}|_{face1} &= u_k^{(1)}|_{face3}, \\
u_k^{(1)}|_{face2} &= u_k^{(1)}|_{face4}, \\
u_k^{(1)}|_{face5} &= u_k^{(1)}|_{face6}.
\end{aligned}
\tag{2.41}
$$

Substitui-se a Equação (2.29) em (2.41) e, como $\tilde{u}_k^{(1)}$ não depende de y, pode-se reescrever a Equação (2.41) como:

$$\begin{aligned} \chi_k^{ij}|_{face1} &= \chi_k^{ij}|_{face3}, \\ \chi_k^{ij}|_{face2} &= \chi_k^{ij}|_{face4}, \\ \chi_k^{ij}|_{face5} &= \chi_k^{ij}|_{face6}. \end{aligned} \tag{2.42}$$

adotando $\tilde{u}_k^{(1)} = 0$.

Dessa forma, a função característica χ pode ser completamente determinada a partir de (2.39) e (2.42). Para evitar movimento de corpo rígido, os deslocamentos e rotações ao menos de um ponto arbitrário da célula unitária devem ser restringidos [7].

2.3.3 Matriz Elástica Homogeneizada

Uma vez calculados os vetores que compõem χ, determina-se a matriz elástica homogeneizada D^h aplicando a Equação (2.37), aqui reescrita na forma discreta:

$$D^h = \sum_{k=1}^{nelm} \frac{Y^k}{Y} D^k \left(I - B^k \chi^k\right) \tag{2.43}$$

em que $nelm$ é o número de elementos na célula unitária; Y^k é o volume do elemento k; Y é o volume total da célula; I é a matriz identidade; B^k é o operador diferencial e χ^k é a matriz de valores de χ nodais associados ao elemento k. Observa-se que no caso de $\chi = 0$, a Equação (2.43) passa a representar a média aritmética das propriedades elásticas dos elementos finitos que compõem o modelo da célula.

A partir das discretizações apresentadas nesta seção, é possível realizar a implementação computacional da homogeneização por expansão assintótica. Portanto, pode-se resumir o procedimento HEA aplicada à elasticidade através do MEF nos seguintes passos [19]:

1. Identificação da célula periódica que representa o problema;

2. Discretização da célula periódica em elementos finitos;

3. Resolução do sistema de Equações descrito pela expressão (2.39), calculando os vetores que compõem χ;

4. Efetuar a operação de média descrita pela Equação (2.43), obtendo o tensor homogeneizado D^h.

Ainda que o emprego de técnicas multiescala torne mais simples a análise de problemas heterogêneos, existem casos em que a composição geométrica e física da célula periódica apresenta tal complexidade que se faz necessário o emprego de malhas de elementos finitos muito refinadas. Daí a justificativa para se empregar a programação paralela, que é o tema do próximo capítulo.

3 Programação Paralela

3.1 Introdução

Alguns tipos de geometrias representadas pelas células periódicas demandam um maior refinamento na malha de elementos finitos para garantir resultados válidos na determinação da matriz elástica homogeneizada. Esse refinamento aumenta as dimensões do problema tratado e pode levar a um sistema de equações a ser resolvido muito extenso e caro computacionalmente. Um exemplo é a célula mostrada na Figura 3.1. A presença de várias inclusões distribuídas no volume demanda malhas refinadas para que o modelo gere um resultado satisfatório. Tal fato justifica a busca por mecanismos que possam reduzir o tempo de computação necessário para determinar a matriz elástica homogeneizada.

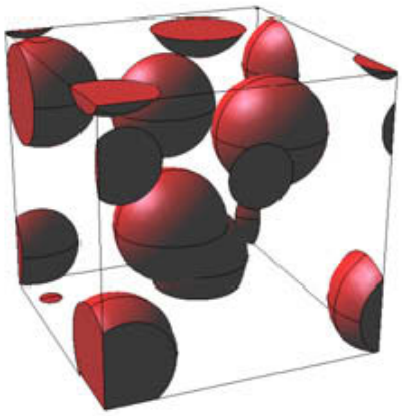


Figura 3.1: Célula periódica com várias inclusões. Extraída de [7].

Tradicionalmente, os aplicativos são desenvolvidos de forma a terem suas instruções executadas sequencialmente em um único núcleo de processamento. No entanto, quando há a necessidade de reduzir o tempo de computação de um aplicativo, uma possível solução é a divisão do problema computacional em partes que possam ser executadas simultaneamente em núcleos distintos. Cada uma dessas partes é chamada de processo ou de fluxo de execução (*thread*), conforme será apresentado a seguir neste capítulo. A sobreposição na execução dos distintos processos leva a uma redução no tempo total necessário para a computação de uma aplicação. Esse mecanismo de implementação que permite a sobreposição da execução de partes de uma aplicação é conhecida como

programação paralela[31]. A forma de se implementar o paralelismo pode variar, e muitas vezes a escolha da melhor abordagem a ser empregada está intimamente ligada a arquitetura de *hardware* disponível.

Neste capítulo serão abordadas de modo introdutório as principais arquiteturas para computação paralela, assim como as principais ferramentas para o desenvolvimento de aplicações paralelas – o capítulo discorre particularmente sobre as ferramentas utilizadas neste trabalho: OpenMP e CUDA.

3.2 Arquiteturas de Computação Paralela

Existem basicamente dois tipos de computadores paralelos[31]: o multiprocessador e o multicomputador. O primeiro é composto por um único computador que possui vários núcleos de processamento, enquanto o segundo é composto por vários computadores independentes, que são interconectados de modo a formar uma única plataforma de computação de alto desempenho. Nesta seção são apresentas com mais detalhes cada uma dessas arquiteturas. Uma terceira alternativa é empregar um ambiente híbrido, resultante do emprego simultâneo das duas arquiteturas anteriores.

3.2.1 Multicomputador

O termo multicomputadores se refere a sistemas de memória distribuída em que vários computadores são interconectados por uma rede de comunicação, conforme o esquema mostrado na Figura 3.2. Cada computador possui seu próprio núcleo de processamento e sua própria memória, que só pode ser acessada diretamente pelo núcleo ao qual ela pertence. Como cada computador possui uma memória local única, opera de forma independente e cada mudança realizada na memória local não surte efeito nas memórias dos demais computadores.

Durante a divisão do problema computacional em partes que possam ser executadas simultaneamente, é comum a necessidade de estabelecer algum mecanismo de troca de dados entre as partes, de modo que a computação possa chegar a resultados corretos. Em um ambiente multicomputador tais dados podem estar localizados na memória privativa de outro computador. A rede de comunicação tem um papel importante neste caso, visto que é utilizada para a troca de dados entre os computadores. Quanto maior o tempo

necessário para que os dados sejam trocados, maior será o impacto no tempo total de processamento. Assim, multicomputadores tipicamente empregam redes de comunicação de baixa latência e protocolos especiais de comunicação com o objetivo de reduzir o impacto no tempo de processamento da aplicação.

Talvez uma das grandes vantagens na utilização de multicomputadores seja a possibilidade de realizar o crescimento incremental do *hardware*. Novos computadores podem ser agregados ao sistema existente, aumentando seu poder computacional. Outra vantagem do uso de multicomputadores é a possibilidade de montar ambientes paralelos a partir dos chamados computadores de prateleira (*off-the-shelf*): máquinas com grande poder de processamento podem ser facilmente montadas e configuradas a partir de computadores e peças disponíveis nas lojas de varejo. Quanto às desvantagens, além da dependência do sistema de comunicação, que pode tornar-se um limitador do desempenho das aplicações executadas nesta arquitetura, também podemos citar o fato do programador ser responsável por grande parte dos detalhes associados à comunicação de dados entre os computadores, algo que pode dificultar o desenvolvimento de aplicações paralelas.

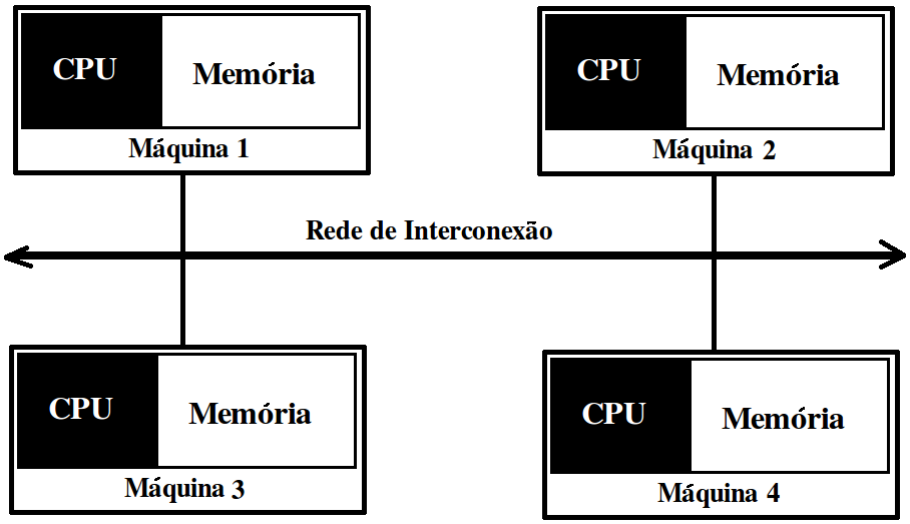


Figura 3.2: Representação esquemática da arquitetura de um multicomputador.

3.2.2 Multiprocessador

Nos multiprocessadores, um conjunto de núcleos de processamento compartilha uma série de recursos de um computador, como memórias e dispositivos de entrada/saída. Tal compartilhamento de memória em um multiprocessador implica na utilização de um

espaço de endereçamento único por parte dos núcleos, o que significa que cada posição de memória possui um endereço único, independente do núcleo que o esteja referenciando[31]. Deste modo, qualquer alteração em uma parte da memória por um núcleo torna-se visível aos demais.

Os multiprocessadores com memória compartilhada ainda podem ser divididos em duas classes, de acordo com o tempo de acesso à memória: UMA e NUMA. Máquinas UMA (*U*nified *M*emory *A*ccess) possuem acesso uniforme à memória, o que significa que qualquer acesso à memória principal por qualquer núcleo levará sempre o mesmo tempo para ser concluída. A Figura 3.3 apresenta um esquema dessa arquitetura. Em máquinas NUMA (*N*on-*U*nified *M*emory *A*ccess) o tempo de acesso à memória pode variar em função do endereço de memória acessada. Alguns núcleos terão tempo de acesso mais rápido a uma determinada faixa de endereços, enquanto o acesso pode ser mais lento a outra faixa de endereços. Isso ocorre pela forma como as máquinas NUMA são construídas, conforme pode ser observado na Figura 3.4. Nesta figura verifica-se a presença de três barramentos distintos, um dos quais é utilizado para interligar os outros dois barramentos. O acesso será mais lento sempre que um núcleo precisar cruzar este barramento de interligação para acessar uma memória.

As vantagens da utilização de multiprocessadores, em relação à arquitetura multicomputador, são: a) menor tempo de acesso à memória, devido principalmente à proximidade física com que qualquer posição de memória se encontra em relação aos núcleos; b) maior facilidade no compartilhamento de dados entre as distintas partes de uma aplicação, visto que os mesmos já se encontram em memória e esta é compartilhada por todos os núcleos. Como principais desvantagens na utilização desta arquitetura, destacam-se: a) a responsabilidade do programador em sincronizar o acesso aos dados de sua aplicação e b) o menor número de núcleos de processamento que podem ser empregados simultaneamente na resolução problemas. Em relação ao item a) das desvantagens, é importante apresentar dois termos, seção crítica e exclusão mútua. Quando processos concorrentes interagem com dados localizados em regiões compartilhadas de memória, a integridade destes dados pode ser violada se não for imposta uma coordenação no acesso a esses dados. A solução para este problema consiste em sincronizar os processos para que apenas um destes possa acessar um dado ou um recurso compartilhado por vez. A imposição de que apenas um processo pode acessar um dado ou

um recurso compartilhado por vez é conhecida como exclusão mútua. Uma seção crítica é um segmento de código onde um dado ou um recurso compartilhado pode ser acessado por mais de um processo por vez, e onde portanto pode ser necessária a imposição da exclusão mútua.

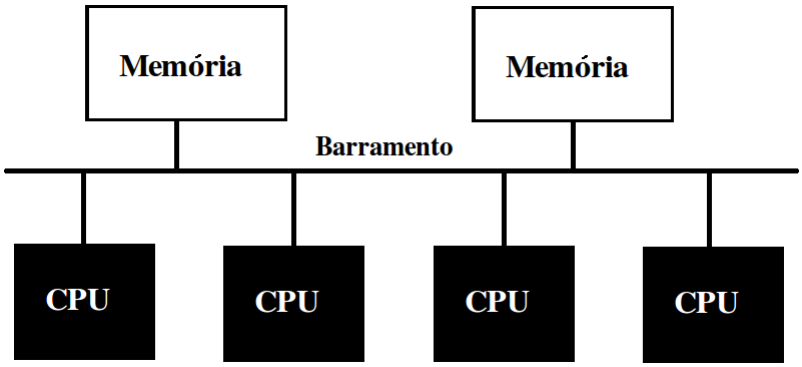


Figura 3.3: Esquema representativo de uma arquitetura com acesso uniforme à memória (UMA).

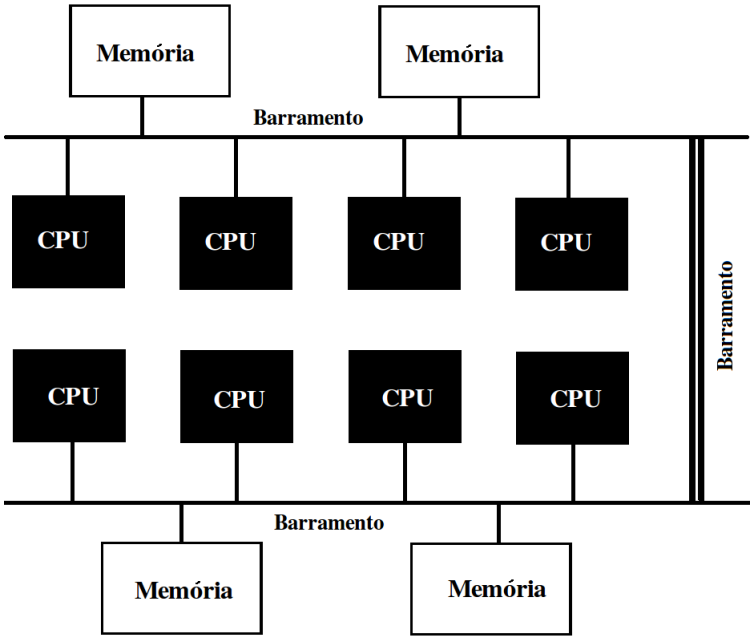


Figura 3.4: Representação de uma arquitetura computacional com diferentes tempos de acesso à memória (NUMA).

3.3 Modelos de Programação Paralela

Existem, basicamente, dois modelos para o desenvolvimento de aplicações paralela, que são: troca de mensagens e memória compartilhada [32]. A escolha de um dos modelos está intimamente relacionada a arquitetura a ser utilizada. Em geral, o modelo com troca de mensagens é utilizado em máquinas multicomputador, enquanto o modelo de memória compartilhada é utilizado em máquinas multiprocessador, apesar de não existir uma imposição rígida para este mapeamento. Nesta seção apresentamos uma introdução aos modelos de programação paralela.

3.3.1 Troca de Mensagens

O modelo de programação com troca de mensagens utiliza um mecanismo explícito de comunicação para que os dados de um processo sejam enviados a outro processo. Cabe ao programador definir que dados devem ser enviados e em que momentos essa comunicação deve se dar. O processo que deseja enviar dados, chamado remetente, invoca uma rotina de envio de dados (*send*), enquanto o processo que deseja receber uma cópia dos dados, chamado destinatário, deve invocar uma rotina para o seu recebimento (*receive*). Essas operações impõem, de modo implícito, uma sincronização entre os processos que dela participam, visto que os dados só serão efetivamente trocados quando o destinatário realizar uma chamada para uma operação de recebimento e o remetente chamar a função para envio de dados [32].

Este modelo, portanto, casa naturalmente com multiprocessadores, onde é necessário o estabelecimento de comunicação entre os processos através da rede de comunicação. Entretanto, nada impede que processos em um multiprocessador utilizem esse mecanismo para estabelecer comunicação. A diferença é que, no primeiro caso, uma mensagem será enviada entre distintos computadores através da rede de comunicação que os liga, enquanto no segundo caso nenhuma mensagem trafegará na rede de comunicação pelo fato dos dois processos se encontrarem em uma mesma máquina.

Existem diversas alternativas para implementação do modelo de programação com troca de mensagens. Pode-se, por exemplo, definir linguagens inteiramente novas que já possuam palavras reservadas para estabelecer a troca de mensagens, ou pode-se, por exemplo, utilizar uma linguagem já conhecida e consagrada, oferecendo ao programador

uma biblioteca que implemente o modelo de troca de mensagens. Um padrão muito conhecido para o modelo de programação paralela por troca de mensagens oferecido na forma de biblioteca é o MPI (*M*essage *P*assing *I*nterface). Este padrão define um conjunto de rotinas para facilitar a comunicação entre os processos que estão executando em paralelo. O programador é responsável por identificar os pontos em que a comunicação se faz necessária, utilizando para tal chamadas às funções de troca de mensagens e sincronismo oferecidas por MPI. Destacam-se em MPI primitivas para comunicação ponto a ponto e para comunicação coletiva. Neste último caso, um único remetente pode enviar uma mensagem para vários destinatários distintos. Existem várias bibliotecas que implementam o padrão MPI. Um exemplo de implementação gratuita é o MPICH [31].

3.3.2 Memória Compartilhada

No modelo de programação com memória compartilhada, distintos processos compartilham uma mesma região de memória. Esta é utilizada para realizar a comunicação entre os processos. Quando um processo precisa passar um dado para outro processo, basta que este copie o dado para a região de memória comum. O dado assim se tornará automaticamente acessível ao processo que irá utilizá-lo [32]. Este modelo de programação se casa naturalmente com multiprocessadores, que já possuem *hardware* que permite o compartilhamento de dados entre núcleos distintos. Entretanto, o modelo pode ser implementado com suporte de *hardware* ou $software$ em multicomputadores, recebendo neste caso o nome de memória compartilhada distribuída.

Assim como no modelo de troca de mensagens, existem diversas alternativas para implementação do modelo de programação com memória compartilhada. Uma das mais interessantes é a utilização de uma linguagem sequencial existente incrementada com diretivas de pré-compilação para especificar o paralelismo. Nesse tipo de implementação é necessário somente o aprendizado das diretivas. Um padrão popular que implementa esse formato é o OpenMP [31].

OpenMP

OpenMP é uma interface de programação de aplicativos - API (*A*pplication *P*rocessing *I*nterfaces) - que implementa o paralelismo em memória compartilhada através de

diretivas de pré-compilação. Cabe ao programador identificar os pontos do código que serão paralelizados e usar as diretivas para definir como deseja que essa paralelização seja feita [33]. Assim, as diretivas de pré-compilação são como anotações no código inseridas pelo programador, mas tratadas de fato pelo pré-compilador, que as identifica e transforma em um conjunto de operações equivalentes na linguagem de programação alvo. Tipicamente essas transformações no código levam a estruturas muito mais complexas do que as estruturas anotadas pelo programador. Talvez esse seja o motivo da popularidade de OpenMP: através de anotações simples, o programador é capaz de realizar operações complexas.

As anotações se parecem com comentários do tipo !$*omp* no Fortran ou #*pragma* em C/C++ e são ignoradas por compiladores que não implementem OpenMP, de forma que se mantém a portabilidade do programa no caso de execução em tais ambientes [34]. Pode-se delimitar com o auxílio das diretivas os trechos do programa que serão executadas por múltiplos processos, ou *threads* no jargão de programação paralela. Uma *thread* pode ser definida como uma sequência independente e concorrente de execução dentro de um processo. Esse fluxo de execução compartilha com os demais fluxos o mesmo espaço de memória e variáveis globais do processo [31].

As diretivas principais do OpenMP serão listadas a seguir:

Diretiva *Parallel* : responsável pela criação de múltiplas *threads* que executam um bloco de instruções em paralelo. De forma geral pode ser utilizado da seguinte forma em FORTRAN (os colchetes representam características opcionais):

!$**omp parallel** [**clausula**[,] [**clausula**...]]

bloco de instruções

!$**omp end parallel**

Em C e C++ o formato é:

#**pragma omp parallel** [**clausula**[,]**clausula**[...]]

bloco de instruções

Algumas das cláusulas que podem ser utilizadas em conjunto a diretiva *parallel* são as seguintes:

- ***private***, que identifica as variáveis que terão uma cópia privada para cada *thread* durante a execução da região paralela, ou seja, que não serão alocadas

na região de memória compartilhada. Neste caso as variáveis serão alocadas na pilha de execução de cada *thread*;

- ***shared***, que especifica que variáveis podem ser compartilhadas entre as *threads*, ou seja, que serão alocadas na região de memória compartilhada.

Diretiva *do* : esta diretiva especifica que o laço "*do*" no FORTRAN ou "for" em C/C++ deve ser executado em paralelo. Neste caso, cada iteração do laço será executada de modo independente por uma *thread* diferente. Uma instrução com a definição do laço deve seguir a diretiva. Pode-se ainda controlar a execução do laço paralelo com a utilização de cláusulas opcionais. São exemplos de cláusulas que podem ser utilizadas [34]:

- cláusulas de escopo, como ***shared*** e ***private***, para determinar, respectivamente, se uma ou mais variáveis serão compartilhadas entre as *threads* ou possuirão uma cópia particular para cada *thread* durante a execução do laço;
- ***schedule***, que controla como as iterações do laço são distribuídas entre as *threads*. O escalonamento pode ser a) estático, quando a escolha de qual *thread* irá executar qual iteração é uma função do número de iterações pelo número de *threads* criadas, sendo determinada em tempo de compilação; e b) dinâmico, em que essa distribuição é definida em tempo de execução, podendo portanto variar de uma execução para outra. Existem várias opções para utilização desta cláusula e mais informações podem ser obtidas na referência [34];

A diretiva *do* deve ser usada em conjunto com a diretiva *parallel*. Isso significa que a) ou um time de *threads* já foi criado anteriormente com uma diretiva *parallel* localizada em um ponto anterior do código, ou b) que a diretiva *parallel* será usada em conjunto com *do*. Este último caso é muito comum. A forma geral de utilização conjunta das diretivas *parallel* e *do* em FORTRAN é a que segue:

!$**omp parallel do** [**clausula**[**,**] [**clausula** ...]]
    do indice=primeiro , ultimo [, incremento]
        corpo do laço
[!$**omp end parallel do**]

Em C/C++:

#pragma omp parallel for [clausula [clausula ...]]
for(indice=primeiro; expressao_teste; incremento)
corpo do laço

Diretiva *Critical Section* : define uma seção crítica no programa. Essa diretiva realiza exclusão mútua em relação a todas as outras diretivas *critical section* no programa, de forma que é permitida a execução ao mesmo tempo em qualquer parte do programa de somente uma seção crítica por vez. Havendo a necessidade de criar mais de uma seção crítica, é conveniente nomear cada bloco de forma distinta usando *nome*. Uma seção nomeada sincroniza apenas com as seções de mesmo nome, mas podem ser executadas simultaneamente às seções com nomes diferentes [34].

Essa diretiva tem a forma geral em Fortran:

!$omp critical [(nome)]
bloco de instruções
!$omp end critical [(nome)]

E em C/C++:

#pragma omp critical [(nome)]
bloco de instruções

Exemplo

Para exemplificar o uso das diretivas apresentadas, utilizaremos um código para multiplicação de um escalar por um vetor. O código sequencial em FORTRAN é apresentado na Figura 3.5. Neste código o bloco "*do*" é responsável pela operação de multiplicação de um escalar por um vetor. Como cada iteração do laço é independente das demais, ou seja, nenhum dado gerado na iteração $i-1$ de um laço será utilizado na iteração i, podem ser executadas em paralelo. A paralelização deste laço pode ser realizada simplesmente com a inclusão da diretiva *parallel do* no código, como ilustra a Figura 3.6. Nesta versão do código, o programa inicia a execução de forma sequencial, portanto com uma única *thread* de execução. Quando uma diretiva do tipo *parallel* é localizada, o pré-compilador OpenMP sabe que se trata de uma indicação do programador de que as instruções seguintes devem ser executadas de forma simultânea. Assim, um "time" de

threads é criado para a execução em paralelo do bloco de instruções que segue a diretiva. Como neste caso a diretiva *parallel* está acompanhada da diretiva *do*, o pré-compilador sabe que cada uma das *threads* criadas deverá executar uma iteração diferente do laço. Ao identificar o fim do bloco, o programa segue sua execução de modo sequencial, com apenas uma única *thread* ativa. A Figura 3.7 mostra a paralelização do mesmo exemplo utilizando a linguagem C e, nesse caso, utiliza-se a diretiva *parallel for*.

```
      subroutine exemplo(z,a,x,n)
      integer i,n
      real z(n), a, x(n)

      do i=1, n
        z(i) = a * x(i)
      enddo
      return
      end
```

Figura 3.5: Algoritmo sequencial para multiplicação de um escalar por um vetor utilizando a linguagem FORTRAN.

```
      subroutine exemplo_omp(z,a,x,n)
      integer i,n
      real z(n), a, x(n)
!$omp parallel do
      do i=1, n
        z(i) = a * x(i)
      enddo
      return
      end
```

Figura 3.6: Exemplo simples da utilização de uma diretiva *parallel do* para paralelização da multiplicação de um escalar por um vetor em FORTRAN.

```
int exemplo_omp(float z, float a, float x, int n){
  int i,n
  float z, a, x
#pragma omp parallel for
  for (i = 0; i <= n ; i++) {
    z(i) = a * x(i)
  }
  return(1);
}
```

Figura 3.7: Utilização de uma diretiva *parallel for* para paralelização da multiplicação de um escalar por um vetor empregando a linguagem de programação C.

CUDA

As unidades de processamento gráfico (*G*raphics *P*rocessing *U*nits - GPUs) são hoje um dos sistemas multiprocessador mais poderosos em uso. Além da utilização principal na aceleração de operações de renderização gráfica, tem crescido bastante o interesse na sua exploração para a computação de propósito geral (*G*eneral *P*urpose *G*raphics *P*rocessing *U*nit - GPGPU). O grande poder de processamento das GPUs aliada a possibilidade de sua utilização para realizar computação de propósito geral tornou as GPGPUs uma escolha atrativa para realizar computação numérica e científica de alto desempenho. A Figura 3.8 ilustra um dos motivos do grande poder de processamento das GPUs. Nesta figura pode ser observado que a quantidade de unidades de processamento presente nas GPUs é muito superior a quantidade presente nas CPUs.

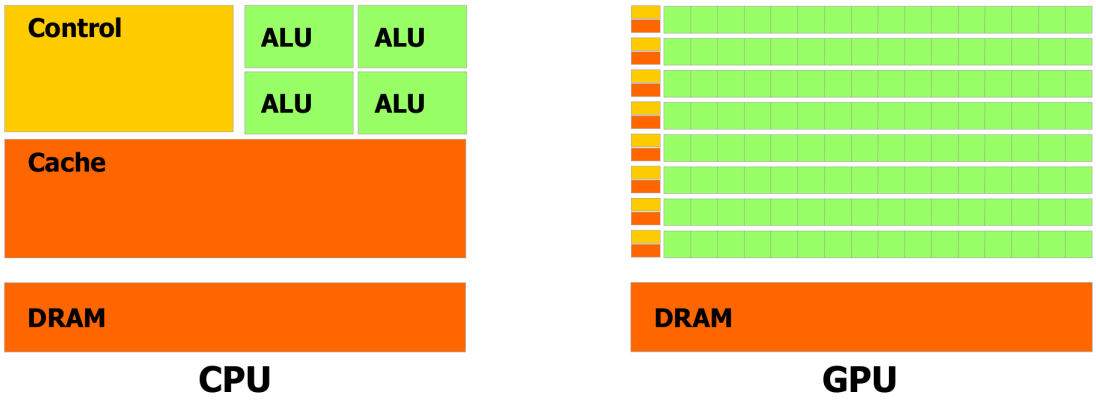


Figura 3.8: As GPUs dedicam muito mais transistores ao processamento do que à cache e ao controle de fluxo. Extraída de [8]

Um exemplo de interface para computação em GPGPUs é a arquitetura CUDA (*C*ompute *U*nified *D*evice *A*rchitecture) [8]. CUDA inclui ferramentas e bibliotecas para desenvolvimento de aplicações em várias linguagens de programação, como C e C++.

Na nomeclatura CUDA, um *kernel* é uma função chamada a partir da CPU e executada na GPU simultaneamente por várias *threads*. Em CUDA, as *threads* são organizadas hierarquicamente em dois grupos, blocos e *grids*. Blocos são compostos de *threads* e *grids* compostos de blocos. Um bloco pode ter *threads* organizadas em três dimensões, x, y e z. Assim, o número de *threads* criadas para a execução de um *kernel* é derivado diretamente do número de *threads* por bloco e de blocos por *grid*. Este dois valores devem ser especificados pelo programador no momento da invocação de um *kernel*. A sintaxe que especifica tais valores é formalmente conhecida como configuração

de execução.

Como todas as *threads* que pertencem a um *grid* executam o mesmo código, é necessário distinguir os dados que devem ser processados por cada uma delas. Um conjunto único de números identificadores é utilizado para este propósito, identificando assim univocamente as *threads*. Esse conjunto é dado pelos valores das variáveis *blockId* e *threadId*, criadas em tempo de execução pelo *CUDA runtime system*. O primeiro valor identifica, dentro de um *grid*, o número do bloco ao qual a *thread* pertence. O segundo valor é o identificador da *thread* dentro de cada uma das dimensões de um bloco. Essas duas variáveis podem ser acessadas dentro das funções *kernel*.

Alguns passos devem ser seguidos para utilizar a GPU, quais sejam:

- Inicialização do dispositivo;
- Alocação de memória na GPU e transferência dos dados, uma vez que a memória da CPU não pode ser acessada diretamente pela GPU, e vice-versa;
- Chamada ao *kernel* que será executado na GPU simultaneamente por várias *threads* e
- Recuperação dos dados de volta para a CPU, de modo a dar continuidade ao processamento sequencial.

Exemplo

A Figura 3.9 mostra um exemplo de *kernel* CUDA C para realização da operação de multiplicação paralela de um escalar por um vetor na GPGPU. A função *kernel* é declarada como __global__ e o número de blocos que compõe o *grid* e de *threads* que compõem o bloco são definidos utilizando a configuração <<< ... >>>. No exemplo, é criado um bloco unidimensional composto por N *threads*, em que N deve ser definido antes da chamada ao *kernel*. Cada *thread* realiza a multiplicação de um índice do vetor, utilizando para isso o valor de $threadIdx.x$, ou seja, o componente na dimensão x do bloco.

```
//definicao do kernel
__global__ void VecMult (float* z, int a, float* x){
  int i = threadIdx.x;

  z[i] = a * x[i];
}

int main(){
  ...
  //chamada ao kernel com N threads
  VecAdd <<<1,N>>> (z, a, x);
  ...
}
```

Figura 3.9: Exemplo de kernel CUDA para a multiplicação de um escalar por um vetor com a chamada a partir do código executado na CPU.

4 Implementação Computacional da HEA Empregando o MEF em 2D

4.1 Introdução

A partir de uma versão preliminar desenvolvida no ambiente de processamento algébrico Matlab® [35] para o cálculo de propriedades homogeneizadas a partir de células bidimensionais, foi implementada uma versão utilizando a linguagem de programação C. Dentre as motivações para a implementação desta versão cita-se: (a) a versão inicial em Matlab® apresentava limitações no cálculo de propriedades homogeneizadas para determinadas geometrias mais complexas; (b) viabilizar melhor compreensão do problema mecânico visando a implementação de uma versão para células tridimensionais.

Além da versão sequencial, foram desenvolvidas ainda duas versões paralelas: utilizando OpenMP e CUDA e os detalhes da implementação computacional dessas versões serão apresentados neste Capítulo.

4.2 Versão Sequencial do Programa HEA2D

4.2.1 Descrição Geral do Programa

O objetivo da aplicação chamada HEA2D é o cálculo das propriedades mecânicas homogeneizadas a partir de células planas utilizando elementos triangulares de seis nós.

Deve-se fornecer como entrada de dados as características geométricas e mecânicas da célula, cujo domínio é retangular. As malhas de elementos finitos devem permitir a imposição das condições de contorno periódicas nos nós situados nas arestas. Obtém-se como resposta o tensor de propriedades efetivas do material.

As etapas do cálculo das propriedades homogeneizadas são apresentadas a seguir.

4.2.2 Imposição das Condições de Contorno Periódicas

Para a imposição das condições de contorno periódicas, utiliza-se uma estratégia que realiza a redução da ordem do sistema de equações a ser resolvido. Primeiramente, os nós do contorno são associados através da numeração das equações globais: os Graus de Liberdade (GLs) dos nós correspondentes em arestas opostas recebem a mesma numeração. A Figura 4.1 representa esta estratégia ilustrando apenas os nós do contorno. Para fins de ilustração, admitindo que cada nó tenha apenas um grau de liberdade, na Figura 4.1 número iguais indicam a periodicidade entre os nós situados em arestas opostas.

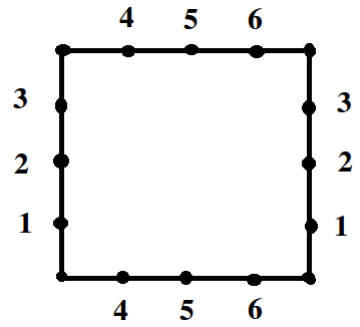


Figura 4.1: Representação da estratégia utilizada para imposição das condições de contorno periódicas.

A estratégia adotada para impor as condições de contorno ao montar o sistema de equações é representada através do exemplo mostrado a seguir. Em (4.1), tem-se um sistema [k]{u}={f} com 5 *G*raus de *L*iberdade (GLs), sendo os GLs 1 e 5 periódicos e, portanto, $u_1 = u_5$:

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \\ k_{31} & k_{32} & k_{33} & k_{34} & k_{35} \\ k_{41} & k_{42} & k_{43} & k_{44} & k_{45} \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix}. \tag{4.1}$$

O sistema modificado é representado em (4.2), em que:

1. à linha 1 soma-se a linha 5;

2. à coluna 1 soma-se a coluna 5;

3. Reduz-se a ordem do sistema a resolver obtendo (4.2).

$$\begin{bmatrix} (k_{11}+k_{51})+(k_{15}+k_{55}) & k_{12}+k_{52} & k_{13}+k_{53} & k_{14}+k_{54} \\ k_{21}+k_{25} & k_{22} & k_{23} & k_{24} \\ k_{31}+k_{35} & k_{32} & k_{33} & k_{34} \\ k_{41}+k_{45} & k_{42} & k_{43} & k_{44} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} (f_1+f_5) \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \quad (4.2)$$

Uma vez calculados os valores de u_i da Equação (4.2), atualiza-se o vetor-solução completo, fazendo $u_5 = u_1$. Esta estratégia possibilita a consideração das condições de contorno periódicas alterando a ordem do sistema de equações e mantendo a matriz de rigidez k simétrica.

4.2.3 Funcionalidades básicas

Entrada de Dados

Devem ser informados como entrada: os dados da malha de elementos finitos, números dos GLs com condições de contorno periódicas, e propriedades mecânicas das fases α que compõem o domínio discretizado da microestrutura (módulo de elasticidade E_α e coeficiente de Poisson ν_α), considerando aqui todas as fases isotrópicas.

Montagem da matriz de rigidez e do tensor de termos independentes

A partir da Equação (2.39), a matriz de rigidez do sistema é calculada como (4.3):

$$\left[\sum_{i=1}^{nelm} B^T D B J\right] \quad (4.3)$$

e o tensor de termos independentes como (4.4):

$$\left[\sum_{i=1}^{nelm} B^T D J\right] \quad (4.4)$$

onde *nelm* é o número de elementos da malha que discretiza a célula, B é operador diferencial, D é o tensor de propriedades elásticas do elemento e J é o tensor jacobiano, que relaciona o sistema de coordenadas com a representação paramétrica da geometria.

Na Figura 4.2 está a representação do algoritmo com a implementação do cálculo e

montagem deste sistema. Utiliza-se integração de Gauss com 7 pontos para as funções.

Algoritmo para o cálculo e montagem de K e F
INICIO
PARA $i = 1$ ATÉ *nume* FAÇA
INICIO
localiza nós do elemento
localiza matriz constitutiva do elemento
inicializa K e F temporários
PARA $j = 1$ ATÉ *npg* FAÇA
INÍCIO
calcula coeficientes da matriz de derivação
calcula K e F dos elementos e acumula nas respectivas estruturas globais
FIM
FIM PARA
FIM
FIM PARA
FIM

Figura 4.2: Pseudo-código do cálculo e montagem da matriz de rigidez K e do tensor de termos independentes F, em que *nume* é o número total de elementos e *npg* é o número de pontos de Gauss.

Solução do sistema de equações

Para obtenção do corretor elástico, representado por χ na Equação (2.39), empregou-se para a solução do sistema de equações uma adaptação da biblioteca $CSPARSE$ [36]. Essa biblioteca oferece a possibilidade de escolha da resolução do sistema dentre os métodos diretos de resolução QR, LU e Cholesky e ainda possui duas opções para armazenar a matriz de rigidez esparsa. A primeira forma é chamada *triplet* e a outra *compressed-column*. O formato escolhido para implementação, que foi o segundo, utiliza três vetores para armazenar os valores não nulos da matriz esparsa. Um deles, *data*, é utilizado para guardar os valores não nulos propriamente ditos, o segundo, chamado *indice*, guarda os ponteiros para as colunas em que se encontram os elementos correspondentes e o terceiro vetor, *ptr*, armazena o número de elementos não-nulos por linha. O primeiro índice do vetor *ptr* sempre será zero e o último conterá o número de elementos não-nulos da matriz.

Para os três métodos disponíveis ainda é possível escolher entre três tipos de permutação que podem ser aplicadas à matriz para minimizar o custo computacional na fatoração, quais sejam: $A + A^T$, que é mais apropriado para Cholesky ou LU, $S^T * S$,

mais adequado para a decomposição LU em matrizes não-simétricas e $A^T * A$, que é mais conveniente para fatorações QR ou LU [36].

Como o tensor de termos independentes possui na verdade 3 colunas, consequentemente o tensor χ também terá a mesma ordem. Para a resolução do sistema, a matriz de rigidez é armazenada no formato *compressed-column* e passada como parâmetro junto com o tensor para o *solver* da biblioteca. Para a resolução das três colunas, optou-se por adaptar o método de forma a resolver cada coluna de forma independente, uma de cada vez, utilizando um laço de repetição for, visando a posterior resolução dessas iterações em paralelo. A Figura 4.3 mostra o pseudo-código do método modificado da biblioteca. Devem ser passados como parâmetros a matriz de rigidez, as 3 colunas do termo independente, a estrutura que irá retornar o resultado e o método numérico escolhido para a resolução (QR, LU ou Cholesky). Nesta versão sequencial, cada coluna é resolvida por vez.

Cálculo do tensor de propriedades homogeneizadas $\mathbf{D}^h$

A partir da resolução do sistema de equações, o tensor de propriedades homogeneizadas (D^h) é calculado a partir da Equação (2.43) e o pseudo-código da implementação é mostrado na Figura 4.4.

Algoritmo para a resolução do sistema de equações

```
INICIO
SE metodo = 1 ENTAO
  INICIO
    PARA i = 1 ATÉ 3 FAÇA
    INICIO
      QR (K,F_i,χ_i)
    FIM
  FIM PARA
  FIM
FIM SE
SE metodo = 2 ENTAO
  INICIO
    PARA i = 1 ATÉ 3 FAÇA
    INICIO
      LU (K,F_i,χ_i)
    FIM
  FIM PARA
  FIM
FIM SE
SE metodo = 1 ENTAO
  INICIO
    PARA i = 1 ATÉ 3 FAÇA
    INICIO
      Cholesky (K,F_i,χ_i)
    FIM
  FIM PARA
  FIM
FIM SE
FIM
```

Figura 4.3: *QR*, *LU* e *Cholesky* representam respectivamente as chamadas às resoluções utilizando um dos 3 métodos à escolha; K é a matriz de rigidez, F_i é a coluna i do termo independente e χ_i a coluna i resultante.

Algoritmo para o cálculo do D^h
INICIO
inicializa(D^h)
PARA $i = 1$ ATÉ $nume$ FAÇA
INICIO
localiza nós do elemento
localiza matriz constitutiva do elemento
calcula área do elemento e acumula na área total
PARA $j = 1$ ATÉ npg FAÇA
INÍCIO
calcula coeficientes da matriz de derivação
$D^h \leftarrow D_e \times (I - B_e \times \chi_e)$
FIM
FIM PARA
FIM
FIM PARA
divide termos de D^h pela área total
FIM

Figura 4.4: $nume$ é o número total de elementos, D_e é a matriz de coeficientes do elemento, I_e é a matriz identidade de ordem 3, B_e é o operador diferencial, χ_e é o corretor elástico χ do elemento e npg é o número de pontos de Gauss.

4.3 Versões Paralelas

A partir da versão sequencial do programa `HEA2D`, foram desenvolvidas duas versões paralelas, utilizando OpenMP e CUDA. As estratégias utilizadas no desenvolvimento destas versões paralelas são descritas a seguir.

4.3.1 Versão Paralela com OpenMP

A primeira opção de paralelização do código explorada emprega as diretivas de compilação do OpenMP. Optou-se por utilizar o OpenMP pelo alto nível de abstração oferecido e pela sua disponibilidade em vários compiladores. Conforme detalhamento no Capítulo 6, a máquina disponível para execução é um multiprocessador e, portanto, a escolha mais natural é a utilização de um modelo de programação com memória compartilhada.

As duas partes do código sequencial que consomem maior tempo de processamento são a montagem da matriz de rigidez e a resolução do sistema de equações lineares. A partir desse cenário, optou-se por utilizar 3 abordagens distintas, para a implementação da versão paralela utilizando as diretivas do OpenMP:

1. Paralelização da resolução do sistema de equações;
2. Paralelização da montagem da matriz de rigidez;
3. Paralelização de ambos.

Paralelização da resolução do sistema de equações

Para a paralelização da resolução do sistema de equações com OpenMP, foi incluído inicialmente na versão sequencial um laço *for* para a chamada da resolução de cada uma das três colunas do tensor. A paralelização dessa resolução consiste na inclusão de uma diretiva do tipo *#pragma omp parallel for* com as devidas cláusulas privadas. Dessa forma, cada iteração do laço *for*, correspondente a uma chamada à resolução de um dos três sistemas, passa a ser resolvido simultaneamente por 3 *threads* distintas. A Figura 4.5 mostra o algoritmo paralelo com as diretivas OpenMP.

```
#pragma omp parallel for private(b) shared (C)
for (i =0; i <= 2 ; i++) {
  switch(i){
    case 0:b = Prob->b1;break;
    case 1:b = Prob->b2;break;
    case 2:b = Prob->b3;break;
  }
  ok = cs_cholsol (C, b, order) ;
  switch(i){
    case 0:Prob->x1 = b;break;
    case 1:Prob->x2 = b;break;
    case 2:Prob->x3 = b;break;
  }
}
```

Figura 4.5: Resolução do sistema de equações em paralelo com OpenMP.

A desvantagem na utilização desta abordagem se deve ao ganho de aceleração com a paralelização estar limitado a esse número de *threads*, ainda que a arquitetura possua um número maior de processadores disponíveis.

Paralelização da montagem da matriz de rigidez

A matriz de rigidez é esparsa, e sua montagem para armazenamento nesse formato requer alocação e realocação de memória de forma dinâmica. Esse procedimento é realizado pelo método *cs_entry* da biblioteca *CSPARSE*. A chamada a esse método foi paralelizada utilizando diretivas específicas do OpenMP e o algoritmo é apresentado na Figura 4.6.

```
#pragma omp parallel for ordered private(j)
#pragma omp shared (gll,kg,list_gll,T)
for (i = 0; i < gll; i++) {
  #pragma omp ordered
  for (j = 0; j < gll; j++) {
    if (kg[list_gll[i]-1][list_gll[j]-1] != 0)
     cs_entry(T,i,j,kg[list_gll[i]-1][list_gll[j]-1]);
  }
}
```

Figura 4.6: Paralelização da montagem da matriz de rigidez com OpenMP.

Como o método *cs_entry* realoca memória utilizada para armazenar a matriz esparsa, esse procedimento deve ser realizado de forma ordenada. Para tanto utiliza-se a diretiva *ordered* de modo a especificar que o código dentro do laço paralelo *for* deve ser executado como um laço sequencial [31].

4.3.2 Versão Paralela com CUDA

Visando um ganho maior de desempenho, foi implementada uma versão do `HEA2D` para GPGPUs utilizando CUDA[8]. No desenvolvimento dessa versão, a biblioteca *C*oncurrent *N*umber *C*runcher (CNC) [37], desenvolvida para ser executada em GPGPUs, foi adaptada para atender a resolução do sistema de equações [24]. Essa biblioteca resolve o sistema através do método iterativo de Gradientes Conjugados com Precondicionamento Jacobi (ou diagonal) [38, 39] e oferece opções para o armazenamento de matrizes esparsas.

As funções que são executadas em paralelo na GPGPU – *kernels* – são as operações algébricas do método iterativo GC otimizadas do pacote BLAS [40], dentre as quais podemos citar a multiplicação matriz(esparsa)-vetor, e o produto interno. Nessa versão, diferentemente da versão OpenMP, as chamadas ao método de resolução de cada sistema são realizadas de forma sequencial. Na versão CUDA o paralelismo se encontra na resolução das operações algébricas internas que ocorrem a cada iteração do método.

5 Implementação Computacional da HEA Empregando o MEF em 3D

5.1 Introdução

Neste trabalho, a técnica da HEA aplicada ao cálculo de propriedades elásticas efetivas em problemas tridimensionais foi implementada em duas versões: uma versão sequencial e uma versão paralela empregando OpenMP. Inicialmente, pretendia-se obter também uma versão paralela do HEA3D empregando GPGPUs, no entanto isto não foi possível, devido a limitações relativas às bibliotecas atuamente disponíveis para a implementação do CUDA para FORTRAN.

O trabalho consistiu nas etapas:

1. Implementação em $3D$ da HEA;
2. Paralelização do programa utilizando OpenMP.

5.2 Versão Sequencial do Programa HEA3D

5.2.1 Descrição Geral do Programa

Para a implementação computacional da HEA, adotou-se um programa-base originalmente desenvolvido na COPPE/UFRJ [41, 42] para a análise não-linear de problemas mecânicos contínuos em $3D$ via MEF, em linguagem FORTRAN que apresenta as seguintes características:

- elementos tetraédricos com quatro nós para problemas tridimensionais;
- resolução do sistema de equações lineares empregando o método dos Gradientes Conjugados Pré-Condicionado [42, 38] implementado segundo uma estratégia elemento por elemento (Element-By-Element - EBE) [43, 44], que dispensa a montagem e a fatoração da matriz de rigidez global. Essa estratégia possibilita a implementação em ambientes computacionais de alto desempenho, explorando

paralelismo dos cálculos e, dessa forma, viabilizando a análise de modelos mais realistas e de grande porte.

A partir desse programa foi desenvolvido o programa `HEA3D` (*H*omogeneização por *E*xpansão *A*ssintótica em 3 *D*imensões). As principais alterações aqui realizadas para adaptar o programa ao cálculo das propriedades elásticas homogeneizadas foram:

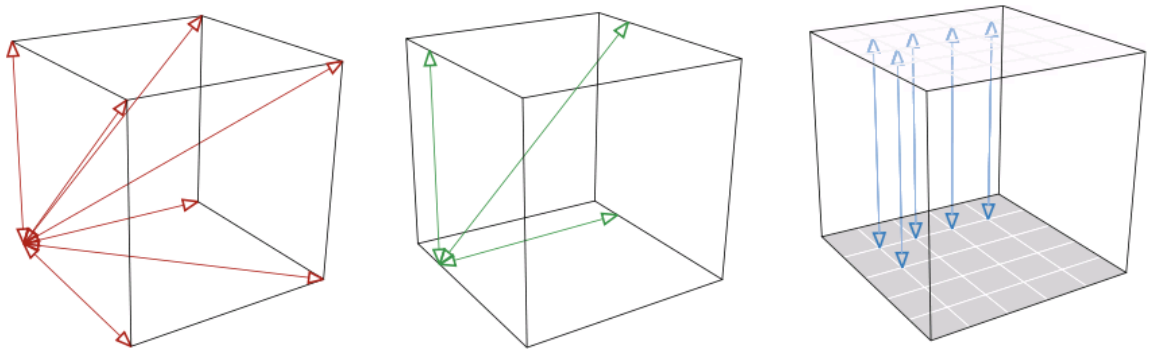
- inclusão das condições de contorno periódicas;
- montagem do termo independente F^D, descrito pela Equação (2.38);
- determinação da matriz elástica homogeneizada (Equação (2.43)).

Além dessas características, uma condição do programa `HEA3D` é que o domínio da malha de elementos finitos utilizada como entrada de dados deve ser hexaédrica e ter condições de contorno periódicas: para cada nó de uma face deve haver um nó correspondente na respectiva face oposta.

5.2.2 Imposição das Condições de Contorno Periódicas

Para a posterior imposição das condições de contorno de periodicidade, um vetor é utilizado para relacionar os nós do contorno. A partir das coordenadas dos nós das faces opostas, estes são associados de forma que um deles é escolhido como mestre e o nó correspondente na face oposta é definido como escravo. O vetor utilizado para essa associação possui dimensão igual ao número de nós e é inicializado com zeros. A Figura 5.1 mostra a associação dos nós periódicos de forma esquemática. Para os vértices, (Figura 5.1(a)) define-se um como mestre e os outros 7 como seus respectivos escravos. O mesmo ocorre para os nós que estão localizados nas arestas, sendo que cada associação terá um nó definido como mestre e os outros 3 como escravos (Figura 5.1(b)). No caso dos demais, cada nó definido como mestre possuirá um nó associado a ele na face oposta (Figura 5.1(c)).

A estratégia adotada para impor as condições de contorno ao montar o sistema de equações é representada através do exemplo mostrado nos sistemas 5.1 e 5.2. Em (5.1), um sistema [k]{u}={f} com 5 *G*raus de *L*iberdade (GLs), sendo 1 o GL mestre e 5 o GL



(a) (b) (c)

Figura 5.1: Esquema de associação dos nós periódicos nos contornos da célula unitária para: (a) vértices, (b) arestas e (c) faces. Adaptada de [7].

escravo e $u_1 = u_5$:

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \\ k_{31} & k_{32} & k_{33} & k_{34} & k_{35} \\ k_{41} & k_{42} & k_{43} & k_{44} & k_{45} \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix}. \tag{5.1}$$

O sistema modificado é representado em (5.2), em que:

1. à linha 1 soma-se a linha 5;
2. à coluna 1 soma-se a coluna 5;
3. $k_{55} = 1$ e todos os demais elementos da linha 5 e da coluna 5 são substituídos por zeros;
4. f_5 é substituído por zero.

Uma vez calculados os valores de u_i da Equação (5.2), atualiza-se o vetor-solução, fazendo $u_5 = u_1$. Esta estratégia possibilita a consideração das condições de contorno periódicas sem que se altere a ordem do sistema de equações e mantendo a matriz de

rigidez k simétrica.

$$\begin{bmatrix} (k_{11}+k_{51})+(k_{15}+k_{55}) & k_{12}+k_{52} & k_{13}+k_{53} & k_{14}+k_{54} & 0 \\ k_{21}+k_{25} & k_{22} & k_{23} & k_{24} & 0 \\ k_{31}+k_{35} & k_{32} & k_{33} & k_{34} & 0 \\ k_{41}+k_{45} & k_{42} & k_{43} & k_{44} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} (f_1+f_5) \\ f_2 \\ f_3 \\ f_4 \\ 0 \end{bmatrix} \quad (5.2)$$

5.2.3 Detalhamento das Rotinas Principais

O fluxo geral do programa é representado esquematicamente na figura 5.2, onde as rotinas de maior relevância para a implementação da HEA estão assinaladas com um asterisco e serão detalhadas a seguir.

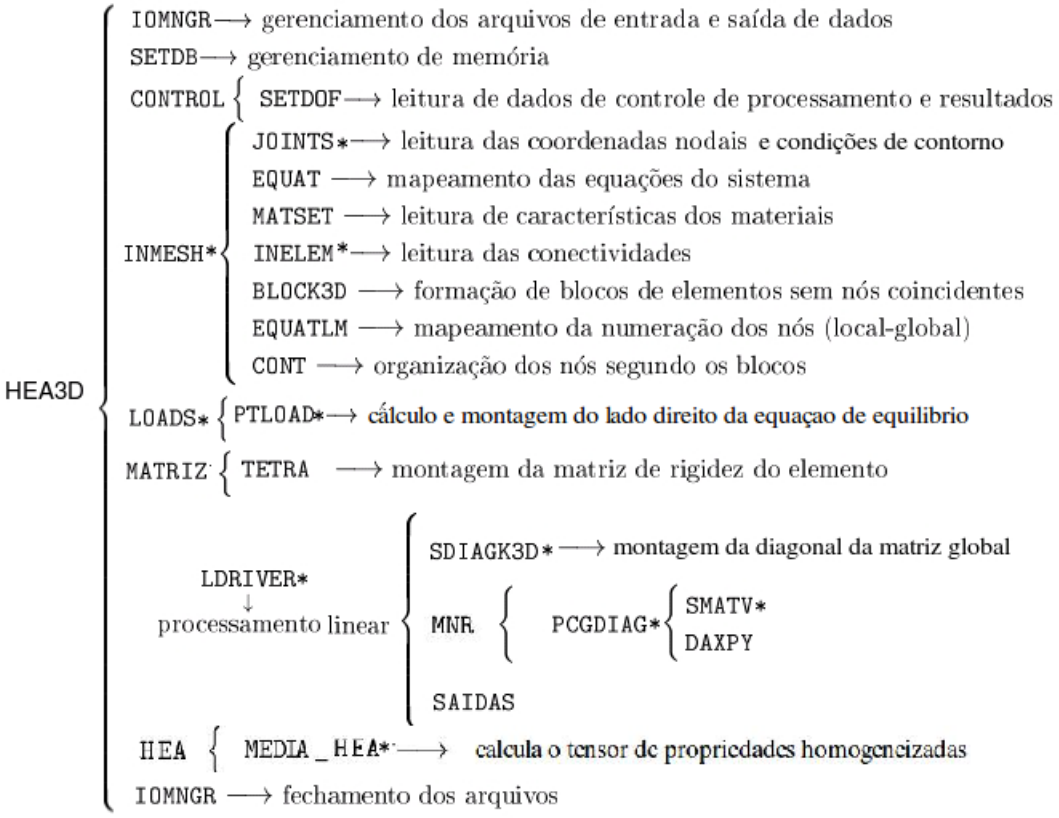


Figura 5.2: Representação esquemática do fluxo do programa `HEA3D`

Rotina `SETDB`

Trata-se de um pacote de rotinas que gerenciam a memória através da alocação dinâmica de ponteiros para estruturas de dados em tempo de execução [45].

Rotina `CONTRL`

Faz a leitura das informações que controlam a entrada de dados e o dimensionamento de várias estruturas de dados internas do programa, como o número de pontos nodais, número de elementos e quantidade de materiais com propriedades distintas, e ainda a saída de resultados em arquivo contendo dados necessários para pós-processamento dos resultados por aplicativos específicos.

Rotina `INMESH`

Rotina responsável pela leitura dos dados relativos à malha periódica, tais como as coordenadas dos pontos nodais no espaço tridimensional $x - y - z$. Essa rotina faz algumas chamadas, dentre elas:

- `JOINTS`, que identifica os conjuntos de nós periódicos, a partir das coordenadas nodais, para posterior imposição das condições de contorno. A Figura 5.3 mostra o pseudo-código da associação dos nós periódicos que tenham coordenadas x opostas;
- `INELEM`, que lê as conectividades e o número de identificação das propriedades físicas dos materiais associados aos elementos;
- `BLOCK3D`, que agrupa os elementos em blocos contendo apenas elementos que não possuam qualquer nó em comum (blocos de elementos disjuntos), procedimento necessário para a estratégia EBE empregada e para cálculos em memória compartilhada.

Rotina `LOADS`

Realiza o cálculo e montagem de F^D, que é dado pela Equação (2.38), a partir das coordenadas nodais e das propriedades dos materiais. Esse tensor de termos independentes possui então dimensões 12×6 em cada elemento e $N \times 6$ globalmente, em que N representa

Algoritmo para associação de nós periódicos

```
INICIO
inicializa(iper)
PARA i = 1 ATÉ nos FAÇA
  INICIO
    no1 ← no(i)
    SE (coordenada x do no1 = menorx) ENTÃO
      INICIO
        PARA j = 1 ATÉ nos FAÇA
            INICIO
              no2 ← no(j)
              SE (coordenada x do no2 = maiorx) E
                (no1 ≠ no2) e (no1 e no2 não são vértices) E
                (coordenada z de no1 e no2 iguais) E
                (coordenada y de no1 e no2 iguais) ENTÃO
                INICIO
                  iper(no1) ← 0
                  iper(no2) ← no1
                FIM
              FIM SE
            FIM
        FIM PARA
      FIM
    FIM SE
  FIM
FIM PARA
FIM
```

Figura 5.3: Pseudo-código da associação dos nós periódicos com coordenadas x opostas da rotina `JOINTS`, em que *nos* representa o número total de nós, *menorx* é o menor valor da coordenada x, *maiorx* é o maior valor da coordenada x e *iper* é o vetor que armazena a associação.

o número total de graus de liberdade livres do problema. Vale ressaltar que o total de graus de liberdade livres considera também os nós periódicos, tanto os mestres quanto os escravos identificados na rotina `JOINTS`. O pseudo-código do cálculo realizado na rotina `PTLOAD` está representado na Figura 5.4.

Algoritmo para o cálculo do termo independente
INICIO
PARA $i = 1$ ATÉ *nume* FAÇA
INICIO
localiza nós do elemento (no1,no2,no3,no4)
calcula coeficientes da matriz de derivação
localiza matriz constitutiva do elemento
localiza equações do elemento
PARA $k = 1$ ATÉ 4 FAÇA
INICIO
SE no_k for periódico ENTÃO
INICIO
troca equação pela do nó mestre respectivo
FIM
FIM SE
calcula F do elemento e acumula nas equações globais respectivas
FIM
FIM PARA
FIM
FIM PARA
FIM

Figura 5.4: Pseudo-código do cálculo de F^D, em que *nume* é o número total de elementos e a localização das equações do elemento se refere a localização do número da equação no sistema global.

Rotina `MATRIZ`

Realiza uma chamada a rotina `TETRA`, que calcula e armazena o triângulo superior das matrizes de rigidez de todos os elementos sendo que essas matrizes de elemento não são alteradas durante os cálculos.

Rotina `LDRIVER`

Rotina na qual é realizada a resolução do sistema de equações lineares. Essa rotina chama as rotinas:

- `SDIAGK3D`, que monta a diagonal principal da matriz global do sistema, que é utilizada como matriz precondicionadora na resolução do sistema de equações, a partir das matrizes de elementos;

- `PCGDIAG`, que resolve o sistema de equações lineares através do Método dos Gradientes Conjugados com Pré-Condicionamento Diagonal, considerando as condições de contorno periódicas nos cálculos. Essa rotina chama outras rotinas que realizam as operações de multiplicação matriz-vetor e produto interno, etapas da resolução do GC [38] sendo uma delas a `SMATV`. A rotina `SMATV` realiza a multiplicação entre as matrizes de elemento e um vetor passado como parâmetro. Essa rotina considera as condições de periodicidade na multiplicação e os cálculos sobre a matrizes de elementos são realizados por blocos, o que permite a execução paralela do algoritmo. O pseudo-código dessa rotina está representado na Figura 5.5;

- `SAIDAS`, que imprime os resultados conforme estipulado na entrada de dados pelas variáveis de controle.

Rotina `HEA`

A rotina `HEA` chama a rotina `MEDIA_HEA`, que é responsável pela operação de média para o cálculo do tensor elástico homogeneizado, dada pela Equação (2.43), utilizando os vetores resultantes da resolução do sistema pela rotina `PCGDIAG`. O pseudo-código relativo a essa rotina está representado na Figura 5.6.

Algoritmo da multiplicação das matrizes de elemento por um vetor

```
INICIO
ielm = 0
PARA i = 1 ATÉ nelblk FAÇA
  INICIO
    nvec = dimensão do bloco
    PARA k_e = ielm + 1 até ielm + nvec faça
      INICIO
        localiza nós do elemento (no1,no2,no3,no4)
        localiza equações do elemento
        PARA j = 1 ATÉ 4 FAÇA
          INICIO
            SE no_j for periódico ENTÃO
              INICIO
                troca equação pela do nó mestre respectivo
              FIM
            FIM SE
          FIM
        FIM PARA
        multiplica matriz de elemento i pelo vetor
        acumula na equação global correspondente
      FIM
    FIM PARA
    ielm = ielm + nvec
  FIM
FIM PARA
FIM
```

Figura 5.5: Pseudo-código da multiplicação das matrizes de elemento por um vetor. *ielm* é usado para controlar o índice do elemento no bloco, *nelblk* é o número de blocos disjuntos, *nvec* é a dimensão do bloco atual.

Algoritmo para o cálculo do D^h
INICIO
inicializa(D^h)
PARA $i = 1$ ATÉ $nume$ FAÇA
INICIO
localiza nós do elemento (no1,no2,no3,no4)
calcula coeficientes da matriz de derivação
calcula volume do elemento e acumula no volume total
localiza matriz constitutiva do elemento
localiza equações do elemento
$D^h \leftarrow D_e \times (I - B_e \times \chi_e)$
FIM
FIM PARA
divide termos de D^h pelo volume total
FIM

Figura 5.6: Pseudo-código do algoritmo que realiza o cálculo do tensor homogeneizado D^h em que $nume$ é o número total de elementos, D_e é a matriz de coeficientes do elemento, I_e é a matriz identidade de ordem 6, B_e é o operador diferencial e χ_e é o corretor elástico χ do elemento.

5.3 Versão paralela com OpenMP

O primeiro passo para o desenvolvimento da versão paralela do código foi escolher o modelo de programação mais adequado para a codificação. Da mesma forma que na versão 2D, optou-se pelo uso de OpenMP.

O passo seguinte foi a identificação das partes do código que possuem maior demanda computacional. Espera-se que a resolução do sistema de equações seja a parte com maiores demandas, uma vez que o número de equações é igual ao número de GLs livres da malha utilizada [46]. Como esperado, confirmou-se experimentalmente, através da instrumentação do código sequencial, que esta é a etapa do código que gasta maior tempo de processamento: por exemplo, para uma malha com 57162 GLs livres, a rotina `LDRIVER` foi responsável por $87,6\%$ do tempo de execução total e a rotina `INMESH` por aproximadamente 12%.

Com base nesta informação, utilizaram-se as diretivas de compilação do OpenMP em duas abordagens distintas na rotina `LDRIVER`: paralelização das chamadas à rotina `PCGDIAG` e paralelização da multiplicação das matrizes de elementos por um vetor na rotina `SMATV`.

Na versão sequencial, a rotina `LDRIVER` realiza 6 chamadas sequenciais à rotina `PCGDIAG`,

sendo cada uma para a resolução de uma coluna de χ. No entanto, essas resoluções são independentes e podem ser realizadas de forma simultânea. Para isso, inseriu-se um laço e paralelizaram-se as iterações utilizando a diretiva *parallel do*. As iterações são executadas por *threads* distintas que operam sobre cada uma das colunas de F^D e da solução χ. Os ponteiros desses vetores são declarados como privados utilizando a cláusula *private* de forma que somente uma coluna de F^D e a respectiva coluna de χ são visíveis para cada *thread*. As matrizes de elementos não sofrem modificações e, portanto, são compartilhadas por todas as *threads* durante a resolução paralela. Uma limitação existente nessa abordagem é o fato de que, ainda que se tenha um número grande de processadores disponíveis, somente poderão ser criadas seis *threads* simultâneas, uma para cada chamada sequencial, fazendo com que o ganho máximo nesta porção do código seja restringido a esse número.

A rotina `SMATV` é chamada pela rotina `PCGDIAG` para a multiplicação das matrizes de elemento por um vetor. A multiplicação é realizada elemento por elemento em blocos disjuntos (Figura 5.5), e pode ser paralelizada utilizando a diretiva *parallel do*. Como os blocos de dados são disjuntos, os valores resultantes dessa multiplicação ocuparão posições distintas do vetor, fazendo com que seja dispensada a definição de seções críticas no código. Espera-se que a eliminação das seções críticas tenha um grande impacto no tempo de computação do código `HEA3D`, já que seções críticas impõem a serialização da execução de um determinado trecho de código.

Embora para essa rotina seja possível explorar o número total de processadores disponíveis, uma desvantagem potencial é a necessidade de se criar e destruir, a cada iteração de `SMATV`, um grande número de threads. Esse custo adicional em termos de tempo de execução introduzido pelo uso de computação paralela é conhecido como *overhead*. Em particular, o *overhead* necessário para o gerenciamento das *threads* pode impor um grande custo a esta segunda versão paralela de `HEA3D`, reduzindo ou mesmo eliminando os ganhos em relação a primeira versão.

6 Aplicações Numéricas

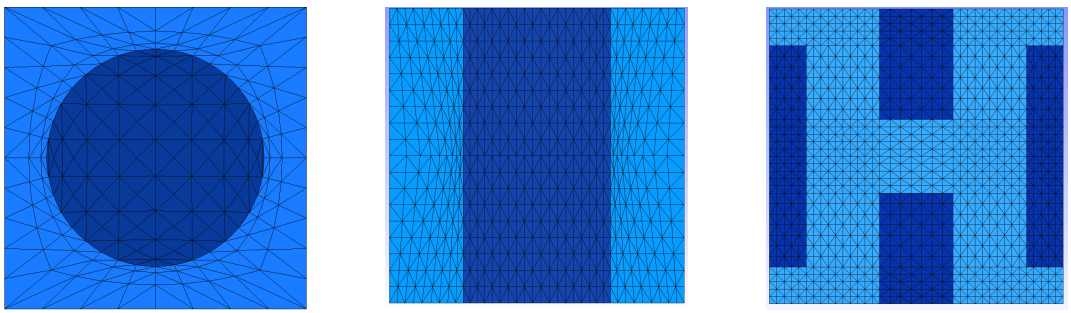
6.1 Introdução

Inicialmente, este capítulo descreve os experimentos numéricos realizados para validação dos programas `HEA2D` e `HEA3D` com base em resultados numéricos e experimentais disponíveis na literatura.

Em seguida, apresenta-se a comparação entre os desempenhos da versão sequencial e da versões paralelas do `HEA2D` e `HEA3D`.

6.2 Validação do programa HEA2D

Três tipos de células periódicas diferentes foram consideradas para representar materiais compósitos distintos para a análise em duas dimensões. A Figura 6.1 apresenta exemplos de malhas de elementos triangulares com 6 nós para as geometrias analisadas. Para a geração de todas as malhas do domínio plano foi empregado o gerador de malhas Gmsh 2.4.2 [47].



(a) inclusão circular (b) laminado (c) fibras curtas

Figura 6.1: Exemplos de malhas de uma célula periódica que possua inclusão circular (a), inclusão de fibras longas ou laminado (b) e reforço por fibras curtas (c)

A Fig. 6.1(a) é representativa de compósitos com inclusões circulares, como aqueles que contêm fibras em sua composição, representados através de um plano transversal à direção longitudinal das fibras. Considerando os componentes como isotrópicos, espera-se deste tipo de volume elementar representativo um comportamento isotrópico devido à sua geometria.

A célula da Fig. 6.1(b) representa um laminado periódico, cujo tensor de propriedades efetivas pode ser calculado analiticamente. Os compósitos laminados estruturados hierarquicamente são uma classe muito importante de compósitos macroscopicamente anisotrópicos [28]. Esta conformação estrutural tem se mostrado de imenso valor na construção de estruturas que devem alcançar propriedades efetivas prescritas [28].

A Fig. 6.1(c) representa um tipo de compósito também muito empregado, que são os reforçados com fibras curtas. O arranjo mostrado é uma proposição de como as fibras estão periodicamente dispostas na estrutura. Esta célula também tem comportamento anisotrópico e o cálculo das propriedades efetivas não é exato, dependendo fortemente do grau de refinamento da malha para que o resultado se aproxime do real.

As 3 versões do `HEA2D` – sequencial e paralelas (OpenMP e CUDA) – retornaram os mesmos resultados, o que era esperado uma vez que todas utilizam precisão simples. Os mesmos tensores de propriedades homogeneizadas foram obtidos para cada caso analisado.

Uma malha representativa de uma geometria com inclusão circular (Fig. 6.1(a)) com 3097 nós foi analisada considerando as propriedades mecânicas para validação dadas na Tabela 6.1, sendo a matriz e a inclusão isotrópicas.

Tabela 6.1: Propriedades mecânicas utilizadas para validação da aplicação.

Propriedade	valor
Módulo de elasticidade da matriz, E_m (GPa)	1
Poisson da matriz, ν_m (-)	0.03
Módulo de elasticidade da inclusão, E_i (GPa)	10
Poisson da inclusão, ν_i (-)	0.3

A partir desses parâmetros, o tensor homogeneizado obtido pelo `HEA2D` é o seguinte (6.1):

$$D^h = \begin{bmatrix} 1.205 & 0.0713 & 0.0000 \\ 0.0713 & 1.205 & 0.0000 \\ 0.0000 & 0.0000 & 0.5582 \end{bmatrix}. \tag{6.1}$$

em que se observa simetria e isotropia, como esperado para a geometria analisada.

Para as células representativas de um material laminado (Fig 6.1(b)) e de um material reforçado com fibras curtas (Fig 6.1(c)), as propriedades do material de cada fase foram retiradas da literatura. Ghosh et al. [9] empregaram a mesma técnica multiescala para analisar materiais compósitos cujas propriedades das fases são dadas na Tabela 6.2.

Tabela 6.2: Propriedades mecânicas retiradas da referência [9] para fibra de Boro e matriz de Alumínio.

Propriedade	valor
Módulo de elasticidade da matriz, E_m (GPa)	72.5
Poisson da matriz, ν_m (-)	0.33
Módulo de elasticidade da inclusão, E_i (GPa)	400
Poisson da inclusão, ν_i (-)	0.2

Para a célula unitária representativa de um material laminado, foi necessário empregar uma malha com mais de 15.000 nós para um resultado semelhante ao obtido por Ghosh et al. [9]. O tensor obtido pela referência é mostrado em (6.2) e o tensor obtido pelo `HEA2D` em (6.3).

$$D^h_{Ghosh} = \begin{bmatrix} 136.1 & 36.08 & 0.0000 \\ 36.08 & 245.8 & 0.0000 \\ 0.0000 & 0.0000 & 46.85 \end{bmatrix}, \tag{6.2}$$

$$D^h_{HEA2D} = \begin{bmatrix} 136.1 & 36.25 & 0.0000 \\ 36.08 & 245.8 & 0.0000 \\ 0.0000 & 0.0000 & 47.31 \end{bmatrix}. \tag{6.3}$$

No caso do material com inclusão de fibras curtas, foi necessário um maior refinamento para obter resultados semelhantes aos da referência [9] e, portanto foram geradas malhas de elementos finitos com refinamentos distintos. Observou-se uma tendência aos mesmos resultados da referência com o refinamento. O tensor obtido por [9] foi o seguinte:

$$D^h_{Ghosh} = \begin{bmatrix} 122.4 & 36.23 & 0.0000 \\ 36.23 & 151.2 & 0.0000 \\ 0.0000 & 0.0000 & 42.10 \end{bmatrix}. \tag{6.4}$$

Empregando uma malha com 2516 elementos o resultado obtido com o programa `HEA2D`

é dado por:

$$D^h_{HEA2D} = \begin{bmatrix} 122.3 & 36.30 & 0.0001 \\ 36.29 & 151.5 & 0.0004 \\ 0.0005 & 0.0009 & 42.22 \end{bmatrix}. \tag{6.5}$$

O tensor resultante para uma malha com 3780 elementos :

$$D^h_{HEA2D} = \begin{bmatrix} 122.3 & 36.31 & 0.0001 \\ 36.30 & 151.4 & 0.0001 \\ 0.0001 & 0.0001 & 42.19 \end{bmatrix}. \tag{6.6}$$

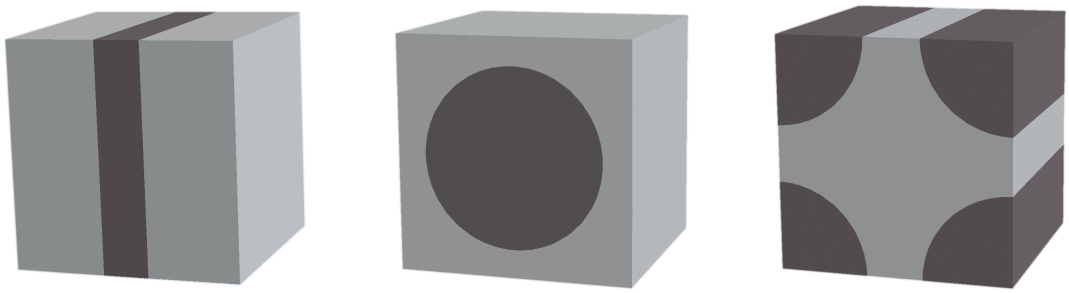
E, por fim a malha mais refinada analisada para essa geometria com 7440 elementos gerou seguinte o tensor de propriedades homogeneizadas:

$$D^h_{HEA2D} = \begin{bmatrix} 122.3 & 36.32 & 0 \\ 36.31 & 151.3 & 0 \\ 0 & 0 & 42.16 \end{bmatrix}. \tag{6.7}$$

Observa-se uma tendência a convergência dos elementos de D^h aos valores utilizados para comparação com o aumento do número de elementos da malha. Há simetria e os elementos D^h_{13}, D^h_{31}, D^h_{23}, D^h_{32}, que para as malhas anteriores apresentaram valores diferentes de zero, passam a ser nulos conforme esperado.

6.3 Exemplos para validação do programa HEA3D

As versões sequencial e paralela forneceram resultados idênticos para todas as análises envolvendo as três células periódicas cúbicas mostradas na Figura 6.2. Trata-se de células compostas por duas fases: matriz e inclusão. A Figura 6.2(a) consiste em uma célula periódica relativa a uma estrutura laminada. As Figuras 6.2(b) e 6.2(c) representam o mesmo tipo de material compósito reforçado com fibras longas de seção transversal circular. Estes três exemplos foram extraídos da referência [7] para validação do programa e os detalhes de cada uma das aplicações são apresentados a seguir. As malhas de elementos finitos empregadas para análise dos domínios 3D foram geradas a partir do Cubit 11.0 [48] e para a visualização dos resultados foi empregado o Gmsh 2.4.2 [47].



(a) Célula A (b) Célula B (c) Célula C

Figura 6.2: Células unitárias representativas da microestrutura periódica de materiais compósitos: (a) laminados e (b) e (c) reforçados com fibras longas.

6.3.1 Célula A

A célula mostrada na Figura 6.2(a) representa um material laminado bifásico cujos valores limites dos módulos de elasticidade homogeneizados podem ser calculados analiticamente através da regra das misturas [49], dada pelas Equações (6.8) e (6.9):

$$E_{max} = f_i \times E_i + (1 - f_i) \times E_m, \tag{6.8}$$

$$E_{min} = \frac{(E_i \times E_m)}{(f_i \times E_m + (1 - f_i) \times E_i)}. \tag{6.9}$$

onde: E_{max} e E_{min} são, respectivamente, os valores máximo e mínimo do módulo de elasticidade do compósito, f_i é a fração volumétrica da inclusão, E_i é o módulo de Young da inclusão e E_m é o módulo de Young da matriz.

No caso em que os coeficientes de Poisson sejam nulos, E_{max} e E_{min} são os valores exatos dos módulos de Young nas direções 1,2 e 3 da célula mostrada na Figura 6.3, que mostra o aspecto típico das malhas de elementos finitos utilizadas.

Para o modelo aqui analisado, foi adotada uma fração volumétrica de 20% para a inclusão e as propriedades mecânicas utilizadas para a validação foram: $E_i = 10$ GPa e $E_m = 1$ GPa e coeficientes de Poisson nulos para os dois materiais, adotados como isotrópicos. Daí espera-se que:

$$\begin{aligned} E_1 &= E_{min} = 1.2195\, GPa \quad e \\ E_2 &= E_3 = E_{max} = 2.80\, GPa. \end{aligned}$$

O tensor elástico homogeneizado obtido com o programa `HEA3D` a partir de uma malha

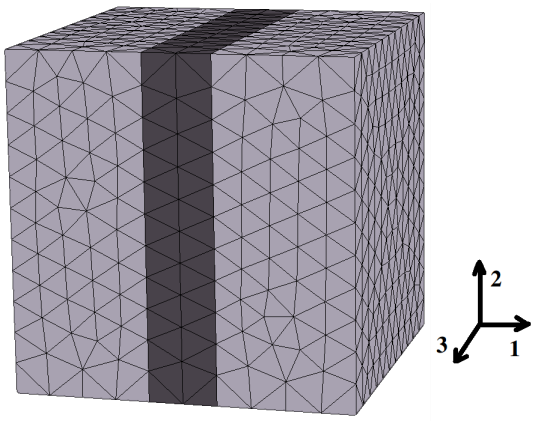


Figura 6.3: Exemplo de malha de elementos finitos tetraédricos lineares a partir da geometria representativa de um material compósito laminado.

com 5178 elementos e 3231 GLs é apresentado na matriz 6.10:

$$D_{h^A} = \begin{bmatrix} 1.2195 & 0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 \\ -0.0000 & 2.8000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 \\ -0.0000 & -0.0000 & 2.8000 & 0.0000 & -0.0000 & -0.0000 \\ -0.0000 & 0.0000 & 0.0000 & 0.6098 & -0.0000 & 0.0000 \\ 0.0000 & -0.0000 & -0.0000 & -0.0000 & 1.4000 & 0.0000 \\ 0.0000 & -0.0000 & -0.0000 & 0.0000 & 0.0000 & 0.6098 \end{bmatrix}. \quad (6.10)$$

Onde se observa que os coeficientes do tensor obtido para a célula A são iguais aos valores analíticos: $E_1 = E_{min} = D_h^A$ [1,1]; $E_2 = E_3 = E_{max} = D_h^A$ [2,2] $= D_h^A$ [3,3]. O resultado é exato, ainda que a malha de elementos finitos não seja muito refinada.

6.3.2 Célula B

A célula periódica B, mostrada na Figura 6.2(b), é representativa de um material com a estrutura repetitiva mostrada na Figura 6.4. Foram utilizados para comparação os resultados apresentados por Oliveira et al. [7], em que algumas implementações numéricas desta técnica com diferentes tipos de elementos finitos são avaliadas.

A fração volumétrica da inclusão (47%) e as propriedades mecânicas adotadas para cada um dos componentes da célula são as mesmas empregadas na referência, mostradas na Tabela 6.3. Este exemplo foi empregado para validação dos programas aqui desenvolvidos, uma vez que a referência [7] traz valores experimentais e numéricos

para o tensor elástico homogeneizado de um compósito com as mesmas propriedades aqui adotadas.

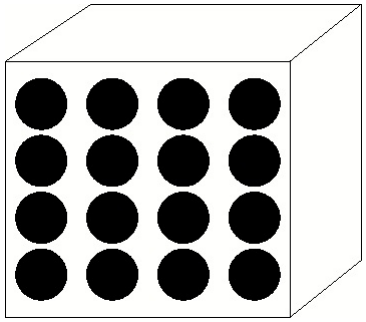


Figura 6.4: Representação de um material composto por matriz reforçada com fibras longas.

Tabela 6.3: Propriedades mecânicas utilizadas para validação da aplicação.

Propriedade	valor
Módulo de elasticidade da inclusão, E_i (GPa)	379.3
Poisson da inclusão, ν_i (-)	0.1
Módulo de elasticidade da matriz, E_m (GPa)	68.3
Poisson da matriz, ν_m (-)	0.3

As matrizes 6.11 e 6.12 apresentam, respectivamente: o tensor elástico da inclusão (D_i) e o tensor elástico da matriz (D_m), ambos isotrópicos.

$$D_i = \begin{bmatrix} 387.9 & 43.10 & 43.10 & 0 & 0 & 0 \\ 43.10 & 387.9 & 43.10 & 0 & 0 & 0 \\ 43.10 & 43.10 & 387.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 172.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 172.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 172.4 \end{bmatrix}, \tag{6.11}$$

$$D_m = \begin{bmatrix} 91.94 & 39.40 & 39.40 & 0 & 0 & 0 \\ 39.40 & 91.94 & 39.40 & 0 & 0 & 0 \\ 39.40 & 39.40 & 91.94 & 0 & 0 & 0 \\ 0 & 0 & 0 & 26.27 & 0 & 0 \\ 0 & 0 & 0 & 0 & 26.27 & 0 \\ 0 & 0 & 0 & 0 & 0 & 26.27 \end{bmatrix}. \tag{6.12}$$

Tabela 6.4: Limite máximo e mínimo para o módulo de elasticidade segundo a regra das misturas para fração volumétrica 0.47% e propriedades das fases dadas pela tabela 6.3.

Propriedade	valor (GPa)
E_{max}	214.47
E_{min}	111.12

A Figura 6.5 mostra uma das malhas de elementos finitos tetraédricos lineares empregadas para modelar o material. Sabendo que o compósito resultante desta associação é transversalmente isotrópico [50], a matriz elástica homogeneizada a calcular apresenta a forma geral mostrada em 6.13.

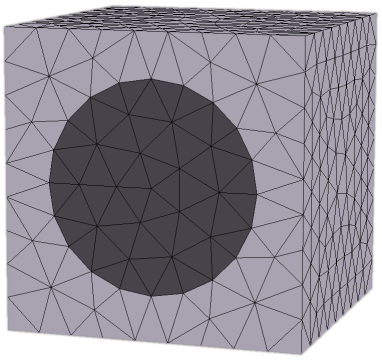


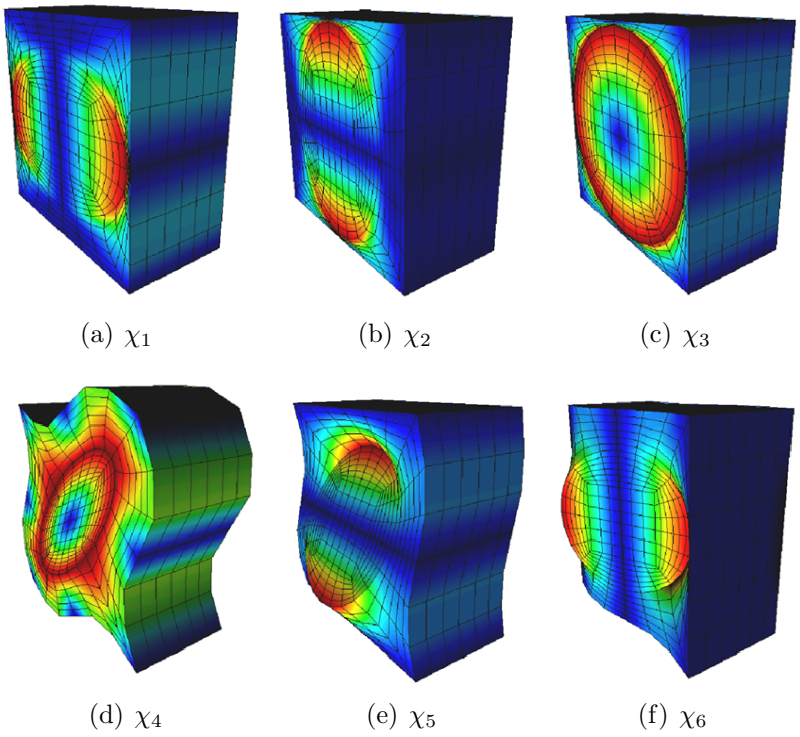
Figura 6.5: Exemplo de malha de elementos finitos tetraédricos lineares a partir da geometria representativa de um material reforçado por fibras longas.

$$D_h = \begin{bmatrix} a & b & c & 0 & 0 & 0 \\ b & a & c & 0 & 0 & 0 \\ c & c & d & 0 & 0 & 0 \\ 0 & 0 & 0 & e & 0 & 0 \\ 0 & 0 & 0 & 0 & f & 0 \\ 0 & 0 & 0 & 0 & 0 & f \end{bmatrix}. \tag{6.13}$$

Os valores de E_{max} e E_{min} dados pela regra das misturas (Equações (6.8) e (6.9)) são apresentados na Tabela 6.4, servindo para verificar se os valores aproximados são adequados.

Autodeformações da Célula B

Os 6 vetores que compõem o corretor elástico (χ) – obtidos como soluções do sistema de equações (2.39) – podem ser representados na forma de deslocamentos. Segundo Oliveira et al. [7], cada uma das 6 deformadas da célula periódica, denominadas autodeformações, é uma medida da heterogeneidade do material. Para fins de análise qualitativa dos resultados aqui obtidos, as autodeformações obtidas por Oliveira et al. [7] a partir de uma malha de elementos hexaédricos são apresentadas na Figura 6.6, em que χ_i, $i = 1, 2, \ldots, 6$, é a coluna i do corretor elástico χ. As autodeformações obtidas a partir do programa



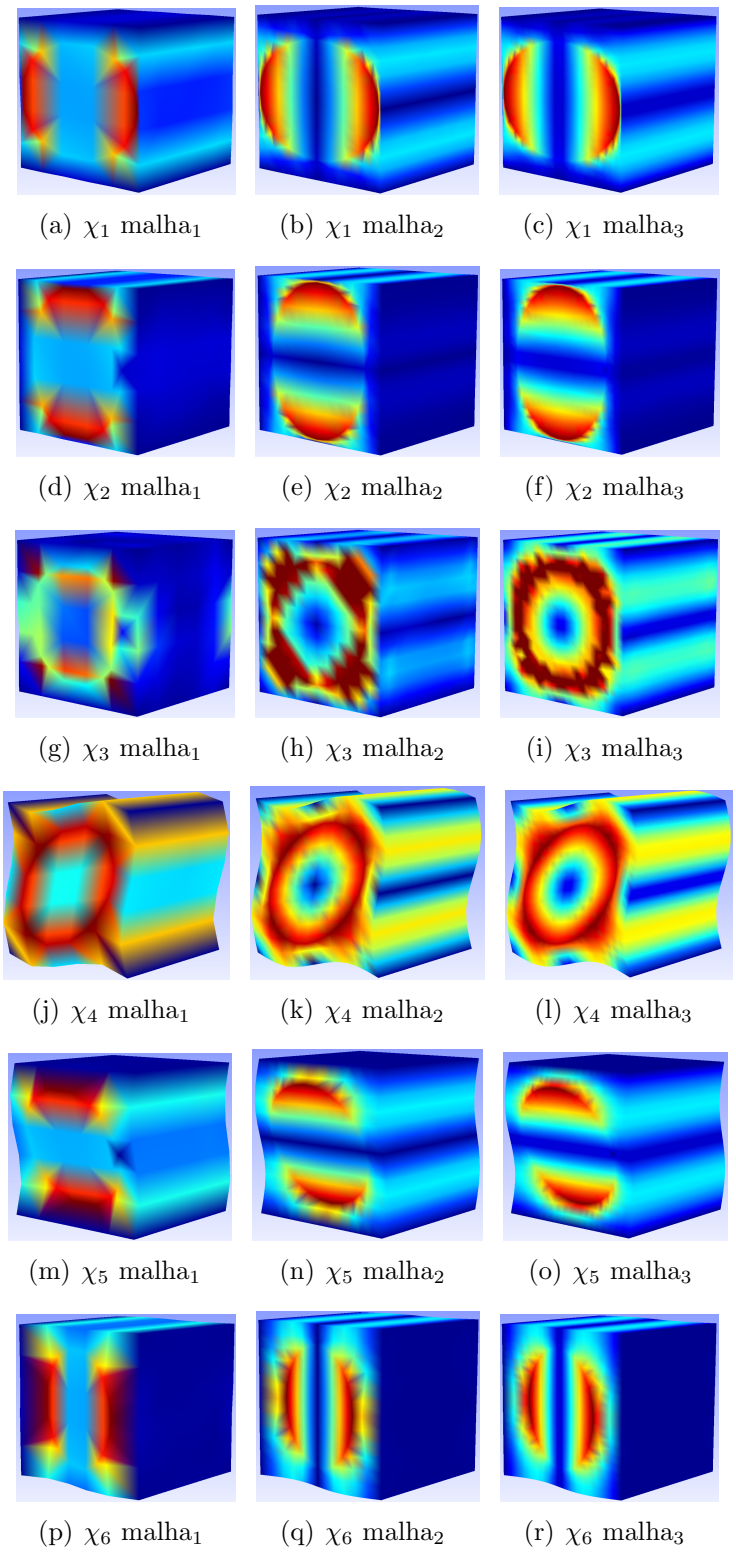
(a) χ_1 (b) χ_2 (c) χ_3

(d) χ_4 (e) χ_5 (f) χ_6

Figura 6.6: Autodeformações obtidas com malha de hexaedros na referência [7].

`HEA3D` são apresentadas na Figura 6.7. São mostrados os resultados para 3 malhas de elementos tetraédricos lineares com refinamentos distintos, com 1407, 13884 e 57885 GLs respectivamente. A comparação é meramente qualitativa, uma vez que se trata de malhas com graus de refinamentos e elementos distintos dos empregados por Oliveira et al. [7].

A Figura 6.7 mostra que, na medida em que se aumenta o refinamento das malhas, os aspectos das autodeformações aproximam-se dos obtidos na referência [7].



(a) χ_1 malha$_1$ (b) χ_1 malha$_2$ (c) χ_1 malha$_3$

(d) χ_2 malha$_1$ (e) χ_2 malha$_2$ (f) χ_2 malha$_3$

(g) χ_3 malha$_1$ (h) χ_3 malha$_2$ (i) χ_3 malha$_3$

(j) χ_4 malha$_1$ (k) χ_4 malha$_2$ (l) χ_4 malha$_3$

(m) χ_5 malha$_1$ (n) χ_5 malha$_2$ (o) χ_5 malha$_3$

(p) χ_6 malha$_1$ (q) χ_6 malha$_2$ (r) χ_6 malha$_3$

Figura 6.7: Autodeformações calculadas pelo HEA3D com malhas de elementos tetraédricos lineares: malha$_1$ com 1407 GLs, malha$_2$ com 13884 GLs e malha$_3$ com 57885 GLs.

Estudo de Convergência da Célula B

As versões do programa `HEA3D` forneceram respostas idênticas para o tensor homogeneizado D^h, que foi determinado empregando-se malhas com graus variados de

refinamento, de modo a permitir um estudo da convergência do resultado. A exemplo do observado na referência [7], ocorreu convergência simultânea de todos os componentes de D^h em todos os estudos e, portanto, adotou-se a norma de Frobenius de D^h, dada pela Equação (6.14) [51], para avaliar a convergência do resultado em função do número de graus de liberdade das malhas.

$$\|D^h\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(d_{ij}^h\right)^2} \tag{6.14}$$

onde D^h é o tensor homogeneizado, n é a ordem da matriz e d_{ij}^h é o termo ij do tensor D^h. Na Figura 6.8, que relaciona $\|D^h\|_F$ e o número de GLs, nota-se que a variação da norma é pequena quando o número de GLs é maior ou igual a 30939. A malha mais refinada que foi analisada para essa célula, possui 57855 GLs: por uma limitação do gerador de malhas empregado, não foi possível analisar um modelo mais refinado com a condição de periodicidade necessária para esta aplicação. Nota-se a convergência em função do número de GLs, mas não foi possível atingir a estabilização do resultado, devido a problemas relacionados à geração da malha periódica. Em Oliveira et al. [7], foram mostrados resultados numéricos para malhas com mais de 100000 GLs usando tetraedros lineares.

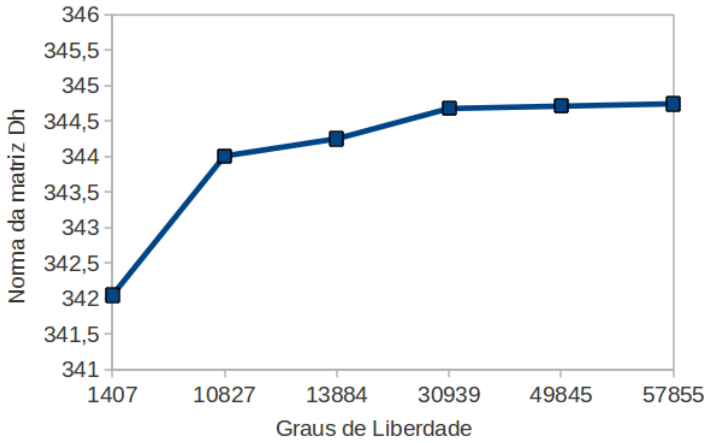


Figura 6.8: Evolução da norma de D^h com o número de graus de liberdade para a célula B.

A Tabela 6.5 compara os valores das propriedades elásticas homogeneizadas fornecidas pelo `HEA3D` com resultados analíticos (A), numéricos (N) e experimentais (E) disponíveis nas referências [7, 52, 53].

A matriz 6.15 mostra o tensor de propriedades homogeneizadas da referência [7] e a

Tabela 6.5: Comparativo de resultados numéricos, analíticos e experimentais para a Célula B.

Resultados	E_{11} (GPa)	E_{22} (GPa)	G_{12} (GPa)	G_{23} (GPa)	ν_{12} $(-)$	ν_{23} $(-)$
N − HEA3D	209.83	140.1	52.85	44.83	0.198	0.259
N − Oliveira [7]	214.6	144.5	54.7	46.2	0.19	0.25
N − Xia et al. [52]	214	143	54.2	45.7	0.195	0.253
N − Sun and Vaidya [53]	215	144	57.2	45.9	0.19	0.29
A − Sun and Chen [7]	214	135	51.1	−	0.19	−
A − Chamis [7]	214	156	62.6	43.6	0.20	0.31
A − Whitney and Riley [7]	215	123	53.9	−	0.19	−
E − Kenaga et al. [7]	216	140	52	−	0.29	−

matriz 6.16 mostra o tensor de propriedades homogeneizadas da célula B, obtido a partir da mesma malha de elementos finitos utilizada na comparação, com 57855 GLs.

$$D_{h^{ref}} = \begin{bmatrix} 160.8106 & 45.2106 & 39.1440 & 0 & 0 & 0 \\ 45.2106 & 160.8106 & 39.1440 & 0 & 0 & 0 \\ 39.1440 & 39.1440 & 229.4797 & 0 & 0 & 0 \\ 0 & 0 & 0 & 46.2000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 54.7000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 54.7000 \end{bmatrix}, \quad (6.15)$$

$$D_{h^B} = \begin{bmatrix} 157.3623 & 46.0804 & 40.2894 & 0.0185 & -0.0004 & -0.0008 \\ 46.0804 & 157.3624 & 40.2893 & 0.0184 & -0.0009 & -0.0005 \\ 40.2894 & 40.2894 & 225.7904 & 0.0005 & 0.0185 & 0.0185 \\ 0.0185 & 0.0184 & 0.0005 & 44.8352 & -0.0007 & -0.0006 \\ -0.0004 & -0.0009 & -0.0001 & -0.0007 & 52.8584 & -0.0010 \\ -0.0008 & -0.0005 & -0.0004 & -0.0006 & -0.0010 & 52.8582 \end{bmatrix}, \quad (6.16)$$

Os valores calculados de E_{11} e E_{22} a partir do tensor homogeneizado D_h^B estão dentro do intervalo limitado pela regra das misturas (Tabela 6.4). As discrepâncias observadas podem ser justificadas pelo tipo de elemento finito aqui empregado – o tetraedro linear – e pelo fato da malha aqui empregada ser menos refinada do que as empregadas nas demais análises numéricas. Levando em consideração a limitação do refinamento da malha

empregada, pode-se considerar os resultados como satisfatórios. Nota-se que a matriz D_h^B (6.16) é bastante similar à de referência (6.15).

6.3.3 Célula C

O material compósito mostrado na Figura 6.4 pode também ser representado pela célula C, mostrada na Figura 6.2(c). Empregando para este exemplo a mesma fração volumétrica para a inclusão (47%) e as mesmas propriedades mecânicas para as duas fases (Tabela 6.3), deve-se então obter como resposta o mesmo tensor homogeneizado D^h (6.16). A Figura 6.9 mostra o aspecto típico das malhas de elementos finitos usadas para este modelo.

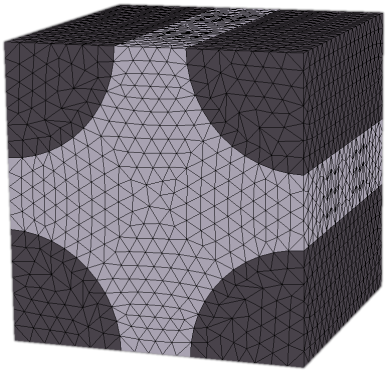
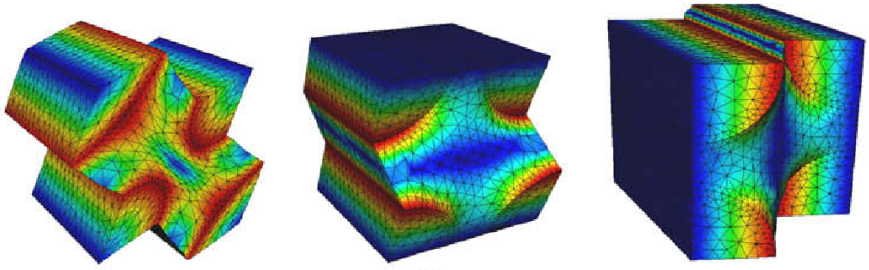


Figura 6.9: Malha de elementos finitos tetraédricos lineares (fração volumétrica da inclusão 47%).

Autodeformações da Célula C

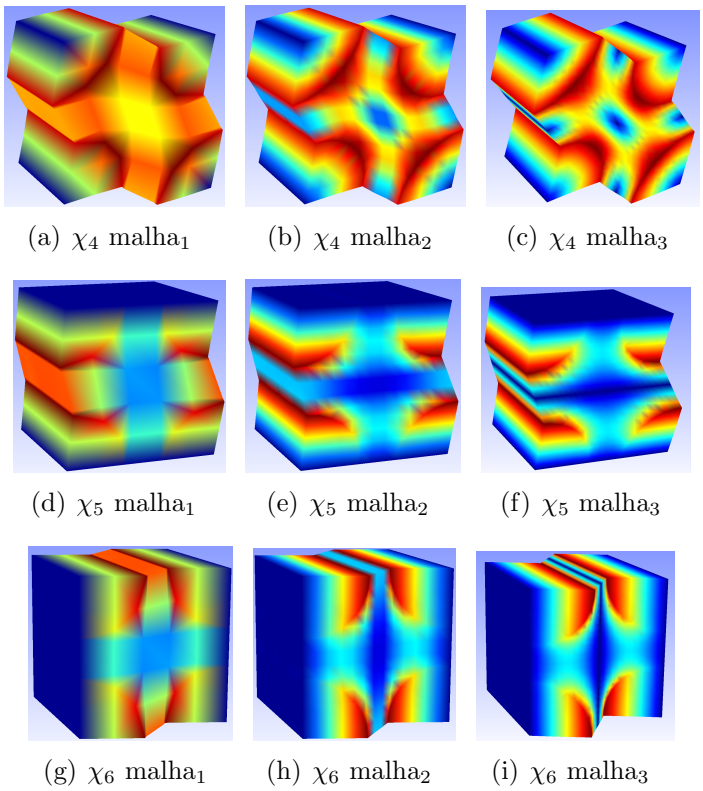
A Figura 6.10 mostra as autodeformações apresentadas na referência [7] a partir de uma malha de elementos tetraédricos lineares. São apresentadas aqui as 3 últimas colunas de χ a fim de permitir uma avaliação qualitativa dos resultados via `HEA3D`.



(a) χ_4 (b) χ_5 (c) χ_6

Figura 6.10: Autodeformações obtidas com malha de tetraedros lineares na referência [7].

As autodeformações obtidas a partir do programa `HEA3D` são apresentadas na Figura 6.11, em que a evolução das visualizações das 3 últimas colunas de χ é mostrada para malhas de elementos tetraédricos lineares com refinamentos distintos. Consideraram-se 3 malhas: malha$_1$ com 1434 GLs, malha$_2$ com 17172 GLs e malh$_3$ com 76131 GLs. Claramente nota-se que quanto maior o refinamento, maior a semelhança entre os aspectos das autodeformações aqui obtidas e os fornecidos na referência.



(a) χ_4 malha$_1$ (b) χ_4 malha$_2$ (c) χ_4 malha$_3$

(d) χ_5 malha$_1$ (e) χ_5 malha$_2$ (f) χ_5 malha$_3$

(g) χ_6 malha$_1$ (h) χ_6 malha$_2$ (i) χ_6 malha$_3$

Figura 6.11: Visualização das 3 últimas colunas de χ para malha$_1$ com 1434 GLs, malha$_2$ com 17172 GLs e malha$_3$ com 76131 GLs (elementos tetraédricos lineares `HEA3D`).

Estudo de Convergência da Célula C

Na Figura 6.12, que relaciona $\|D^h\|_F$ (Equação 6.14) e o número de GLs, nota-se que a variação da norma é suficientemente pequena quando o número de GLs é maior ou igual a 49887.

A matriz (6.17) mostra o tensor homogeneizado obtido empregando-se 141436 elementos finitos e 76131 GLs.

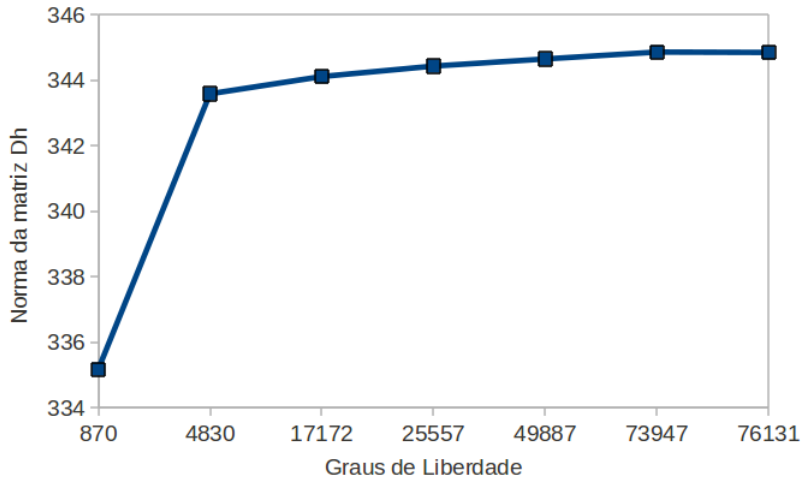


Figura 6.12: Evolução da norma de D_h com o número de GLs para a célula B.

$$D_{hC} = \begin{bmatrix} 157.4007 & 46.1058 & 40.2919 & 0.0139 & 0.0002 & -0.0008 \\ 46.1058 & 157.4025 & 40.2928 & 0.0137 & -0.0010 & 0.0002 \\ 40.2919 & 40.2919 & 225.8840 & 0.0003 & 0.0139 & 0.0139 \\ 0.0139 & 0.0137 & 0.0003 & 44.8376 & -0.0005 & -0.0001 \\ 0.0002 & -0.0010 & 0.0005 & -0.0005 & 52.8738 & -0.0001 \\ -0.0008 & 0.0002 & -0.0009 & -0.0001 & -0.0001 & 52.8717 \end{bmatrix}, \quad (6.17)$$

Conforme esperado, o resultado é praticamente igual ao obtido pra a Célula B (6.16), o que também serve para validar o modelo aqui implementado.

6.3.4 Considerações Sobre os Resultados Numéricos

As análises realizadas com o `HEA3D` mostram a influência da qualidade da malha para a obtenção de resultados adequados. Dada a particularidade imposta pelas condições de contorno periódicas, as análises aqui efetuadas de forma geral empregaram malhas menos refinadas do que as usadas nas referências, ainda assim os resultados são bastante semelhantes.

Levando em conta as dificuldades encontradas na geração de malhas, consideram-se os resultados obtidos para a célula B como satisfatórios e coerentes. Os aspectos das autodeformações e a convergência alcançada na célula C com malhas mais refinadas reforçam esta observação.

Para a célula A, obteve-se resultado exato – idêntico aos valores analíticos – por se tratar de um modelo que demanda menor grau de refinamento. As células B e C, que representam o mesmo compósito, geraram tensores elásticos homogeneizados muito semelhantes, conforme esperado.

6.4 Experimentos Computacionais

Utilizou-se para avaliar o ganho obtido com a implementação paralela o fator de aceleração (*speedup*), que é uma medida de desempenho relativo de quanto a resolução do problema pode ser acelerada em um multiprocessador [31]. O ganho pode ser calculado utilizando a Equação (6.18):

$$S(p) = \frac{t_s}{t_p}, \tag{6.18}$$

em que t_s é o tempo de execução sequencial e t_p é o tempo de execução paralela utilizando p processadores. Para esses cálculos foram utilizadas as médias dos tempos de execução sequencial e paralelos, respectivamente. O ganho máximo relativo, que seria igual a p utilizando p processadores, é chamado de aceleração linear. No entanto, nem todos os códigos são passíveis de serem completamente paralelizados, de modo a obter acelerações lineares. De fato, a grande maioria dos códigos é composta por parte que necessariamente devem ser executadas de modo sequencial, enquanto outras partes podem ser executadas de modo paralelo. A partir de tal observação, Amdahl propôs a Equação (6.19) [31] para que seja possível avaliar a aceleração máxima que pode ser obtida por uma aplicação, considerando quanto de seu código é executado de forma sequencial. Seja f a fração de tempo de execução sequencial de um código. Assim, a aceleração máxima S obtida quando p processadores são empregados pode ser calculada por:

$$S(p) = \frac{p}{1 + (p-1) \times f}, \tag{6.19}$$

6.4.1 Experimentos Computacionais HEA2D

Os experimentos foram realizados no seguinte ambiente computacional: processador Intel Core i7-860 1.2 GHz, com 8 GB RAM, cache nível 2 com 8 MB e uma placa GTX 285. O sistema operacional instalado é o Linux versão 2.6.31.

Os ganhos computacionais obtidos com as versões paralelas (OpenMP e CUDA), são

apresentados nas Tabelas 6.6 e 6.7. Foi utilizado para comparação o fator de aceleração dado pela Equação (6.18).

A aceleração de cada versão foi obtida a partir da divisão do tempo de execução sequencial da aplicação `HEA2D` pelo tempo da versão paralela, sendo que para a versão com OpenMP executando na CPU e a versão CUDA executando na GPGPU. Mesmo que se tratem de arquiteturas distintas, os ganhos da versão OpenMP são apresentados para efeitos de comparação. Para a comparação de desempenho, considerou-se para a versão paralela com OpenMP apenas a abordagem em que foi parelizada a resolução do sistema de equações, por ser esta a que obteve melhores resultados [23].

Tabela 6.6: Ganhos obtidos para a malha circular (primeira coluna) e para as malhas representativas do material laminado com diferentes refinamentos (número de nós).

	3097 (circular)	2775	9171	10877	15445
$CUDA$	1.31	2.50	6.02	6.60	6.76
$OpenMP$	1.90	1.94	2.33	2.37	2.37

Tabela 6.7: Ganhos obtidos para as malhas representativas de um material com inclusão de fibras curtas com refinamentos variados (número de nós).

	5175	7735	15125	21903	33485	45353
$CUDA$	6.33	8.50	9.63	17.02	12.44	19.02
$OpenMP$	2.22	2.35	2.37	2.43	2.47	2.54

Como indicado nas Tabelas 6.6 e 6.7, a versão OpenMP obteve melhor ganho sobre a versão CUDA somente para o caso da malha circular. Para o caso da malha laminada com 2775 nós, a versão OpenMP apresentou ganho similar ao da versão da GPGPU. Esse fato se deve ao reduzido número de nós em ambas as malhas utilizadas, uma vez que trabalhar na GPU se torna ineficiente quando somente pequenas porções de dados são transferidos para o dispositivo e os *kernels* são chamados frequentemente. Dessa forma, a inicialização e as transferências de dados para a GPU e a recuperação dos dados de volta a partir da GPU consomem a maior parte do tempo de processamento. Para todas as outras malhas, a versão CUDA teve um desempenho pelo menos 2.5 vezes superior ao da versao OpenMP.

Considerações Sobre os Resultados via HEA2D

O melhor resultado, em termos de ganho de tempo de computação, foi a aceleração de 19 vezes obtida ao utilizar uma malha com 45353 nós. Pode-se observar que, quanto mais

refinada for a malha e, consequentemente quanto maior for o problema a ser resolvido, maior é o ganho que pode ser obtido com a versão CUDA.

6.4.2 Experimentos Computacionais HEA3D

O ambiente utilizado para a realização dos experimentos foi o seguinte: máquinas com dois processadores Intel Xeon E5620, cada processador com quatro núcleos, totalizando assim oito núcleos físicos. Cada núcleo possui 64 KB de cache de nível 1 (32KB de dados + 32KB de instruções) e 256KB de cache de nível 2. Os quatro núcleos compartilham 12MB de cache de nível 3 e 12 GB de memória principal. Os processadores possuem tecnologia $Hyper-Threading$(HT), o que faz com que os processadores tenham suporte para executar até 16 *threads* simultaneamente. O sistema operacional instalado no computador é o Linux versão 2.6.18-194.17.4 de 64 bits.

Conforme descrito no capítulo anterior, foram utilizadas duas abordagens para a implementação da versão paralela com OpenMP:

1. chamadas à rotina `PCGDIAG` de forma a resolver cada coluna do sistema de equações simultaneamente;

2. multiplicação matriz-vetor na rotina `SMATV` por blocos de elementos disjuntos;

Para facilitar o entendimento dos resultados, chamaremos de versão paralela PD (de PCGDIAG) a primeira abordagem para a implementação da versão paralela de OpenMP e de versão paralela ST (de SMATV) a segunda abordagem.

Para analisar o desempenho da aplicação paralela, a seguinte metodologia foi empregada: três tipos de diferentes de células, A(Figura 6.2(a)), B(Figura 6.2(b)) e C(Figura 6.2(c)), com 5 níveis distintos de refinamento, foram utilizadas como conjunto de teste (*benchmarks*). O conjunto de testes foi utilizado como entrada para cada uma das versões paralelas PD e ST do código. Para cada versão paralela do código e para cada uma das entradas, foram coletados os tempos de execução da aplicação com o aplicativo *time* do Linux. O mesmo foi feito com a versão sequencial do código. A média dos tempos coletados é então calculada, bem como a variância dos resultados. A variância observada ficou abaixo de 5.9%.

Ainda que a célula A representativa de um material laminado não tenha demandado malhas muito refinadas para que o resultado esperado fosse alcançado, o uso de

computação paralela se justifica. Neste caso a versão paralela do código garante o aproveitamento dos recursos computacionais disponíveis no computador.

Resultados Obtidos Pelas Versões Paralelas do Código

Serão apresentados resultados empregando apenas o OpenMP, devido a limitações encontradas na implementação da versão paralela utilizando CUDA para FORTRAN.

A Tabela 6.8 apresenta os ganhos para as malhas representativas das células A, B e C com diferentes refinamentos, obtidos quando as entradas são executadas com a versão paralela PD do código. Todos os experimentos foram realizados com 6 processadores, já que este é o número de *threads* criadas por esta versão paralela do código. Em relação à célula A (Figura 6.2(a)), a fração não-paralelizável de tempo para a malha A_1, por exemplo, é de 2% do tempo total de processamento da aplicação. Dessa forma, utilizando-se a Lei de Amdahl (Equação 6.19), para 6 processadores, conclui-se que é possível obter para essa malha ganho de até 5.5 vezes no tempo de processamento. Como pode ser visto na Tabela 6.8, observou-se um ganho de 5.3 vezes com esta abordagem, bem próximo ao máximo ganho possível.

Tabela 6.8: Acelerações (*Sps*) para as malhas representativas das células A, B e C com refinamentos variados, obtidas pelas versão PD

Malhas	GLs	*Sps*	Malhas	GLs	*Sps*	Malhas	GLs	*Sps*
A_1	3231	5.3	B_1	10827	4.9	C_1	17172	4.6
A_2	4590	5.2	B_2	13884	5.1	C_2	25557	4.3
A_3	6264	5.1	B_3	30939	4.9	C_3	49887	4.0
A_4	10764	4.8	B_4	49845	4.3	C_4	73947	3.6
A_5	13668	5.2	B_5	57855	3.9	C_5	76131	3.5

Um efeito interessante que pode ser observado na Tabela 6.8 é uma tendência a redução na aceleração a medida que o número de GLs cresce. A princípio esse efeito não seria esperado já que, com malhas maiores, maior seria o custo computacional para resolver os sistemas de equações. Entretanto as operações de entrada de dados também aumentam a medida que malhas maiores são lidas do arquivo de entrada. Como o tempo total de computação considera essas operações, que são realizadas sequencialmente, o resultado é, pela Equação (6.19), uma redução na aceleração. Como exemplo, para a malha C_5, o trecho do código que pode ser paralelizado representa aproximadamente 89.8% do tempo total de execução, enquanto aproximadamente 10% do tempo correspondem às rotinas que

realizam a leitura e verificação dos dados da malha. Com isso a aceleração máxima que pode-se alcançar para a malha C_5, ainda segundo a Equação (6.19) é 3.98 e a alcançada com essa abordagem foi de 3.5 (Tabela 6.8).

Para os testes da versão ST do código foram variadas, além das entradas utilizadas, o número de *threads* criadas durante a execução. Limitou-se a execução a 16 *threads* por ser esse o número máximo de *threads* suportada para execução simultânea pela arquitetura. Caso mais *threads* fossem criadas, ocorreria uma disputa pelo uso do processador entre elas, o que acarretaria em um aumento do tempo de processamento.

As Tabelas 6.9, 6.10 e 6.11 mostram as acelerações obtidas utilizando 2, 4, 6, 8 e 16 *threads* para as malhas representativas da célula A, B e C, respectivamente, apesar de experimentos terem sido realizados variando o número de *threads* criadas em uma unidade entre 1 e 16. Os demais valores não foram listados porque não se diferenciam de modo substancial dos valores apresentados.

Tabela 6.9: Ganhos obtidos para as malhas representativas de um material laminado (Célula A) com refinamentos variados e diferentes números de *threads*, para a versão ST do código paralelo.

Malhas	*Sps* 2	*Sps* 4	*Sps* 6	*Sps* 8	*Sps* 16
A_1	1.6	1.6	2.1	2.2	2.1
A_2	1.1	1.7	2.2	2.5	2.4
A_3	1.0	1.7	2.2	2.5	2.4
A_4	1.7	2.7	3.9	4.0	3.9
A_5	1.4	2.2	2.9	3.5	3.3

De forma geral observa-se aumento na aceleração com o aumento do número de *threads*. No entanto, o ganho com 16 *threads* não supera o ganho com 8 *threads* em nenhum dos casos analisados para a célula A. Este resultado pode ter ocorrido porque o processador possui, de fato, apenas 8 núcleos físicos. A tecnologia HT permite uma troca rápida de contexto entre as *threads*, mas, de fato, duas *threads* irão disputar os recursos de um único núcleo de computação. A troca de contexto entre as *threads* ocorre, por exemplo, quando o núcleo de processamento sofre falhas caras, que demandem vários ciclos, como ocorre quando acessos à cache L3, memória principal e operações de entrada e saída são realizados. Como os GLs da célula A são menores do que as demais malhas, os sistemas de equações e as estruturas de dados que as representam são também menores. Apesar de mais experimentos serem necessários para que seja possível fazer uma afirmação mais conclusiva, é possível que as estruturas de dados sejam quase que

inteiramente mapeadas nas caches de nível 2, reduzindo assim o número de falhas as caches de nível 3 e consequentemente diminuindo o número de trocas de contexto entre as *threads* ocasionados por uma falha cara. Com um menor número de trocas de contexto cria-se um cenário de disputa real entre as *threads* por tempo de execução no processador, reduzindo assim as acelerações obtidas. Este cenário só ocorre quando 9 ou mais *threads* são criadas.

Tabela 6.10: Ganhos obtidos para as malhas representativas de um material reforçado com fibras longas (Célula B) com refinamentos variados para a versão ST do código paralelo.

Malhas	*Sps* 2	*Sps* 4	*Sps* 6	*Sps* 8	*Sps* 16
B_1	2.2	2.6	3.4	4.2	4.0
B_2	1.7	2.1	2.8	3.4	3.3
B_3	1.1	1.9	2.6	3.1	3.8
B_4	1.1	2.1	2.7	3.2	3.5
B_5	1.1	2.0	2.6	3.1	3.3

Tabela 6.11: Ganhos obtidos para as malhas representativas de um material reforçado com fibras longas (Célula C) com refinamentos variados para aversão ST do código paralelo

Malhas	*Sps* 2	*Sps* 4	*Sps* 6	*Sps* 8	*Sps* 16
C_1	1.4	2.6	3.3	4.1	4.3
C_2	1.1	1.9	2.4	3.0	3.1
C_3	1.1	2.0	2.7	3.3	3.8
C_4	1.1	1.7	2.3	2.6	2.8
C_5	1.1	2.0	2.5	2.9	3.0

Para os experimentos realizados com a célula B e C, observamos que as acelerações aumentam a medida que mais *threads* são criadas. Os resultados desta vez indicam melhores desempenhos quando 16 *threads* são utilizadas. Esse é um forte indício de que a especulação em relação ao efeito do mapeamento dos dados esteja correta, visto que o desempenho com 16 *threads* só é superior nas configurações com um maior número de GLs. Para as malhas B_1 e B_2, por exemplo, os melhores resultados foram obtidos com 8 *threads*.

Considerações Sobre os Resultados via HEA3D

Pode-se observar para o programa *HEA3D*, que a abordagem que permitiu maior aceleração de forma geral foi a PD. Para a abordagem ST foi possível alcançar valores próximos aos da abordagem PD em alguns cenários, como por exemplo quando 16 *threads* são criadas para computar as células C.

Os resultados da abordagem ST ficaram abaixo da expectativa inicial, principalmente por conta da limitação em relação ao número de *threads* da abordagem PD. Com mais *theads* criadas para realizar o processamento paralelo, a expectativa era de que, a partir de algum ponto, as acelerações da abordagem ST superassem os resultados da abordagem PD. Esse resultado pode ser explicado em parte pelo *overhead* introduzido por OpenMP para o gerenciamento das *threads*. Na abordagem PD as *threads* são criadas uma única vez, enquanto na abordagem ST as *threads* são criadas a cada chamada da função $SMATV$. Assim, esse custo adicional de gerenciamento deve ser pago a cada chamada a essa função. Esse efeito multiplicador anula por completo os ganhos que o uso de mais *threads* traz ao código de $SMATV$.

Foram realizados alguns testes adicionais na abordagem ST. Em especial, cláusulas adicionais para mudar a forma como as iterações do laço são distribuídas entre as *threads* foi testada. Por padrão, utiliza-se o escalonamento estático (*static*). Foi testado o uso do escalonamento dinâmico (*dynamic*). Os resultados gerais mostraram um aumento no tempo de computação, e consequentemente, uma redução na aceleração.

7 Considerações Finais e Perspectivas Futuras

O objetivo inicial proposto neste trabalho, a implementação da técnica da HEA aplicada à elasticidade para obtenção de propriedades mecânicas efetivas de células tridimensionais, empregando técnicas de processamento paralelo, foi alcançado com pleno êxito.

Uma ferramenta computacional que permite a análise mecânica de materiais compósitos periódicos de modo adequado e eficiente, chamada *HEA3D*, foi desenvolvida. Inicialmente obteve-se uma versão sequencial da aplicação em FORTRAN. Os resultados foram validados a partir de comparações dos resultados gerados pelo *HEA3D* com resultados disponíveis na literatura, apresentando uma boa concordância.

A partir dessa versão sequencial, foi desenvolvida uma versão paralela para multiprocessadores empregando OpenMP. A escolha do ambiente de multiprocessadores se deveu, principalmente, a disponibilidade de recursos computacionais. Para o desenvolvimento de aplicações paralelas nesta arquitetura foi escolhido o modelo de programação com memória compartilhada, por esta casar naturalmente com o ambiente de *hardware* disponível. Dentre as opções disponíveis para o desenvolvimento de aplicações com memória compartilhada, destacou-se o OpenMP pelo alto nível de abstração oferecido ao programador para a definição de operações paralelas no código e pela sua disponibilidade em vários compiladores. A análise comparativa de desempenho entre as versões sequencial e paralela indicou que o objetivo de redução no tempo de computação foi alcançado: ganhos de até 5.3 vezes foram obtidos quando 6 processadores foram utilizados.

A partir do desenvolvimento da ferramenta *HEA3D* criaram-se as bases necessárias para o desenvolvimento de uma série de trabalhos. Dessa forma, como perspectivas futuras, espera-se:

avaliar os resultados obtidos via *HEA3D* quando geometrias mais complexas e realistas forem empregadas;

desenvolver um aplicativo para geração automática de malhas com condições de contorno periódicas, de modo a possibilitar a aplicação de *HEA3D* em modelos com um grau elevado de refinamento;

implementar versões do *HEA3D* com elementos tetraédricos quadráticos e elementos hexaédricos, de modo exigir malhas menos refinadas do que as necessárias pelas atuais versões do programa;

incorporar ao modelo tipos variados de materiais, com propriedades e comportamento variados, de modo a diversificar o tipo de compósito resultante, inclusive considerando a associação de fases fluidas e vazios às células em estudo;

estender o programa à análise macroscópica, empregando as propriedades calculadas, e simulando evoluções causadas por agentes externos, como cargas e condições ambientais, dentre outros;

incorporar não-linearidade mecânica para analisar outros materiais que apresentam comportamento inelástico;

validar a ferramenta a partir de experimentos realizados em estruturas reais.

desenvolver uma nova versão paralela da ferramenta, utilizando para isso Unidades de Processamento Gráfico (GPGPUs - *General-purpose computing on graphics processing units*). Tais unidades de processamento gráfico são hoje amplamente utilizadas no desenvolvimento de aplicações paralelas por suas características, como o grande número de unidades de processamento disponíveis e a presença em grande parte dos computadores. GPUs oferecem uma capacidade de processamento muitas vezes superior a capacidade de processamento obtida com CPUs. A versão 2D de nosso código utilizou GPUs para a resolução dos sistemas de equações, atingindo acelerações de cerca de 20 vezes em relação ao código sequencial.

REFERÊNCIAS

[1] SANZ-HERRERA, J., GARCÍA-AZNAR, J., DOBLARÉ, M., "On scaffold designing for bone regeneration: A computational multiscale approach", *Acta Biomaterialia*, v. 5, n. 1, pp. 219 – 229, 2009.

[2] KIM, M., PARK, Y.-B., OKOLI, O. I., ZHANG, C., "Processing, characterization, and modeling of carbon nanotube-reinforced multiscale composites", *Composites Science and Technology*, v. 69, n. 3-4, pp. 335 – 342, 2009.

[3] UVA, G., SALERNO, G., "Towards a multiscale analysis of periodic masonry brickwork: A FEM algorithm with damage and friction", *International Journal of Solids and Structures*, v. 43, n. 13, pp. 3739 – 3769, 2006.

[4] LEE, K., PARK, J., "A numerical model for elastic modulus of concrete considering interfacial transition zone", *Cement and Concrete Research*, v. 38, n. 3, pp. 396 – 402, 2008.

[5] CIORANESCU, D., DONATO, P., *An Introduction to Homogenization*. 1st ed. Oxford University Press, 1999.

[6] WANG, W., LUO, D., TAKAO, Y., KAKIMOTO, K., "New solution method for homogenization analysis and its application to the prediction of macroscopic elastic constants of materials with periodic microstructures", *Computers & Structures*, v. 84, n. 15-16, pp. 991–1001, June 2006.

[7] OLIVEIRA, J., DA CRUZ, J. P., TEIXEIRA-DIAS, F., "Asymptotic homogenisation in linear elasticity. Part II: Finite element procedures and multiscale applications", *Computational Materials Science*, v. 45, pp. 1081–1096, 2009.

[8] NVIDIA, *NVIDIA CUDA Programming Guide 2.0*. NVIDIA, 2008.

[9] GHOSH, S., LEE, K., MOORTHY, S., "Multiple scale analysis of heterogeneous elastic structures using homogenization theory and Voronoi cell finite element method", *International Journal of Solids and Structures*, v. 32, n. 1, 1995.

[10] PIASTA, J., SAWICZ, Z., RUDZINSKI, L., "Changes in the structure of hardened paste due to high temperature", *Matériaux et Constructions*, v. 100, pp. 291–296, 1984.

[11] ROSTASY, F. S., WEIB, R., WIEDEMANN, G., "Changes of pore structure of cement mortars due to temperature", *Cement and Concrete Research*, v. 10, pp. 157–164, 1980.

[12] SAAD, M., EL ENEIN, A. A. A., HANNA, G. B., KOTKATA, M. F., "Effect of temperature on physical and mechanical properties of concrete containing silica fume", *Cement and Concrete Research*, v. 26(5), pp. 669–675, 1996.

[13] MURAD, M. A., GUERREIRO, J. N., LOULA, A. F. D., "Micromechanical computational modeling of secondary consolidation and hereditary creep in soils", *Comput. Methods Appl. Mech. Engrg.*, v. 190, pp. 1985–2016, 2001.

[14] MURAD, M. A., MOYNE, C., "Micromechanical computational modeling of expansive porous media", *C. R. Mechanique*, v. 330, pp. 865–870, 2002.

[15] MURAD, M. A., MOYNE, C., "Eletro-chemo-mechanical couplings in swelling clays derived from a micro/macro homogenization procedure", *Int. Journal of Solids and Structures*, v. 39, pp. 6159–6190, 2002.

[16] ROMKES, A., ODEN, J. T., "Adaptive modeling of wave propagation in heterogeneous elastic solid", *Comput. Methods Appl. Mech. Engrg.*, v. 193, pp. 539–559, 2004.

[17] LNCC, P., "Modelagem multiescala", I Escola em Modelagem Computacional Multiescala, 2005.

[18] SANCHEZ-PALENCIA, E., *Non-homogeneous media and vibration theory*. J. Ehlers Lecture Notes in Physics, Springer, 1980.

[19] CHUNG, P. W., TAMMA, K. K., NAMBURU, R. R., "Asymptotic expansion homogenization for heterogeneous media: computational issues and applications", *Composites: Part A*, v. 32, pp. 1291–1301, 2001.

[20] FERREIRA, A. P. G., FARAGE, M. C. R., DA SILVA BARRA, L. P., "Aplicação da Homogeneização por Expansão Assintótica à Modelagem Mecânica de Corpos Heterogêneos Bidimensionais". In: *X Encontro de Modelagem Computacional*, 2007.

[21] FARAGE, M. C. R., FERREIRA, A. P. G., BARRA, L. P. S., BEAUCOUR, A.-L., "Modelagem Multiescala de Concretos Feitos com Agregados Leves". In: *VIII Simpósio de Mecânica Computacional*, 2008.

[22] QUINTELA, B. M., FERREIRA, A. P. G., FARAGE, M. C. R., LOBOSCO, M., "Parallel Implementation of the AEH Technique for the Solution of Plane Multiphasic Problems". In: *Anais do III MCSul*, pp. 363–368, 2009.

[23] QUINTELA, B. M., FERREIRA, A. P. G., FARAGE, M. C. R., LOBOSCO, M., "Parallel Implementation of the AEH Technique for the Solution of Plane Multiphasic Problems", *Vetor (FURG)*, v. 20, n. 1, pp. 30–34, 2010.

[24] QUINTELA, B. M., FARAGE, M. C. R., LOBOSCO, M., "Evaluation of effective properties of heterogeneous media through a GPGPU based algorithm". In: *XXXI CILAMCE*, v. XXIX, pp. 7085–7094, 2010.

[25] HOU, T. Y., WU, X.-H., "A multiscale finite element method for elliptic problems in composite materials and porous media", *Journal of computational physics*, v. 134, pp. 169–189, 1997.

[26] AURIAULT, J.-L., BOUTIN, C., GEINDREAU, C., *Homogenization of Coupled Phenomena in Heterogeneus Media*. Iste, Wiley, 2009.

[27] OLIVEIRA, J., DA CRUZ, J. P., TEIXEIRA-DIAS, F., "Asymptotic homogenisation in linear elasticity. Part I: Mathematical formulation and finite element modeling", *Computational Materials Science*, v. 45, pp. 1073–1080, 2009.

[28] TORQUATO, S., *Random heterogeneous materials: microstructure and macroscopic properties*. Springer, 2001.

[29] HOLLISTER, S. J., "Computational Modeling of Biological Tissues", University of Michigan Biomedical Engeneering, disponível em: http://www.engin.umich.edu/class/bme506.

[30] JI-WEI, D., MIAO-LIN, F., "Asymptotic expansion homogenisation for simulating progressive damage of 3D braided composites", *Composite Structures*, v. 92, pp. 873–882, 2010.

[31] WILKINSON, B., ALLEN, M., *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Upper Saddler River, 2005.

[32] DONGARRA, J. J. E. A., *Sourcebook of Parallel Computing*. Morgan Kaufmann, 2003.

[33] CHAPMAN, B., JOST, G., VAN DER PAS, R., *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, October 2007.

[34] CHANDRA, R. E. A., *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001.

[35] MATHWORKS, *MATLAB Online Reference Documentation*, http://www.math.ufl.edu, 2007, acessado em setembro de 2007.

[36] DAVIS, T., *Direct Methods for Sparse Linear Systems*. 1st ed. SIAM, 2006.

[37] BUATOIS, L., CAUMON, G., LÉVY, B., "Concurrent number cruncher An efficient sparse linear solver on the GPU". In: *High Performance Computation Conference 2007*, Lecture Notes in Computer Science (LNCS), 2007.

[38] SHEWCHUK, J. R., "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", School of Computer Science Carnegie Mellon University Pittsburgh, 1994.

[39] HEATH, M. T., *Scientific Computing: An Introductory Survey*. McGrawHill, 2002.

[40] BUATOIS, L., CAUMON, G., LÉVY, B., "Concurrent number cruncher A GPU implementation of a general sparse linear solver", *The International Journal of Parallel, Emergent and Distributed Systems*, v. 00, n. 00, 2008.

[41] D., A. J. L., *Programa TriMC - Manual de Entrada de Dados*, PEC/COPPE/UFRJ, 1996.

[42] COUTINHO, A. L. G. A., MARTINS, M. A. D., ALVES, J. L. D., L, L., MORAES, A., "A.: Edge-Based Finite Element Techniques for Nonlinear Solid Mechanics Problems", *Int. J. Num. Meth. Eng*, v. 50, pp. 2053–2058, 2000.

[43] HUGHES, T. J. R., LEVIT, J., WINGET, J., "Implicit, unconditionality stable element-by-element algorithms for heat conduction analysis". In: *J. Eng. Mech.*, 1983.

[44] LIU, Y., ZHOU, W., QIANG, Y., "A distributed memory parallel element-by-element scheme based on Jacobi-conditioned conjugate gradient for 3D finite element analysis". In: *J. Eng. Mech.*, 1983.

[45] WILSON, E. L., HOIT, M. I., "A computer adaptive language for the development of structural analysis programs", *Computers & Structures*, v. 19, n. 3, pp. 321 – 338, 1984.

[46] HUGHES, T. J. R., *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000.

[47] GEUZAINE, C., REMACLE, J.-F., "Gmsh 2.4.2", 2009, disponível em: http://www.geuz.org/gmsh/.

[48] CORPORATION, S., "Cubit 11.0", 2007, disponível em: http://cubit.sandia.gov/.

[49] REDDY, J. N., MIRAVETE, A., *Practical Analysis of Composite Laminates*. CRC Press.

[50] GIBSON, R. F., *Principles of Composite Material Mechanics*. 2nd ed. CRC Press, 2007.

[51] CAMPOS-FILHO, F. F., *Algoritmos Numéricos*. LTC Editora, 2001.

[52] XIA, Z., ZHANG, Y., ELLYIN, F., "A unified periodical boundary conditions for representative volume elements of composites and applications", *International Journal of Solids and Structures*, v. 40, n. 8, pp. 1907 – 1921, 2003.

[53] SUN, C. T., VAIDYA, R. S., "Prediction of Composite Properties from a Representative Volume Element", *Composite Science and Technology*, v. 56, pp. 171–179, 1995.