Thiago de Sousa Goveia

Avaliação quantitativa do método do gradiente conjugado precondicionado para a solução paralela e concorrente de um modelo de elementos finitos

Proposta de trabalho apresentada como requisito das disciplinas trabalho de conclusão de curso I e metodologia de pesquisa.

Centro Federal de Educação Tecnológica de Minas Gerais

Campus Timóteo

Graduação em Engenharia de Computação

Orientador: Márcio Matias

Timóteo

2017

Sumário

1 Introdução

Devido à popularização da computação de alto desempenho (HPC), trabalhos recentes têm retomado problemas tradicionais a fim de adequá-los aos novos paradigmas e arquiteturas de computação. De acordo com Kiss et al. (2012) a conformidade entre o problema a ser resolvido e a estrutura do ambiente de execução é capaz de ampliar a performance e reduzir a energia dispendida no processamento. Devido ao aumento da demanda por recursos computacionais, dispositivos que possibilitam a execução paralela acabaram se estabelecendo no mercado da informática. Por meio das arquiteturas *manycore* e *multicore*, é possível se executar paralela ou concorrentemente tanto tarefas corriqueiras como a exibição de vídeos e jogos até cálculos complexos da ciência e da engenharia. Podem ser citados como processadores *multicore* as linhas Core e Xeon da Intel, Opteron e Ryzen da AMD e a linha Power da IBM. Os dispositivos manycore por sua vez, têm como principais representantes as unidades de processamento gráfico (GPU) da qual fazem parte as placas GeForce, Quadro e Tesla da NVIDIA e Radeon e FirePro da AMD.

O método dos elementos finitos (FEM) é um método numérico para a resolução de problemas de valor de contorno (PVC) modelados por equações diferenciais e também de problemas associados à minimização de um funcional de energia (SZABO; BABUŠKA, 2009). O algoritmo clássico do FEM foi concebido em sua forma sequencial e consiste principalmente na solução de um sistema linear esparso. A tarefa de se resolver tal sistema é geralmente custosa em termos de memória quando adotados métodos diretos como a eliminação gaussiana e computacionalmente custosa quando adotados métodos iterativos como o método de Jacobi. O método dos gradientes conjugados (CG) e suas variantes pertencem à família dos métodos exatos/iterativos do subespaço de Krylov (ANZT et al., 2016) e tem sido adotados na literatura para a solução paralela de elementos finitos. Alguns trabalhos correlatos que utilizam a família CG são apresentados por Yao et al. (2015), Ahamed e Magoulès (2016) e Iwashita et al. (2017).

A fim de adequar o FEM às arquiteturas modernas será adotado neste trabalho a abordagem elemento a elemento (EbE-FEM) proposta por Hughes, Levit e Winget (1983). Esta técnica baseia-se no fato de que a matriz do sistema de elementos finitos é caracterizada como uma função parcialmente separável (DAYDE; L'EXCELLENT; GOULD, 1995), resultado da soma das matrizes elementares. Assim sendo, as operações da solução do sistema de elementos finitos podem ser realizadas em nível elementar, sem a necessidade de se montar a matriz global do sistema. Adicionalmente tem-se a vantagem de que elementos não adjacentes podem ser calculados simultaneamente por meio dos métodos CG (WATHEN, 1989). Esta última característica torna a técnica EbE-FEM propícia para a implementação paralela nas arquiteturas modernas.

1.1 Problema de Pesquisa

Neste trabalho é feita uma análise quantitativa do desempenho do algoritmo do EbE-FEM nas linguagens concorrentes C++, Erlang e Scala e nas linguagens de computação paralela, CUDA e Harlan. O problema *benchmark* a ser resolvido refere-se à equação de Laplace originada do cálculo da distribuição de potencial e do campo elétrico de um capacitor de placas paralelas (BOYLESTAD, 2011, Exemplo 10.3). A escolha deste problema se deve à simplicidade de sua modelagem e ao seu comportamento já explorado em livros de eletromagnetismo e circuitos elétricos.

1.2 Justificativa

A justificativa deste trabalho se baseia na contínua transformação dos paradigmas de programação e arquiteturas de hardware. Como coloca Guo et al. (2014), com aumento de núcleos de processamento, ocorre a redução da razão memória por núcleo, o que impõe uma forte demanda para que os algoritmos utilizem eficientemente todos os níveis de paralelismo disponíveis enquanto minimizam a movimentação de dados. Tal evolução não se limita ao cenários dos *clusters* e *grids* mas alcança inclusive os dispositivos móveis, que atualmente já possuem até oito núcleos. Pensando em um futuro próximo, com os avanços da computação ubíqua que introduz temas como internet das coisas, dispositivos "usáveis", realidade aumentada e realidade virtual, a necessidade de se aproveitar ao máximo todo o poder de processamento disponível se torna ainda mais evidente, uma vez que nessas tecnologias há alta demanda de processamento e/ou pouco espaço físico para comportar um processador adequado. A justificativa para a escolha das linguagens C++, Scala e Erlang se dá devido à ausência de trabalhos acadêmicos relacionando tais tecnologias e seu desempenho. C/C++ é uma linguagem abrangente, robusta e atual, presente por trás de grande parte das aplicações *desktop* e *mobile*. Scala e Erlang (e/ou Elixir) são linguagens naturalmente concorrentes e que possuem crescente *market share* .

1.3 Objetivos

1.3.1 Objetivos Gerais

- Demonstrar o processo de mudança do paradigma sequencial para uma solução paralela do método do elementos finitos;
- Avaliar quantitativamente, segundo as métricas propostas (tempo de execução, uso de memória, e *speedup*) o desempenho do processo de solução do EbE-FEM.

- Apresentar à comunidade resultados acadêmicos experimentais do desempenho das linguagens C++, Scala, Erlang, CUDA e Harlan.

1.3.2 Objetivos Específicos

- Investigar a viabilidade da técnica EbE-FEM como alternativa dos *solvers* iterativos e do solver do MATLAB®;
- Desenvolver um material acessível e de fácil compreensão de introdução ao FEM para o nível da graduação;
- Investigar o quão otimizada é a convergência do EbE a partir do emprego de diferentes precondicionadores;
- Relacionar as métricas e conceitos de estatísticas necessários para a avaliação adequada de performance;
- Comparar conforme as métricas o desempenho das linguagens nativamente concorrentes e paralelas na resolução do problema *benchmark* proposto neste trabalho.

1.4 Resultados Esperados

Espera-se com este trabalho iniciar no CEFET-MG campus Timóteo uma nova linha de pesquisa a ser continuada nos trabalhos futuros, voltada para a análise numérica de problemas de valor de contorno da física aplicada. De forma similar, espera-se o incentivo e a adoção por parte da universidade de paradigmas e linguagens emergentes, a fim de diversificar o currículo dos graduandos.

2 Revisão bibliográfica

2.1 Estado da arte

Como coloca Kiss et al. (2012), o processamento paralelo de um problema modelado pelo FEM pode ser feito a partir da decomposição do domínio do problema. Esta decomposição pode ser feita por meio do particionamento da malha que representa o domínio do problema ou por meio da decomposição apropriada da matriz de coeficientes. Os trabalhos de Boehmer et al. (2011) e Ahamed e Magoulès (2016) apresentam a solução paralela do FEM por meio do particionamento da malha. No primeiro caso o processamento é executado comparativamente nas arquiteturas de memória compartilhada e distribuída com o uso da API OpenMP e da biblioteca MPI respectivamente. No segundo caso, a comparação é feita entre as linguagens CUDA e OpenCL que são executadas em uma arquitetura *manycore*.

A técnica EbE é uma forma de decomposição da matriz de coeficientes e será tratada neste trabalho. Por meio desta, as operações são realizadas na matriz de cada elemento, sem que seja necessária a montagem do sistema global. Esta estrutura de dados foi proposta originalmente por Hughes, Levit e Winget (1983) como uma fatoração especial para a matriz de coeficientes de forma a melhorar sua condição a acelerar a convergência (CAREY et al., 1988). Devido ao seu desempenho, precisão, economia de memória e à possibilidade de processamento paralelo (LEVIT, 1987; HU; QUIGLEY; CHAN, 2008; KISS et al., 2012) a adoção da abordagem EbE tem sido recorrente à medida em que surgem novas tecnologias de HPC.

Carey et al. (1988) apresenta uma implementação EbE do gradiente biconjugado (BiCG) para solucionar um sistema de elementos finitos (FE). Segundo ele, o surgimento de novas arquiteturas tais como os processadores vetoriais, paralelos e estações de trabalho microprocessadas foram responsáveis por se repensar o algoritmo original do FEM. A execução vetorial foi realizada no computador CRAY-XMP e foi 8 vezes mais rápida em relação ao processamento sequencial e apresentou *speed-up* variando de 4.25 à 6.5. A execução paralela foi feita no ALLIANT-FX/8, um mini supercomputador com 8 CPUs. Foi adotado um esquema de ordenação de nós a fim de se evitar condições de corrida no acesso às variáveis globais. Com a utilização de todas as unidade de processamento, o *speed-up* foi em torno de 7 em relação ao uso de uma única CPU.

O trabalho de Dayde, L'Excellent e Gould (1995) faz uma análise comparativa entre 5 precondicionadores de nível elementar para o algoritmo do gradiente conjugado (CG), a saber: *Element matrix factorization* (EMF), *finite element preconditioner* (FEP), *one-pass*

element-by-element preconditioner (EbE), *two-pass element-by-element preconditioner* (EbE2) e Gauss-Seidel *element-by-element preconditioner* (GS-EBE). Como os autores colocam, uma vez que se têm problemas de larga escala mas parcialmente separáveis, torna-se necessário explorar diferentes alternativas a fim de se aproveitar as vantagens oferecidas pela estrutura do problema. A abordagem EbE se mostrou a melhor opção entre seus concorrentes. EMF e FEP requerem a montagem parcial do sistema, o que agrega maior custo de processamento. EBE2 e GS-EBE não apresentaram boas aproximações para elementos com pouca vizinhança. A fim de se realizar a paralelização, é sugerida a coloração da malha de forma que elementos vizinhos sejam processados sequencialmente.

Uma implementação EbE-CG em FPGA (Field Programmable Gate Array) é proposta por Hu, Quigley e Chan (2008). Neste trabalho as operações sobre as matrizes elementares foram realizadas por meio da configuração de um circuito lógico no chip 4VLX160 da Xilinx. O processamento sequencial foi feito em um PC 2.01 GHz Athlon 64 com as devidas otimizações na compilação. Graças à implementação diretamente em hardware foi alcançado um *speed-up* máximo igual a 40.

O surgimento da linguagem CUDA em 2006 e a popularização das GPGPU possibilitaram que a técnica de EbE pudesse ser revisitada e aplicada nas arquiteturas modernas. Kiss et al. (2012) soluciona o um modelo de elementos finitos por meio do BiCG com precondicionador de Jacobi. De acordo com este trabalho, a abordagem EbE é adequada para o processamento em GPU, cuja arquitetura embora seja massivamente paralela, possui um gargalo, que é a capacidade limitada de memória. O caráter localizado do EbE faz com que o trânsito de dados seja mínimo, o que reduz o consumo de energia e maximiza o potencial do dispositivo. Para a realização dos testes foram utilizados o processador quad-core Xeon X3440 da Intel e a placa GTX 590 da NVIDIA, contendo 2 GPUs. A execução com aceleração em GPU consumiu 20 vezes menos memória e foi 10 vezes mais rápida que a execução unicamente em CPU.

O trabalho de Wu et al. (2015) também apresenta o método BiCG elemento a elemento com precondicionador de Jacobi implementado em GPU. Foi utilizado o processador Xeon E5-2696v2 da Intel e a placa Tesla K20c da NVIDIA. Nos dois refinamentos de malha testados foi alcançado um *speed-up* de 4.63, e como coloca o autor, o resultado obtido se torna ainda mais efetivo à medida em que o número de elementos aumenta. No trabalho mais recente dos mesmos autores (YAN et al., 2017) é apresentado o uso do precondicionador de Gauss-Seidel para a abordagem EbE, o qual apresentou melhores resultados para o número de iterações e tempo de execução em relação ao precondicionamento de Jacobi, mantendo a mesma precisão.

Outros trabalhos importantes para esta monografia são os de Xu, Yin e Mao (2005), Yao et al. (2015) e Chou e Chen (2016). O primeiro apresenta a implementação do EbE-CG e a técnica de atribuição das condições de contorno que também será empregada no

presente trabalho. O segundo apresenta a comparação da performance da implementação do BiCG estabilizado (BiCGStab) em CPU e GPU, utilizando-se diferentes formas de armazenamento de matriz esparsa e diferentes ferramentas de resolução de sistema linear em GPU, a saber: CUSPARSE, CUSP e CULA Sparse. O último trabalho apresenta a comparação entre diferentes tecnologias de processamento paralelo (Cuda C, Cuda Fortran, MPI e OpenMP) na resolução de dois problemas *benchmark*

2.2 Fundamentação teórica

Neste capítulo é apresentado o referencial teórico para a compreensão dos métodos e conceitos utilizados ao longo deste trabalho. Na primeira seção é apresentada a definição de Problemas de Valor de Contorno (PVC) bem como o problema a ser solucionado neste trabalho. Na seção seguinte, são abordados os Sistemas de Elementos Discretos a fim de fornecer a compreensão intuitiva e genérica do Método dos Elementos Finitos. Em seguida, na seção 2.2.2 é feita a introdução ao Método dos Elementos Finitos (FEM). As seções **??**, 2.2.4 e **??** abordam respectivamente, as etapas de pré-processamento, análise e pós-processamento do FEM. Nas seção **??** são apresentadas as equações de Maxwell para o Eletromagnetismo e na seção **??** são mostradas algumas técnicas iterativas para a resolução de sistemas lineares, tais como o Método do Gradiente Conjugado e o uso de Precondicionadores. Estas técnicas são as estratégias utilizadas para a realização da abordagem Elemento a Elemento (EbE) do FEM. Por fim, a última seção deste capítulo apresenta os principais fundamentos da programação de Uso Geral em Unidades de Processamento Gráfico (GPGPU).

2.2.1 Problema de Valor de Contorno

Um problema de valor inicial PVI,pode ser definido como uma equação ou sistema de equações diferenciais no qual são dadas as condições iniciais do fenômeno. Tais condições são impostas sobre a variável dependente e suas derivadas em um único instante de tempo (inicial) t_0. Um problema de valor de contorno PVC por sua vez, apresenta tais condições impostas em pontos distintos, como por exemplo em x_i e x_f. Geralmente os PVI são dados em função do tempo enquanto os PVC são dados em função do espaço (BOYCE; DIPRIMA, 2010).

Como pode ser visto na tabela 1. as condições estabelecidas sobre a variável dependente, são chamadas de *condições de Dirichlet* ou *essenciais*, enquanto as que são estabelecidas sobre as derivadas da variável dependente são conhecidas como *condições de Neumann* ou *naturais*. Além destas duas, existem as restrições específicas do fenômeno modelado, como por exemplo, condições de radiação ou de impedância para problemas do eletromagnetismo (JIN, 2002).

Condição	**Tipo**
$u(x_i) = \alpha$	Dirichlet
$u(x_i) = 0$	Dirichlet Homogênea
$u'(x_i) = \beta$	Neumann
$u'(x_i) = 0$	Neumann Homogênea

Tabela 1 – Exemplos de condições de contorno

A solução analítica de um PVC pode ser obtida por meio da integração direta ou a partir da aplicação de técnicas como a separação de variáveis, expansão em séries ou pela transformada de Laplace. No entanto, a maioria dos problemas da engenharia e da ciência não são lineares e apresentam geometria ou condições de contorno complexas. Estas características fazem com que a resolução analítica de tais problemas seja impraticável, sendo necessário recorrer a métodos numéricos para se obter uma solução aproximada (BOYCE; DIPRIMA, 2010; POWERS, 2006).

2.2.2 Métodos numéricos para PVCs

O método dos elementos finitos (FEM) é uma alternativa numérica para a solução de PVCs.Neste método, o domínio do problema é visto como uma coleção de subdomínios, chamados de elementos finitos, sobre os quais, a equação que modela o problema é aproximada por um método variacional ou de resíduos ponderados Reddy (2006). Essas diferentes vertentes do método surgiram graças aos esforços independentes de matemáticos, cientistas e engenheiros (ZIENKIEWICZ, 2005).

O método das diferenças finitas (FDM) assim como o FEM é uma abordagem numérica para a aproximação de PVCs. Este método consiste na discretização do domínio do problema por meio de uma grade de pontos e na aproximação de cada derivada da equação por um quociente-diferença adequado (FAIRES, 2008). Embora este método seja útil em muitos casos, se torna difícil aplicá-lo em problemas com geometria irregular ou com condições de contorno não usuais. Um exemplo pode ser visto na figura 1.

Diferentemente do FDM, como coloca Huebner et al. (2001), o FEM divide o domínio não em pontos, mas em subdomínios, sobre os quais as equações são aproximadas por polinômios definidos por partes. O FEM também é capaz de representar mais fielmente o contorno (ou a borda) do problema. Desta forma, ele se apresenta como uma técnica mais poderosa e versátil para a modelagem de fenômenos com geometria complexa e meios não homogêneos (SADIKU, 2001).

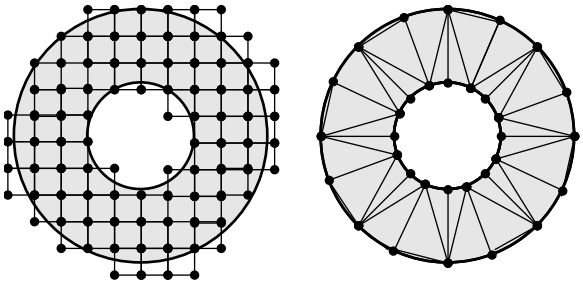


Figura 1 – Método das diferenças Finitas e método dos Elementos Finitos

2.2.3 Método dos Elementos Finitos

O FEM surgiu originalmente como uma técnica de análise de deslocamentos e elasticidade de estruturas mecânicas, mas em seguida foi estendido para solucionar problemas de outros campos da física e da engenharia (JIN, 2002; DESAI, 1972; ZIENKIEWICZ, 2005).

As primeiras formulações do FEM são conhecidas como *abordagem direta* ou *formulação física*, que embora forneça a interpretação intuitiva do método, é util apenas para a resolução de problemas relativamente simples (HUEBNER et al., 2001; DESAI, 1972; ZIENKIEWICZ, 2005). O uso do princípio do trabalho virtual, para a determinação de forças na abordagem direta, levou à generalização do FEM, por meio da estratégia de minimização do funcional de energia. Esta técnica mais genérica ficou conhecida como *formulação variacional* (DESAI, 1972; ZIENKIEWICZ, 2005; JIN, 2002) e tem como principal representante o método de Rayleigh-Ritz. Uma terceira abordagem, conhecida como *Método dos Resíduos Ponderados* ou *formulação generalizada* (ZIENKIEWICZ, 2005; HUEBNER et al., 2001) é tradicionalmente utilizada e é ainda mais genérica que o princípio variacional, pois resolve diretamente as equações diferenciais do modelo, sem necessitar da existência de um funcional de energia (DESAI, 1972), ou ainda, sem exigir que o sistema seja positivo-definido. Neste trabalho será adotado o método dos resíduos ponderados, mais especificamente, o método de Galerkin.

2.2.4 Método dos Resíduos Ponderados

De acordo com Jin (2002), a aplicação do FEM pode ser feita a partir de 4 passos básicos, os quais serão detalhados nas subseções a seguir.

Discretização do Domínio

A discretização do domínio consiste na transformação do contínuo Ω em uma malha de elementos finitos (discretos). Cada elemento Ω_e dessa malha representa um subdomínio de Ω. Nesta etapa são definidas a forma, a quantidade e o tamanho dos elementos, de

forma que a representação em malha seja a mais próxima possível do objeto em análise (DESAI, 1972).

Conforme pode ser visto na figura 2, cada elemento é identificado na malha a partir de um número que lhe é atribuído. Os vértices (ou nós) de cada elemento também são numerados. Cada nó possui dois valores vinculados a ele, um atuando como identificador global (numeração do nó na malha) e o outro como identificador local (numeração dentro de um dado elemento). A numeração local é geralmente feita no sentido anti-horário, a fim de se obter um valor positivo no cálculo da área ou volume por meio do determinante (SADIKU, 2001; JIN, 2002).

Um fator que deve ser levado em consideração é o balanceamento entre o refinamento da malha e o esforço computacional necessário (DESAI, 1972). Como o valor da solução é aproximado para cada elemento, o excesso de elementos pode causar a propagação do erro de aproximação, levando a resultados indesejáveis.

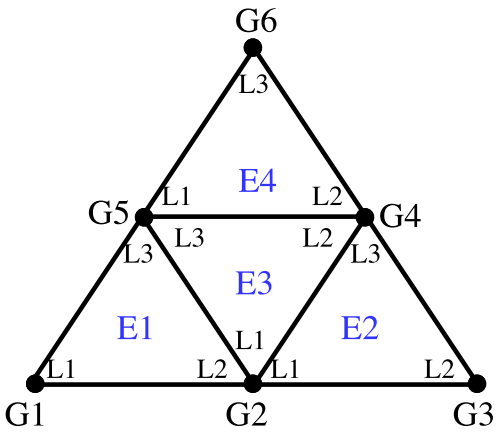


Figura 2 – Identificadores de elementos e nós

Seleção das funções de interpolação

O passo seguinte é a escolha das funções de interpolação (ou função de base ou de forma) (HUEBNER et al., 2001) que fornece uma aproximação da equação original dentro de cada subdomínio (JIN, 2002). A função interpoladora escolhida geralmente é um polinômio, e isso ocorre por dois motivos a priori (DESAI, 1972):

- Facilidade de manipulação matemática, principalmente derivação e integração;
- Aproximação satisfatória quando truncado em uma ordem qualquer.

Na prática são escolhidos polinômios de primeira ou segunda ordem, mas ordens superiores podem ser adotadas para reduzir o erro de aproximação, sobretudo em bordas com curvaturas, no entanto, ocorre também o aumento da carga computacional (JIN, 2002).

Seja a função ϕ desconhecida, da qual se deseja obter uma aproximação. A nível elementar, a função $\tilde{\phi^e}$ aproxima ϕ dentro do domínio do elemento e. O valor $\tilde{\phi_i^e}$ é a aproximação do valor de ϕ sobre o nó i de e. Considerando um subdomínio bidimensional triangular, o valor aproximado em cada nó de um elemento é dado pela equação 2.1.

$$\tilde{\phi_i^e} = a^e + b^e x_i^e + c^e y_i^e \tag{2.1}$$

Desta forma, a função que aproxima os valores dentro de todo o domínio e é pode ser dada pela equação 2.2.

$$\tilde{\phi^e} = \sum_{j=1}^{n} N_j^e(x,y)\phi_j^e = \{N^e\}^T\{\phi^e\} \tag{2.2}$$

Para manter esta aproximação em conformidade com os valores $\tilde{\phi_1^e}$, $\tilde{\phi_e^e}$ e $\tilde{\phi_3^e}$ em cada nó, a função N_i^e deve ser escolhida de forma que seu comportamento seja o mesmo do delta de Kronecker, como mostra a equação 2.3 a seguir.

$$N_i^e(x_j^e, y_j^e) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{2.3}$$

As funções de interpolação mais comuns pertencem às famílias de funções Lagrange e Serendipity (ZIENKIEWICZ, 2005; VOLAKIS; CHATTERJEE; KEMPEL, 1998).

Formulação do sistema de equações

Na abordagem variacional destaca-se o método de Ritz, ou Rayleigh-Ritz, o qual tem por objetivo, minimizar o funcional variacional, ou funcional de energia, do problema aproximado (VOLAKIS; CHATTERJEE; KEMPEL, 1998). Embora tal abordagem tenha sido historicamente utilizada e possua fundamentação física e matemática, sua adoção, em muitos casos, é mais complicada em relação ao Método dos Resíduos Ponderados, pois demanda a formulação variacional do problema. Desta forma, se um problema é dado por um modelo diferencial, é necessário se obter a partir deste modelo a forma variacional equivalente, para só então se aplicar o método para a obtenção do sistema de equações. No eletromagnetismo, a formulação variacional das equações de Maxwell não é bem estabelecida (JIN, 2002).

Para problemas que apresentam explicitamente condições de Dirichlet no contorno e cujo operador $\mathcal{L}$ é linear e auto-adjunto, é possível se obter imediatamente o funcional de energia correspondente (ZIENKIEWICZ, 2005), no entanto, como mostra (VOLAKIS; CHATTERJEE; KEMPEL, 1998, p. 29), para este tipo de operador a mesma integral da formulação variacional é obtida pelo método de Galerkin.

Seja a equação 2.4, uma equação de um PVC na qual $\mathcal{L}$ é um operador diferencial, f é uma função de excitação conhecida e ϕ é a solução procurada.

$$\mathcal{L}\phi = f \tag{2.4}$$

Se substituirmos a solução exata pela sua aproximação apresentada na equação 2.2, um resíduo r surge, como pode ser visto na equação 2.5, em decorrencia dos erros de aproximação.

$$r = \mathcal{L}\tilde{\phi} - f \neq 0 \tag{2.5}$$

Embora o resíduo em cada elemento seja diferente de zero em sua maioria, deseja-se que o resíduo total seja igual a zero. Para que isso seja possível, uma função de poderação w_i é introduzida para cada subdomínio e. Desta forma na média dos M subdomínios de Ω a condição residual é atendida (VOLAKIS; CHATTERJEE; KEMPEL, 1998). As equações 2.6 e 2.7 correspondem ao sistema de resíduos ponderados. A ideia chave dessa abordagem é transformar o problema de valor de contorno para um sistema linear que forneça o valor aproximado.

$$R_i^e = \int_\Omega w_i^e r \; d\Omega = 0 \qquad i = 1, 2, 3. \tag{2.6}$$

$$R_i^e = \int_\Omega w_i^e [\mathcal{L}\tilde{\phi} - f] \; d\Omega = 0 \tag{2.7}$$

A escolha particular de $w_i^e = N_i^e$ configura o método de Galerkin, o qual, para um operador $\mathcal{L}$ auto adjunto a matriz do sistema é simétrica e equivalente à matriz gerada pelo método de Ritz. Após a substituição da equação 2.2 em 2.7 obtem-se a equação 2.8, a qual forma um sistema esparso de equações lineares como o apresentado em 2.9.

$$R_i^e = \int_\Omega N_i^e \mathcal{L}\{N\}^T\{\phi\} \; d\Omega\} - \int_\Omega N_i^e f \; d\Omega = 0 \tag{2.8}$$

$$[K]\{\phi\} = b \tag{2.9}$$

É importante colocar que o parâmetro w_i deve ser um conjunto de funções integraveis, linearmente independentes (REDDY, 2006, p. 60). Alguns casos especiais do método

dos resíduos ponderados são obtidos a partir da escolha de w_i:

$$\begin{array}{ll} \text{Método de Petrov-Galerkin} & w_i = \psi_i \neq \phi_i \\ \text{Método de Galerkin} & w_i = \phi_i \\ \text{Método dos Mínimos quadrados} & w_i = \frac{d}{dx}\left(a(x)\frac{d\phi_i}{dx}\right) \\ \text{Método da colocação} & \delta(x - x_i) \end{array} \tag{2.10}$$

Método dos elementos finitos em 2 dimensões

2.2.5 Método dos gradientes conjugados

3 Desenvolvimento

O desenvolvimento deste trabalho está dividido nas etapas de *validação do problema* e *aplicação das estratégias*. Na primeira são apresentados os recursos utilizados para modelar o problema e para definir as possíveis estratégias de solução. De posse das informações obtidas nesta etapa, são definidos na etapa seguinte os métodos e as estruturas de dados adotados em cada linguagem de programação e arquitetura de processamento.

3.1 Materiais e métodos

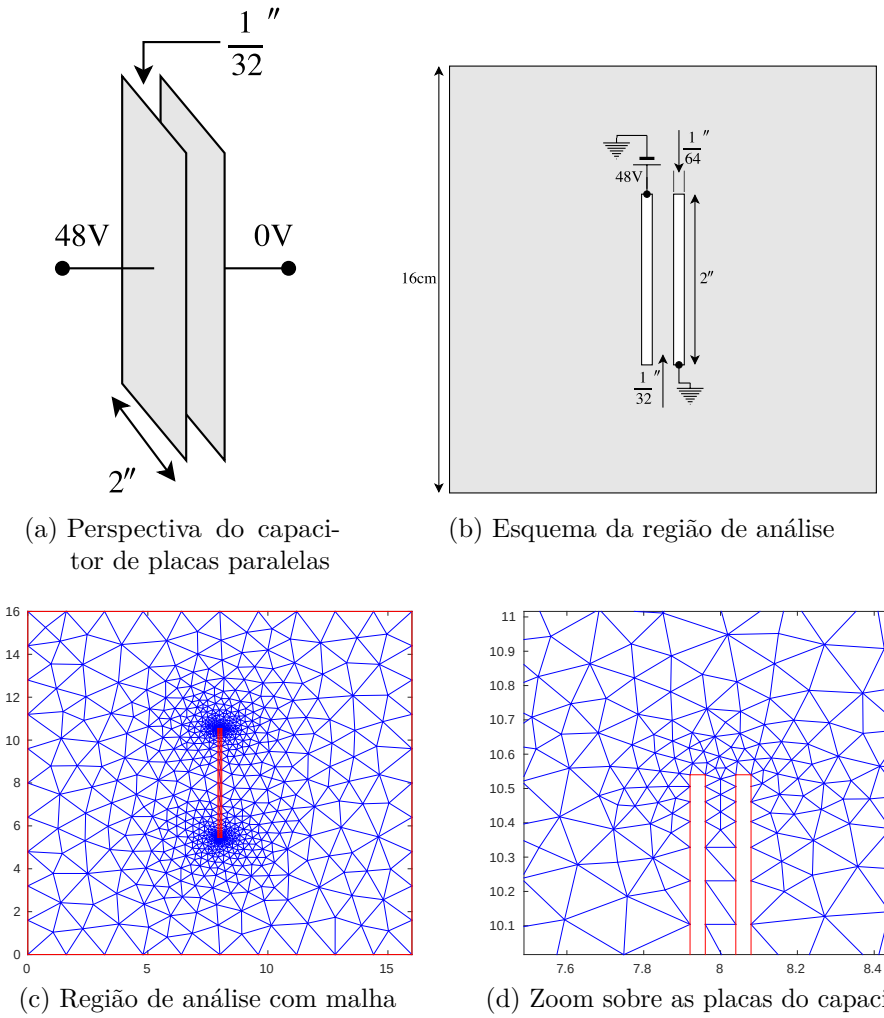
O ambiente de desenvolvimento MATLAB®foi adotado para a realização da etapa de validação do problema. A motivação dessa escolha se deu pela facilidade da manipulação de matrizes oferecida pelo ambiente e pela existência de ferramentas dedicadas à especificação de PVCs e à programação paralela. A aplicação das estratégias propostas na etapa de validação nas linguagens e arquiteturas adotadas neste trabalho é feita pela adequação da sintaxe e do paradigma de programação da implementação realizada no MATLAB®.

3.1.1 Especificação da geometria

O problema *benchmark* utilizado neste trabalho é o exemplo 10.3 do livro Análise de Circuitos de Boylestad (2011) e é ilustrado na figura 3(a). O capacitor é composto de duas placas quadradas paralelas com lado de 2 polegadas, cuja distância entre elas é de $\frac{1}{32}$ polegadas. O problema informa que a diferença de potencial entre as placas é de $48V$, dessa forma foi assumido que uma das placas possui $48V$ e e que a outra está aterrada.

A fim de se realizar uma análise mais ampla do problema a distribuição do campo elétrico será calculada entre as placas mas também no espaço ao redor. Será considerada uma região quadrada de $16cm$ de lado como mostra o esquema na figura 3(b). Também foi considerado que a espessura de cada placa é a metade da largura da região entre elas, ou seja, $\frac{1}{64}$ polegadas.

A fim de se obter um parâmetro da corretude dos resultados obtidos neste trabalho, foi feita a simulação do problema na *partial differential equation toolbox* do MATLAB®. Por meio dessa ferramenta é possível programaticamente e via interface gráfica definir a equação diferencial que rege o problema, a geometria, as condições de contorno, a malha inicial e seus refinamentos bem como a solução e exibição dos resultados do PVC. As figuras 3(c) e 3(d) mostram a geometria e a malha inicial modeladas com o auxílio da interface gráfica da *pdetool*.



(a) Perspectiva do capacitor de placas paralelas

(b) Esquema da região de análise

(c) Região de análise com malha

(d) Zoom sobre as placas do capacitor

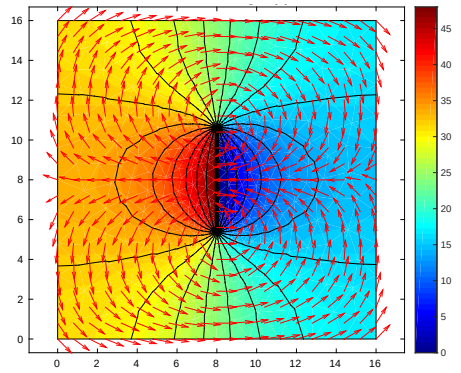
Figura 3 – Geometria do capacitor de placas paralelas

3.1.2 Obtenção dos valores de referência

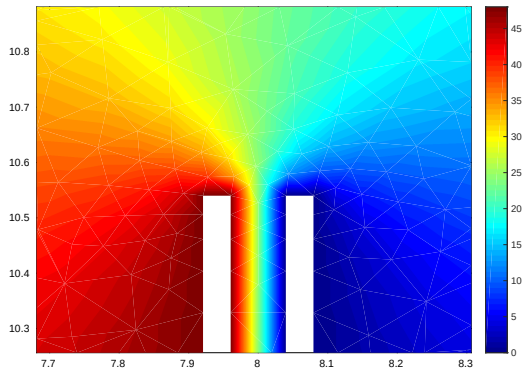
Definidas a geometria e a malha, foram feitas as atribuições das condições de contorno de Dirichlet nas 4 bordas da seção de cada placa. À placa da esquerda foram atribuídos $48V$ e $0V$ para a placa da direita. Para se definir a EDP do problema é necessário atribuir valores aos coeficientes da fórmula geral de EDPs elípticas, mostrada na equação 3.1.

$$- \bigtriangledown \cdot (c \bigtriangledown (u)) + au = f \tag{3.1}$$

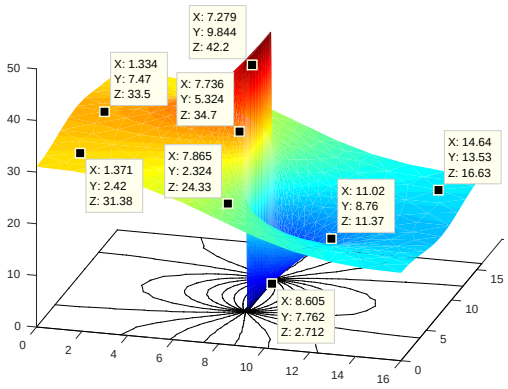
Na equação de Laplace, que modela a distribuição do potencial elétrico, apenas o coeficiente c é diferente de zero. A constante c corresponde à permissividade no vácuo na equação de Poisson, mas como a densidade volumétrica de carga representada por f e o auto valor a são iguais a zero na equação de Laplace, c passa a valer 1. Estabelecidos os valores das constantes a solução do problema pode ser obtida, como mostram as figuras 4(a) e 4(b).



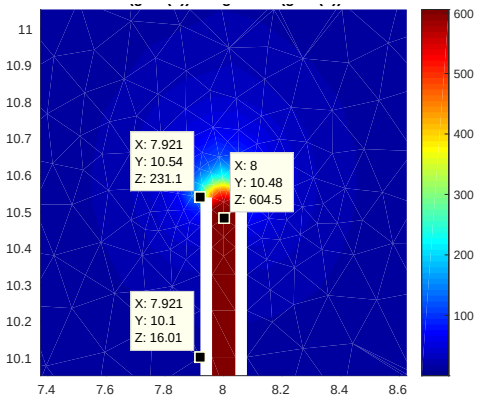
(a) Distribuição do potencial em Volts e fluxo do campo elétrico em Volts/centímetro (vetores normalizados)



(b) Distribuição do potencial entre as placas



(c) Pontos de referência do potencial elétrico



(d) Pontos de referência do campo elétrico

Figura 4 – Solução do PVC com a *pdetool*

De posse da solução da *pdetool*, foram tomados 8 pontos de referência da distribuição do potencial e 3 pontos de referência do campo elétrico. A localização e os valores destes pontos são apresentados na figuras 4(c) e 4(d).

3.1.3 Implementação do FEM

Após a simulação e obtenção dos valores de referência foi realizada a implementação do método dos elementos finitos em duas dimensões, conforme o apresentado na seção 2.2.4 do referencial teórico. A fim de não tirar proveito das funções do MATLAB®de forma a manter a generalidade da solução implementada e a facilidade de transcrevê-la para outras linguagens, foram utilizadas nos algoritmos deste trabalho apenas as funções da *PDE toolbox* relativas à geometria, geração da malha e exibição dos resultados. No pseudocódigo 1 as palavras destacadas em itálico são variáveis, as funções destacadas em negrito fazem parte do MATLAB®e as funções descritas nas linhas de 9 a 12 são detalhadas no exemplo da seção 2.2.4. A tabela 2 contém a descrição das funções utilizadas.

Algorithm 1 Pseudocódigo do FEM

1: **load**($dadosGeometria$);
2: **load**($coordReferencia$);
3: **load**($dadosContorno$);
4: g = **decsg**($dadosGeometria$);
5: m = **createpde**(1);
6: **geometryFromEdges**(m,g);
7: **generateMesh**(m, 'Hmax',$valHmax$);
8: (nos,tri) = **meshToPet**($m.Mesh$);
9: C = geraMatElementares(nos, tri);
10: G = geraMatGlobal(C, tri);
11: (A, b) = atribuiContorno(G, $dadosContorno$);
12: sol = resolveSistLinear(A,b);
13: **pdeplot**(m,...);

A função *resolveSistLinear* é o ponto central deste trabalho, uma vez que a análise é feita sobre o desempenho das implementações na solução do sistema de equações originado do FEM. Foram utilizados 2 *solvers* a fim de se verificar a performance e a gestão de memória. O primeiro deles é o *solver* "\" do MATLAB®e o segundo é a implementação do método do gradiente conjugado proposta por Barrett et al. (1995). O CG como explicitado no referencial teórico, é um método iterativo não estacionário do subespaço de Krylov. Como coloca Wathen (1989), o uso de métodos iterativos se tornou competitivo à medida em que os sistemas de equações se tornavam maiores. A técnica EBE foi proposta por Hughes, Levit e Winget (1983) como como sendo um precondicionador para o CG de forma a possibilitar que as computações com matrizes sejam realizadas no nível elementar (KISS et al., 2012) de forma a possibilitar a vetorização (ou paralelização) e a economia de memória.

A motivação da escolha do *solver* do MATLAB®("\" ou *mldivide*) está no fato de ele utilizar diferentes algoritmos dependendo da estrutura da matriz de coeficientes, tais como simetria e esparsidade (**??**). Especificamente para o problema deste trabalho, cuja matriz é Hermitiana (auto-adjunta) e positiva definida, o *solver* adota a fatoração de Cholesky ou a LDL.

A motivação para a escolha do CG implementado por Barrett et al. (1995) está no fato de que seu material é citado em diferentes trabalhos da revisão bibliográfica, como em Kiss et al. (2012), Dolwithayakul, Chantrapornchai e Chumchob (2012), Yang, Liu e Chen (2016). Além de um completo *survey* de métodos numéricos para a solução de sistemas lineares, os autores disponibilizam as implementações na linguagem do MATLAB®e em C++. O pseudocódigo do CG precondicionado com a matriz M é apresentado em 2 e a descrição de cada etapa é apresentada na seção **??** da fundamentação teórica.

Função	**Descrição**
load	Carrega para o *workspace* as variáveis salvas nos arquivos *.mat*.
decsg	Cria a geometria do problema por meio da associação de regiões primitivas. O espaço de análise do capacitor de placas paralelas, por exemplo é composto por 2 retângulos e 1 quadrado.
createpde	Instancia um modelo de PDE contendo n equações.
geometryFromEdges	Vincula a geometria originada da função *decsg* ao modelo de PDE gerado por *createpde*.
generateMesh	Cria uma malha sobre a geometria do modelo de PDE. Parâmetros adicionais podem ser incluídos para modificar a qualidade malha ou a ordem dos elementos (ver 2.2.4). O parâmetro $Hmax$ determina o tamanho máximo das arestas dos elementos.
meshToPet	Obtém as matrizes de pontos, arestas e triângulos, as quais serão utilizadas na geração das matrizes elementares e da matriz global.
geraMatElementares	À partir dos dados da triangulação da malha, gera a matriz de cada elemento por meio das funções de aproximação e interpolação.
geraMatGlobal	Realiza a agregação ou o mapeamento das matrizes elementares no sistema global. A matriz resultante é esparsa e sua ordem corresponde ao número de nós da malha.
atribuiContorno	Atribui os m valores de contorno pré-estabelecidos e com isso realiza a redução dos sistema em m ordens, fazendo com que deixe de ser homogêneo.
resolveSistLinear	Consiste na aplicação de métodos numéricos para a solução eficiente de sistemas lineares esparsos.
pdePlot	Função utilizada para a exibição da malha e dos resultados. Conforme os parâmetros adicionais, são plotados gráficos tridimensionais, campos vetoriais e linhas de campo.

Tabela 2 – Descrição das funções utilizadas no algoritmo do FEM

3.1.4 Implementação do EBE-FEM

A implementação do EBE-FEM consiste basicamente em antecipar a solução do sistema de equações, passando da montagem das matrizes elementares diretamente para a atribuição das condições de contorno e solução do sistema, sem que seja necessária a montagem da matriz de coeficientes global. Para realizar esta mudança de rota na solução do problema é necessário modificar tanto o algoritmo quanto a estrutura de dados

Algorithm 2 Pseudocódigo do CG

1: $x_0 = \{0\}$
2: $r_0 = b - Ax_0$
3: **for** i = 1,2... **do**
4: z_{i-1} = resolve(M, r_{i-1})
5: $\rho_{i-1} = {r_{i-1}}^T z_{i-1}$
6: **if** i = 1 **then**
7: $p_1 = z_0$
8: **else**
9: $\beta_{i-1} = \rho_{i-1}$ / ρ_{i-2}
10: $p_i = z_{i-1} + \beta_{i-1} p_{i-1}$
11: **end if**
12: $q_i = Ap_i$
13: $\alpha_i = \rho_{i-1}$ / ${p_i}^T q_i$
14: $x_i = x_{i-1} + \alpha_i p_i$
15: $r_i = r_{i-1}$ - $\alpha_i q_i$
16: **if** converge(r,tol) **then**
17: break
18: **end if**
19: **end for**

utilizados.

A adaptação do bloco referente à atribuição das condições de contorno (linha 11 do algoritmo 1) é baseada na abordagem proposta por Xu, Yin e Mao (2005) a qual consiste na criação de vetores *rhs* (*right hand side*) elementares. Ao invés de se atribuir os valores de contorno às células do vetor *rhs* global que correspondem à cada nó da malha, a atribuição é feita no nível elementar. Por exemplo, se um nó compartilhado por 2 elementos possui valor de contorno igual a 10, a atribuição é feita como uma média no vetor *rhs* de cada um dos 2 elementos. Assim sendo, cada vetor recebe o valor de contorno igual a 5. A figura 5 contém o esquema que ilustra este exemplo. Yan et al. (2017) e Wu et al. (2015) propõe uma abordagem parecida baseada não no número de conexões do nó, mas na média ponderada do valor de contorno original pelos coeficientes de cada elemento que contém o nó.

De posse das matrizes e vetores elementares com as condições de contorno atribuídas, é feita a adaptação no algoritmo do CG a fim de que os cálculos envolvendo matrizes sejam feitos elemento a elemento. As operações entre vetores e escalares são mantidas no nível global, visto que não requerem tanto esforço computacional como as operações matriciais. As adaptações consistem basicamente em percorrer os N elementos, realizando os cálculos pertinentes e em seguida, armazenar nos vetores globais os resultados gerados para cada nó do elemento. As linhas 4 e 12 do algoritmo 2 possuem operações matriciais e são modificadas conforme apresentado nos pseudocódigos 3 e 4. Além disso, o resíduo inicial da linha 2 do 2 é igual a b , dado que o chute x_0 vale zero para todos os nós. Como os

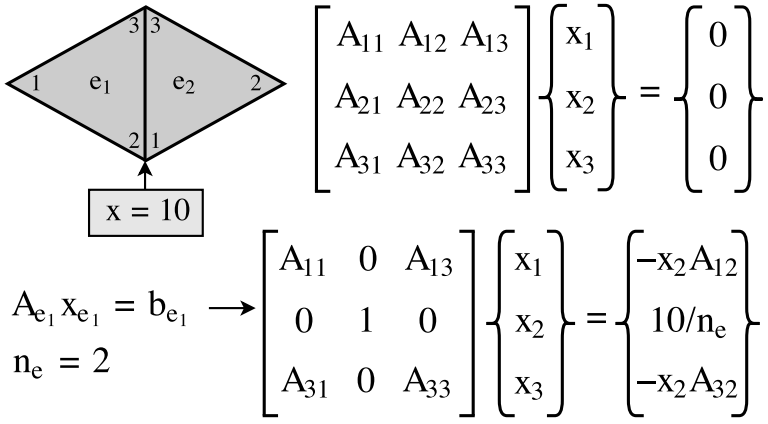


Figura 5 – Esquema da aplicação das condições de contorno de um elemento

valores de b foram atribuídos pelas condições de contorno em nível elementar, é necessário realizar a montagem do vetor b global para a determinação do resíduo inicial aconteça. Este processo de montagem é apresentado no pseudocódigo 5.

Algorithm 3 Adaptação da declaração $z_{i-1} = \text{resolve}(M,\ r_{i-1})$

```
1: z = {0}
2: for j = 1,2...|tri| do
3:     map = tri(j)
4:     M_e = precond(A_e(j))
5:     z(map) = z(map) + M_e r(map)
6: end for
```

Algorithm 4 Adaptação da declaração $q_i = Ap_i$

```
1: q = {0}
2: for j = 1,2...|tri| do
3:     map = tri(j)
4:     q(map) = q(map) + A_e(j) q(map)
5: end for
```

Algorithm 5 Adaptação da declaração $r_0 = b - Ax_0$

```
1: b = {0}
2: for j = 1,2...|tri| do
3:     map = tri(j)
4:     b(map) = b(map) + b_e(j)
5: end for
6: r_0 = b
```

3.1.5 Coloração da malha

Proposta por Wathen (1989) e implementada por Kiss et al. (2012) a coloração da malha possibilita que os trechos do CG descritos nos algoritmos 3, 4 e 5 sejam paralelizados.

A coloração assim como o processo de exclusão mútua são estratégias adotadas para evitar condições de corrida. Condição de corrida é o estado inconsistente dos dados causado por transações não atômicas. Desta forma, se por exemplo, dois processos ou *threads* alteram ao mesmo tempo um dado em memória compartilhada, não se pode prever o valor final do dado, mas sabe-se que será inconsistente. A coloração deve acontecer de tal forma que elementos que compartilhem um mesmo nó possuam cores distintas. Uma característica desejável num algoritmo otimizado é que haja uma distribuição equalizada na quantidade elementos de cada cor, ao invés de uma quantidade mínima de cores. Por exemplo, numa malha de 1000 elementos é preferível se utilizar 20 cores com 50 elementos de cada cor, ao invés de 6 cores com uma distribuição desequilibrada de elementos, como por exemplo {500 50 200 150 10 90}. Contudo, como o objetivo da etapa de *validação do problema* não é obter resultados ótimos, foi adotada uma heurística gulosa sem balanceamento de cores, mas que apenas prioriza a aplicação das cores menos utilizadas. A figura 6 mostra um zoom na coloração da malha no topo entre as duas placas.

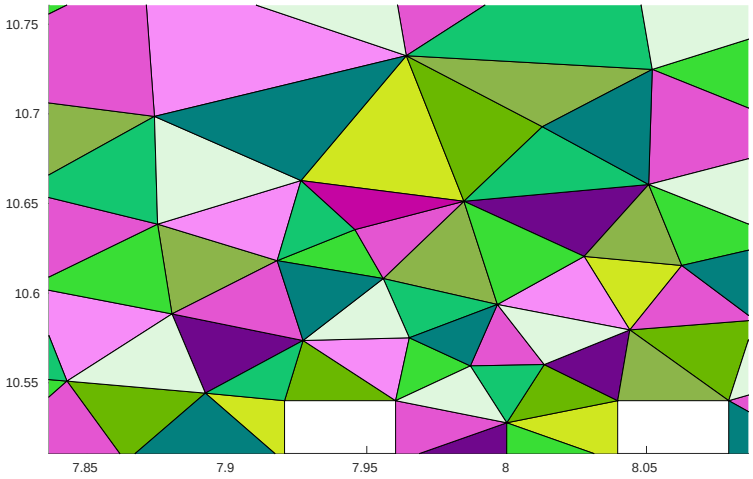


Figura 6 – Exemplo de coloração da malha

Uma vez realizada a coloração, o processamento pode ocorrer de tal forma que elementos da mesma cor seja executados simultaneamente e elementos de cores distintas sequencialmente. Como elementos de uma determinada cor não possuem vértices em comum, não há a possibilidade de acontecer condição de corrida durante o acesso às variáveis globais z e r do algoritmo 6 e q e d dos algoritmos 7 e 8 respectivamente. A estrutura de dados *cores* contém os dados da localização do elemento na malha.

3.1.6 Paralelização do algoritmo

A fim de manter a conformidade com o ambiente de desenvolvimento optou-se por utilizar a *parallel Computing toolbox* do MATLAB®. Esta ferramenta possibilita a implementação para plataformas *multicore*, GPUs, *clusters*, *grids* e *cloud*. De acordo com a documentação do MATLAB®(**??**), funções de álgebra linear como fft, $mldivide$, eig,

Algorithm 6 Aplicação da coloração no algoritmo 3

$z = \{0\}$
for c = 1,2...$|cores|$ **do**
    $triCor = cores(c)$
    for j = 1,2...$|triCor|$ **do**
        $idG = triCor(j)$
        $map = tri(idG)$
        $M_e = precond(A_e(idG))$
        $z(map) = z(map) + M_e r(map)$
    end for
end for

Algorithm 7 Aplicação da coloração no algoritmo 4

$q = \{0\}$
for c = 1,2...$|cores|$ **do**
    $triCor = cores(c)$
    for j = 1,2...$|triCor|$ **do**
        $idG = triCor(j)$
        $map = tri(id)$
        $q(map) = q(map) + A_e(idG)q(map)$
    end for
end for

Algorithm 8 Aplicação da coloração no algoritmo 5

$b = \{0\}$
for c = 1,2...$|cores|$ **do**
    $triCor = cores(c)$
    for j = 1,2...$|triCor|$ **do**
        $idG = triCor(j)$
        $map = tri(idG)$
        $b(map) = b(map) + b_e(idG)$
    end for
end for
$r_0 = b$

svd, e *sort* são *multithread* desde a versão R2018a. No entanto, é possível melhorar a performance de funções customizadas por meio da *parallel toolbox*.

A fim de garantir a robustez do código paralelizado, o MATLAB®disponibiliza a programação em múltiplos *workers* ao invés de *threads*. Cada *worker* é um processo que trabalha em função do processo principal (*client*) que o originou. Desta forma, as tarefas são divididas entre os *workers* e coordenadas pelo *client*, num modelo mestre-escravo. A fim de evitar condições de corrida e o *overhead* de comunicação entre os processos, a ferramenta trabalha por meio de distribuição e troca de mensagens (MPI) ao invés do compartilhamento de memória. Assim sendo, os dados são copiados para cada *worker* e ao final do processamento, agregados novamente no *client*. A tabela 3 contém a descrição das funções utilizadas da *parallel toolbox*.

Para que a paralelização do código ocorra por meio das funções *parfor* e *spmd* é necessário que a estrutura de dados da coloração seja modificada, a fim de que possa ser "fatiada" e distribuída entre os *workers*. Como pode ser visto nos algoritmos de 6 a 8, as variáveis globais são indexadas pela variável *map*, que contém a lista dos identificadores de cada nó do elemento, como por exemplo {25 12 96}. Como estes índices não são contíguos, ocorre um erro durante a compilação do *parfor* e o *overhead* de comunicação no *spmd*. Esta restrição do MATLAB®demanda uma nova adaptação da estrutura de dados e impede que a atribuição às variáveis globais seja feita de forma paralela.

A nova estrutura de dados gerada durante a coloração já retorna o vetor b (ou rhs) completo, dessa forma, não há a necessidade do processo de montagem à partir dos vetores elementares, como descrito nos algoritmos 5 e 8. Além dessa melhoria, o objeto retornado da coloração possui para cada cor, as matrizes elementares e o vetor rhs referente aos elementos da cor. Em resumo, a nova abordagem retorna os próprios objetos que serão utilizados na paralelização ao invés do seu "endereço" na malha. Os pseudocódigos 9 e 10 mostram a aplicação das funções *parfor* e *spmd* no algoritmo 6. A mesma adaptação é feita para o algoritmo 7.

Algorithm 9 Aplicação do *parfor* no algoritmo 6

```
1:  for c = 1,2...|cores| do
2:      matElCor = cores(c).matEl
3:      rhsCor = cores(c).rhs
4:      R = reshape(r(rhsCor), 3, [ ])
5:      PARFOR
6:      for j = 1,2...|cores(c)| do
7:          M_e = precond(matElCor(j))
8:          Z = M_e R(j)
9:      end for
10:     END PARFOR
11:     z(rhsCor) = z(rhsCor) + reshape(Z, 1, [ ])
12: end for
```

Função	Descrição
gcp	*Get current parallel pool* retorna as informações sobre os *workers* ativos.
parpool	Cria uma nova *parallel pool*, ou seja, um conjunto de *workers* e os demais processos necessários para o funcionamento da *toolbox*.
parfor	Executa as iterações de um *loop* em *workers*. Para que isso seja possível as iterações devem ser independentes entre si e as estruturas de dados (coleções) utilizadas devem ter a característica de serem "fatiadas". Por exemplo, uma matriz bidimensional pode ser fatiada de 2 formas, pelas linhas ou pelas colunas.
spmd	A função *single program, multiple data* executando o mesmo código em diferentes *workers*. Diferente do *parfor*, os dados não precisam necessariamente estar particionados, visto que o *spmd* permite a comunicação entre os *workers*.
codistributed	Cria uma coleção que é distribuída entre os *workers* e não copiada integralmente.
getLocalPart	Obtém os dados de uma determinada partição de uma coleção distribuída.
codistributor	Função que realiza a distribuição de uma coleção. Diferente da distribuição limitada que ocorre no *parfor*, essa função possibilita que a distribuição seja feita de diferentes formas em uma coleção.
gather	Função que realiza a agregação de uma coleção distribuída.
drange	Executa um *loop* entre as partições de uma coleção. Este bloco de código faz com que os *workers* trabalhem simultaneamente em cada distribuição da coleção, mas impede a comunicação entre eles.

Tabela 3 – Descrição das funções utilizadas na paralelização do EbE-FEM

Enquanto a paralelização por *parfor* é feita elemento a elemento, a função *spmd* distribui o processamento em q partes, sendo q a quantidade de *workers*. Cada *worker* executa sequencialmente a parcela do problema que lhe é atribuída. Conforme a recomendação da documentação, para se obter melhor desempenho, a quantidade de *workers* deve ser a mesma de núcleos físicos do processador. Ao se utilizar o bloco $for - drange$ na linha 13 ao invés do for convencional, o comportamento do *spdm* passa a ser equivalente ao do $parfor$.

Algorithm 10 Aplicação do *spmd* no algoritmo. 6

```
1: for c = 1,2...|cores| do
2:     matElCor = cores(c).matEl
3:     rhsCor = cores(c).rhs
4:     R = reshape(r(rhsCor), 3, [ ])
5:     SPMD
6:     codMat = codistributor(formaDistMat)
7:     MatDist = codistributed(matElCor, codMat)
8:     codR = codistributor(formaDistR)
9:     RDist = codistributed(R, codR)
10:    MatPart = getLocalPart(MatDist)
11:    RPart = getLocalPart(RDist)
12:    for j = 1,2...qtdWorkers do
13:        for k = 1,2...|cores(c)| do
14:            M_e = precond(MatPart(j, k))
15:            Z = M_e RPart(j, k)
16:        end for
17:    end for
18:    END SPMD
19:    z(rhsCor) = z(rhsCor) + reshape(Z, 1, [ ])
20: end for
```

3.2 Resultados preliminares

A ferramenta *pdetool* forneça a opção de refinamento da malha, que consiste na subdivisão dos triângulos já existentes, no entanto, o refinamento da malha do objeto gerada pela função generateMesh é feito de forma diferente, por meio da alteração do parâmetro *Hmax* que estabelece o tamanho máximo de cada aresta. A vantagem dessa abordagem é o maior controle das propriedades da malha, como por exemplo a quantidade de elementos, no entanto, a desvantagem é que a cada alteração neste parâmetro uma nova malha é gerada, em diferentes pontos. Dessa forma, não se pode precisar com exatidão do valor em cada um dos pontos de referência apresentados nas figuras 4(c) e 4(d), no entanto, a corretude da aproximação pode ser verificada na média pelos nós que estão entorno de cada ponto de referência ou ainda considerando o valor do nó mais próximo. A figura **??** destaca os nós da vizinhança de cada ponto de referência. Foram adotadas regiões quadradas de lado igual a $3mm$, $5mm$ e $10mm$, de acordo com a localização do ponto de referencia.

Referências

AHAMED, A.-K. C.; MAGOULÈS, F. Conjugate gradient method with graphics processing unit acceleration: CUDA vs OpenCL. *Advances in Engineering Software*, v. 47, n. 1, p. 164–169, 2016. ISSN 0965-9978. Disponível em: <http://dx.doi.org/10.1016/j.advengsoft.2011.12.013>. Citado 2 vezes nas páginas 3 e 6.

ANZT, H. et al. Preconditioned Krylov solvers on GPUs. *Parallel Computing*, Elsevier B.V., v. 0, p. 1–13, 2016. ISSN 01678191. Disponível em: <http://dx.doi.org/10.1016/j.parco.2017.05.006>. Citado na página 3.

BARRETT, R. et al. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. *Mathematics of Computation*, v. 64, n. 211, p. 1349, 1995. ISSN 00255718. Disponível em: <http://www.jstor.org/stable/2153507?origin=crossref>. Citado na página 18.

BOEHMER, S. et al. Numerical simulation of electrical machines by means of a hybrid parallelisation using MPI and OpenMP for finite-element method. *IET Science, Measurement & Technology*, v. 6, n. 5, p. 339, 2011. ISSN 17518822. Disponível em: <http://digital-library.theiet.org/content/journals/10.1049/iet-smt.2011.0126>. Citado na página 6.

BOYCE, W. E.; DIPRIMA, R. C. *Equacões Diferenciais Elementares e Problemas de Valores de Contorno*. [S.l.]: LTC, 2010. ISBN 978521617563. Citado 2 vezes nas páginas 8 e 9.

BOYLESTAD, R. L. *Análise de Circuitos*. [S.l.]: Pearson, 2011. ISBN 978-85-64574-20-5. Citado 2 vezes nas páginas 4 e 15.

CAREY, G. F. et al. Element-by-element vector and parallel computations. *Communications in Applied Numerical Methods*, v. 4, n. 3, p. 299–307, 1988. ISSN 0748-8025. Disponível em: <http://doi.wiley.com/10.1002/cnm.1630040303>. Citado na página 6.

CHOU, C.-y.; CHEN, K.-t. Performance Evaluations of Different Parallel Programming Paradigms for Pennes Bioheat Equations and Navier-Stokes Equations. p. 503–508, 2016. Citado na página 7.

DAYDE, M. J.; L'EXCELLENT, J.-Y.; GOULD, N. I. M. On the Use of Element-by-Element Preconditioners to Solve Large Scale Partially Separable Optimization. *Computing*, 1995. Citado 2 vezes nas páginas 3 e 6.

DESAI, J. A. C. S. *Introduction to the Finite Element Method: A Numerical Method for Engineering Analysis*. [S.l.]: Van Nostrand Reinhold Company, 1972. Citado 2 vezes nas páginas 10 e 11.

DOLWITHAYAKUL, B.; CHANTRAPORNCHAI, C.; CHUMCHOB, N. An efficient asynchronous approach for Gauss-Seidel iterative solver for FDM/FEM equations on multi-core processors. *JCSSE 2012 - 9th International Joint Conference on Computer Science and Software Engineering*, p. 357–361, 2012. Citado na página 18.

ERLANG Market Share. <https://w3techs.com/technologies/details/pl-erlang/all/all>. Acesso em: 19/07/2017. Nenhuma citação no texto.

FAIRES, R. L. B. J. D. *Análise Numérica*. [S.l.]: CENGAGE Learning, 2008. ISBN 978-85-221-0601-1. Citado na página 9.

GUO, X. et al. Developing a scalable hybrid MPI/OpenMP unstructured finite element model. *Computers and Fluids*, Elsevier Ltd, v. 110, p. 227–234, 2014. ISSN 00457930. Disponível em: <http://dx.doi.org/10.1016/j.compfluid.2014.09.007>. Citado na página 4.

HU, J.; QUIGLEY, S. F.; CHAN, A. An element-by-element preconditioned conjugate gradient solver of 3d tetrahedral finite elements on an FPGA coprocessor. *Proceedings - 2008 International Conference on Field Programmable Logic and Applications, FPL*, p. 575–578, 2008. Citado 2 vezes nas páginas 6 e 7.

HUEBNER, K. H. et al. *The Finite Element Method for Engineers*. [S.l.]: John Wiley & sons, 2001. ISBN 978-0-471-37078-9. Citado 3 vezes nas páginas 9, 10 e 11.

HUGHES, T.; LEVIT, I.; WINGET, J. Element-by-element implicit algorithms for heat conduction. *Journal of Engineering Mechanics*, v. 109, n. 2, p. 576–585, 1983. ISSN 07339399. Citado 3 vezes nas páginas 3, 6 e 18.

IWASHITA, T. et al. Software framework for parallel BEM analyses with H-matrices. *IEEE CEFC 2016 - 17th Biennial Conference on Electromagnetic Field Computation*, Elsevier B.V., v. 108, n. June, p. 2200–2209, 2017. ISSN 18770509. Disponível em: <http://dx.doi.org/10.1016/j.procs.2017.05.263>. Citado na página 3.

JIN, J. *The Finite Element Method in Electromagnetics*. [S.l.]: John Wiley & sons, 2002. ISBN 0471438189. Citado 4 vezes nas páginas 8, 10, 11 e 12.

KISS, I. et al. High locality and increased intra-node parallelism for solving finite element models on GPUs by novel element-by-element implementation. In: *2012 IEEE Conference on High Performance Extreme Computing, HPEC 2012*. [S.l.: s.n.], 2012. ISBN 9781467315760. Citado 5 vezes nas páginas 3, 6, 7, 18 e 21.

LEVIT, I. Element by element solvers of order n. v. 27, n. 3, p. 357–360, 1987. Citado na página 6.

MATHWORKS. *Documentation: mldivide*. [S.l.]. Nenhuma citação no texto.

POWERS, D. L. *Boundary Value Problems and Partial Differential Equations*. [S.l.]: Elsevier, 2006. ISBN 139780125637381. Citado na página 9.

PROCESSADOR Power9 da IBM. <http://www.ibmsystemsmag.com/power/businessstrategy/competitiveadvantage/POWER9-Plans/>. Acesso em: 19/07/2017. Nenhuma citação no texto.

PROCESSADORES Intel. <https://www.intel.com/content/www/us/en/products/processors.html>. Acesso em: 19/07/2017. Nenhuma citação no texto.

PRODUTOS AMD. <http://www.amd.com/pt-br/products>. Acesso em: 19/07/2017. Nenhuma citação no texto.

PRODUTOS NVIDIA. <http://www.nvidia.com.br/page/products.html>. Acesso em: 19/07/2017. Nenhuma citação no texto.

REDDY, J. N. *An Introduction to the Finite Element Method*. [S.l.]: McGraw Hill, 2006. ISBN 0071244735. Citado 2 vezes nas páginas 9 e 13.

SADIKU, M. N. O. *Numerical Techniques in Electromagnetics*. [S.l.]: CRC Press, 2001. ISBN 0-8493-1395-3. Citado 2 vezes nas páginas 9 e 11.

SCALA Market Share. <https://w3techs.com/technologies/details/pl-scala/all/all>. Acesso em: 19/07/2017. Nenhuma citação no texto.

SZABO, B.; BABUŠKA, I. *An Introduction to Finite Element Analysis - Ch4 Generalized Formulations*. [S.l.: s.n.], 2009. 109–144 p. ISBN 9780470977286. Citado na página 3.

VOLAKIS, J. L.; CHATTERJEE, A.; KEMPEL, L. C. *Finite Element Method for Electromagnetics*. [S.l.]: John Wiley & sons, 1998. ISBN 0780334256. Citado 2 vezes nas páginas 12 e 13.

WATHEN, A. J. An analysis of some element-by-element techniques. *Computer Methods in Applied Mechanics and Engineering*, v. 74, n. 3, p. 271–287, 1989. ISSN 00457825. Citado 3 vezes nas páginas 3, 18 e 21.

WU, D. et al. GPU Acceleration of EBE Method for 3-D Linear Steady Eddy Current Field. 2015. Citado 2 vezes nas páginas 7 e 20.

XU, J.; YIN, W. Y.; MAO, J. Capacitance extraction of high-density 3D interconnects using finite element method. *Asia-Pacific Microwave Conference Proceedings, APMC*, v. 2, p. 3–5, 2005. Citado 2 vezes nas páginas 7 e 20.

YAN, X. et al. Research on Preconditioned Conjugate Gradient Method Based on EBE-FEM and the Application in Electromagnetic Field Analysis. v. 53, n. 6, 2017. Disponível em: <http://www.ieee.org/publications{_}standards/publications/rights/index.h>. Citado 2 vezes nas páginas 7 e 20.

YANG, B.; LIU, H.; CHEN, Z. Preconditioned GMRES solver on multiple-GPU architecture. *Computers and Mathematics with Applications*, Elsevier Ltd, v. 72, n. 4, p. 1076–1095, 2016. ISSN 08981221. Disponível em: <http://dx.doi.org/10.1016/j.camwa.2016.06.027>. Citado na página 18.

YAO, L. et al. Parallel implementation and performance comparison of BiCGStab for massive sparse linear system of equations on GPU libraries. *Proceedings - 2015 IEEE 12th International Conference on Ubiquitous Intelligence and Computing, 2015 IEEE 12th International Conference on Advanced and Trusted Computing, 2015 IEEE 15th International Conference on Scalable Computing and Communications, 20*, p. 603–608, 2015. Citado 2 vezes nas páginas 3 e 7.

ZIENKIEWICZ, R. L. T. J. Z. C. *The Finite Element Method its Basis & Fundamentals*. [S.l.]: Elsevier, 2005. ISBN 0750663200. Citado 3 vezes nas páginas 9, 10 e 12.